Digital Signal Processing Laboratory

Experiment 1

By: Hardik Tibrewal (18EC10020)

Sampling

Objectives:

(a) To sample a sinusoidal waveform at a frequency greater than the Nyquist rate and observe its spectrum in the frequency domain.
(b) To sample a sinusoidal waveform at a frequency below its Nyquist rate and observe the effect of aliasing.
(c) To observe the frequency spectrum of a square wave
(d) To use interpolation/upsampling on a signal to get twice the number of samples and compare with the original signal sampled at twice the initial sampling frequency.

Theoretical Background:

All real practical signals are continuous-time analog signals. However, it is impossible to store all the information of a continuous-time signal in finite memory without any changes. This is why discrete-time signals are preferred, since they have a finite (or countably infinite) number of values that need to be stored. The Nyquist-Shannon sampling theorem gives us a minimum sampling frequency, which if used to sample the a continuous-time signal, will result a discrete-time signal which can give back to original signal without any changes to it. The Nyquist theorem is for baseband signals, or lowpass signals, and the Nyquist sampling rate is twice the bandwidth of the signal. If the sampling rate is less than the Nyquist rate, it results in "aliasing", which causes a distortion in the signal when it is being recreated to a continuous-time signal.

Using the Fourier series expansion of a square wave, we know that it can be represented as an infinite sum of sine waves of odd harmonics.

We can see this by generating a square wave in MATLAB and plotting its frequency spectrum.

Finally, if a signal was sampled at a frequency greater than its Nyquist rate, it is possible to generate more samples of the signal using interpolation techniques. In the discrete-time domain, we can use zero-interpolation(add zeroes between the samples) and then pass the new array of samples through a lowpass filter of a suitable cut-off frequency. This saves us the unnecessary process of generating the continuous-time signal and sampling it again at a higher frequency. In order to upsample to L times the initial sampling frequency, we need to add $L-1$ zeros.

Pseudocode:

Parts A & B:

1. Set the sampling frequency to the desired value, and accordingly generate a discrete-time interval such that the difference between two consecutive points is 1/(sampling frequency)
2. Use the time interval to get samples of the desired sinusoidal wave using its equation at the desired sampling frequency.
3. Plot the signal in its continuous-time representation as well as discrete-time, to see the waveforms.
4. Use N-point FFT to perform Discrete Fourier Transform on the samples and observe the frequency spectrum for N = [12, 64, 128, 256].
5. Shift the zero-frequency component to the centre of the spectrum for better visualisation. Divide the absolute values by the number of points so that the average energy remains the same.
6. Plot all the frequency spectra and observe the similarities and differences.

Part C:

1. Set the frequency of your square wave and the sampling frequency. Generate time intervals corresponding to these frequencies.

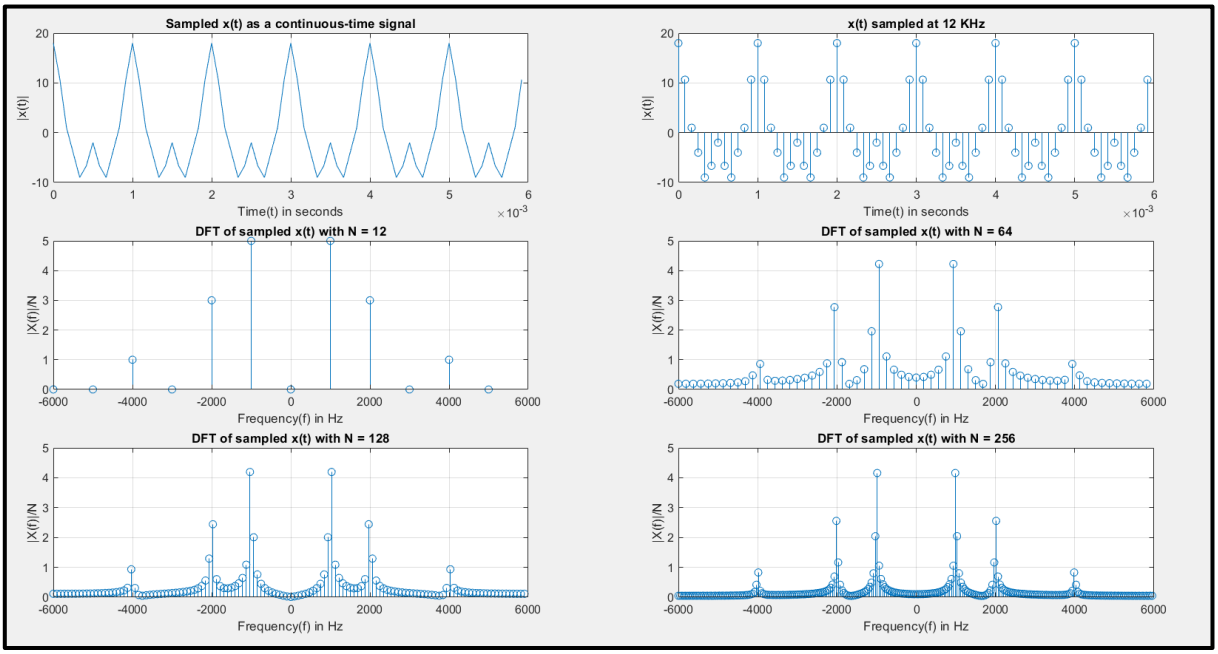
2. Generate and plot a square wave for both of the time intervals for observing the differences generated due to sampling.
3. Following the same process as used in Parts A & B, get the frequency spectrum of the sampled signal for N = [128, 256].

