# D Y Patil International University, Akurdi, Pune

**Date of Performance:** 6th Sept' 21

## Experiment No: 01

## AIM: To generate the discrete time sequences of:

(a) Unit impulse sequence
(b) Unit step sequence
(c) Ramp sequence
(d) Exponential sequence
(e) Sinusoidal sequence
(f) Cosine sequence.

**APPARATUS:** MATLAB Version 7.8 (R2009a)

### PROGRAM CODE

#### a) PROGRAM for the generation of unit impulse signal

```
t=-2:1:2 ;
y=[zeros(1,2),ones(1,1),zeros(1,2)] ;
subplot(3,2,1) ;
stem(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(a)n ---->') ;
title(' unit impulse signal ') ;
```

#### b) PROGRAM for the generation of unit step

**Sequence [u(n)-u(n-N)]**

```
n=input('enter the N value ') ;
t=0:1:n-1 ;
y=ones(1,n) ;
subplot(3,2,2) ;
stem(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(b)n ---->') ;
title(' unit step sequence ') ;
```

**c) PROGRAM for the generation of ramp sequence**

```
n=input('enter the length of ramp sequence ') ;
t=0:n-1 ; subplot(3,2,3) ;
stem(t,t) ;
ylabel('amplitude ---->') ;
xlabel('(c)n ---->') ;
title(' ramp sequence ') ;
```

**d) PROGRAM for the generation of exponential sequence**

```
n=input('enter the length of exponential sequence') ;
t=0:n ;
a=input('enter the value of a ') ;
y=exp(a*t) ;
subplot(3,2,4) ;
stem(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(d)n ---->') ;
title(' exponential sequence ') ;
```

**e) PROGRAM for the generation of sinsoidal sequence**

```
t=0:0.01:pi ;
y=sin(2*pi*t) ;
subplot(3,2,5) ;
plot(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(e)n ---->') ;
title(' sine sequence ') ;
```

**f) PROGRAM for the generation of cosine sequence**

```
t=0:0.01:pi ;
y=cos(2*pi*t) ;
subplot(3,2,6) ;
plot(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(f)n ---->') ;
title (' cosine sequence ') ;
```

# OUTPUT

MATLAB R2020b - academic use

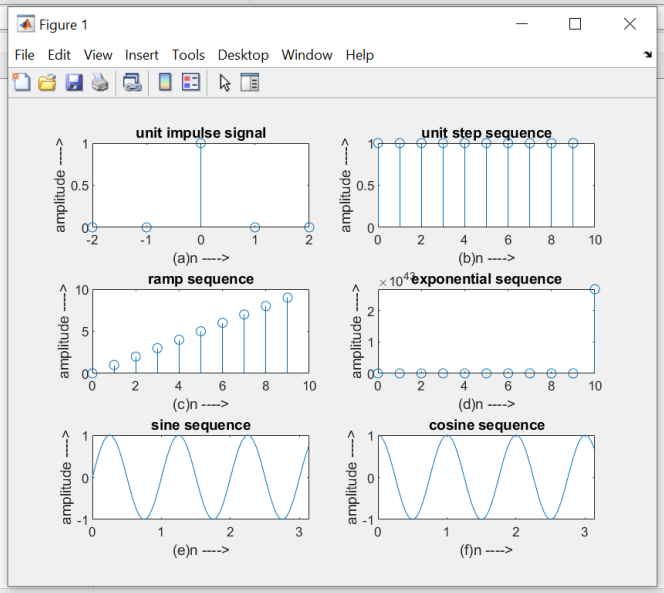
```
clc ;
clear all ;
close all ;

% PROGRAM for the generation of unit impulse signal

t=-2:1:2 ;
y=[zeros(1,2),ones(1,1),zeros(1,2)] ;
subplot(3,2,1) ;
stem(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(a)n ---->') ;
title(' unit impulse signal ') ;

% PROGRAM for the generation of unit step

n=input('enter the N value ') ;
t=0:1:n-1 ;
y=ones(1,n) ;
subplot(3,2,2) ;
stem(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(b)n ---->') ;
title(' unit step sequence ') ;

% PROGRAM for the generation of ramp sequence

n=input('enter the length of ramp sequence ') ;
t=0:n-1 ; subplot(3,2,3) ;
stem(t,t) ;
ylabel('amplitude ---->') ;
xlabel('(c)n ---->') ;
title(' ramp sequence ') ;


% PROGRAM for the generation of exponential sequence

n=input('enter the length of exponential sequence') ;
t=0:n ;
a=input('enter the value of a ') ;
y=exp(a*t) ;
subplot(3,2,4) ;
stem(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(d)n ---->') ;
title(' exponential sequence ') ;
```

Command Window

Ready | UTF-8 | script | Ln 48 | Col 1

MATLAB R2020b - academic use

```
% PROGRAM for the generation of sinusoidal sequence

t=0:0.01:pi ;
y=sin(2*pi*t) ;
subplot(3,2,5) ;
plot(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(e)n ---->') ;
title(' sine sequence ') ;


% PROGRAM for the generation of cosine sequence

t=0:0.01:pi ;
y=cos(2*pi*t) ;
subplot(3,2,6) ;
plot(t,y) ;
ylabel('amplitude ---->') ;
xlabel('(f)n ---->') ;
title (' cosine sequence ') ;
```




Command Window

UTF-8 | script | Ln 48 | Col 1

**RESULT**:- Thus the generation of discrete-time sequences of unit impulse, unit step, ramp, exponential, sinusoidal and cosine sequence is simulated using MATLAB.

