Index

Experiment Number: 01

Experiment Name: Representation of Basic signals

i) Sine wave
ii) Cosine wave
iii) Unit impulse
iv) Unit step wave
v) Square wave
vi) Exponential waveform.

Objective: The main objective of this experiment is to implement the basic signals and identify basic difference between them.

Theory:

Sine wave: A sine wave is a periodic signal that oscillates between two values, one positive and one negative.

Cosine wave: A cosine wave is similar to a sine wave, but it oscillates between zero and one.

Unit impulse: A unit impulse is a signal that has infinite amplitude and zero width. It is often represented by the Dirac delta function, which is a mathematical function that is zero everywhere except at $t = 0$, where it is infinite.

Unit step wave: A unit step wave is a signal that is zero for negative values of t and one for positive values of t.

Square wave: A square wave is a signal that alternates between two values, one for positive values of t and one for negative values of t.

Exponential waveform: An exponential waveform is a signal that increases or decreases exponentially with time.

Algorithm:

Step I: Get the time interval.

Step II: Put Amplitude and Frequency.

Step III: Calculate the equation.

Step IV: Plot and stem the calculated equation.

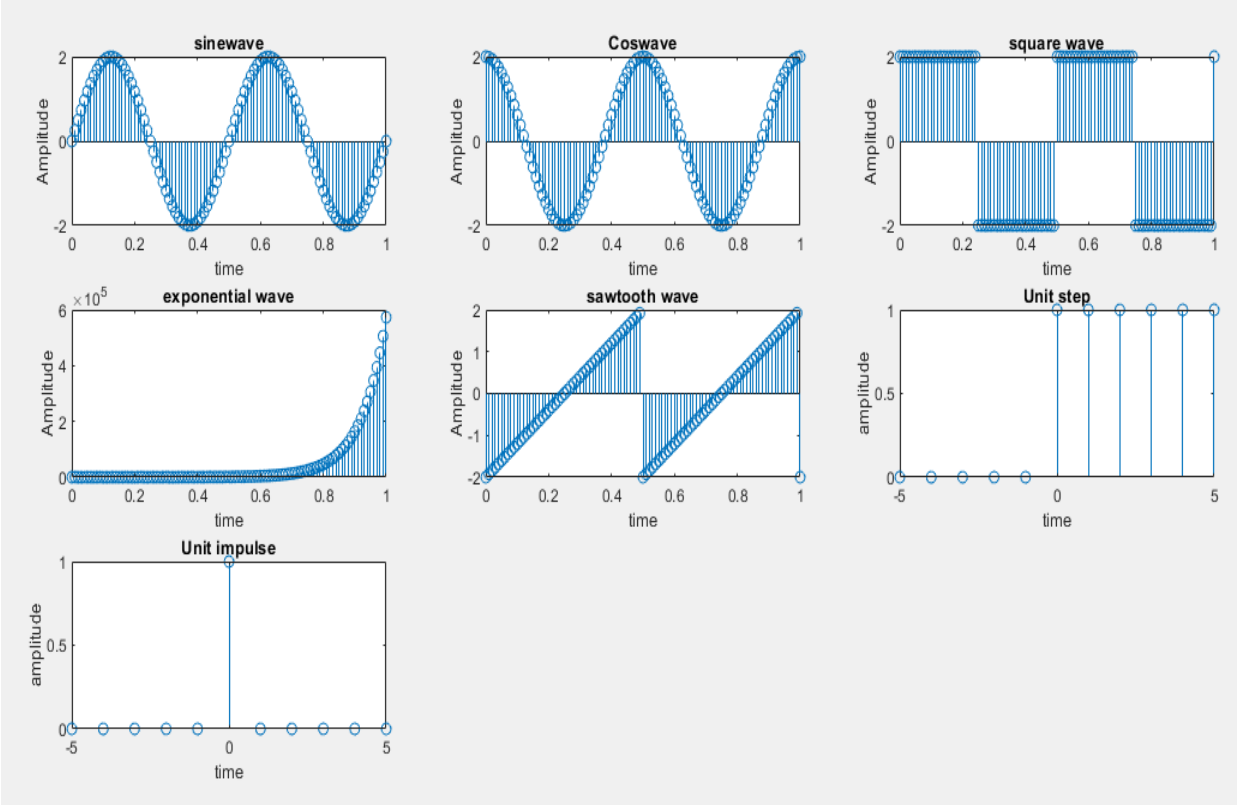
Step V: Display the above outputs.

Code:

```
% sine wave
t=0:0.01:1;
a=2;
b=a*sin(2*pi*2*t);
subplot(3,3,1);
stem(t,b);
xlabel('time');
ylabel('Amplitude');
title('sinewave');

% Cosine wave
t=0:0.01:1;
a=2;
b=a*cos(2*pi*2*t);
subplot(3,3,2);
stem(t,b);
xlabel('time');
ylabel('Amplitude');
title ('Coswave');


% Square wave
t=0:0.01:1;
a=2;
b=a*square(2*pi*2*t);
subplot(3,3,3);
stem(t,b);
xlabel('time');
ylabel('Amplitude');
title('square wave');

% Exponential waveform
t=0:0.01:1;
a=2;
b=a*exp(2*pi*2*t);
subplot(3,3,4);
stem(t,b);
xlabel('time');
ylabel('Amplitude');
title ('exponential wave');
```

```
%sawtooth
t=0:0.01:1;
a=2;
b=a*sawtooth(2*pi*2*t);
subplot(3,3,5);
stem(t,b);
xlabel('time');
ylabel('Amplitude');
title ('sawtooth wave');

% unit step signal
n=-5:5;
a = [zeros(1,5),ones(1,6)];
subplot(3,3,6);
stem(n,a);
xlabel ('time');
ylabel ('amplitude');
title('Unit step');

% unit impulse
n=-5:5;
a = [zeros(1,5),ones(1,1),zeros(1,5)];
subplot(3,3,7);
stem(n,a);
xlabel ('time');
ylabel ('amplitude');
title('Unit impulse');
```

