Digital Signal Processing Lab

Laboratory report submitted for the partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

by

Mohit Akhouri - 19ucc023

Course Coordinator
Dr. Divyang Rawal

Department of Electronics and Communication Engineering
The LNM Institute of Information Technology, Jaipur

September 2021

Contents

Chapter *1*

Experiment - 1

1.1 Aim of the Experiment

- Signal Sampling and Reconstruction
- Introduction to Simulink

1.2 Software Used

- MATLAB
- Simulink

1.3 Theory

1.3.1 About Sampling :

In signal processing, Sampling is the reduction of a continuous-time signal to a discrete-time signal. A common example could be the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal). A **sample** is a value or set of values at a point in time and/or space. A **sampler** is a subsystem or operation that extracts samples from a continuous signal. For functions that vary with time, let s(t) be a continuous function (or ”signal”) to be sampled, and let sampling be performed by measuring the value of the continuous function every T seconds, which is called the sampling interval or the sampling period.Then the sampled function is given by the sequence:

$$S(nT) \tag{1.1}$$

where **n** can be any integer. The **sampling frequency** or **sampling rate**, f_s, is the average number of samples obtained in one second (samples per second), thus $f_s = 1/T_s$. The original signal is retrievable from a sequence of samples, up to the **Nyquist limit**, by passing the sequence of samples through a type of low pass filter called a **reconstruction filter**.

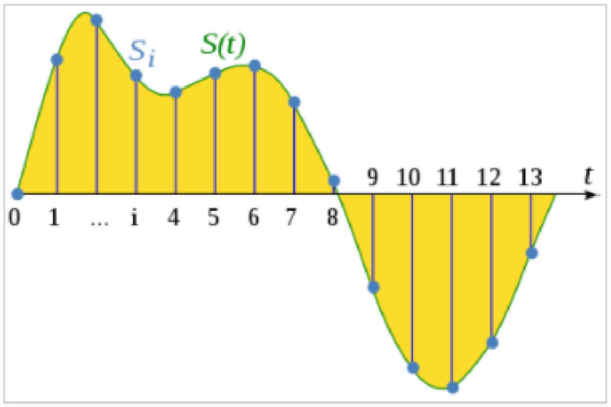


Figure 1.1 Signal Sampling

In the above figure, **green lines** represent the continuous signal and **blue vertical lines** represent the sampled signal.

1.3.2 About Signal Reconstruction :

In signal processing, reconstruction usually means the determination of an original continuous signal from a sequence of equally spaced samples. The process of reconstruction, also commonly known as **interpolation**, produces a continuous time signal that would sample to a given discrete time signal at a specific sampling rate. Reconstruction can be mathematically understood by first generating a continuous time impulse train

$$x_{imp}(t) = \sum_{n=-\infty}^{\infty} x_s(n)\delta(t - nT_s) \tag{1.2}$$

from the sampled signal x_s with sampling period T_s and then applying a lowpass filter G that satisfies certain conditions to produce an output signal $\overline{x}$. Here $\overline{x}$ is given as :

$$\overline{x} = (x_{imp} * g)(t) = \sum_{n=-\infty}^{\infty} x_s(n)g(t - nT_s) \tag{1.3}$$

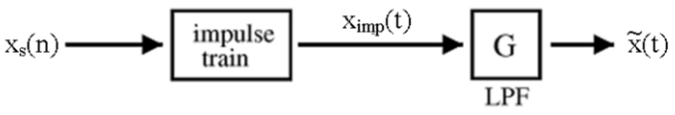


Figure 1.2 Block diagram for Signal Reconstruction

1.3.3 About Interpolation :

In the domain of digital signal processing, the term interpolation refers to the process of converting a sampled digital signal (such as a sampled audio signal) to that of a higher sampling rate (**Upsampling**) using various digital filtering techniques (for example, convolution with a frequency-limited impulse signal). In this application there is a specific requirement that the harmonic content of the original signal be preserved without creating aliased harmonic content of the original signal above the original **Nyquist** limit of the signal (that is, above $f_s/2$ of the original signal sample rate). There are different types of Interpolation techniques which are as follows :

- Nearest-neighbour interpolation
- Linear Interpolation
- Polynomial Interpolation
- Spline Interpolation

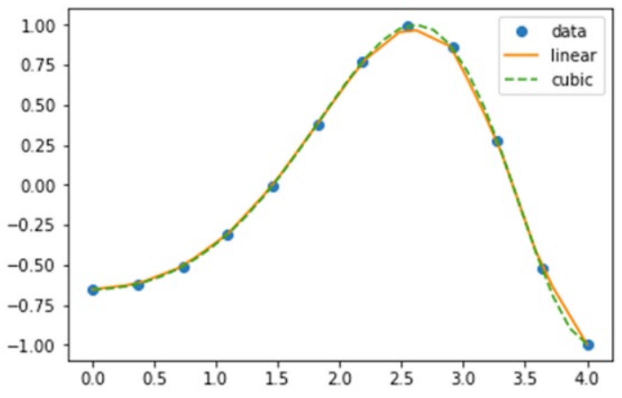


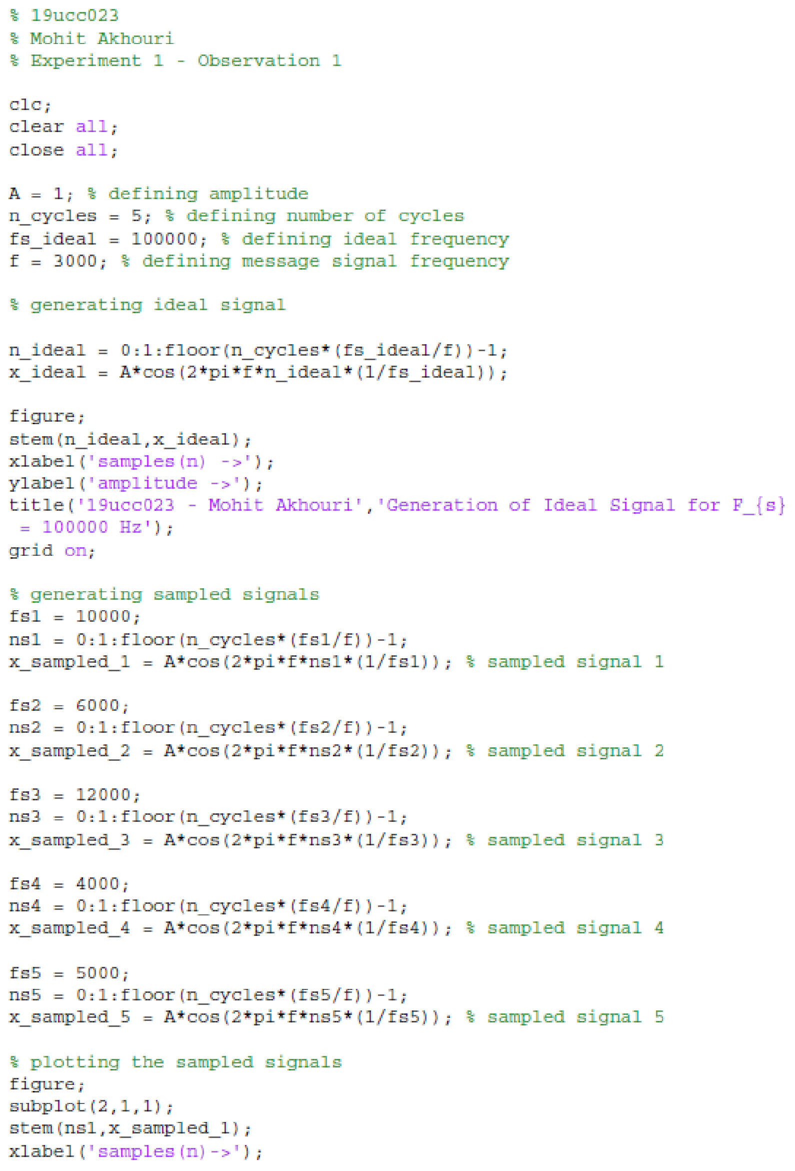
Figure 1.3 Linear and Cubic Interpolation

1.3.4 About Simulink :

Simulink is a MATLAB-based **graphical programming** environment for modeling, simulating and analyzing **multidomain** dynamical systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in **automatic control** and **digital signal processing** for **multidomain** simulation and model-based design.

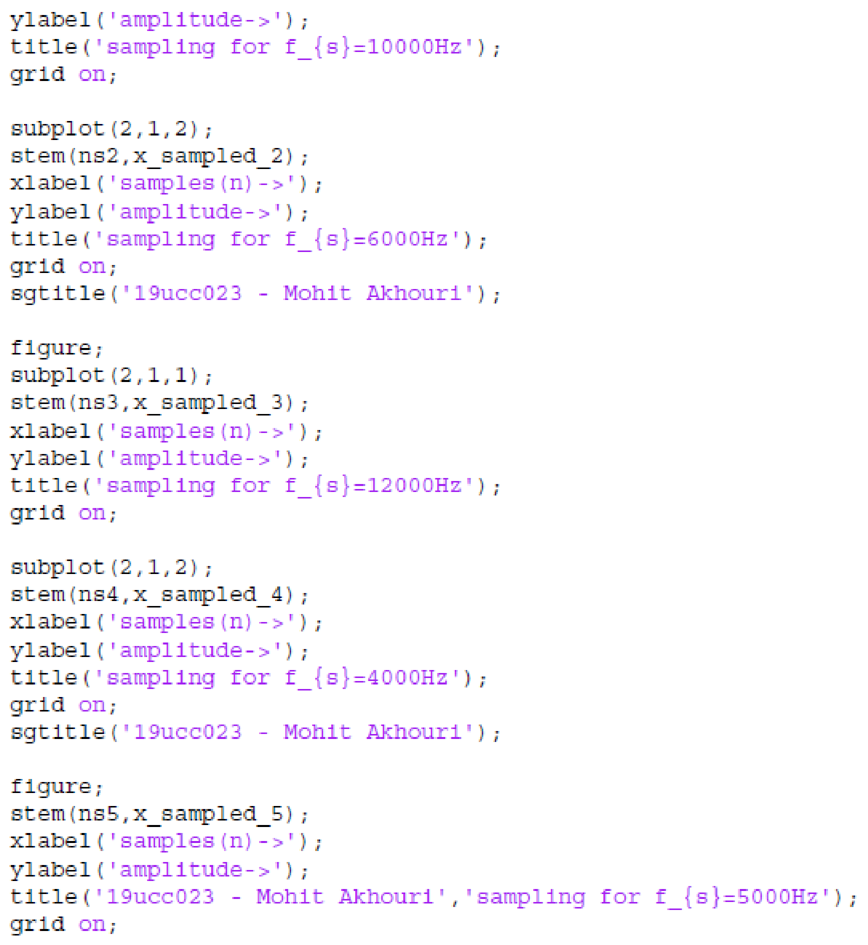
1.4 Code and results

1.4.1 Plot 5 cycles of input signal for different sampling frequencies :



```
% 19ucc023
% Mohit Akhouri
% Experiment 1 - Observation 1

clc;
clear all;
close all;

A = 1; % defining amplitude
n_cycles = 5; % defining number of cycles
fs_ideal = 100000; % defining ideal frequency
f = 3000; % defining message signal frequency

% generating ideal signal

n_ideal = 0:1:floor(n_cycles*(fs_ideal/f))-1;
x_ideal = A*cos(2*pi*f*n_ideal*(1/fs_ideal));

figure;
stem(n_ideal,x_ideal);
xlabel('samples(n) ->');
ylabel('amplitude ->');
title('19ucc023 - Mohit Akhouri','Generation of Ideal Signal for F_{s}
 = 100000 Hz');
grid on;

% generating sampled signals
fs1 = 10000;
ns1 = 0:1:floor(n_cycles*(fs1/f))-1;
x_sampled_1 = A*cos(2*pi*f*ns1*(1/fs1)); % sampled signal 1

fs2 = 6000;
ns2 = 0:1:floor(n_cycles*(fs2/f))-1;
x_sampled_2 = A*cos(2*pi*f*ns2*(1/fs2)); % sampled signal 2

fs3 = 12000;
ns3 = 0:1:floor(n_cycles*(fs3/f))-1;
x_sampled_3 = A*cos(2*pi*f*ns3*(1/fs3)); % sampled signal 3

fs4 = 4000;
ns4 = 0:1:floor(n_cycles*(fs4/f))-1;
x_sampled_4 = A*cos(2*pi*f*ns4*(1/fs4)); % sampled signal 4

fs5 = 5000;
ns5 = 0:1:floor(n_cycles*(fs5/f))-1;
x_sampled_5 = A*cos(2*pi*f*ns5*(1/fs5)); % sampled signal 5

% plotting the sampled signals
figure;
subplot(2,1,1);
stem(ns1,x_sampled_1);
xlabel('samples(n)->');
```

Figure 1.4 Part 1 of the code for observation 1



```
ylabel('amplitude->');
title('sampling for f_{s}=10000Hz');
grid on;

subplot(2,1,2);
stem(ns2,x_sampled_2);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=6000Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

figure;
subplot(2,1,1);
stem(ns3,x_sampled_3);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=12000Hz');
grid on;

subplot(2,1,2);
stem(ns4,x_sampled_4);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=4000Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

figure;
stem(ns5,x_sampled_5);
xlabel('samples(n)->');
ylabel('amplitude->');
title('19ucc023 - Mohit Akhouri','sampling for f_{s}=5000Hz');
grid on;
```