**VIVA QUESTIONS** :

1. Define unit sample(impulse) sequence.
2. Define unit step sequence.
3. Define discrete-time system.
4. Distinguish the terms ‘discrete’ and ‘digital’ w.r.t DSP.
5. Define sampling theorem.

**ANSWERS:**

1. The unit impulse sequence is denoted as $\delta(n)$ and is defined as $\delta(n) = 1$ for $n = 0$ $\delta(n) = 0$ for $n \neq 0$.

2. The unit step sequence is denoted as $u(n)$ and is defined as $u(n) = 1$ for $n \geq 0$ $u(n) = 0$ for $n < 0$.

3. A discrete-time system is a device or algorithm that operates on a discrete-time input Signal x(n), according to some well-defined rule, to produce another discrete-time signal y(n) called the output signal.

4. Discrete may have the various amplitudes at respective interval but the digital signal should have either zero or one.

5. The sampling theorem states that perfect reconstruction of a signal is possible when the sampling frequency is greater than twice the maximum frequency of the signal being sampled, or equivalently, when the frequency (half the sample rate) exceeds the highest frequency of the signal being sampled. If lower sampling rates are used, the original signal's information may not be completely recoverable from the sampled signal. For example, if a signal has an upper band limit of 100 Hz, a sampling frequency greater than 200 Hz will avoid aliasing and allow theoretically perfect reconstruction.

**Date of Performance:** 20th Sept' 21

# Experiment No: 02

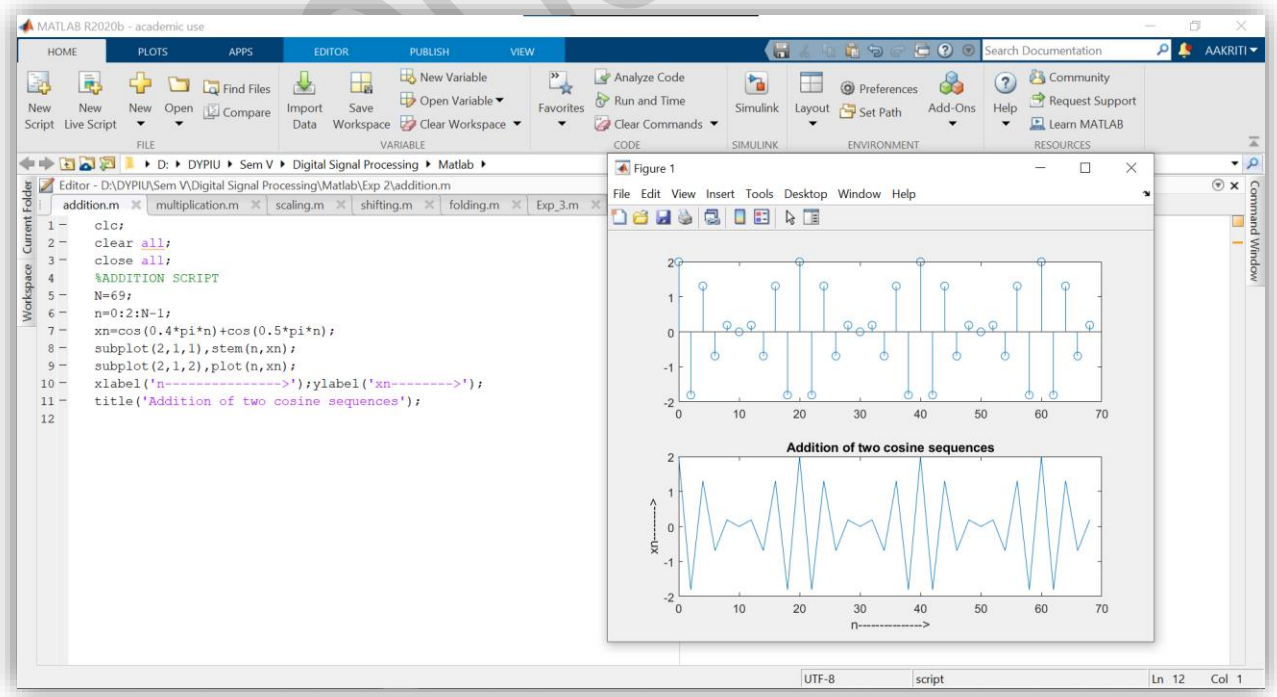
## AIM: IMPLEMENTATION OF OPERATION ON SEQUENCES (ADDITION, MULTIPLICATION, SCALING, SHIFTING, FOLDING ETC)