Part D:

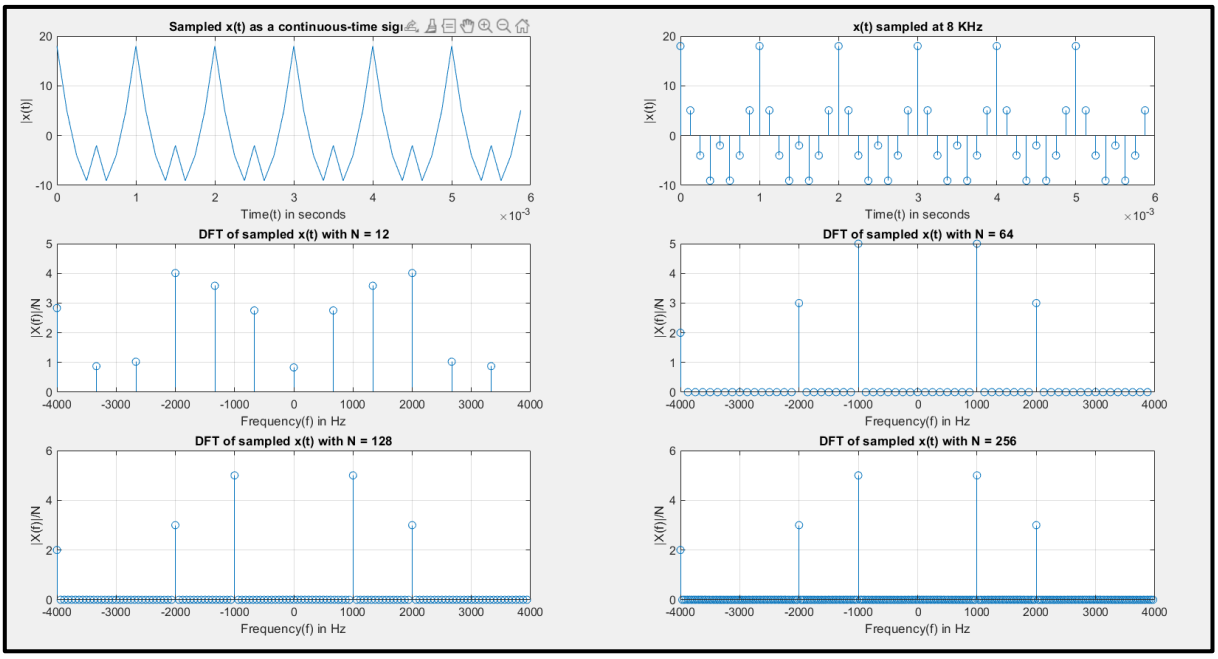
1. Set a desired frequency for the sinusoidal wave, and choose a sampling frequency above the Nyquist rate. Also set a variable for twice the sampling frequency.
2. Generate time intervals corresponding to the sampling frequencies, and another time interval with much a shorter time period to represent the original signal.
3. Sample the signal at the both the frequencies and plot them.
4. Add a zero between each sample to get twice the number of samples
5. Pass the zero-padded samples through a lowpass filter with a cut-off frequency of 6 kHz.
6. Plot the upsampled signal and compare with the samples taken at 24 kHz.
7. Plot the frequency spectra of both the samples and compare them as well.