Figure 1.5 Part 2 of the code for observation 1

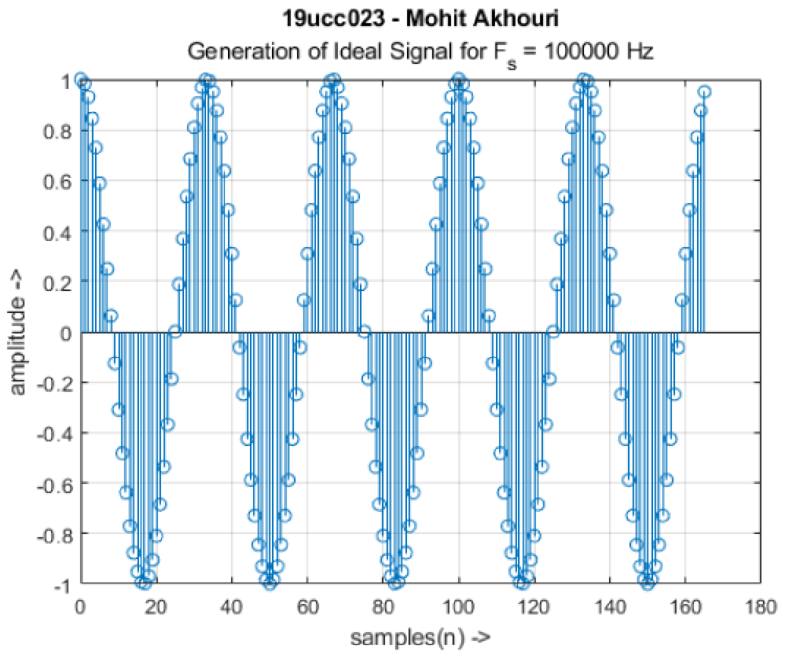


Figure 1.6 Ideal Signal For Sampling frequency = 100000 Hz

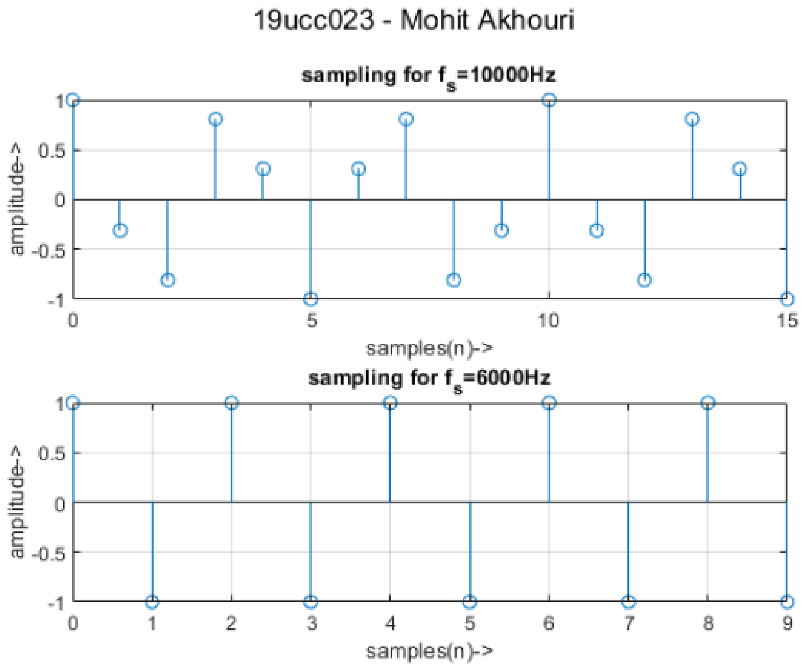


Figure 1.7 Plot of the sampled signals for F_s = 10000 Hz and 6000 Hz

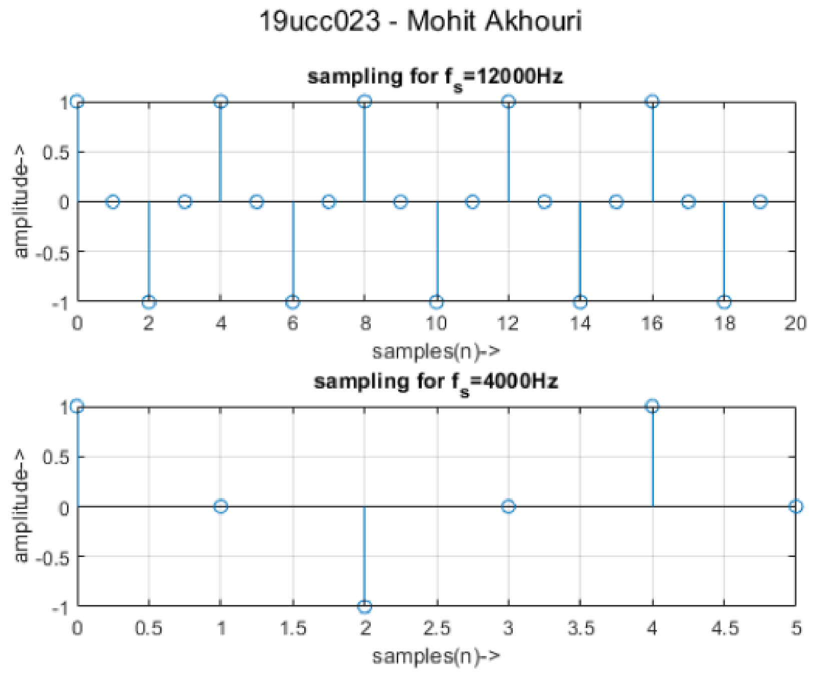


Figure 1.8 Plot of the sampled signals for F_s = 12000 Hz and 4000 Hz

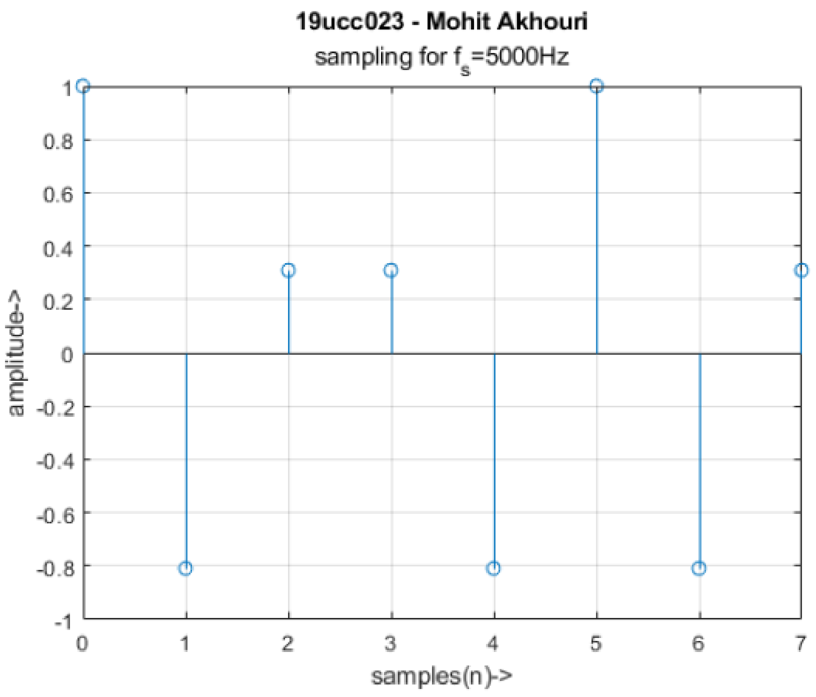
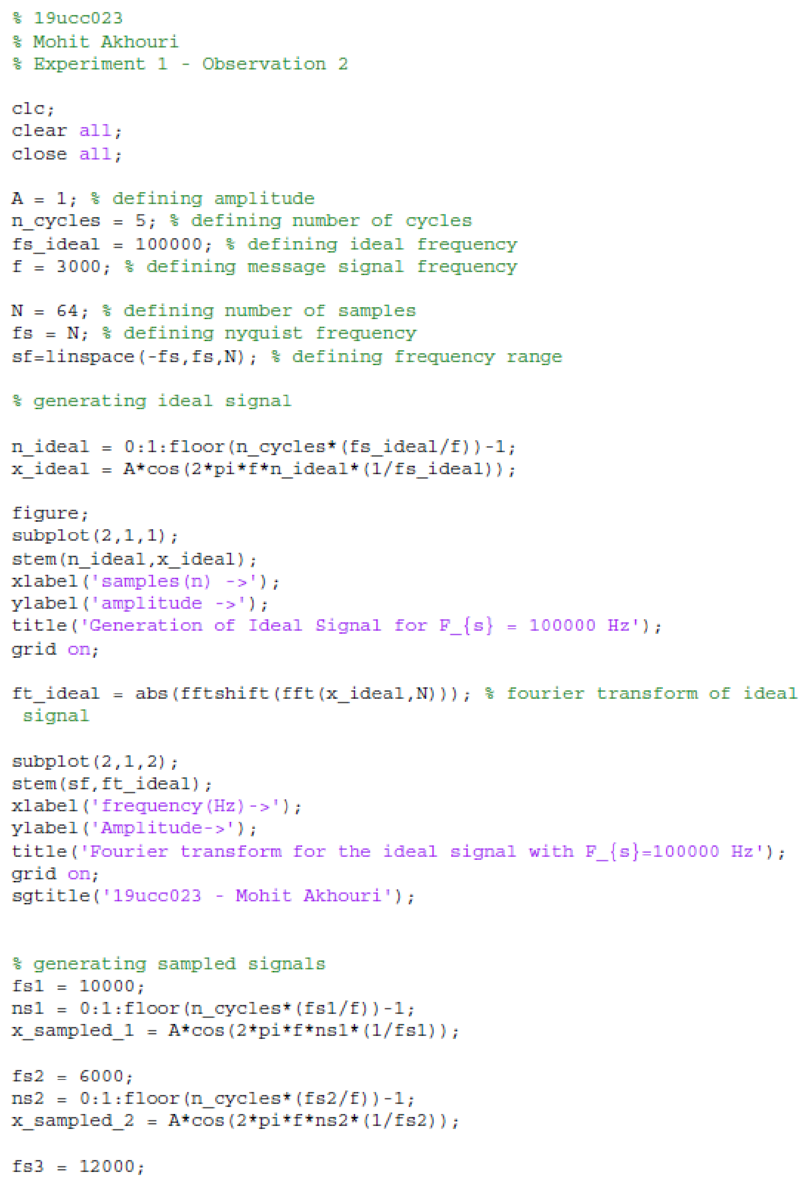


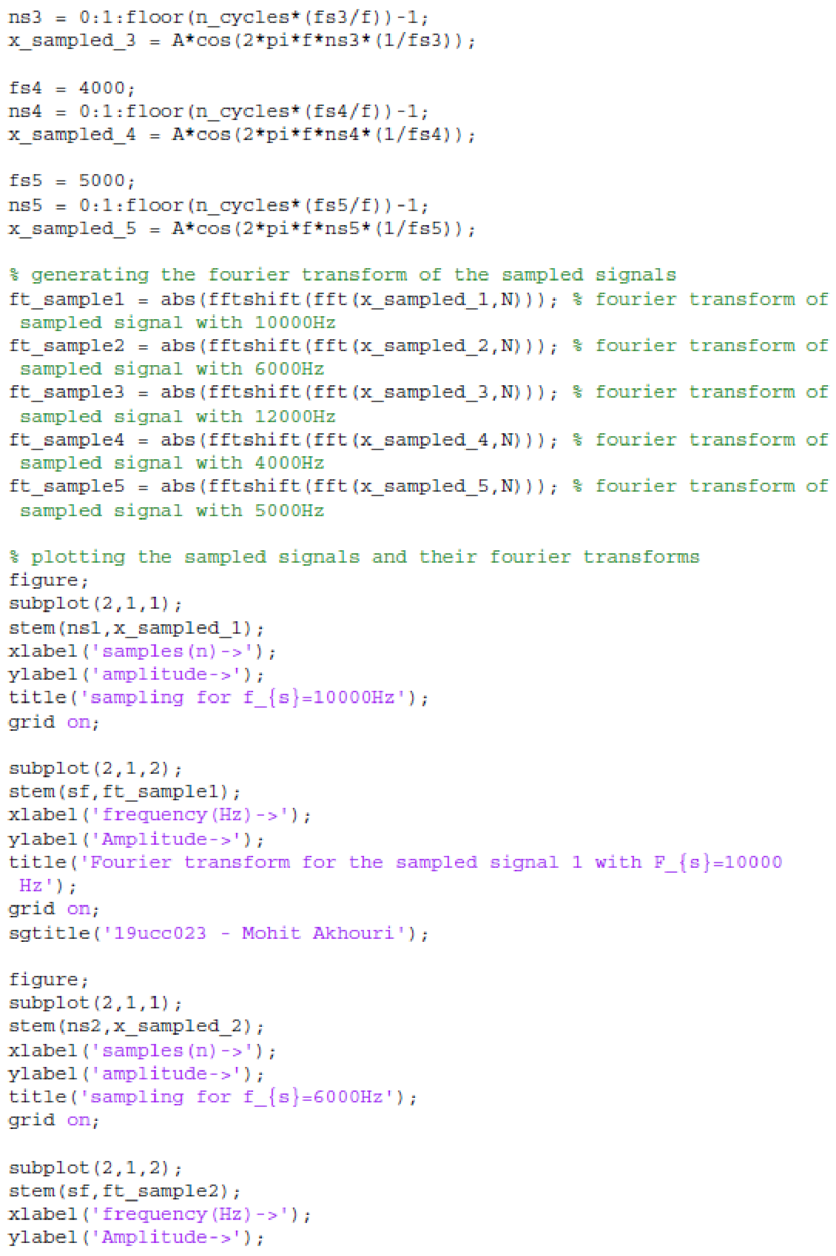
Figure 1.9 Plot of the sampled signal for F_s = 5000 Hz