## TOOLS REQUIRED: PC, MATLAB

## SCRIPT

```
%ADDITION SCRIPT
N=69;
n=0:2:N-1;
xn=cos(0.4*pi*n)+cos(0.5*pi*n);
subplot(2,1,1),stem(n,xn);
subplot(2,1,2),plot(n,xn);
xlabel('n--------------->');ylabel('xn-------->');
title('Addition of two cosine sequences');
```

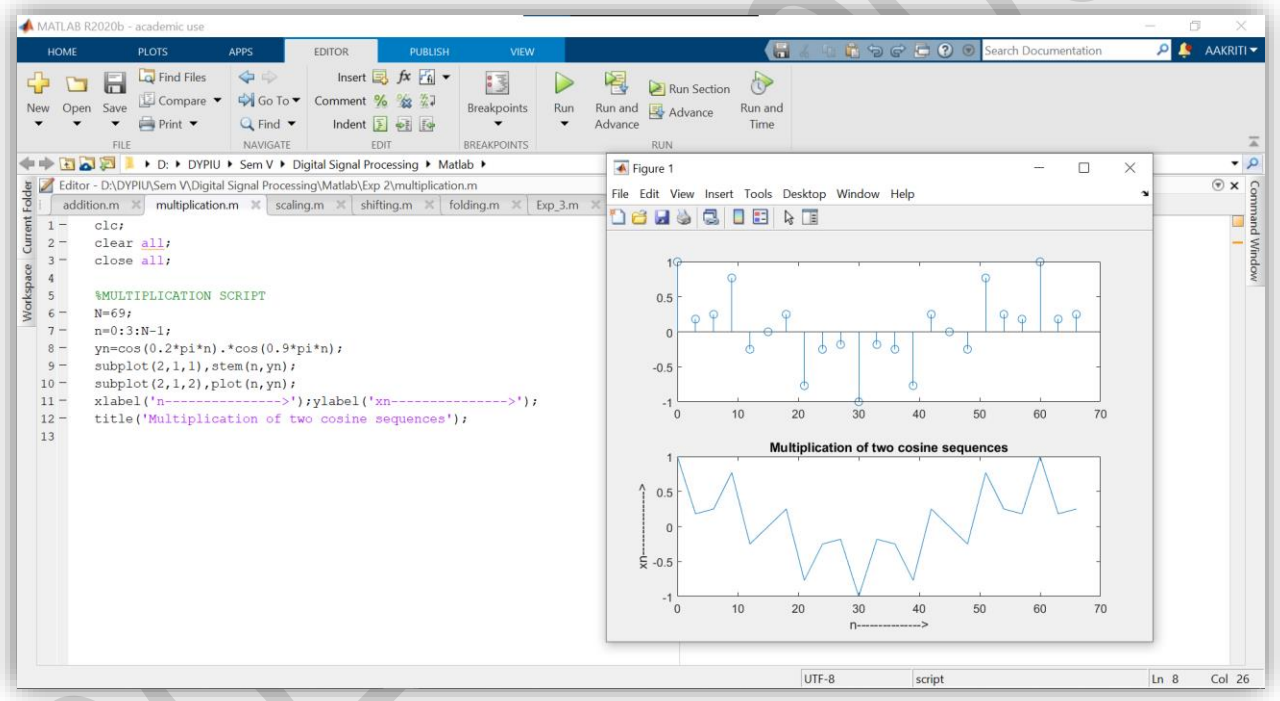
## OUTPUT



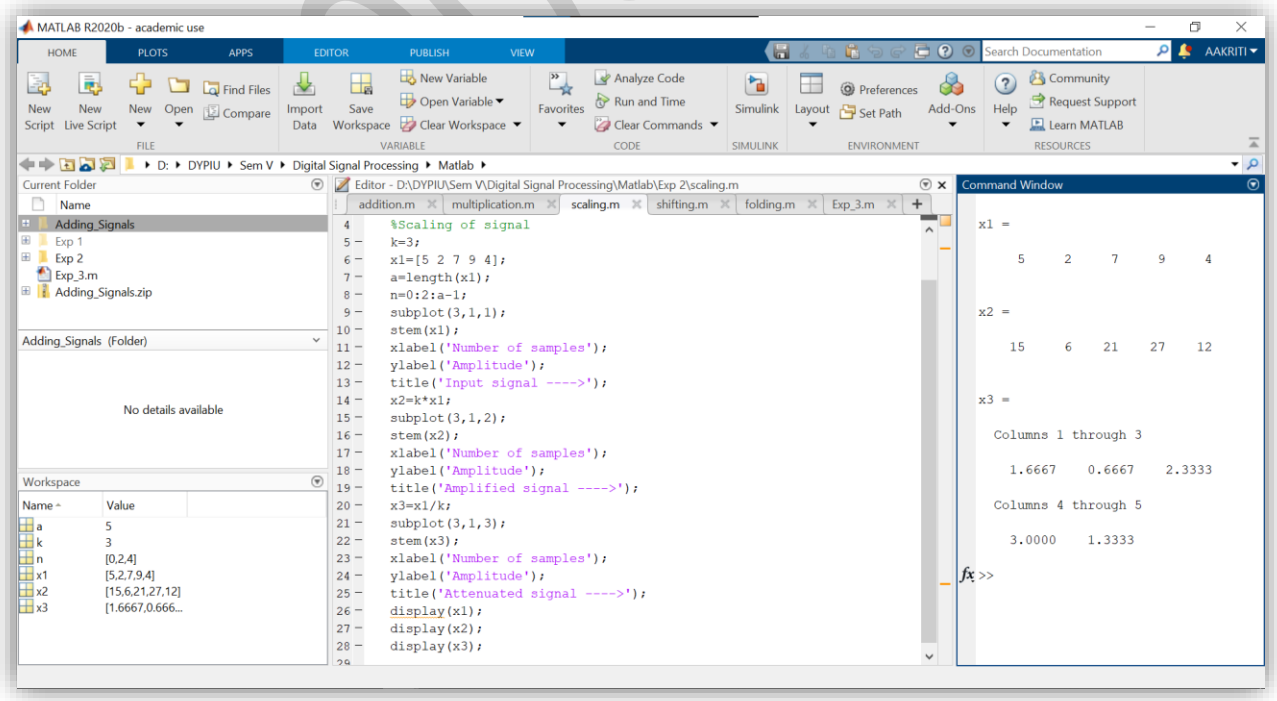
**%MULTIPLICATION SCRIPT**

```
N=69;
n=0:3:N-1;
yn=cos(0.2*pi*n).*cos(0.9*pi*n);
subplot(2,1,1),stem(n,yn);
subplot(2,1,2),plot(n,yn);
xlabel('n--------------->');ylabel('xn--------------->');