Output:



Discussion: These are just a few of the basic signals that are used in signal processing. The following table summarizes the properties of the basic signals:

Signal	Amplitude	Frequency	Phase
Sine wave	A	ω	θ
Cosine wave	A	ω	-θ
Unit impulse	∞	0	0
Unit step wave	1	0	0
Square wave	A	T/2	0
Exponential waveform	A	β	0

Experiment Number: 02

Experiment Name: To find DFT / IDFT of given DT signal.

Objective: The main motto of this experiment is to convert the time domain of a signal into frequency domain and vice versa.

Theory: The Discrete Fourier Transform (DFT) and the Inverse Discrete Fourier Transform (IDFT) are two closely related mathematical operations that are used to analyze and synthesize signals. The DFT converts a signal from the time domain to the frequency domain, while the IDFT converts a signal from the frequency domain to the time domain.

The DFT is defined as follows:

$X(k) = \sum_{n=0}^{N-1} x(n) e^{(-j*2*pi*k*n/N)}$

The IDFT is defined as follows:

$x(n) = 1/N \sum_{k=0}^{N-1} X(k) e^{(j*2*pi*k*n/N)}$

Algorithm:

Step I: Get the input sequence.

Step II: Find the DFT of the input sequence using direct equation of DFT.

Step III: Find the IDFT using the direct equation.

Step IV: Plot DFT and IDFT of the given sequence using matlab command stem.

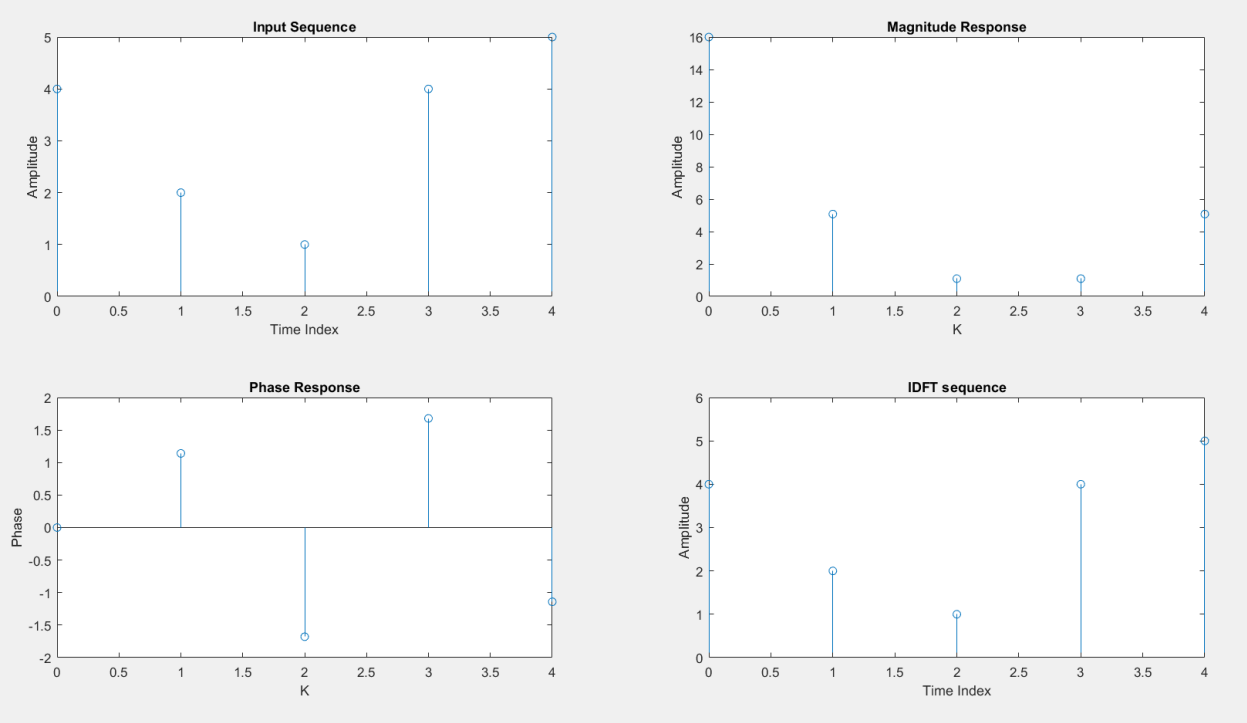
Step V: Display the above outputs.