1.4.2 Plot Fourier Spectra and determine the aliasing effect :



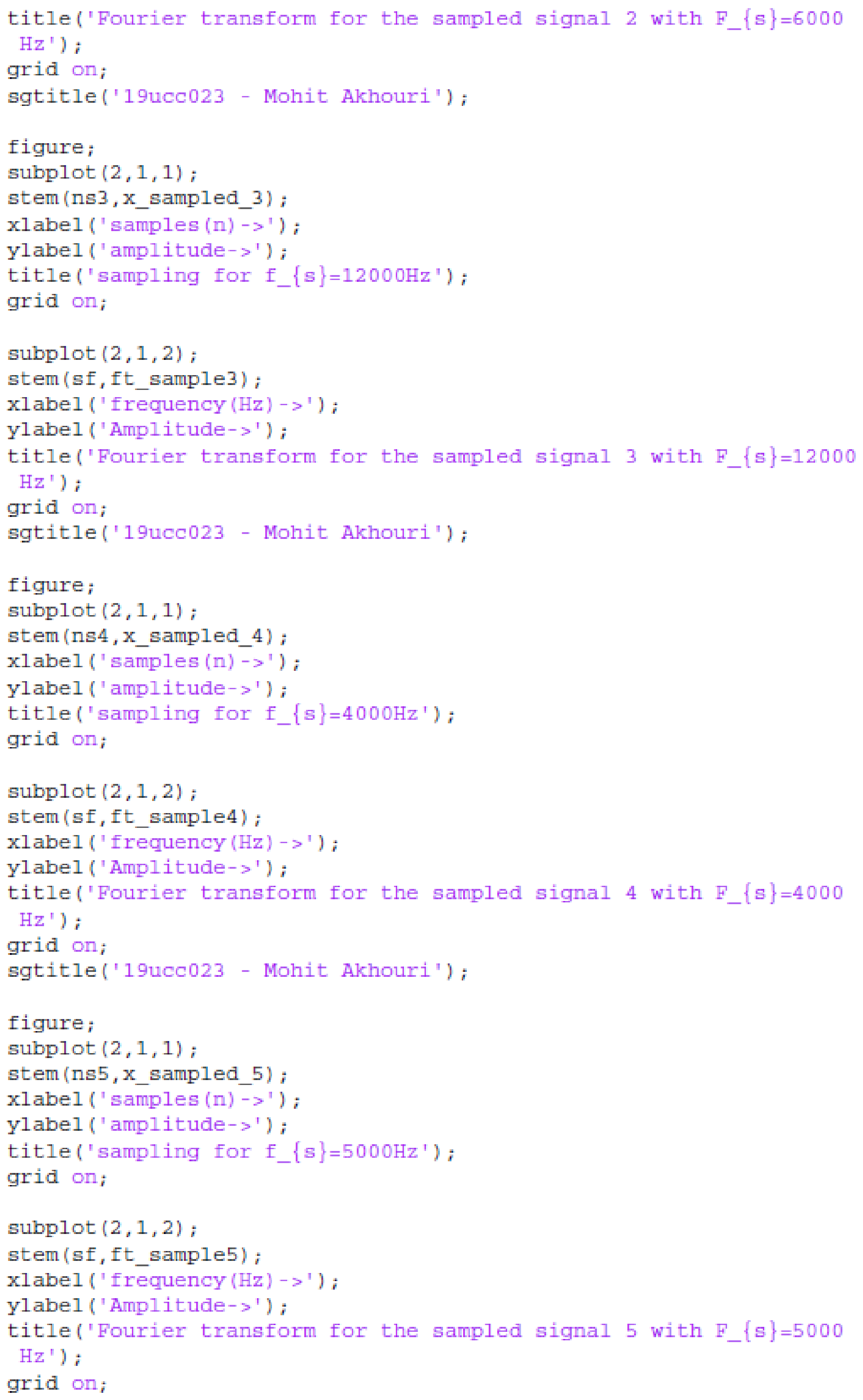
```
% 19ucc023
% Mohit Akhouri
% Experiment 1 - Observation 2

clc;
clear all;
close all;

A = 1; % defining amplitude
n_cycles = 5; % defining number of cycles
fs_ideal = 100000; % defining ideal frequency
f = 3000; % defining message signal frequency

N = 64; % defining number of samples
fs = N; % defining nyquist frequency
sf=linspace(-fs,fs,N); % defining frequency range

% generating ideal signal

n_ideal = 0:1:floor(n_cycles*(fs_ideal/f))-1;
x_ideal = A*cos(2*pi*f*n_ideal*(1/fs_ideal));

figure;
subplot(2,1,1);
stem(n_ideal,x_ideal);
xlabel('samples(n) ->');
ylabel('amplitude ->');
title('Generation of Ideal Signal for F_{s} = 100000 Hz');
grid on;

ft_ideal = abs(fftshift(fft(x_ideal,N))); % fourier transform of ideal
 signal

subplot(2,1,2);
stem(sf,ft_ideal);
xlabel('frequency(Hz)->');
ylabel('Amplitude->');
title('Fourier transform for the ideal signal with F_{s}=100000 Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');


% generating sampled signals
fs1 = 10000;
ns1 = 0:1:floor(n_cycles*(fs1/f))-1;
x_sampled_1 = A*cos(2*pi*f*ns1*(1/fs1));

fs2 = 6000;
ns2 = 0:1:floor(n_cycles*(fs2/f))-1;
x_sampled_2 = A*cos(2*pi*f*ns2*(1/fs2));

fs3 = 12000;
```

Figure 1.10 Part 1 of the code for observation 2



```
ns3 = 0:1:floor(n_cycles*(fs3/f))-1;
x_sampled_3 = A*cos(2*pi*f*ns3*(1/fs3));

fs4 = 4000;
ns4 = 0:1:floor(n_cycles*(fs4/f))-1;
x_sampled_4 = A*cos(2*pi*f*ns4*(1/fs4));

fs5 = 5000;
ns5 = 0:1:floor(n_cycles*(fs5/f))-1;
x_sampled_5 = A*cos(2*pi*f*ns5*(1/fs5));

% generating the fourier transform of the sampled signals
ft_sample1 = abs(fftshift(fft(x_sampled_1,N))); % fourier transform of
 sampled signal with 10000Hz
ft_sample2 = abs(fftshift(fft(x_sampled_2,N))); % fourier transform of
 sampled signal with 6000Hz
ft_sample3 = abs(fftshift(fft(x_sampled_3,N))); % fourier transform of
 sampled signal with 12000Hz
ft_sample4 = abs(fftshift(fft(x_sampled_4,N))); % fourier transform of
 sampled signal with 4000Hz
ft_sample5 = abs(fftshift(fft(x_sampled_5,N))); % fourier transform of
 sampled signal with 5000Hz

% plotting the sampled signals and their fourier transforms
figure;
subplot(2,1,1);
stem(ns1,x_sampled_1);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=10000Hz');
grid on;

subplot(2,1,2);
stem(sf,ft_sample1);
xlabel('frequency(Hz)->');
ylabel('Amplitude->');
title('Fourier transform for the sampled signal 1 with F_{s}=10000
 Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

figure;
subplot(2,1,1);
stem(ns2,x_sampled_2);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=6000Hz');
grid on;

subplot(2,1,2);
stem(sf,ft_sample2);
xlabel('frequency(Hz)->');
ylabel('Amplitude->');
```

Figure 1.11 Part 2 of the code for observation 2



```
title('Fourier transform for the sampled signal 2 with F_{s}=6000
 Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

figure;
subplot(2,1,1);
stem(ns3,x_sampled_3);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=12000Hz');
grid on;

subplot(2,1,2);
stem(sf,ft_sample3);
xlabel('frequency(Hz)->');
ylabel('Amplitude->');
title('Fourier transform for the sampled signal 3 with F_{s}=12000
 Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

figure;
subplot(2,1,1);
stem(ns4,x_sampled_4);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=4000Hz');
grid on;

subplot(2,1,2);
stem(sf,ft_sample4);
xlabel('frequency(Hz)->');
ylabel('Amplitude->');
title('Fourier transform for the sampled signal 4 with F_{s}=4000
 Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

figure;
subplot(2,1,1);
stem(ns5,x_sampled_5);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampling for f_{s}=5000Hz');
grid on;

subplot(2,1,2);
stem(sf,ft_sample5);
xlabel('frequency(Hz)->');
ylabel('Amplitude->');
title('Fourier transform for the sampled signal 5 with F_{s}=5000
 Hz');
grid on;
```