title('Multiplication of two cosine sequences');
```

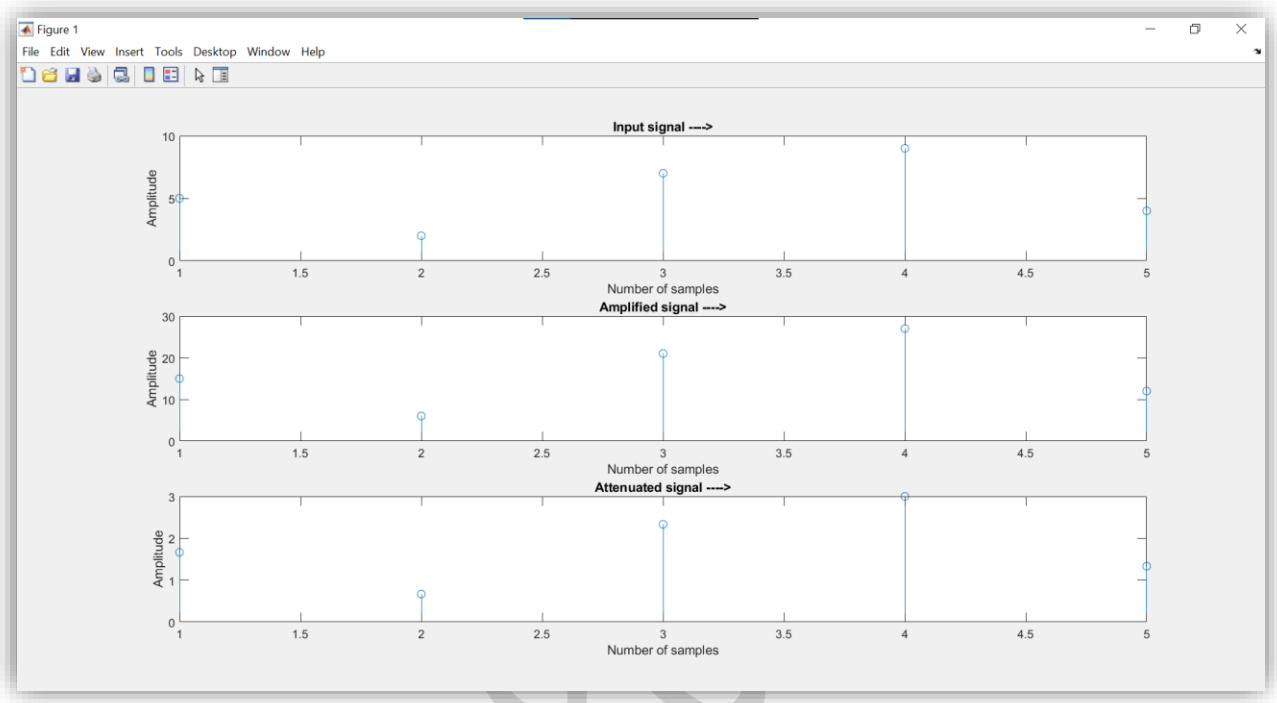
## OUTPUT



```
%Scaling of signal

clc;
k=3;
x1=[5 2 7 9 4];
a=length(x1);
n=0:2:a-1;
subplot(3,1,1);
stem(x1);
xlabel('Number of samples');
ylabel('Amplitude');
title('Input signal ---->');
x2=k*x1;
subplot(3,1,2);
stem(x2);
xlabel('Number of samples');
ylabel('Amplitude');
title('Amplified signal ---->');
x3=x1/k;
subplot(3,1,3);
stem(x3);
xlabel('Number of samples');
ylabel('Amplitude');
title('Attenuated signal ---->');
display(x1);
display(x2);
display(x3);
```