Code:

```
xn=input('Enter the sequence x(n): ');
ln=length(xn);
xk=zeros(1,ln);

for k=0:ln-1
    for n=0:ln-1
        xk(k+1)=xk(k+1)+(xn(n+1)*exp(-(1i)*2*pi*k*n/ln));
    end
end

t=0:ln-1;
subplot(2, 2, 1);
stem(t,xn);
ylabel ('Amplitude');
xlabel ('Time Index');
title('Input Sequence');

magnitude=abs(xk);
t=0:ln-1;
subplot(2, 2, 2);
stem(t,magnitude);
ylabel ('Amplitude');
xlabel ('K');
title('Magnitude Response');

phase=angle(xk);
t=0:ln-1;
subplot(2, 2, 3);
stem(t,phase);
ylabel ('Phase');
xlabel ('K');
title ('Phase Response');

ixk=zeros(1,ln);
for n=0:ln-1
    for k=0:ln-1

ixk(n+1)=ixk(n+1)+real((xk(k+1)*exp((1i)*2*pi*k*n/ln)));
    end
end

ixk=ixk./ln;
```

```
t=0:ln-1;
subplot(2, 2, 4);
stem(t,ixk);
ylabel ('Amplitude');
xlabel ('Time Index');
title ('IDFT sequence');
```

Output: Enter the sequence x(n): [4 2 1 4 5]



Discussion: The DFT and IDFT are both linear operations, which means that they can be combined to perform more complex operations, such as filtering and convolution. The DFT and IDFT are widely used in many different areas of signal processing, including audio processing, image processing, and digital communications.

Experiment Number: 03

Experiment Name: Linear convolution of two sequences.

Objective: The main motto of this experiment is to take two sequences as input and produces a third sequence as output using convolution of two sequences.

Theory: The linear convolution of two sequences is a mathematical operation that takes two sequences as input and produces a third sequence as output. The output sequence is the result of multiplying the elements of the two input sequences at corresponding time positions and summing the products.

The formal definition of linear convolution is as follows:

$y[n] = \sum_{m=-\infty}^{\infty} x[m]\, h[n-m]$

Algorithm:

Step I: Get the time interval.

Step II: Put the input signal sequence.

Step III: Calculate the equation.

Step IV: Stem the calculated equation.

Step V: Display the above outputs.

Code:

```
x1 = input('Enter the first sequence: ');
subplot(3, 1, 1);
stem(x1);
ylabel('Amplitude');
title('Plot of the first sequence');

x2 = input('Enter the second sequence: ');
subplot(3, 1, 2);
stem(x2);
ylabel('Amplitude');
title('Plot of the second sequence');

f = conv(x1, x2);
disp("Convolution: ");
disp(f);

subplot(3, 1, 3);
stem(f);
xlabel('Time index n');
ylabel('Amplitude f');
title('Linear convolution of sequence');
```

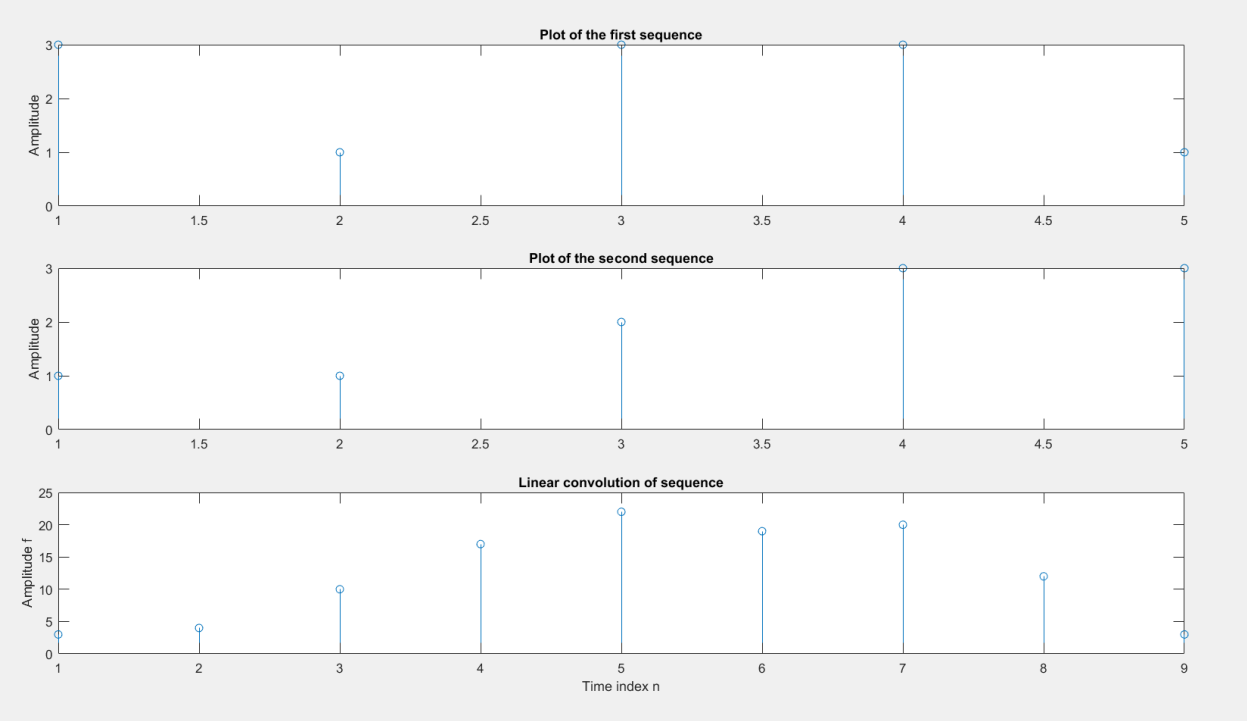
Output:

Enter the first sequence: [3 1 3 3 1]

Enter the second sequence: [1 1 2 3 3]

Convolution:

3 4 10 17 22 19 20 12 3



Discussion: The linear convolution operation can be implemented in a number of different ways. One common way is to use a recursive algorithm. Another is Fast Fourier Transform (FFT). The linear convolution operation is a very important operation in signal processing. It is used in a wide variety of applications, including digital filtering, image processing, and speech recognition.