Simulation Results: x(t) = 10cos(2π*1000t)+6cos(2π*2000t)+2*cos(2π*4000t)

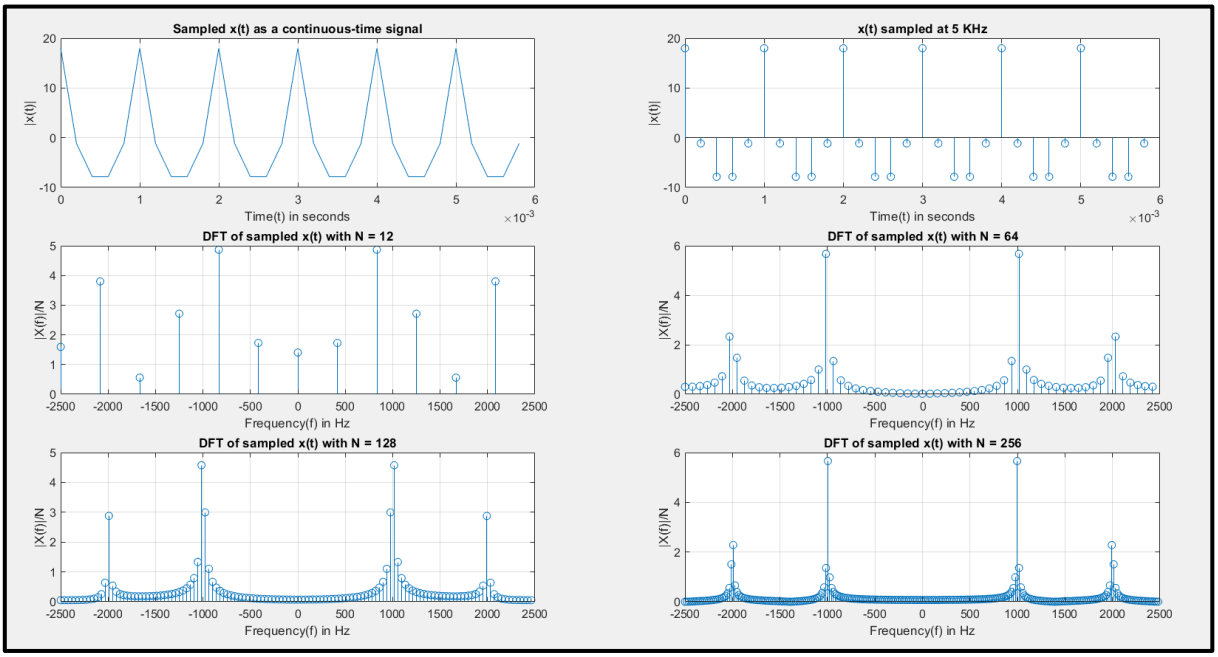
x(t) sampled at F_s = 12 kHz