Figure 1.12 Part 3 of the code for observation 2

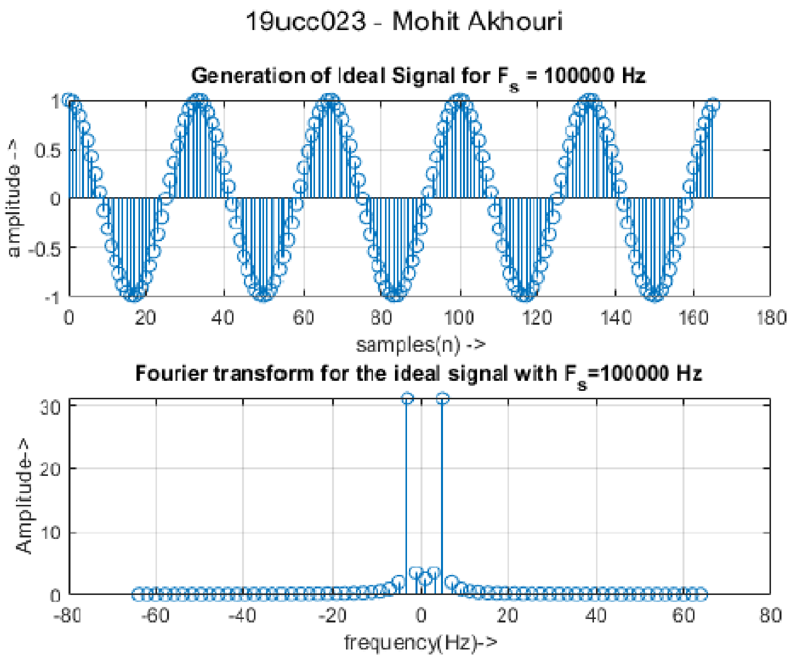


Figure 1.13 Fourier Transform for the Ideal signal with F_s = 100000 Hz

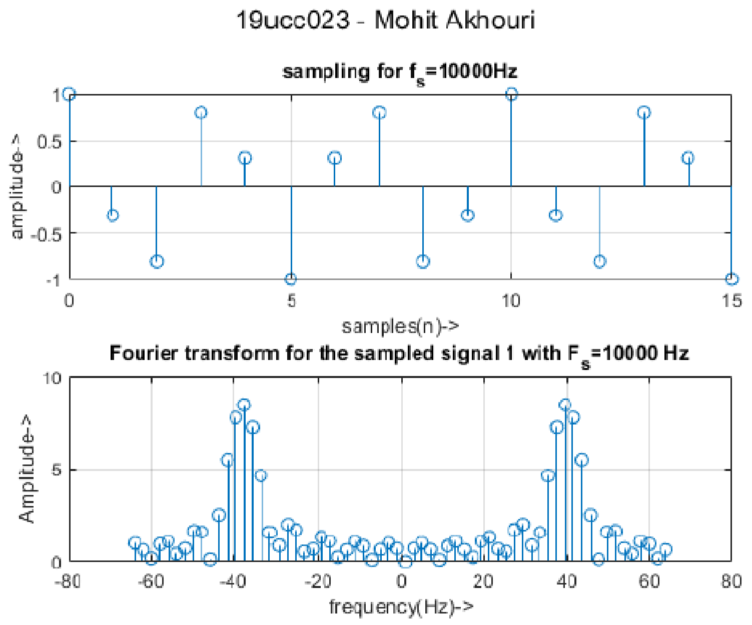


Figure 1.14 Fourier Transform for the Sampled signal with F_s = 10000 Hz

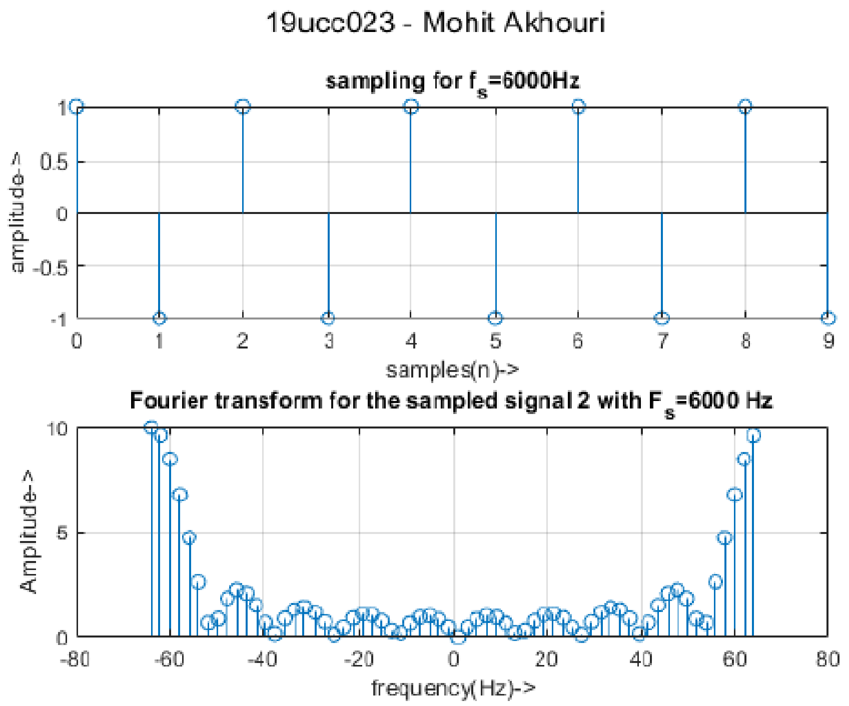


Figure 1.15 Fourier Transform for the Sampled signal with F_s = 6000 Hz

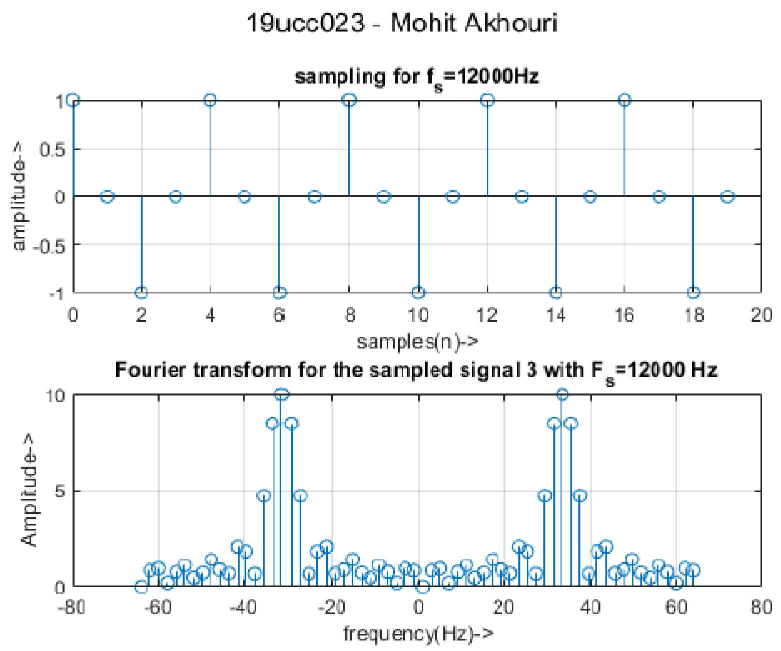


Figure 1.16 Fourier Transform for the Sampled signal with F_s = 12000 Hz

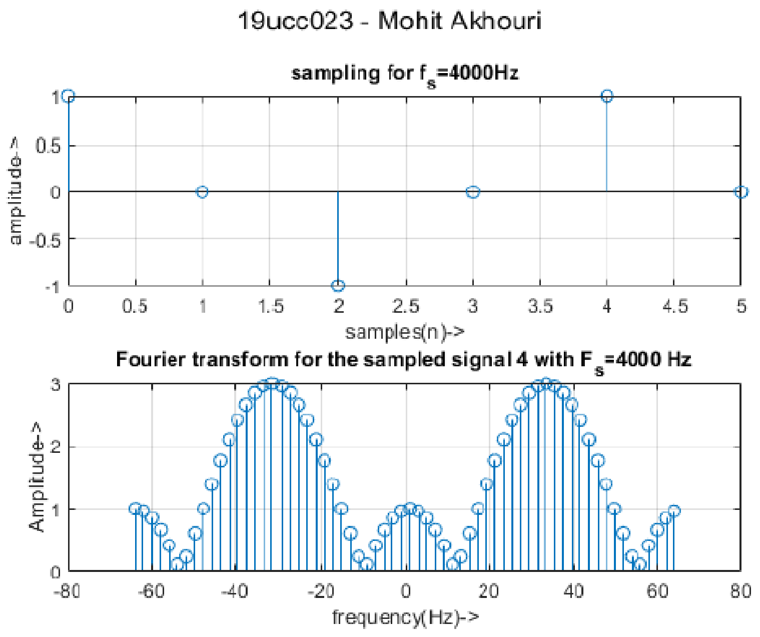
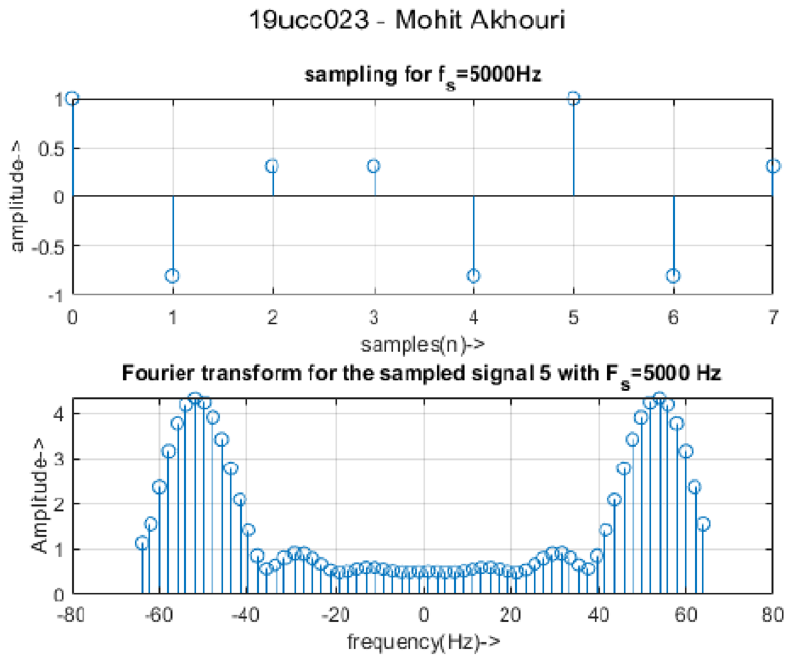


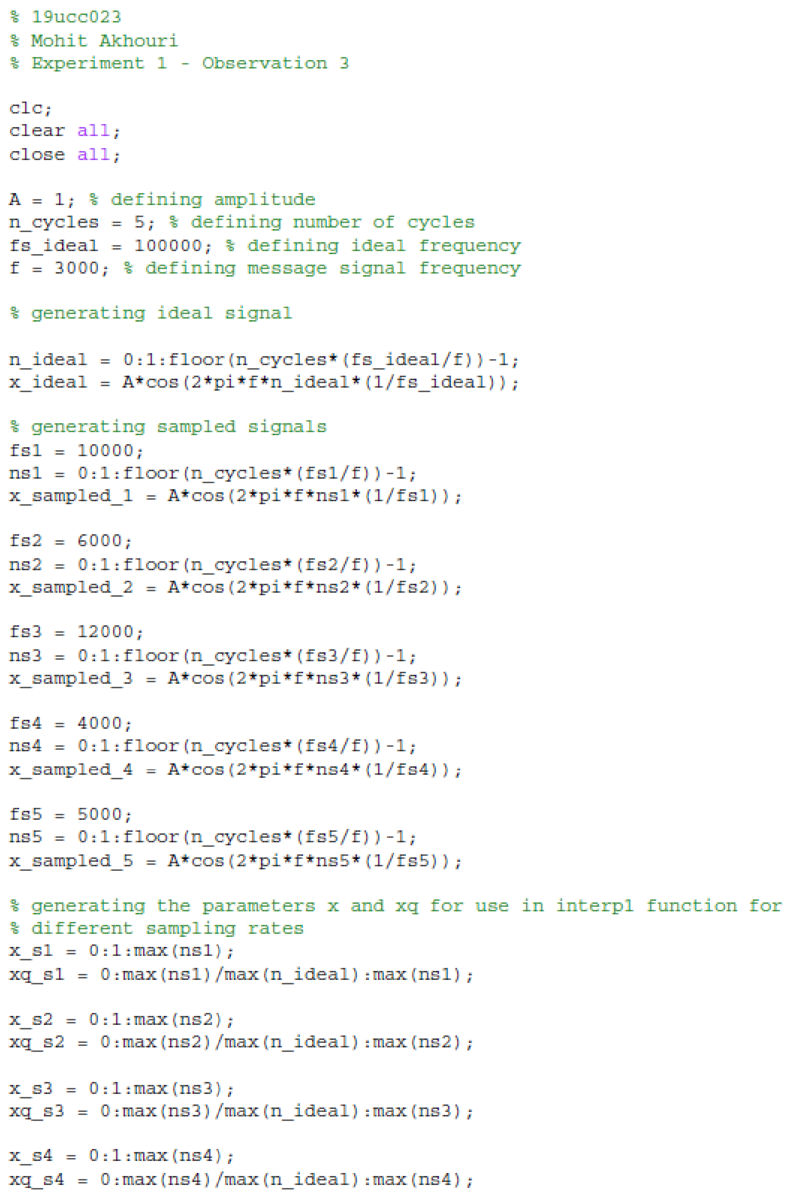
Figure 1.17 Fourier Transform for the Sampled signal with F_s = 4000 Hz



Published with MATLAB® R2020b

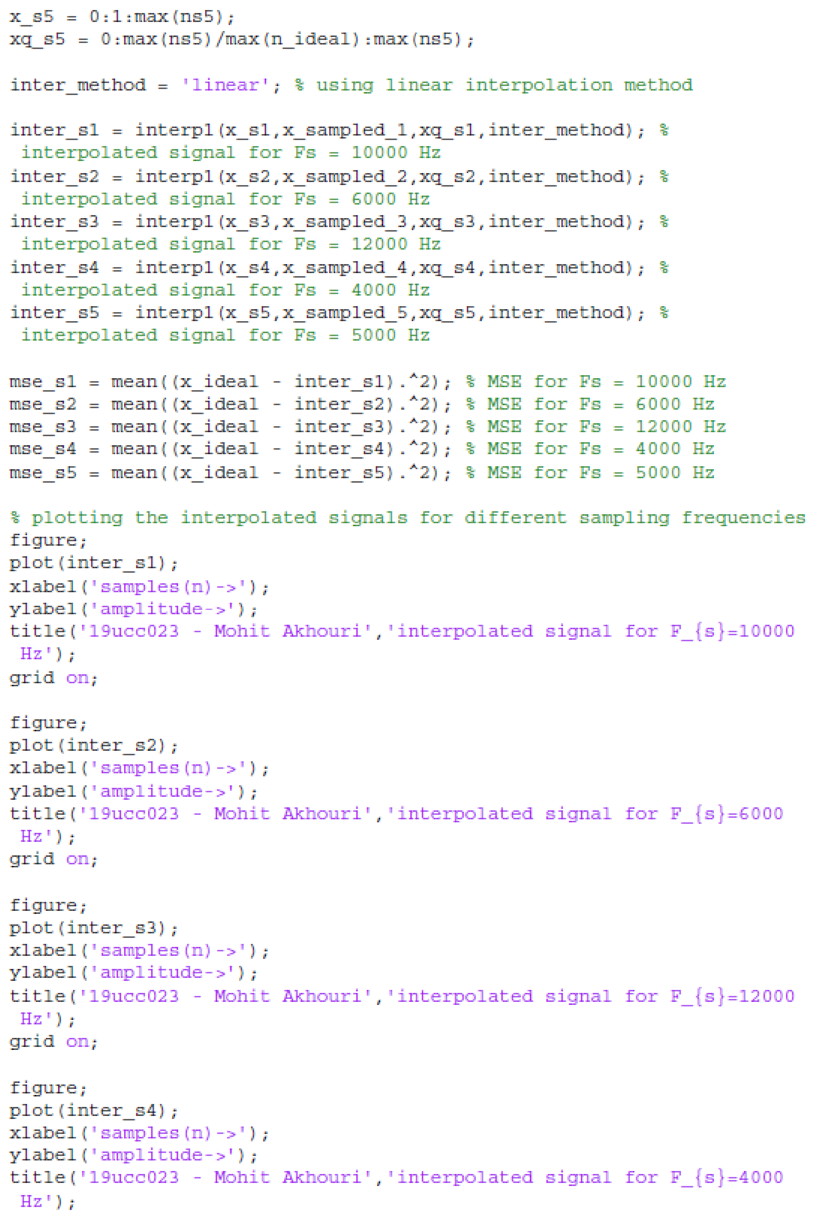
Figure 1.18 Fourier Transform for the Sampled signal with F_s = 5000 Hz