Experiment Number: 04

Experiment Name: Auto correlation.

Objective: The main motto of this experiment is to compute auto correlation between two sequences.

Theory: Autocorrelation is a statistical measure of the similarity between a time series and a lagged version of itself. It is a measure of the correlation between observations of a random variable as a function of the time lag between them.

The autocorrelation function (ACF) is a plot of the autocorrelation of a time series as a function of the lag. The ACF can be used to determine the presence of autocorrelation in a time series and to identify the order of an autoregressive model.

Algorithm:

Step I : Give input sequence x[n].

Step II : Give impulse response sequence h (n)

Step III: Find auto correlation using the matlab command xcorr.

Step IV: Plot x[n], h[n], z[n].

Step V: Display the results

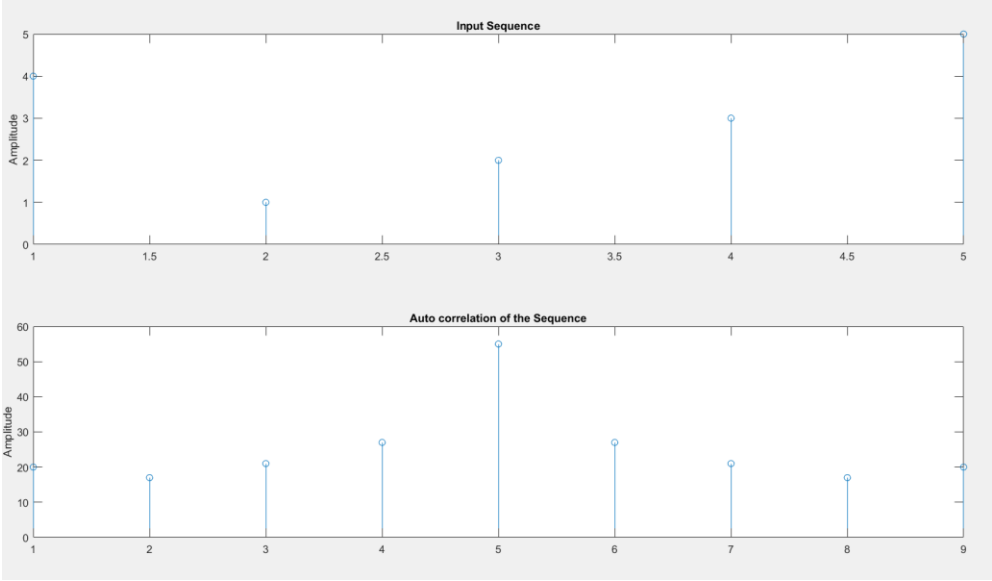
Code:

```
x = input('Enter sequence: ');
subplot(2,1,1);
stem(x);
ylabel('Amplitude');
title('Input Sequence');

subplot(2,1,2);
y = xcorr(x,x);
stem(y);
ylabel('Amplitude');
title('Auto correlation of the Sequence');

disp("Auto correlation: ");
disp(y);
```

Output:

Enter sequence: [4 1 2 3 5]

Auto correlation: 20 17 21 27 55 27 21 17 20



Discussion: The autocorrelation function is a powerful tool for analyzing time series data. It can be used to identify the presence of autocorrelation in a time series and to determine the order of an autoregressive model. Autocorrelation is a versatile statistical tool that can be used to analyze a wide variety of data.

Experiment Number: 05

Experiment Name: To find the FFT of a given sequence.

Objective: The main motto of this experiment is to find the FFT of a given sequence.

Theory: The Fast Fourier Transform (FFT) is a mathematical operation that converts a signal from the time domain to the frequency domain. The FFT can be used to analyze the frequency content of a signal. The fft function then calls itself recursively to compute the FFT of the sequence. The recursive calls are used to divide the sequence into smaller and smaller pieces until the FFT of each piece can be computed directly.

Algorithm:

Step I: Give input sequence x[n].

Step II: Find the length of the input sequence using length command.