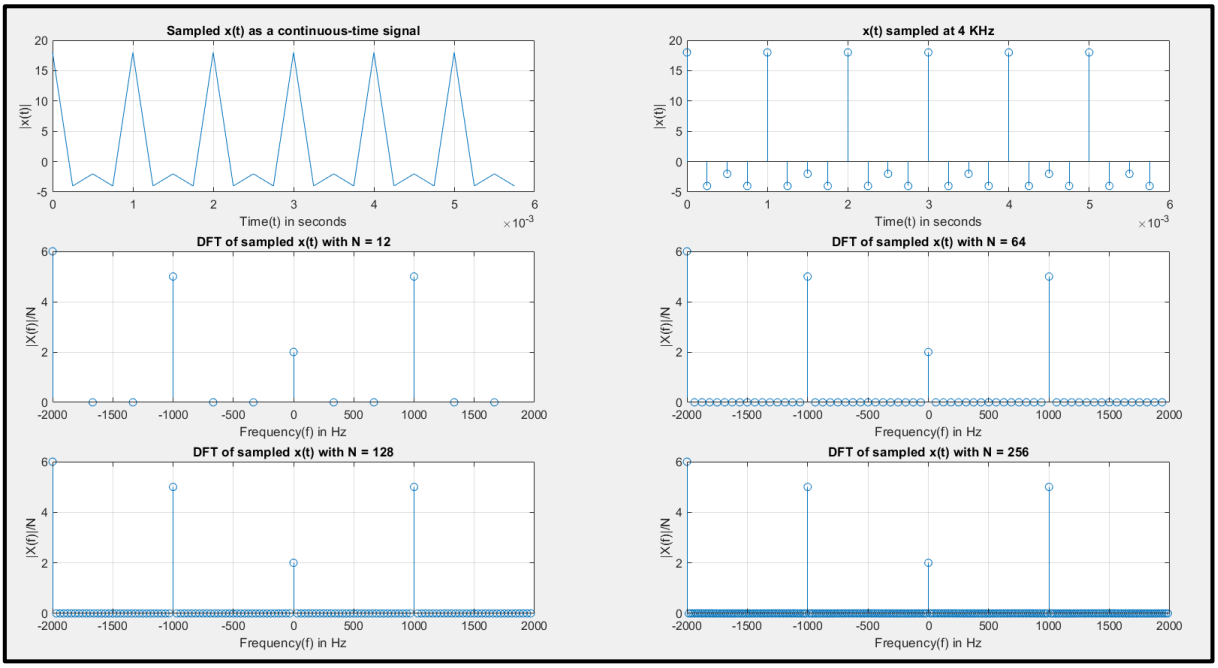
x(t) sampled at F_s = 8 kHz



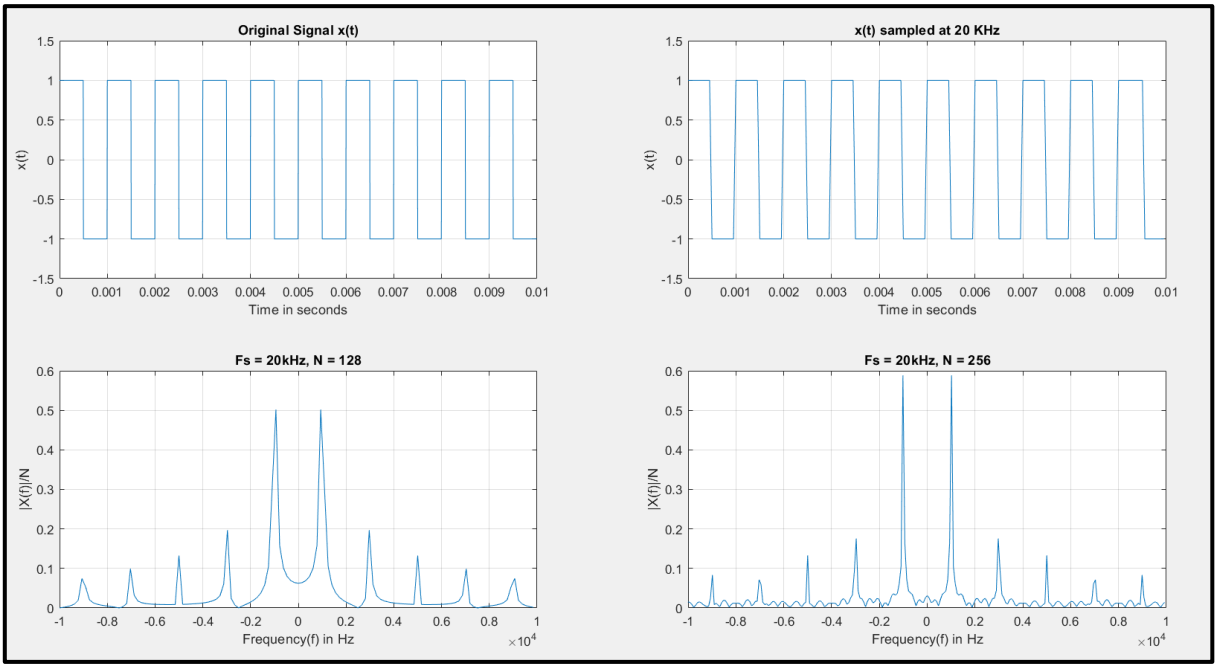
x(t) sampled at F_s = 5 kHz



x(t) sampled at F_s = 4 kHz