1.4.3 Find MSE for linear interpolation with different sampling rates (Plot MSE vs. F_s) :



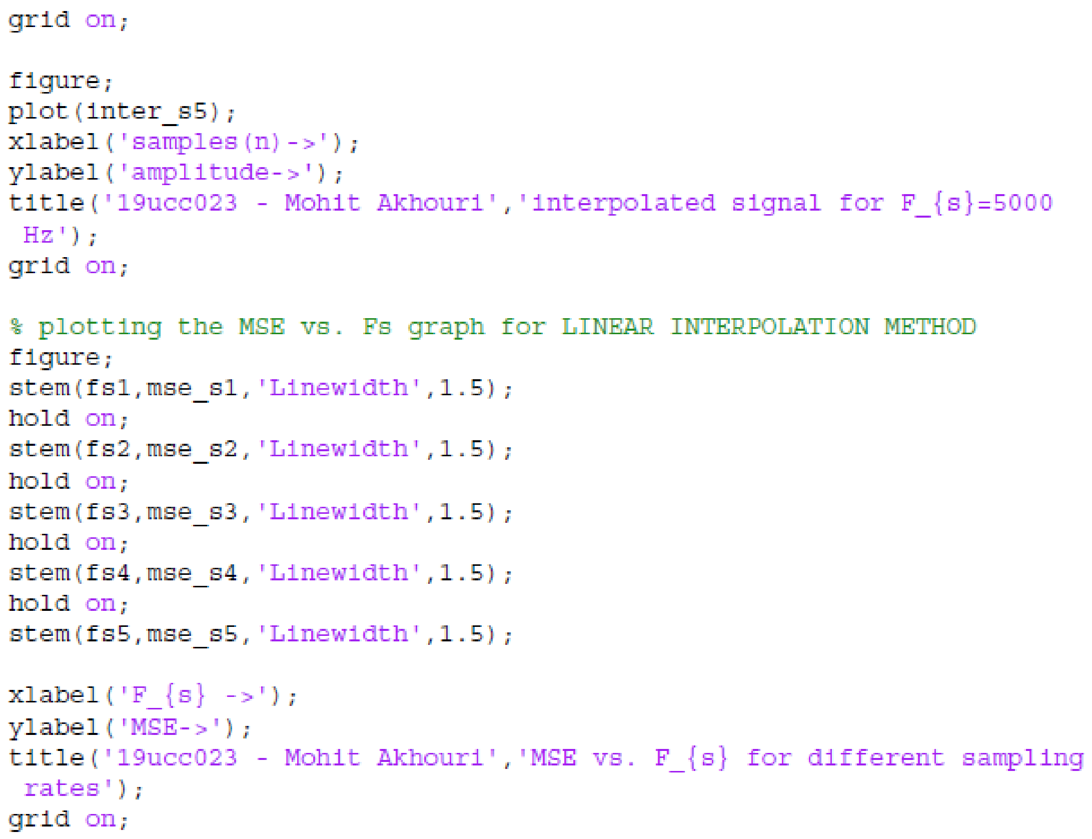
```
% 19ucc023
% Mohit Akhouri
% Experiment 1 - Observation 3

clc;
clear all;
close all;

A = 1; % defining amplitude
n_cycles = 5; % defining number of cycles
fs_ideal = 100000; % defining ideal frequency
f = 3000; % defining message signal frequency

% generating ideal signal

n_ideal = 0:1:floor(n_cycles*(fs_ideal/f))-1;
x_ideal = A*cos(2*pi*f*n_ideal*(1/fs_ideal));

% generating sampled signals
fs1 = 10000;
ns1 = 0:1:floor(n_cycles*(fs1/f))-1;
x_sampled_1 = A*cos(2*pi*f*ns1*(1/fs1));

fs2 = 6000;
ns2 = 0:1:floor(n_cycles*(fs2/f))-1;
x_sampled_2 = A*cos(2*pi*f*ns2*(1/fs2));

fs3 = 12000;
ns3 = 0:1:floor(n_cycles*(fs3/f))-1;
x_sampled_3 = A*cos(2*pi*f*ns3*(1/fs3));

fs4 = 4000;
ns4 = 0:1:floor(n_cycles*(fs4/f))-1;
x_sampled_4 = A*cos(2*pi*f*ns4*(1/fs4));

fs5 = 5000;
ns5 = 0:1:floor(n_cycles*(fs5/f))-1;
x_sampled_5 = A*cos(2*pi*f*ns5*(1/fs5));

% generating the parameters x and xq for use in interp1 function for
% different sampling rates
x_s1 = 0:1:max(ns1);
xq_s1 = 0:max(ns1)/max(n_ideal):max(ns1);

x_s2 = 0:1:max(ns2);
xq_s2 = 0:max(ns2)/max(n_ideal):max(ns2);

x_s3 = 0:1:max(ns3);
xq_s3 = 0:max(ns3)/max(n_ideal):max(ns3);

x_s4 = 0:1:max(ns4);
xq_s4 = 0:max(ns4)/max(n_ideal):max(ns4);
```

Figure 1.19 Part 1 of the code for observation 3



```
x_s5 = 0:1:max(ns5);
xq_s5 = 0:max(ns5)/max(n_ideal):max(ns5);

inter_method = 'linear'; % using linear interpolation method

inter_s1 = interp1(x_s1,x_sampled_1,xq_s1,inter_method); %
 interpolated signal for Fs = 10000 Hz
inter_s2 = interp1(x_s2,x_sampled_2,xq_s2,inter_method); %
 interpolated signal for Fs = 6000 Hz
inter_s3 = interp1(x_s3,x_sampled_3,xq_s3,inter_method); %
 interpolated signal for Fs = 12000 Hz
inter_s4 = interp1(x_s4,x_sampled_4,xq_s4,inter_method); %
 interpolated signal for Fs = 4000 Hz
inter_s5 = interp1(x_s5,x_sampled_5,xq_s5,inter_method); %
 interpolated signal for Fs = 5000 Hz

mse_s1 = mean((x_ideal - inter_s1).^2); % MSE for Fs = 10000 Hz
mse_s2 = mean((x_ideal - inter_s2).^2); % MSE for Fs = 6000 Hz
mse_s3 = mean((x_ideal - inter_s3).^2); % MSE for Fs = 12000 Hz
mse_s4 = mean((x_ideal - inter_s4).^2); % MSE for Fs = 4000 Hz
mse_s5 = mean((x_ideal - inter_s5).^2); % MSE for Fs = 5000 Hz

% plotting the interpolated signals for different sampling frequencies
figure;
plot(inter_s1);
xlabel('samples(n)->');
ylabel('amplitude->');
title('19ucc023 - Mohit Akhouri','interpolated signal for F_{s}=10000
 Hz');
grid on;

figure;
plot(inter_s2);
xlabel('samples(n)->');
ylabel('amplitude->');
title('19ucc023 - Mohit Akhouri','interpolated signal for F_{s}=6000
 Hz');
grid on;

figure;
plot(inter_s3);
xlabel('samples(n)->');
ylabel('amplitude->');
title('19ucc023 - Mohit Akhouri','interpolated signal for F_{s}=12000
 Hz');
grid on;

figure;
plot(inter_s4);
xlabel('samples(n)->');
ylabel('amplitude->');
title('19ucc023 - Mohit Akhouri','interpolated signal for F_{s}=4000
 Hz');
```

Figure 1.20 Part 2 of the code for observation 3



```
grid on;

figure;
plot(inter_s5);
xlabel('samples(n)->');
ylabel('amplitude->');
title('19ucc023 - Mohit Akhouri','interpolated signal for F_{s}=5000
 Hz');
grid on;

% plotting the MSE vs. Fs graph for LINEAR INTERPOLATION METHOD
figure;
stem(fs1,mse_s1,'Linewidth',1.5);
hold on;
stem(fs2,mse_s2,'Linewidth',1.5);
hold on;
stem(fs3,mse_s3,'Linewidth',1.5);
hold on;
stem(fs4,mse_s4,'Linewidth',1.5);
hold on;
stem(fs5,mse_s5,'Linewidth',1.5);

xlabel('F_{s} ->');
ylabel('MSE->');
title('19ucc023 - Mohit Akhouri','MSE vs. F_{s} for different sampling
 rates');
grid on;
```