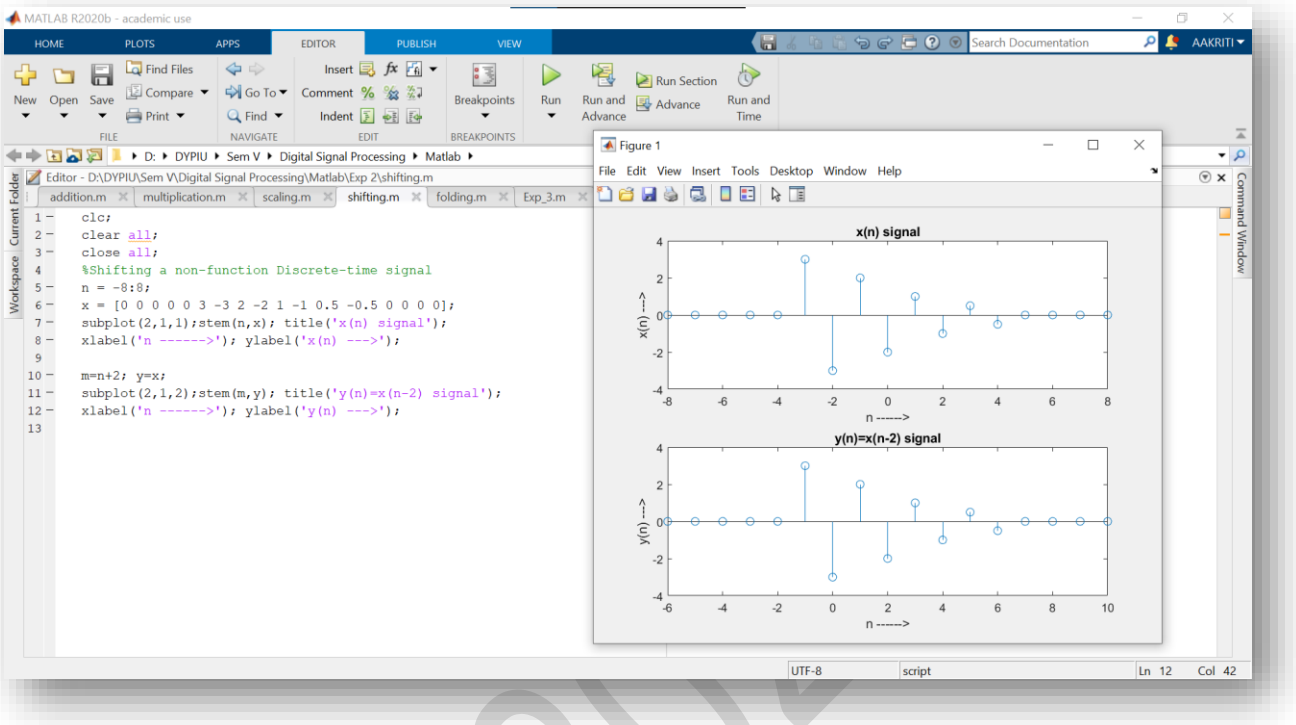


## OUTPUT



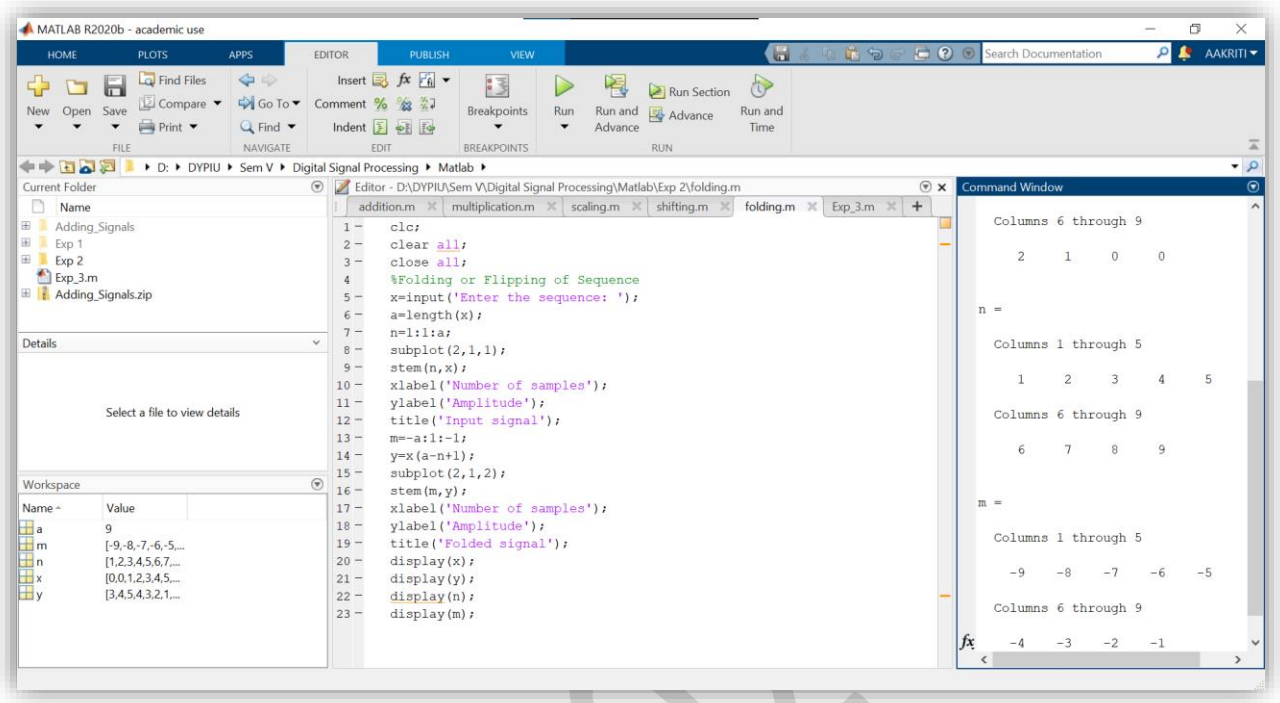

```
%Shifting a non-function Discrete-time signal

n = -8:8;
x = [0 0 0 0 0 3 -3 2 -2 1 -1 0.5 -0.5 0 0 0 0];
subplot(2,1,1);stem(n,x); title('x(n) signal');
xlabel('n ------>'); ylabel('x(n) --->');

m=n+2; y=x;
subplot(2,1,2);stem(m,y); title('y(n)=x(n-2) signal');
xlabel('n ------>'); ylabel('y(n) --->');
```