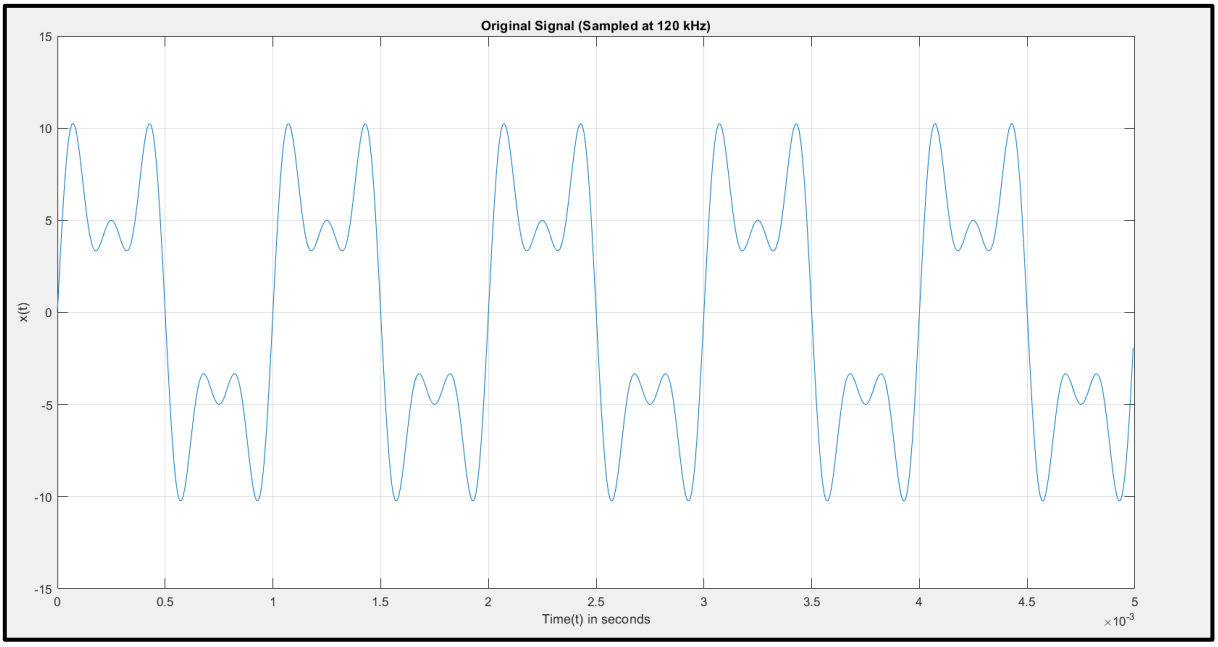


Frequency Spectrum of a Square Wave

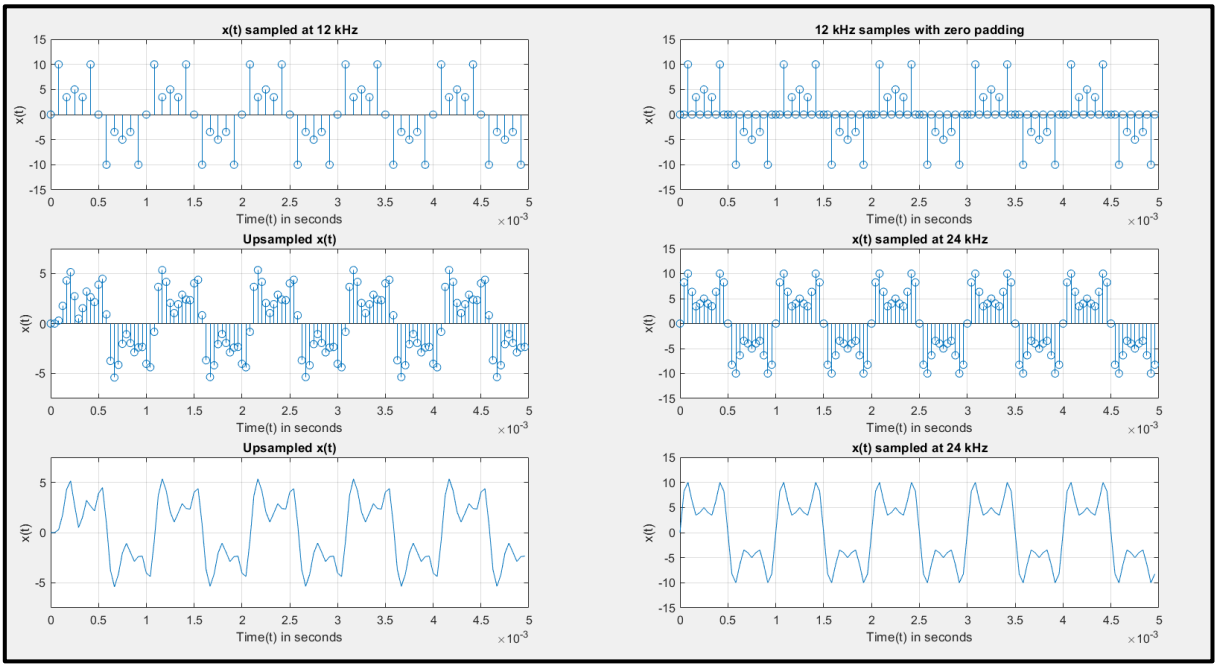


Original Signal (Actually Sampled at 120 kHz)