Step III: Find FFT and IFFT using matlab commands fft

Step IV: Plot magnitude and phase response

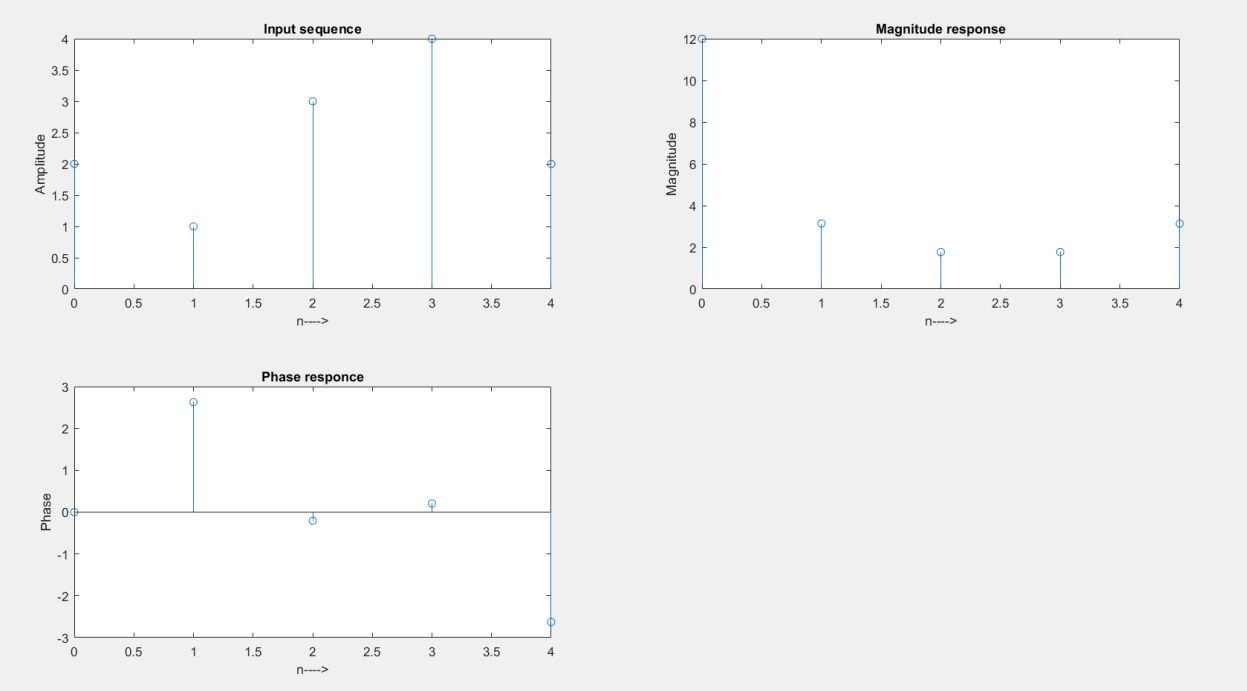
Step V: Display the results

Code:

```
x=input('Enter the sequence : ');
N=length(x);
xk=fft(x);

n=0:N-1;
subplot (2,2,1);
stem(n,x);
xlabel('n---->');
ylabel('Amplitude');
title('Input sequence');

subplot (2,2,2);
stem(n,abs(xk));
xlabel('n---->');
ylabel('Magnitude');
title('Magnitude response');

subplot (2,2,3);
stem(n,angle(xk));
xlabel('n---->');
ylabel('Phase');
title('Phase responce');
```

Output:

Enter the sequence : [2 1 3 4 2]



Discussion: The FFT algorithm is a very efficient algorithm for computing the DFT of a sequence. The number of operations required to compute the DFT of a sequence using the FFT algorithm is O(N log N), where N is the length of the sequence. This is much more efficient than the direct implementation of the DFT, which requires O(N^2) operations.

Experiment Number: 06

Experiment Name: Implementation of interpolation process.

Objective: The main motto of this experiment is to verify the decimation of given sequence.

Theory: Interpolation is the process of estimating the value of a function at a point that is not within the range of the function's given data points. There are many different interpolation methods:

Linear interpolation: This method estimates the value of the function by drawing a straight line between the two nearest data points.

Polynomial interpolation: This method estimates the value of the function by fitting a polynomial to the data points.

Spline interpolation: This method estimates the value of the function by fitting a spline to the data points.

Algorithm:

Step I: Define up sampling factor and input frequencies f1 and f2

Step II: Represent input sequence with frequencies f1 and f2

Step III: Perform the interpolation on input signal using MATLAB command interp.

Step IV: Plot the input and output signal/sequence.

Code:

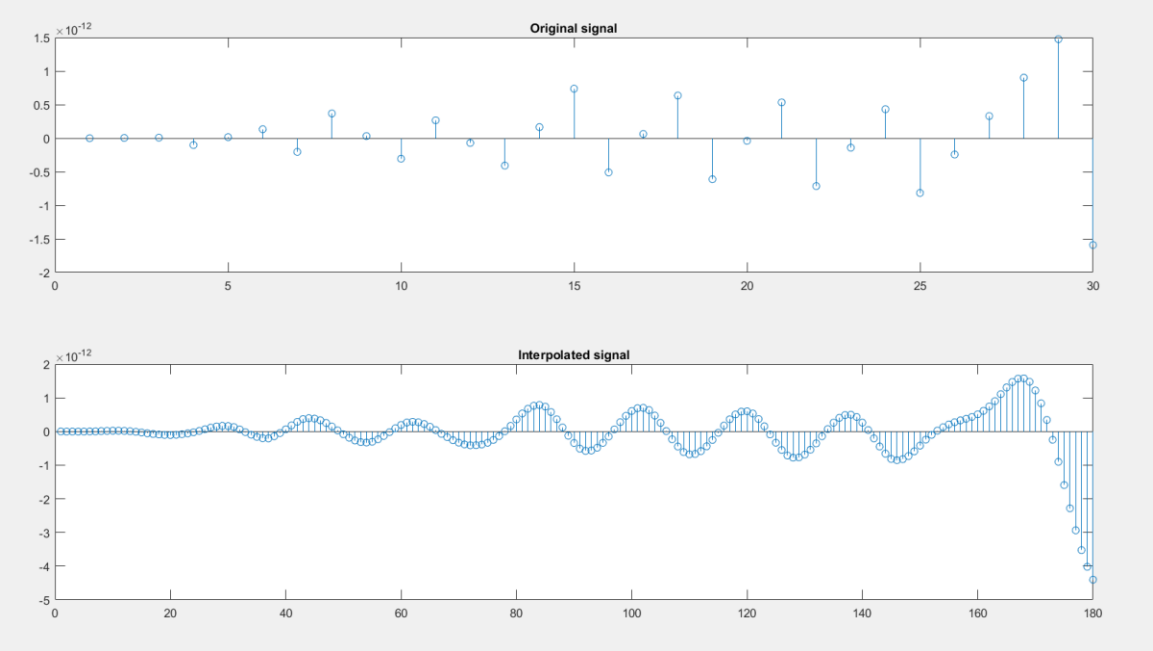
```
F=input('Enter the frequency of the signal: ');
P=input('Enter the interpolator factor: ');
N=input('Enter the length of the signal: ');

t=0:1:N-1;
X = sin(2*pi*F*t);
subplot(2,1,1);
stem(X);
title("Original signal");

i=interp(X,P);
subplot(2,1,2);
stem(i);
title("Interpolated signal");
```