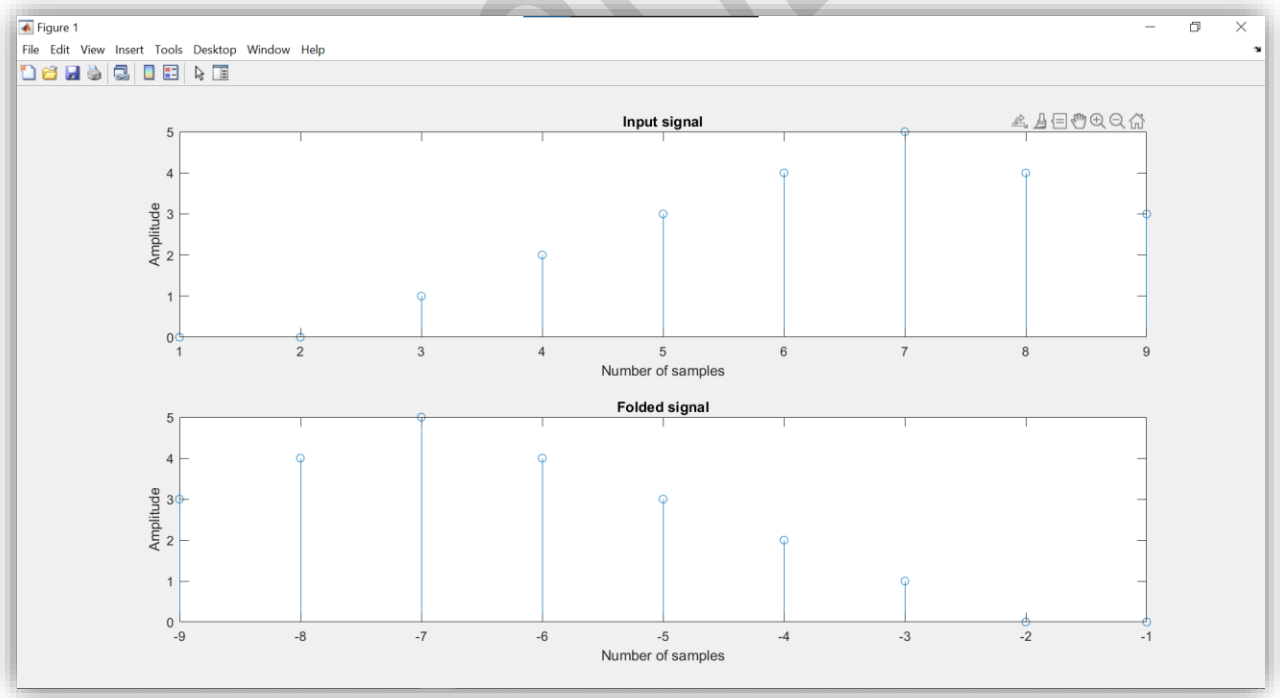
## OUTPUT




```
%Folding or Flipping of Sequence

x=input('enter the sequence');
a=length(x);
n=1:1:a;
subplot(2,1,1);
stem(n,x);
xlabel('number of samples');
ylabel('amplitude');
title('input signal');
m=-a:1:-1;
y=x(a-n+1);
subplot(2,1,2);
stem(m,y);
xlabel('number of samples');
ylabel('amplitude');
title('folded signal');
display(x);
display(y);
display(n);
display(m);
```