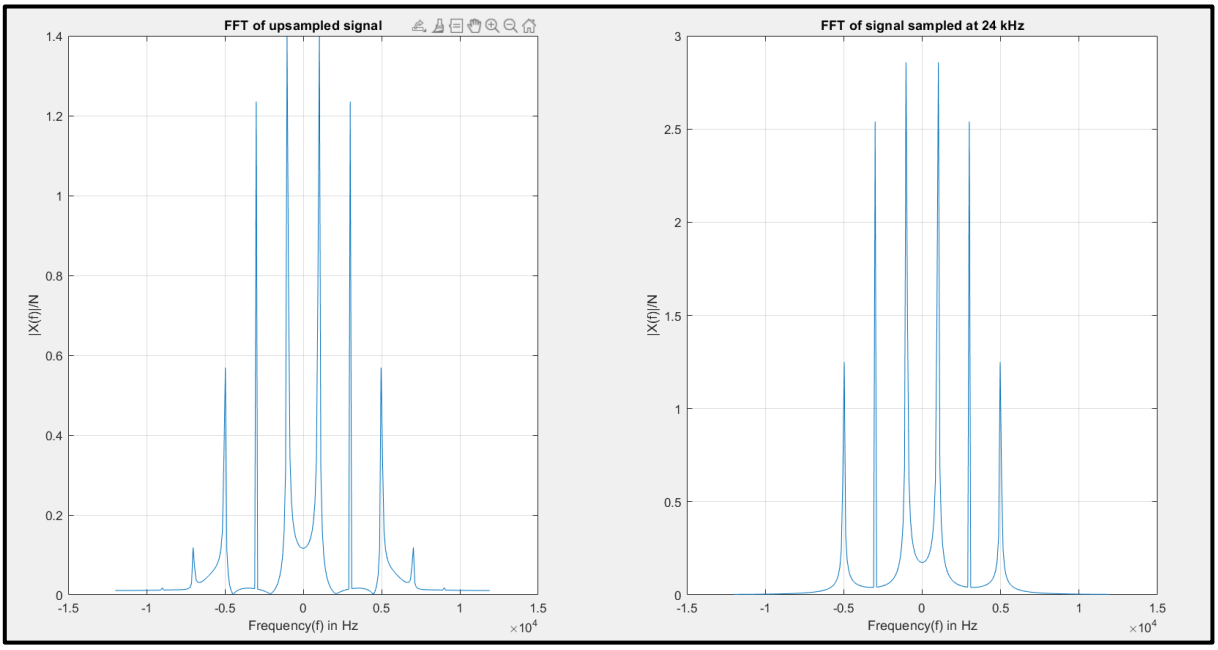
Equation: 3sin(2π*5000t)+5sin(2π*3000t)+7sin(2π*1000t)



Comparison Between Upsampled Signal and Signal Sampled at 24 kHz



FFT Plots of Upsampled Signal and Signal Sampled at 24 kHz



Results:

Thus, the benefits and limitations of sampling continuous-time signals in order to represent them as discrete-time signals were observed. The frequency spectrum of a square wave was also plotted, and the benefits of upsampling were made apparent through the simulation.

Discussion:

From this simulation experiment, we see how it is much more feasible to store signals sampled at discrete intervals of time rather than attempting to store a continuous-time signal, since we can recreate the original signal from its samples.

We see that sampling above the Nyquist sampling rate allows us to recreate the original faithfully, whereas sampling at a frequency less than the Nyquist rate results in aliasing, in which any component of the original signal with a frequency greater than half the sampling rate gets distorted.

An interesting point to note is that even though we have a limited number of frequency components in our signal, when we plot the FFT of the signal in frequency domain, we get non-zero values at many

other frequencies as well. This is because we are not sampling an infinite signal. The signal exists only within a time window. This windowing in the time-domain leads to the emergence of other frequencies. This is called the Gibb's Phenomenon, which states that a discontinuity in the time-domain requires an infinite frequency content.