Output:

Enter the frequency of the signal: 100

Enter the interpolator factor: 6

Enter the length of the signal: 30



Discussion: The interpolation process is a powerful tool that can be used to estimate the value of a function at a point that is not explicitly defined by the function. It can be used in a variety of applications to improve the accuracy and reliability of data analysis, curve fitting, image processing, and signal processing.

Experiment Number: 07

Experiment Name: Computation of n-point DFT of a given sequences.

Objective: The main motto of this experiment is to compute the N (=4/8/16) point DFT of the given sequence.

Theory: The n-point Discrete Fourier Transform (DFT) of a given sequence is a mathematical operation that converts the sequence from the time domain to the frequency domain. The DFT can be used to analyze the frequency content of a signal.

The DFT of a sequence is defined as follows:

X[k] = ∑_n=0^N-1 x[n] e^(-j*2*pi*k*n/N)

Algorithm:

Step I: Get the time interval.

Step II: Put the input signal.

Step III: Calculate the equation.

Step IV: Stem the calculated equation.

Step V: Display the above outputs.

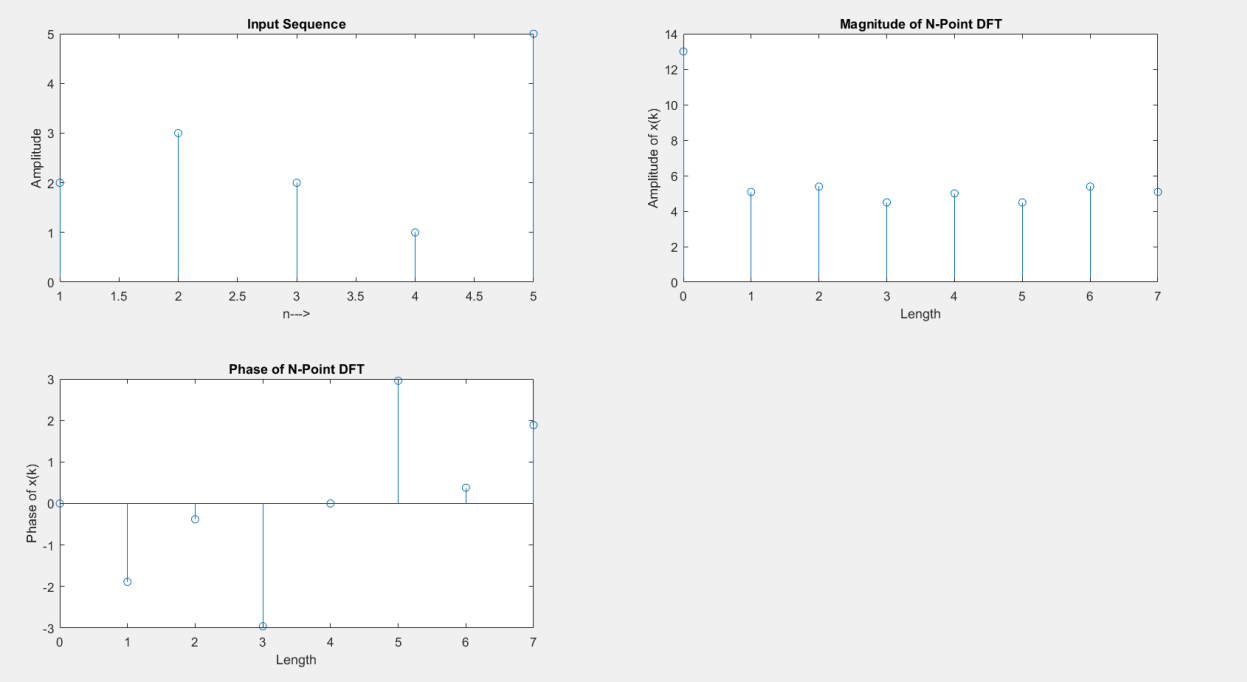
Code:

```
x = input('Enter the sequence: ');
n = input('Enter the value of N: ');
ndft = fft(x,n);

subplot(2,2,1);
stem(x);
xlabel("n--->");
ylabel("Amplitude");
title("Input Sequence");

m = abs(ndft);
N = 0:1:n-1;
subplot(2,2,2);
stem(N,m);
xlabel('Length');
ylabel('Amplitude of x(k)');
title('Magnitude of N-Point DFT');

p = angle(ndft);
subplot(2,2,3);
stem(N,p);
xlabel('Length');
ylabel('Phase of x(k)');
title('Phase of N-Point DFT');
```

Output:

Enter the sequence: [2 3 2 1 5]

Enter the value of N: 8



Discussion: The DFT can be computed using a recursive algorithm, Fast Fourier Transform (FFT). The FFT is a very efficient algorithm for computing the DFT of a sequence. The FFT can be used to compute the DFT of a sequence of any length. We have studied the computation of N point DFT of a given sequence and also plot magnitude and phase spectrum.