## OUTPUT




**RESULT**: Thus implementation of different operations on sequences is simulated using MATLAB

**VIVA**

1. Define Scaling of signal.
2. Define shifting.
3. Define Folding.

**ANSWERS:**

1. Scaling of a signal means, a constant is multiplied with the time or amplitude of the signal.

2. Shifting means movement of the signal, either in time domain aroundY−axis or in amplitude domain aroundX−axis.

3. Folding transforms an operation from a unit-time processing to *N* unit-times processing where *N* is called **folding factor**. Therefore, multiple same operations (less than *N*) used in original system could be replaced with a signal operation block in transformed system.

**Date of Performance:** 20th Sept' 21

# Experiment No: 03

## IMPLEMENTATION OF DISCRETE-TIME SYSTEMS

**AIM:** To find the impulse response of a system with the transfer function
y(n)+1/2y(n-1)+1/3y(n-2)=x(n)

**APPARATUS**: MATLAB Version 7.8 (R2009a)

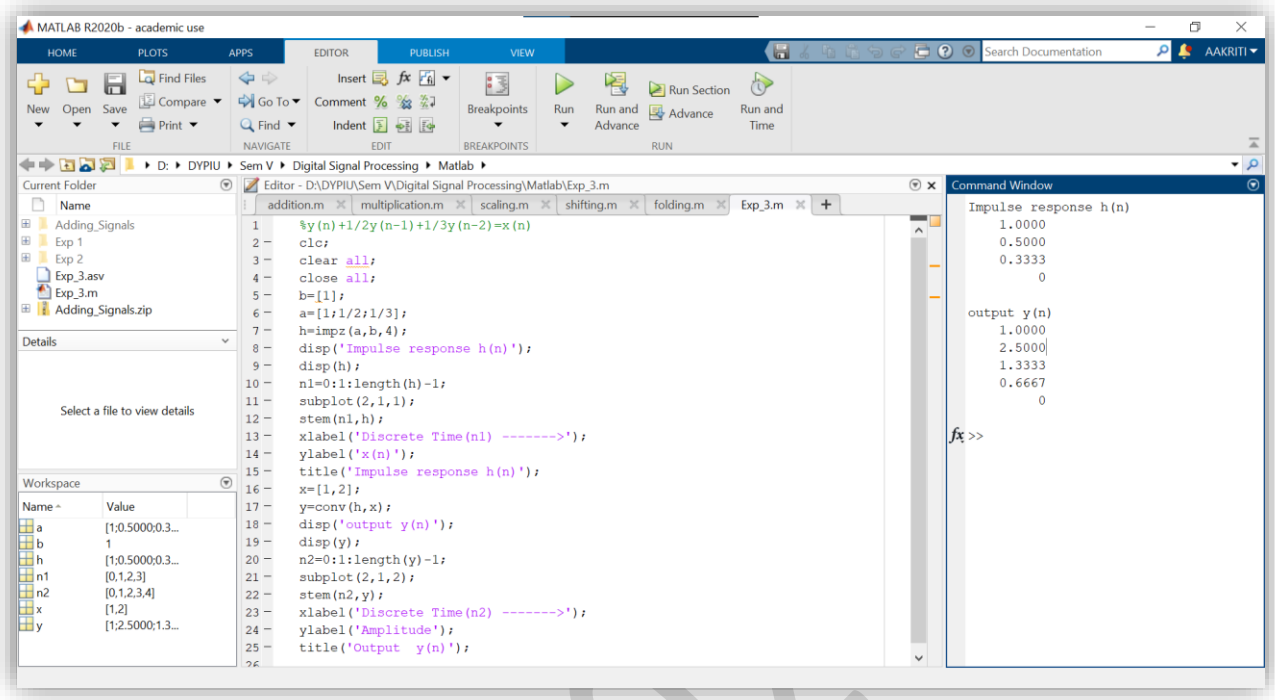
**PROGRAM:**

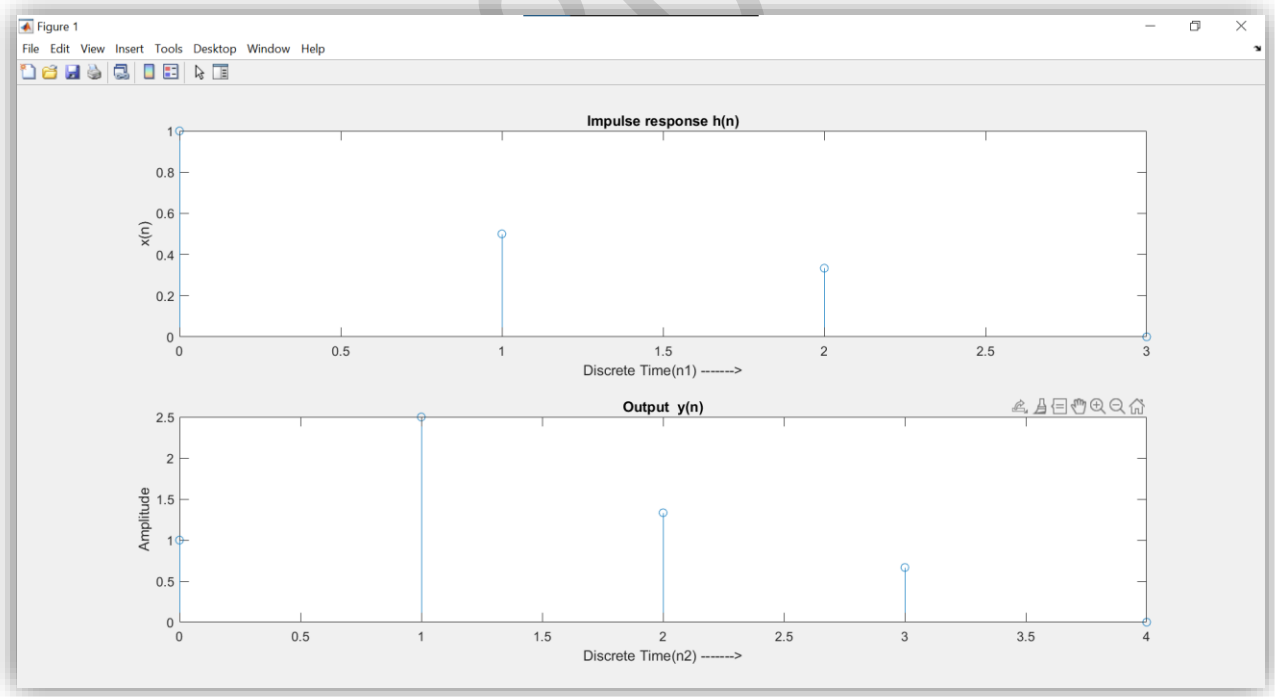
**IMPULSE RESPONSE OF SYSTEM FOR GIVEN TRANSFER FUNCTION**

%y(n)+1/2y(n-1)+1/3y(n-2)=x(n)

```
clc;
clear all;
close all;
b=[1];
a=[1, 1/2, 1/3];
h=impz(a,b,4);
disp('Impulse response h(n)');
disp(h);
n1=0:1:length(h)-1;
subplot(2,1,1);
stem(n1,h);
xlabel('Discrete Time(n1) ------->');
ylabel('x(n)');
title('Impulse response h(n)');
x=[1,2];
y=conv(h,x);
disp('output y(n)');
disp(y);
n2=0:1:length(y)-1;
subplot(2,1,2);
stem(n2,y);
xlabel('Discrete Time(n2) ------->');