While observing the frequency spectrum of a square wave with a frequency of 1 kHz, we observed peaks at odd harmonics of 1 kHz, which was expected because of the Fourier series of a square wave. However, we see that there are other non-zero components as well. They exist because of the Gibb's Phenomenon, since the square wave is also existing only within a time-window rather than extending to infinity. We also see that sampling at a lower rate leads to a slight imperfection during the rise and the fall of the square wave. It is not a line with infinite slope, which is there in an ideal square wave, but has a finite slope due to the limited number of samples and the interpolation of the plot using those samples.

Finally, we see how to upsample a signal, which is interpolating from a set of samples sampled above the Nyquist rate and generating a greater number of samples. In other words, it is possible to interpolate the intermediate *L-1* samples to obtain samples at a frequency of $F_{S2} = F_{S1}$, where F_{S1} is the original sampling frequency.

One reason to use upsampling rather than just sampling at a higher frequency from the start is that sampling at high frequencies requires better equipment, which is not cost-effective. Also, storing more samples leads to issues with storage and transmission, whereas with upsampling, we can increase the number of samples if and when needed, otherwise just use a lesser number of samples.

Appendix:

1. MATLAB Code for Parts A & B:

```
clc
clear all
close all

f = 1000; %frequency in Hz
fs = 4000; %sampling frequency in Hz
t = 0:1/fs:0.1; %time interval

x = 10*cos(2*pi*f*t)+6*cos(2*pi*2*f*t)+2*cos(2*pi*4*f*t); %Sampled Signal

subplot(3,2,1);
%Using plot to have an idea of what the continuous-time signal looks like
plot(t(1:6*fs/f), x(1:6*fs/f));
grid on;
title('Sampled x(t) as a continuous-time signal');
xlabel('Time(t) in seconds');
ylabel('|x(t)|');

subplot(3,2,2);
%Using stem rather than plot to show that this is a sampled signal
stem(t(1:6*fs/f), x(1:6*fs/f));
grid on;
title("x(t) sampled at " + fs/1000 + " KHz");
xlabel('Time(t) in seconds');
ylabel('|x(t)|');


plot_num = 3;
for N = [12, 64, 128, 256] %Iterating over desired values of N for N-point FFT
    dft = fftshift(fft(x, N)); % N-point DFT
    mag = abs(dft)/N; %Dividing by N to get similar energy distribution for all N
    f1 = -fs/2:fs/N:(fs/2 - fs/N); %frequency interval
    subplot(3,2,plot_num);
    plot_num = plot_num + 1;
    stem(f1, mag);
    grid on;
    title("DFT of sampled x(t) with N = " + N);
    xlabel('Frequency(f) in Hz');
    ylabel('|X(f)|/N');
end
```

2. MATLAB Code for Part C:

```
clc
clear all
close all

f = 1000; %frequency in Hz
fs = 20000; %sampling frequency in Hz
t = 0:1/(10*fs):0.1; %time interval to sample the signal at a much higher rate
ts = 0:1/fs:0.1; %time interval to sample to signal at desired rate of 20 kHz
```

```
x = square(2*pi*f*t); %A representation of the original signal
subplot(2,2,1);
plot(t(1:2000), x(1:2000));
grid on;
axis([0 0.01 -1.5 1.5]);
xlabel('Time in seconds'); ylabel('x(t)'); title('Original Signal x(t)');

xs = square(2*pi*f*ts); %Signal sampled at 20 kHz
subplot(2,2,2);
plot(ts(1:200), xs(1:200));
grid on;
axis([0 0.01 -1.5 1.5]);
xlabel('Time in seconds'); ylabel('x(t)'); title('x(t) sampled at 20 kHz');

plot_num = 3;
for N=[128, 256]
    Xs = fftshift(fft(xs, N)); %N-point DFT
    mag_s = abs(Xs)/N;
    f1 = -fs/2:fs/N:(fs/2 - fs/N); %frequency interval in Hz
    subplot(2,2,plot_num);
    plot_num = plot_num + 1;
    plot(f1, mag_s); %Using plot instead of stem for clarity
    grid on;
    xlabel('Frequency(f) in Hz'); ylabel('|X(f)|/N'); title("Fs = 20 kHz, N = " + N);
end
```