Experiment Number: 08

Experiment Name: Verification of sampling theorem.

Objective: The main motto of this experiment is to verify sampling theorem.

Theory: The sampling theorem is a fundamental principle in signal processing that states that a bandlimited signal can be perfectly reconstructed from its samples if the sampling frequency is greater than twice the highest frequency component of the signal.

The sampling theorem states that the sampling frequency must be greater than twice the highest frequency component of the analog signal in order to perfectly reconstruct the signal from the samples. If the sampling frequency is not high enough, then aliasing will occur. Aliasing is the phenomenon where high-frequency components of the signal are mistaken for low-frequency components.

Algorithm:

Step I: Get the time interval.

Step II: Put Amplitude and Frequency.

Step III: Calculate the equation.

Step IV: Plot and stem the calculated equation.

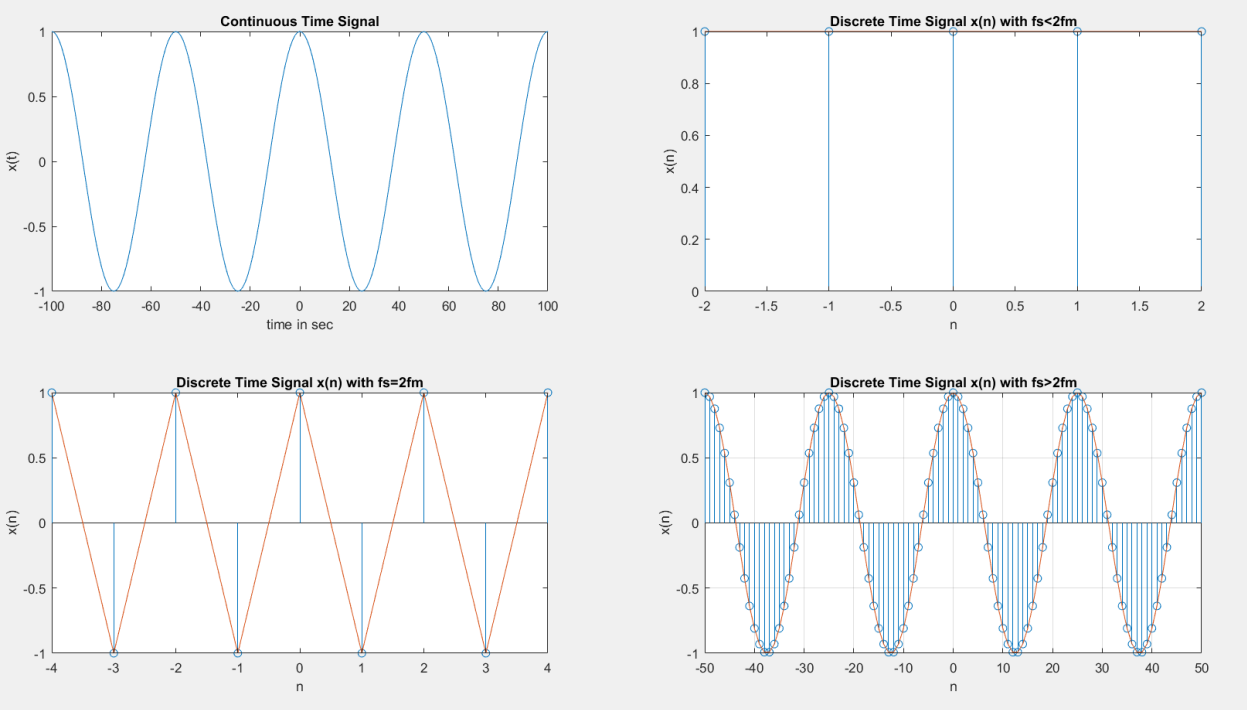
Step V: Display the above outputs.

Code:

```
t=-100:01:100;
fm=0.02;
x=cos(2*pi*t*fm);
subplot(2,2,1);
plot(t,x);
xlabel('time in sec');
ylabel('x(t)');
title('Continuous Time Signal');

fs1=0.02;
n=-2:2;
x1=cos(2*pi*fm*n/fs1);
subplot(2,2,2);
stem(n,x1);
hold on

subplot(2,2,2);
plot(n,x1);
title('Discrete Time Signal x(n) with fs<2fm');
xlabel('n');
ylabel('x(n)');

fs2=0.04;
n1=-4:4;
x2=cos(2*pi*fm*n1/fs2);
subplot(2,2,3);
stem(n1,x2);
hold on

subplot(2,2,3);
plot(n1,x2);
title('Discrete Time Signal x(n) with fs=2fm');
xlabel('n');
ylabel('x(n)');


n2=-50:50;
fs3=0.5;
x3=cos(2*pi*fm*n2/fs3);
subplot(2,2,4);
stem(n2,x3);
hold on
```

```
subplot(2,2,4);
plot(n2,x3);
grid on
xlabel('n');
ylabel('x(n)');
title('Discrete Time Signal x(n) with fs>2fm');
```

Output:



Discussion: The sampling theorem is a powerful tool that can be used to convert analog signals to digital signals and to analyze the frequency content of signals. The sampling theorem has many important applications in signal processing. It is used in digital audio and video, telecommunications, and radar.