Figure 1.21 Part 3 of the code for observation 3

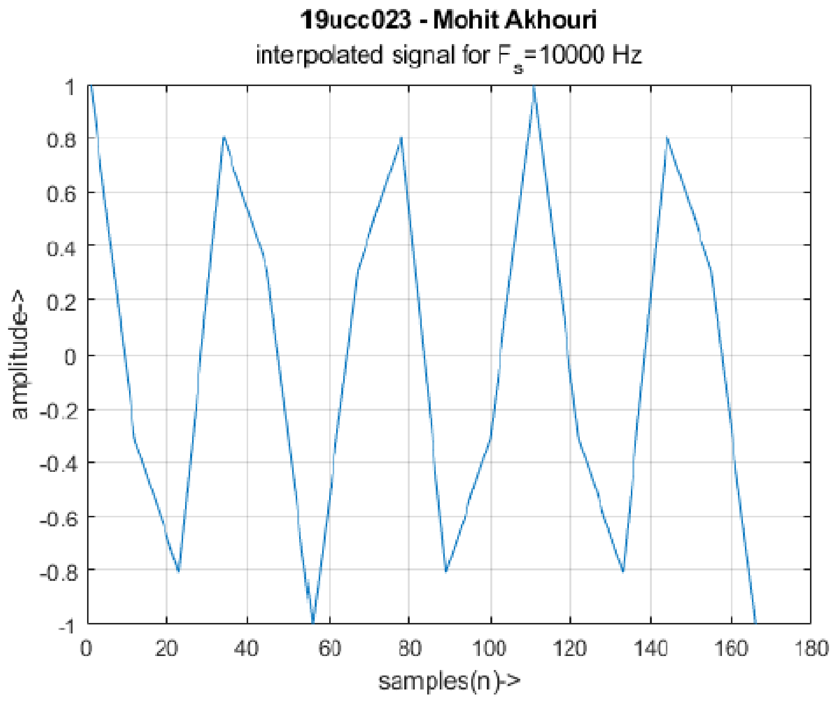


Figure 1.22 Linear Interpolation for the sampled signal with F_s = 10000 Hz

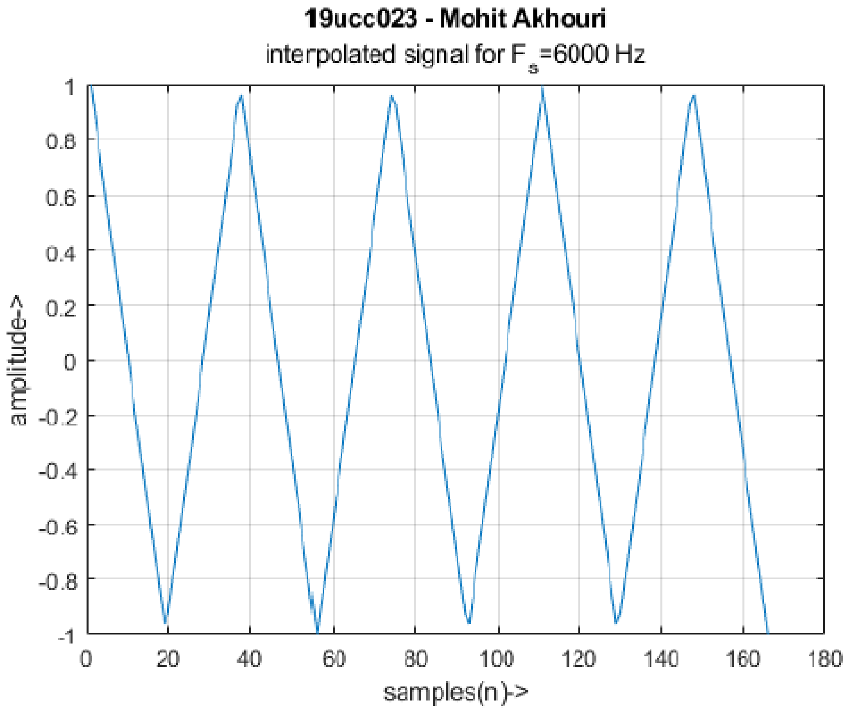


Figure 1.23 Linear Interpolation for the sampled signal with F_s = 6000 Hz

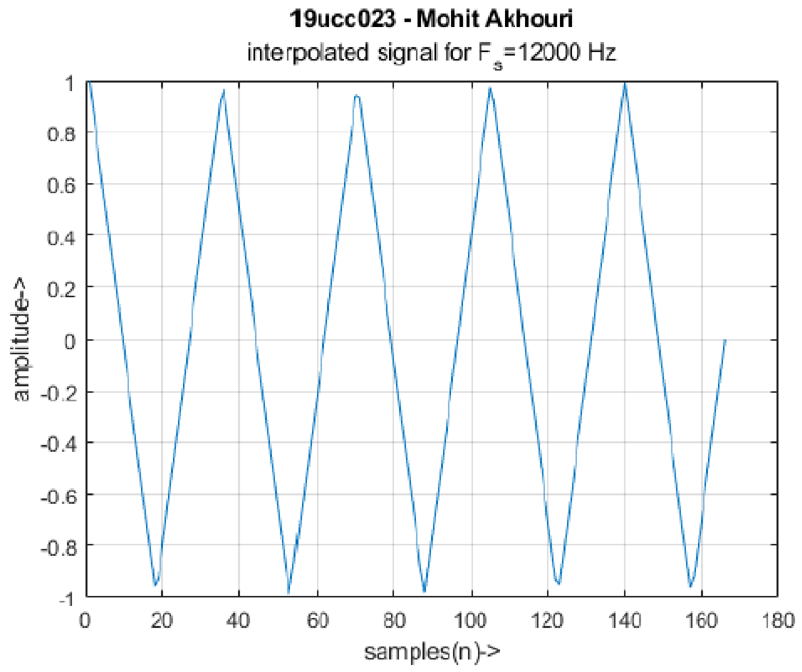


Figure 1.24 Linear Interpolation for the sampled signal with F_s = 12000 Hz

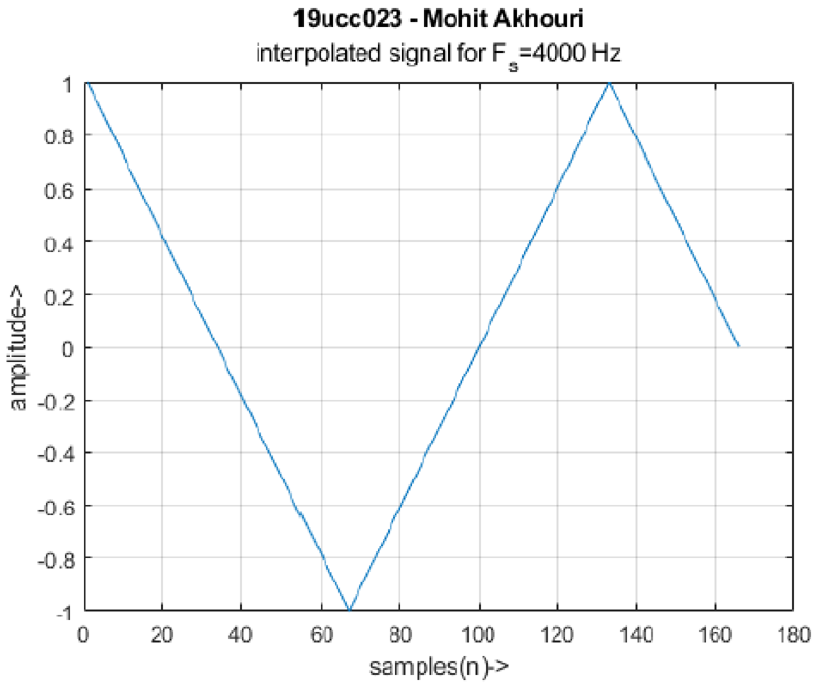


Figure 1.25 Linear Interpolation for the sampled signal with F_s = 4000 Hz

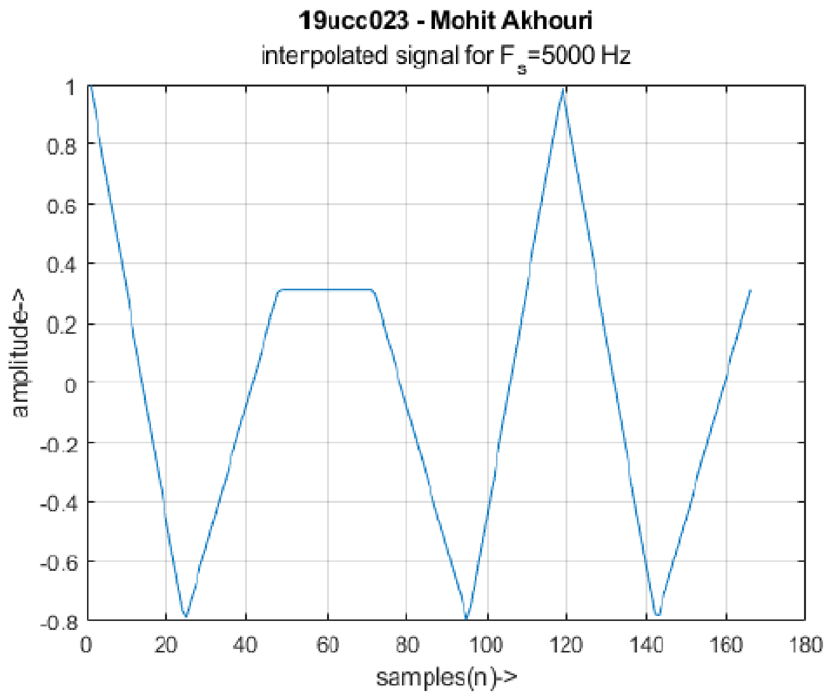


Figure 1.26 Linear Interpolation for the sampled signal with F_s = 5000 Hz

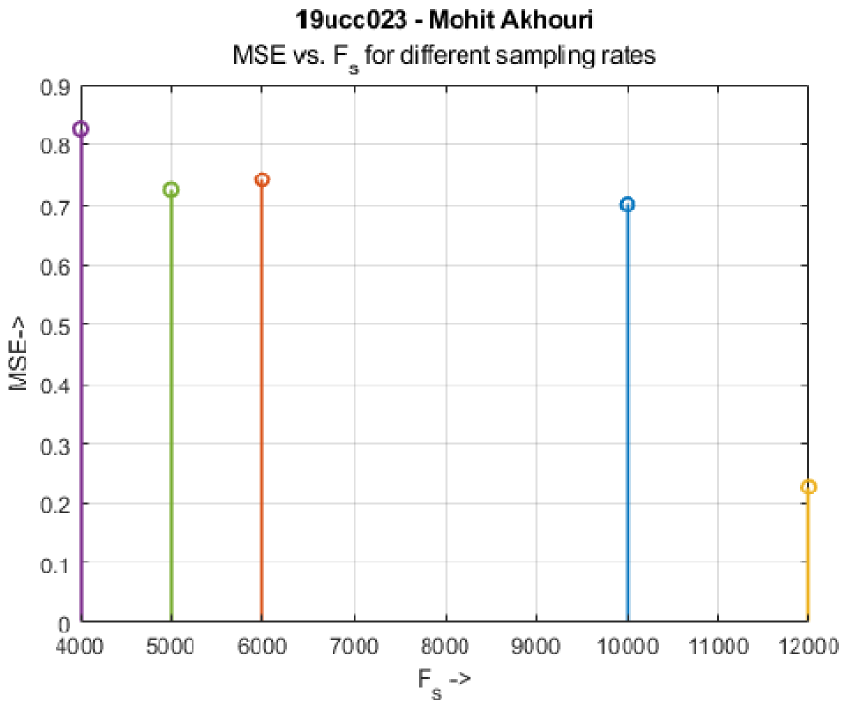
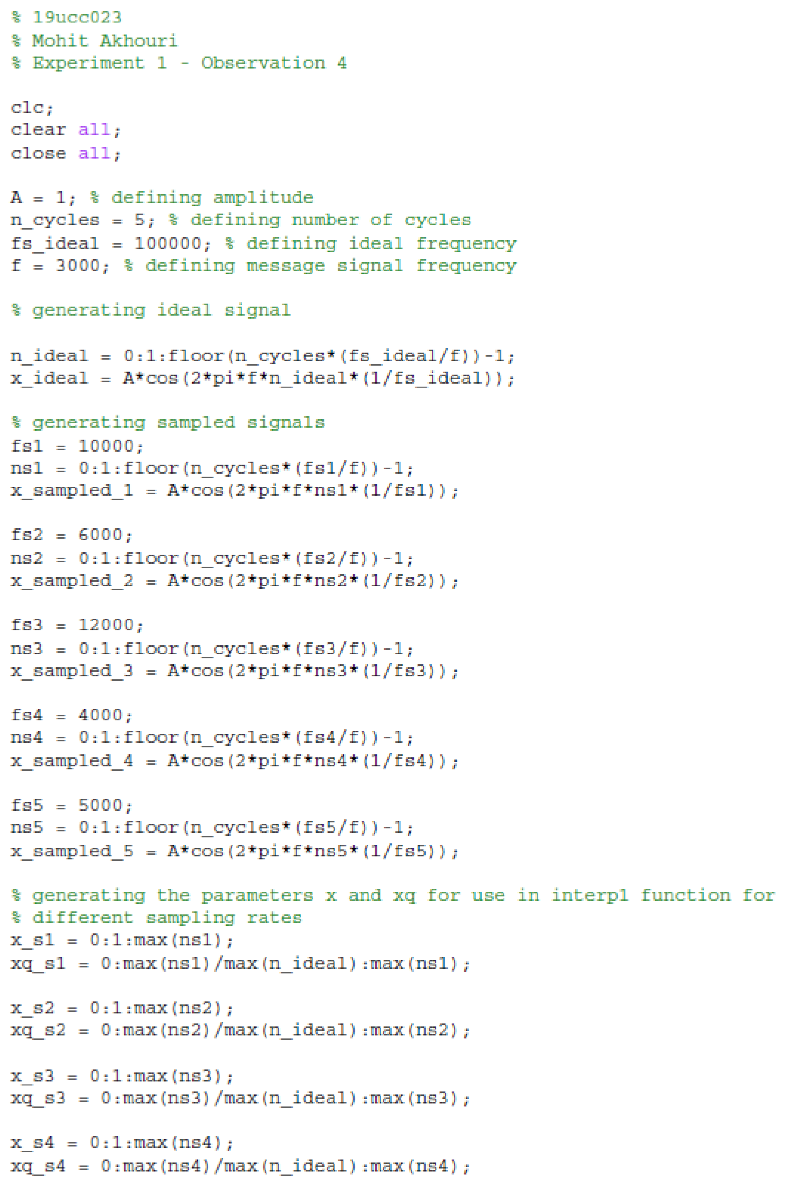


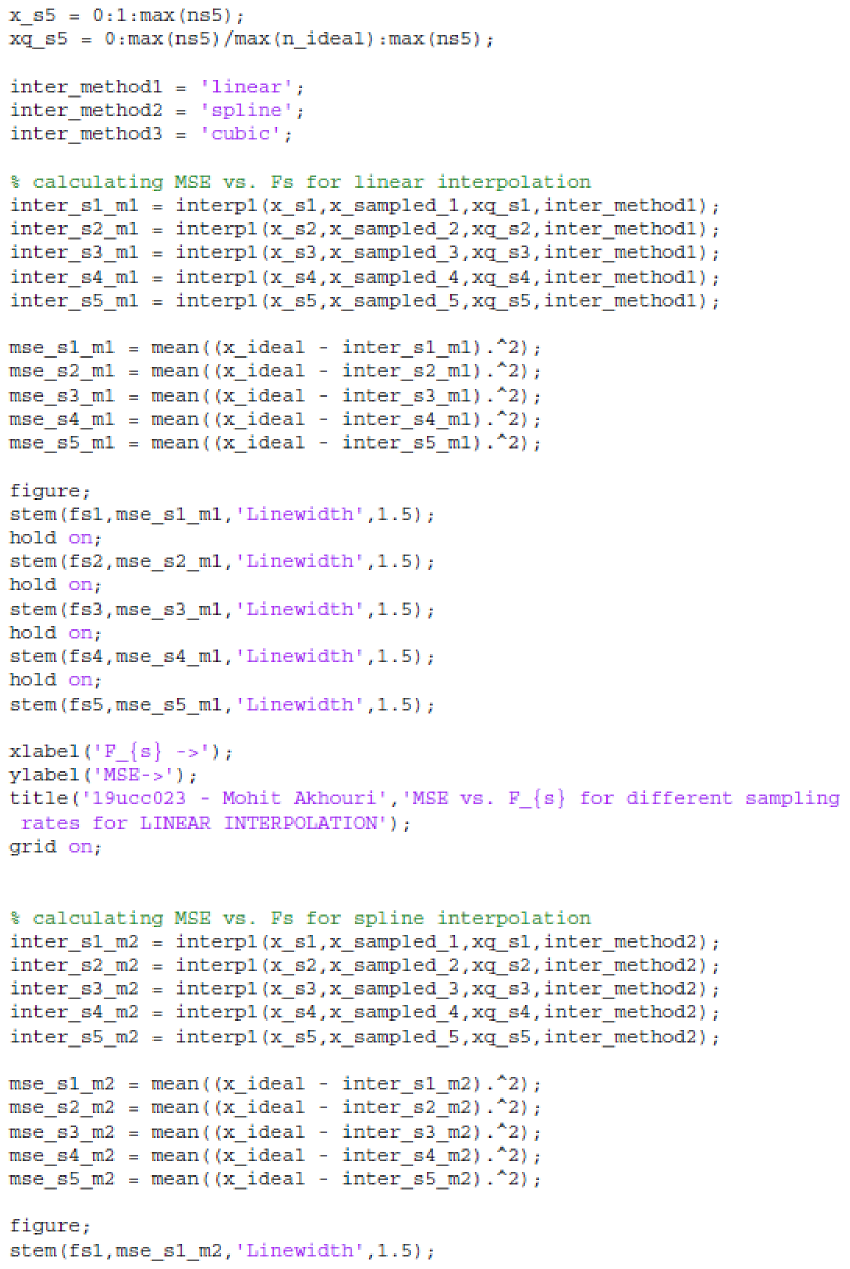
Figure 1.27 MSE vs. F_s for Linear Interpolation

1.4.4 Repeat MSE vs. F_s for different interpolation techniques :



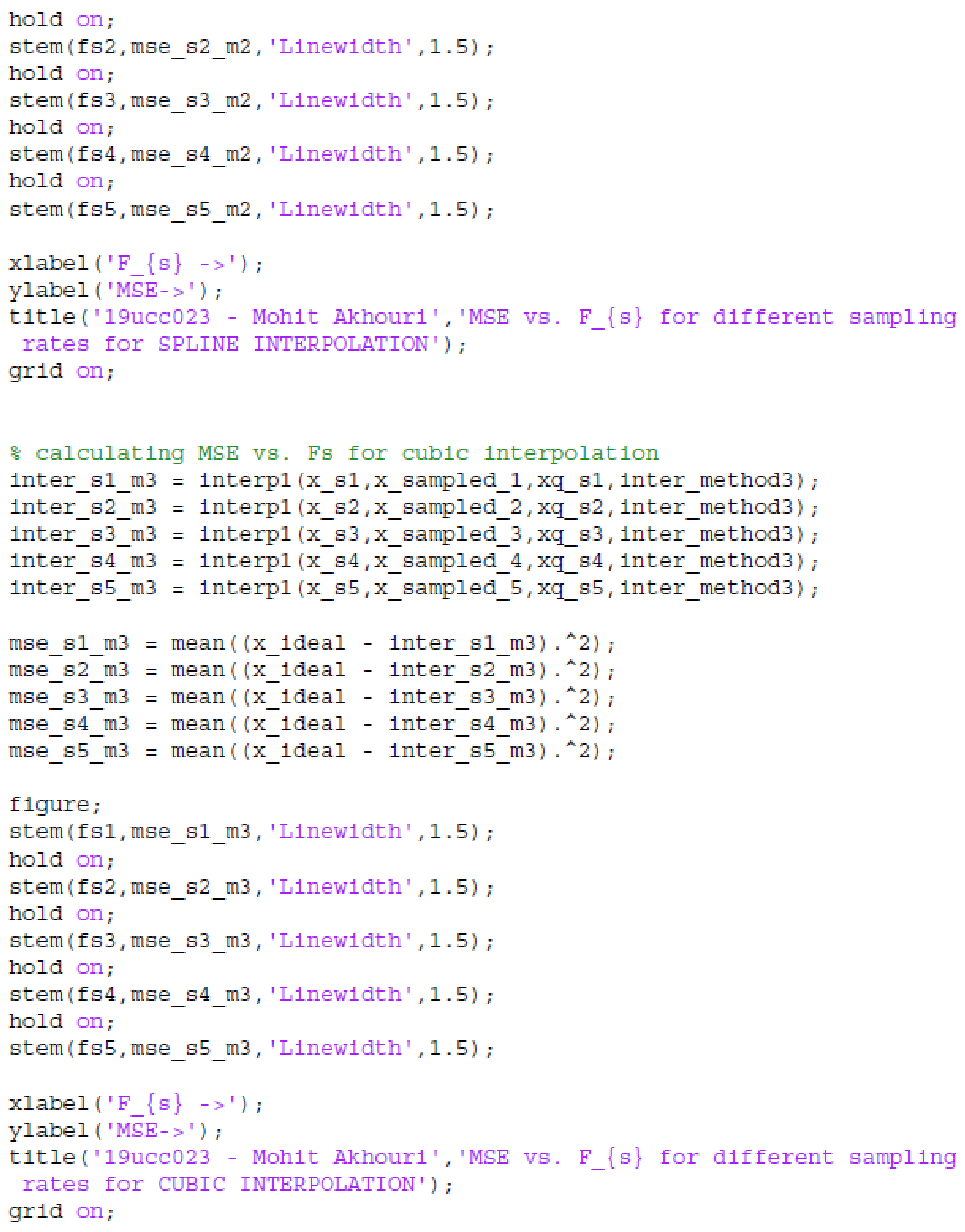
```
% 19ucc023
% Mohit Akhouri
% Experiment 1 - Observation 4

clc;
clear all;
close all;

A = 1; % defining amplitude
n_cycles = 5; % defining number of cycles
fs_ideal = 100000; % defining ideal frequency
f = 3000; % defining message signal frequency

% generating ideal signal

n_ideal = 0:1:floor(n_cycles*(fs_ideal/f))-1;
x_ideal = A*cos(2*pi*f*n_ideal*(1/fs_ideal));

% generating sampled signals
fs1 = 10000;
ns1 = 0:1:floor(n_cycles*(fs1/f))-1;
x_sampled_1 = A*cos(2*pi*f*ns1*(1/fs1));

fs2 = 6000;
ns2 = 0:1:floor(n_cycles*(fs2/f))-1;
x_sampled_2 = A*cos(2*pi*f*ns2*(1/fs2));

fs3 = 12000;
ns3 = 0:1:floor(n_cycles*(fs3/f))-1;
x_sampled_3 = A*cos(2*pi*f*ns3*(1/fs3));

fs4 = 4000;
ns4 = 0:1:floor(n_cycles*(fs4/f))-1;
x_sampled_4 = A*cos(2*pi*f*ns4*(1/fs4));

fs5 = 5000;
ns5 = 0:1:floor(n_cycles*(fs5/f))-1;
x_sampled_5 = A*cos(2*pi*f*ns5*(1/fs5));

% generating the parameters x and xq for use in interp1 function for
% different sampling rates
x_s1 = 0:1:max(ns1);
xq_s1 = 0:max(ns1)/max(n_ideal):max(ns1);

x_s2 = 0:1:max(ns2);
xq_s2 = 0:max(ns2)/max(n_ideal):max(ns2);

x_s3 = 0:1:max(ns3);
xq_s3 = 0:max(ns3)/max(n_ideal):max(ns3);

x_s4 = 0:1:max(ns4);
xq_s4 = 0:max(ns4)/max(n_ideal):max(ns4);
```