3. MATLAB Code for Part D:

```
clc
clear all
close all

f = 1000; %frequency in Hz
fs1 = 12000; %initial sampling frequency
fs2 = 24000; %higher sampling frequency

t = 0:1/(10*fs1):0.1; %time interval to represent the original signal
ts1 = 0:1/fs1:0.1; %time interval to represent the signal sampled at 12 kHz
ts2 = 0:1/fs2:0.1; %time interval to represent the signal sampled at 24 kHz

x = 3*sin(2*pi*5*f*t)+5*sin(2*pi*3*f*t)+7*sin(2*pi*f*t); %Signal equation

figure();

len = (length(t)-1)/20;
plot(t(1:len), x(1:len)); %Only plotting first few samples for clarity
grid on;
axis([0, 0.005 -15 15]); %Adjusting axes
xlabel('Time(t) in seconds'); ylabel('x(t)'); title('Original Signal (Sampled at 120 kHz)');

figure();

xs1 = 3*sin(2*pi*5*f*ts1)+5*sin(2*pi*3*f*ts1)+7*sin(2*pi*f*ts1); %12 kHz sampling

subplot(321);
len = (length(ts1)-1)/20;
```

```
stem(ts1(1:len), xs1(1:len)); %Only plotting first few samples for clarity
grid on;
axis([0, 0.005 -15 15]); %Adjusting axes
xlabel('Time(t) in seconds'); ylabel('x(t)'); title('x(t) sampled at 12 kHz');

%Adding zeros between samples for interpolation
z = zeros(1,length(ts1));
xs2 = [xs1(:) z(:)]';
xs2 = xs2(:);

subplot(322);
len = (length(ts2)-1)/20;
stem(ts2(1:len), xs2(1:len)); %Only plotting first few samples for clarity
grid on;
axis([0, 0.005 -15 15]); %Adjusting axes
xlabel('Time(t) in seconds'); ylabel('x(t)'); title('12 kHz samples with zero padding');

%Passing the samples through a 6th order butterworth filter of cut-off
%frequency of 6 kHz (normalized w.r.t half of sampling frequency
%as per function specification)
[b,a] = butter(6,(6*f)/(fs2/2));
filt_out = filter(b,a,xs2);

%plotting upsampled signal
up_xs1 = filt_out(1:length(filt_out)-1);
len = (length(ts2)-1)/20;
subplot(323);
stem(ts2(1:len), up_xs1(1:len)); %Only plotting first few samples for clarity
grid on;
axis([0, 0.005 -7.5 7.5]); %Adjusting axes
xlabel('Time(t) in seconds'); ylabel('x(t)'); title('Upsampled x(t)');

len = (length(ts2)-1)/20;
subplot(325);
plot(ts2(1:len), up_xs1(1:len)); %Using plot rather than stem for clarity
grid on;
axis([0, 0.005 -7.5 7.5]); %Adjusting axes
xlabel('Time(t) in seconds'); ylabel('x(t)'); title('Upsampled x(t)');

xs2 = 3*sin(2*pi*5*f*ts2)+5*sin(2*pi*3*f*ts2)+7*sin(2*pi*f*ts2); %24 kHz sampling

subplot(324);
len = (length(ts2)-1)/20;
stem(ts2(1:len), xs2(1:len)); %Only plotting first few samples for clarity
grid on;
axis([0, 0.005 -15 15]); %Adjusting axes
xlabel('Time(t) in seconds'); ylabel('x(t)'); title('x(t) sampled at 24 kHz');

subplot(326);
len = (length(ts2)-1)/20;
plot(ts2(1:len), xs2(1:len)); %Using plot rather than stem for clarity
grid on;
axis([0, 0.005 -15 15]); %Adjusting axes
xlabel('Time(t) in seconds'); ylabel('x(t)'); title('x(t) sampled at 24 kHz');

%Comparing the upsampled signal with signal sampled at a higher rate in the
%frequency domain
figure();
```

```
N = 256;
up_ys1 = fftshift(fft(up_xs1, N)); %getting N-point DFT of upsampled signal
mag_ys1 = abs(up_ys1)/N;
f1 = -fs2/2:fs2/N:(fs2/2 - fs2/N); %frequency interval
subplot(1,2,1);
plot(f1, mag_ys1); %Using plot rather than stem for clarity
grid on;
xlabel('Frequency(f) in Hz'); ylabel('|X(f)|/N'); title('FFT of upsampled signal');

ys2 = fftshift(fft(xs2, N)); %getting N-point DFT of signal sampled at 24 kHz
mag_ys2 = abs(ys2)/N;
f1 = -fs2/2:fs2/N:(fs2/2 - fs2/N);
subplot(1,2,2);
plot(f1, mag_ys2); %Using plot rather than stem for clarity
grid on;
xlabel('Frequency(f) in Hz'); ylabel('|X(f)|/N'); title('FFT of signal sampled at 24 kHz');
```