Experiment Number: 09

Experiment Name: To Design a Butterworth digital IIR filter.

Objective: The main motto of this experiment is to design a Butterworth digital IIR filter.

Theory: A Butterworth digital IIR filter is a type of filter that is designed to have a flat frequency response in the passband and a sharp roll-off in the stopband. Butterworth filters are often used in audio applications, where it is important to preserve the quality of the signal while removing unwanted frequencies.

The Butterworth filter is a type of infinite impulse response (IIR) filter, which means that its output depends on both the present input and past inputs. IIR filters are typically more computationally efficient than finite impulse response (FIR) filters, but they can also be more difficult to design.

Algorithm:

Step-I: Get the pass band and stop band ripples.

Step-II: Get the pass band and stop band frequencies.

Step-III: Get the sampling frequency.

Step-IV: Calculate the order of the filter using

$\log \sqrt{[10^{kp} - 1 / 10^{kp} - 1]}\ N$

$\log < s / < p$

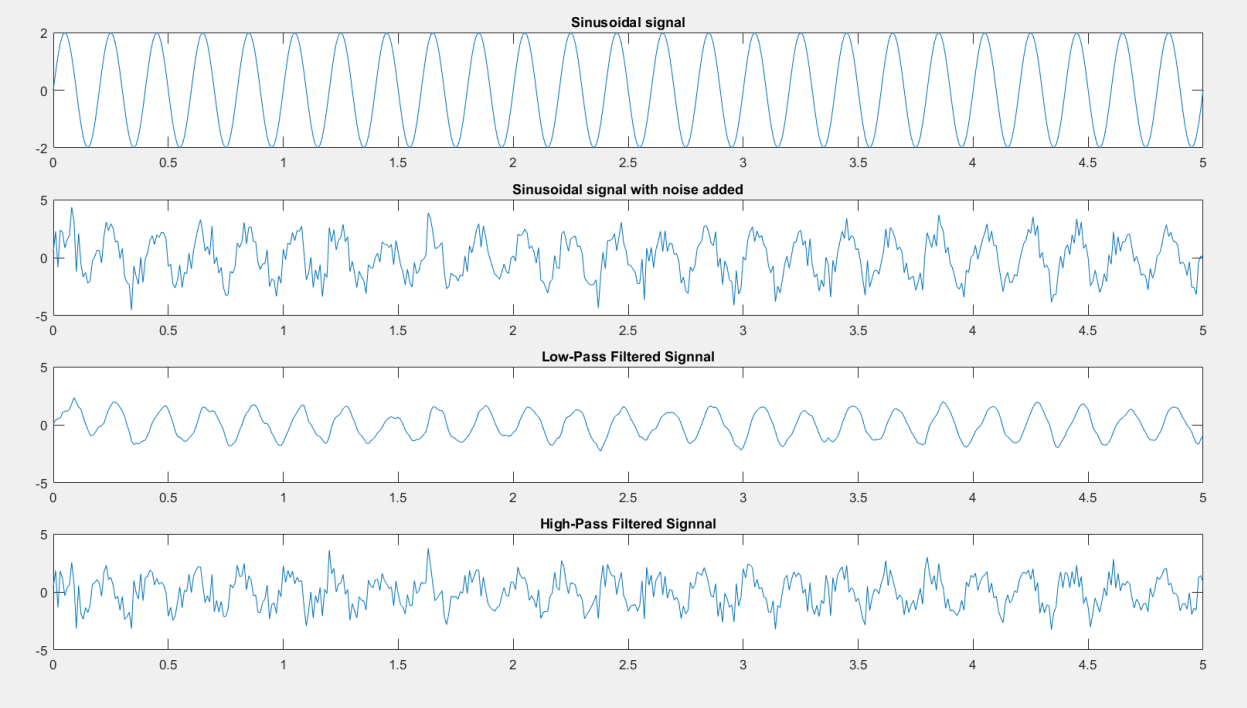
Step-V: Find the filter co-efficient.

Step-VI: Draw the magnitude and phase response.

Code:

```
fs=100;
f=5;
t=5;
n= 0: 1/fs :t;
x=2*sin(2*pi*f*n);
subplot(4,1,1);
plot(n,x);
title('Sinusoidal signal');

z=awgn(x,1);
subplot(4,1,2);
plot(n,z);
title('Sinusoidal signal with noise added');

wc = 0.1;
o = 1;

%Low - pass Filter
[b,a]=butter(o,wc,'low');
iir=filter(b,a,z);
subplot(4,1,3);
plot(n,iir);
title("Low-Pass Filtered Signnal");

%High - pass Filter
[b,a]=butter(o,wc,'high');
iir=filter(b,a,z);
subplot(4,1,4);
plot(n,iir);
title("High-Pass Filtered Signnal");
```

Output:



Discussion: Butterworth filters are often used in audio applications, where it is important to preserve the quality of the signal while removing unwanted frequencies. For example, a Butterworth filter can be used to remove the high-frequency noise from an audio signal, while preserving the low-frequency components of the signal.