Figure 1.28 Part 1 of the code for observation 4



```
x_s5 = 0:1:max(ns5);
xq_s5 = 0:max(ns5)/max(n_ideal):max(ns5);

inter_method1 = 'linear';
inter_method2 = 'spline';
inter_method3 = 'cubic';

% calculating MSE vs. Fs for linear interpolation
inter_s1_m1 = interp1(x_s1,x_sampled_1,xq_s1,inter_method1);
inter_s2_m1 = interp1(x_s2,x_sampled_2,xq_s2,inter_method1);
inter_s3_m1 = interp1(x_s3,x_sampled_3,xq_s3,inter_method1);
inter_s4_m1 = interp1(x_s4,x_sampled_4,xq_s4,inter_method1);
inter_s5_m1 = interp1(x_s5,x_sampled_5,xq_s5,inter_method1);

mse_s1_m1 = mean((x_ideal - inter_s1_m1).^2);
mse_s2_m1 = mean((x_ideal - inter_s2_m1).^2);
mse_s3_m1 = mean((x_ideal - inter_s3_m1).^2);
mse_s4_m1 = mean((x_ideal - inter_s4_m1).^2);
mse_s5_m1 = mean((x_ideal - inter_s5_m1).^2);

figure;
stem(fs1,mse_s1_m1,'Linewidth',1.5);
hold on;
stem(fs2,mse_s2_m1,'Linewidth',1.5);
hold on;
stem(fs3,mse_s3_m1,'Linewidth',1.5);
hold on;
stem(fs4,mse_s4_m1,'Linewidth',1.5);
hold on;
stem(fs5,mse_s5_m1,'Linewidth',1.5);

xlabel('F_{s} ->');
ylabel('MSE->');
title('19ucc023 - Mohit Akhouri','MSE vs. F_{s} for different sampling
 rates for LINEAR INTERPOLATION');
grid on;


% calculating MSE vs. Fs for spline interpolation
inter_s1_m2 = interp1(x_s1,x_sampled_1,xq_s1,inter_method2);
inter_s2_m2 = interp1(x_s2,x_sampled_2,xq_s2,inter_method2);
inter_s3_m2 = interp1(x_s3,x_sampled_3,xq_s3,inter_method2);
inter_s4_m2 = interp1(x_s4,x_sampled_4,xq_s4,inter_method2);
inter_s5_m2 = interp1(x_s5,x_sampled_5,xq_s5,inter_method2);

mse_s1_m2 = mean((x_ideal - inter_s1_m2).^2);
mse_s2_m2 = mean((x_ideal - inter_s2_m2).^2);
mse_s3_m2 = mean((x_ideal - inter_s3_m2).^2);
mse_s4_m2 = mean((x_ideal - inter_s4_m2).^2);
mse_s5_m2 = mean((x_ideal - inter_s5_m2).^2);

figure;
stem(fs1,mse_s1_m2,'Linewidth',1.5);
```

Figure 1.29 Part 2 of the code for observation 4



```
hold on;
stem(fs2,mse_s2_m2,'Linewidth',1.5);
hold on;
stem(fs3,mse_s3_m2,'Linewidth',1.5);
hold on;
stem(fs4,mse_s4_m2,'Linewidth',1.5);
hold on;
stem(fs5,mse_s5_m2,'Linewidth',1.5);

xlabel('F_{s} ->');
ylabel('MSE->');
title('19ucc023 - Mohit Akhouri','MSE vs. F_{s} for different sampling
 rates for SPLINE INTERPOLATION');
grid on;


% calculating MSE vs. Fs for cubic interpolation
inter_s1_m3 = interp1(x_s1,x_sampled_1,xq_s1,inter_method3);
inter_s2_m3 = interp1(x_s2,x_sampled_2,xq_s2,inter_method3);
inter_s3_m3 = interp1(x_s3,x_sampled_3,xq_s3,inter_method3);
inter_s4_m3 = interp1(x_s4,x_sampled_4,xq_s4,inter_method3);
inter_s5_m3 = interp1(x_s5,x_sampled_5,xq_s5,inter_method3);

mse_s1_m3 = mean((x_ideal - inter_s1_m3).^2);
mse_s2_m3 = mean((x_ideal - inter_s2_m3).^2);
mse_s3_m3 = mean((x_ideal - inter_s3_m3).^2);
mse_s4_m3 = mean((x_ideal - inter_s4_m3).^2);
mse_s5_m3 = mean((x_ideal - inter_s5_m3).^2);

figure;
stem(fs1,mse_s1_m3,'Linewidth',1.5);
hold on;
stem(fs2,mse_s2_m3,'Linewidth',1.5);
hold on;
stem(fs3,mse_s3_m3,'Linewidth',1.5);
hold on;
stem(fs4,mse_s4_m3,'Linewidth',1.5);
hold on;
stem(fs5,mse_s5_m3,'Linewidth',1.5);

xlabel('F_{s} ->');
ylabel('MSE->');
title('19ucc023 - Mohit Akhouri','MSE vs. F_{s} for different sampling
 rates for CUBIC INTERPOLATION');
grid on;
```

Figure 1.30 Part 3 of the code for observation 4

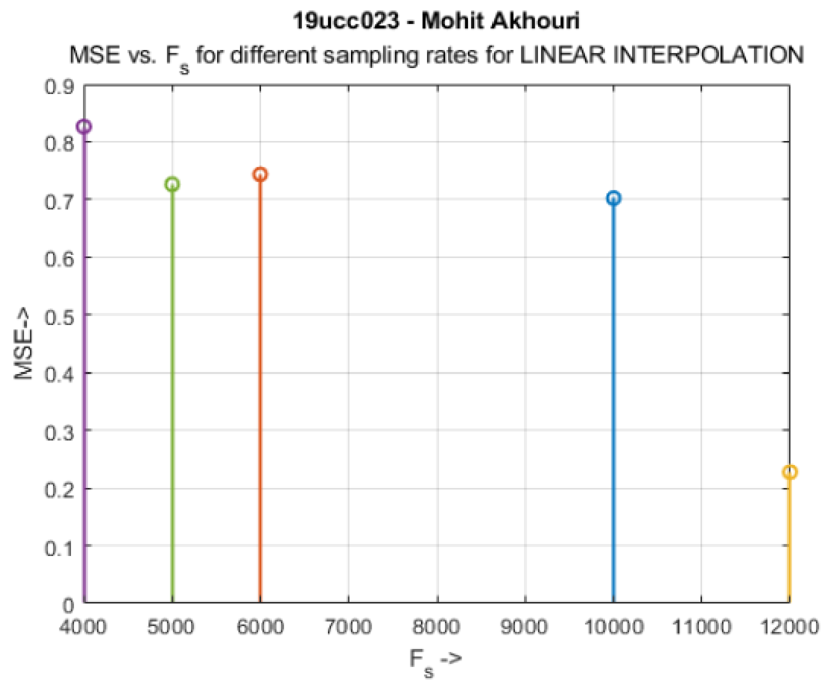


Figure 1.31 MSE vs. F_s for Linear interpolation

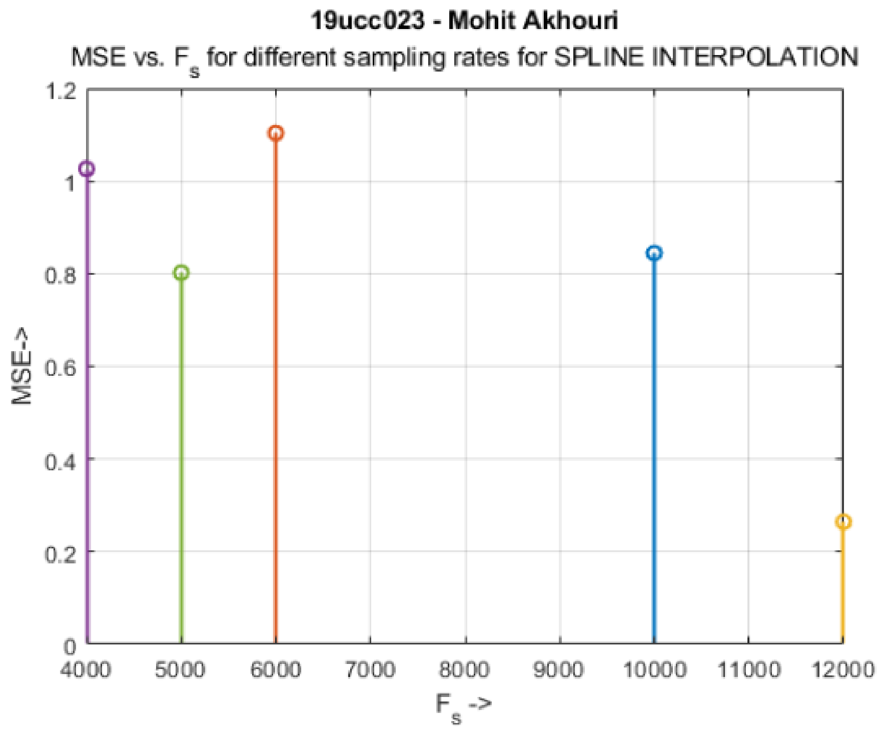
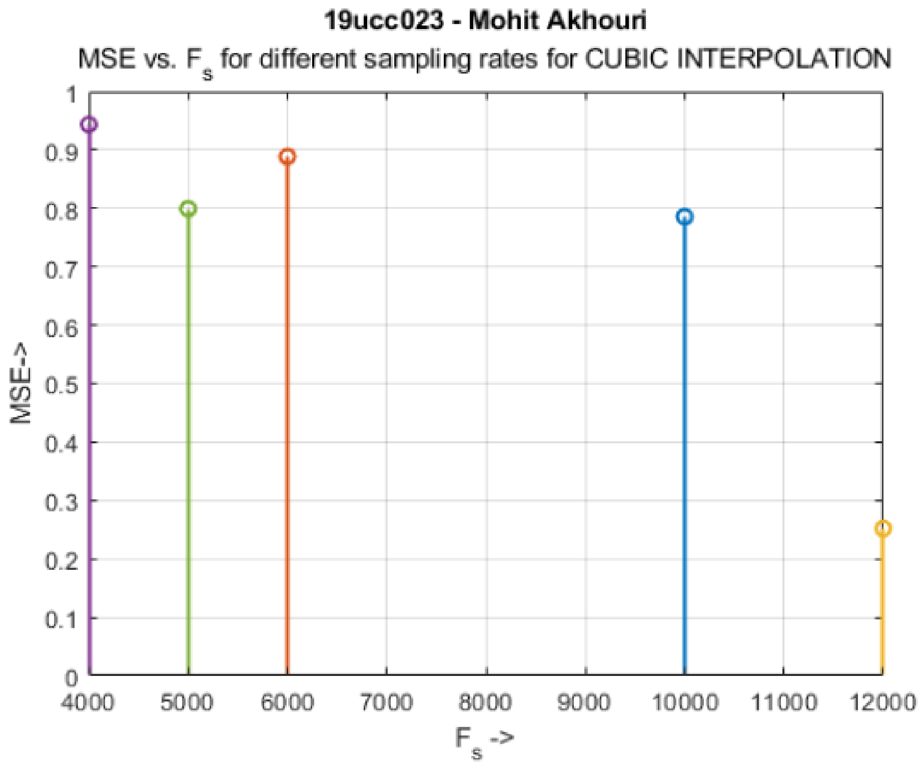


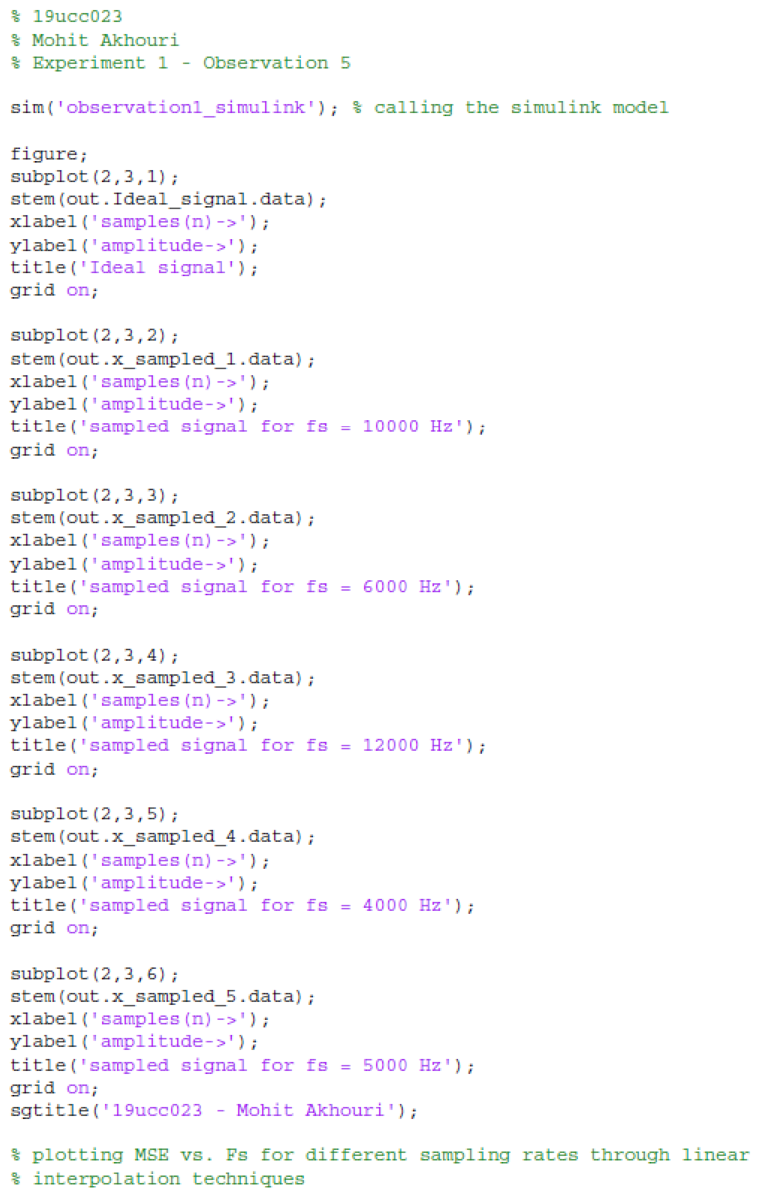
Figure 1.32 MSE vs. F_s for Spline interpolation



Published with MATLAB® R2020b

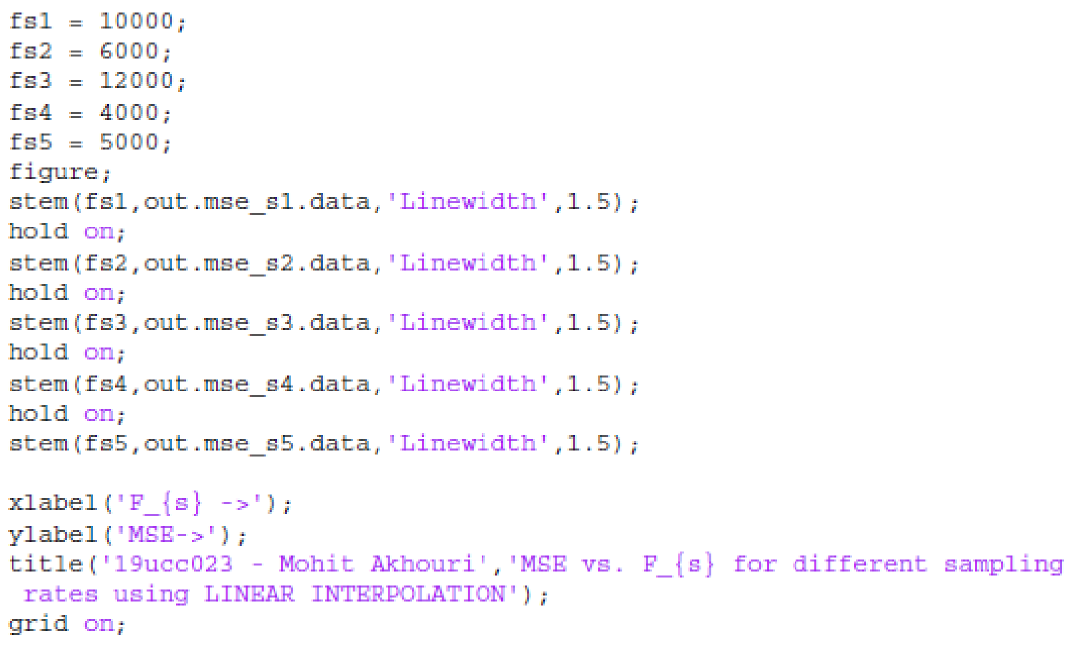
Figure 1.33 MSE vs. F_s for Cubic interpolation

1.4.5 Perform sampling and interpolation in simulink :



```matlab
% 19ucc023
% Mohit Akhouri
% Experiment 1 - Observation 5

sim('observation1_simulink'); % calling the simulink model

figure;
subplot(2,3,1);
stem(out.Ideal_signal.data);
xlabel('samples(n)->');
ylabel('amplitude->');
title('Ideal signal');
grid on;

subplot(2,3,2);
stem(out.x_sampled_1.data);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampled signal for fs = 10000 Hz');
grid on;

subplot(2,3,3);
stem(out.x_sampled_2.data);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampled signal for fs = 6000 Hz');
grid on;

subplot(2,3,4);
stem(out.x_sampled_3.data);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampled signal for fs = 12000 Hz');
grid on;

subplot(2,3,5);
stem(out.x_sampled_4.data);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampled signal for fs = 4000 Hz');
grid on;

subplot(2,3,6);
stem(out.x_sampled_5.data);
xlabel('samples(n)->');
ylabel('amplitude->');
title('sampled signal for fs = 5000 Hz');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

% plotting MSE vs. Fs for different sampling rates through linear
% interpolation techniques
```

Figure 1.34 Part 1 of the code for observation 5



```
fs1 = 10000;
fs2 = 6000;
fs3 = 12000;
fs4 = 4000;
fs5 = 5000;
figure;
stem(fs1,out.mse_s1.data,'Linewidth',1.5);
hold on;
stem(fs2,out.mse_s2.data,'Linewidth',1.5);
hold on;
stem(fs3,out.mse_s3.data,'Linewidth',1.5);
hold on;
stem(fs4,out.mse_s4.data,'Linewidth',1.5);
hold on;
stem(fs5,out.mse_s5.data,'Linewidth',1.5);

xlabel('F_{s} ->');
ylabel('MSE->');
title('19ucc023 - Mohit Akhouri','MSE vs. F_{s} for different sampling
 rates using LINEAR INTERPOLATION');
grid on;
```

Figure 1.35 Part 2 of the code for observation 5

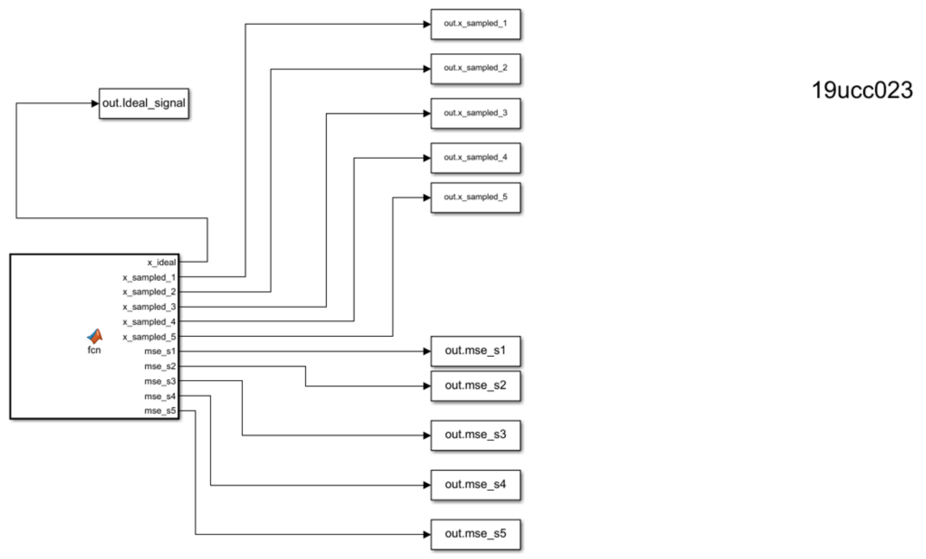


Figure 1.36 Simulink Block Diagram for observation 5 for sampling and interpolation

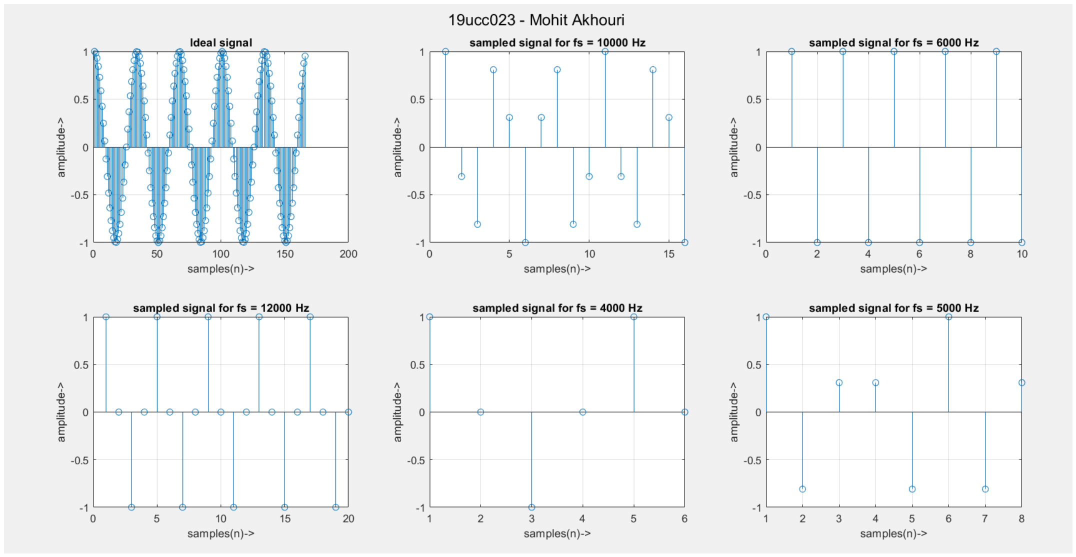
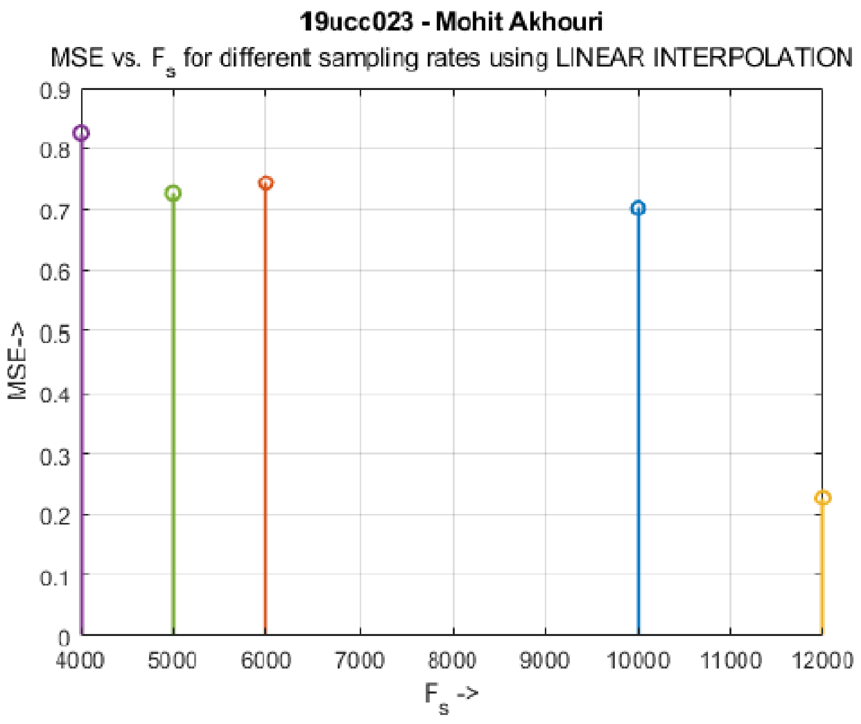


Figure 1.37 Plot of the sampling done via Simulink Model



Published with MATLAB® R2020b

Figure 1.38 MSE vs. F_s for linear interpolation done via Simulink Model

1.5 Conclusion

In this experiment, we learnt about the two concepts of Digital Signal Processing which are **Signal Sampling** and **Signal Reconstruction via Interpolation** . We also learnt how to write code for the above methods in MATLAB and how to create models for the same using **Simulink**. In simulink , we learnt about two blocks - **MATLAB function** and **To workspace**. We plotted the MSE vs. F_s curves for different sampling rates. We also learnt about various Interpolation techniques such as **Linear** , **Spline** and **Cubic** Interpolation. At Last , we concluded that **Cubic interpolation** is **better** than other interpolation techniques due to the **smoothness** of the function and **higher accuracy** in achieving the original function.