ylabel('Amplitude');
title('Output  y(n)');
```

**OUTPUT**




**RESULT**: Thus the generation of impulse sequence for given difference equation is simulated using MATLAB.

**VIVA**

1. What is meant by impulse response of a system?
2. How to generate impulses using MATLAB?
3. What is the role of convolution in finding the impulse response of a system?
4. What is an LTI system?

**ANSWERS:**

1. In signal processing, the **impulse response**, or **impulse response function (IRF)**, of a dynamic system is its output when presented with a brief input signal, called an impulse.

2. >> freq = fft(impulse Train, Len);
>> stem([(0:Len-1)/(Len*Ts)], abs(freq)).

3. Convolution is a very powerful technique that can be used to calculate the zero state response (i.e., the response to an input when the system has zero initial conditions) of a system to an arbitrary input by using the impulse response of a system.

4. **Linear time-invariant system theory**, commonly known as **LTI system theory,** comes from applied mathematics and has direct applications in NMR spectroscopy, seismology, circuits, signal processing, control theory, and other technical areas. It investigates the response of a linear and time-invariant system to an arbitrary input signal. Trajectories of these systems are commonly measured and tracked as they move through time (e.g., an acoustic waveform)

# D Y Patil International University, Akurdi, Pune

**Date of Performance:** 26th Sept' 21

## Experiment No: 04

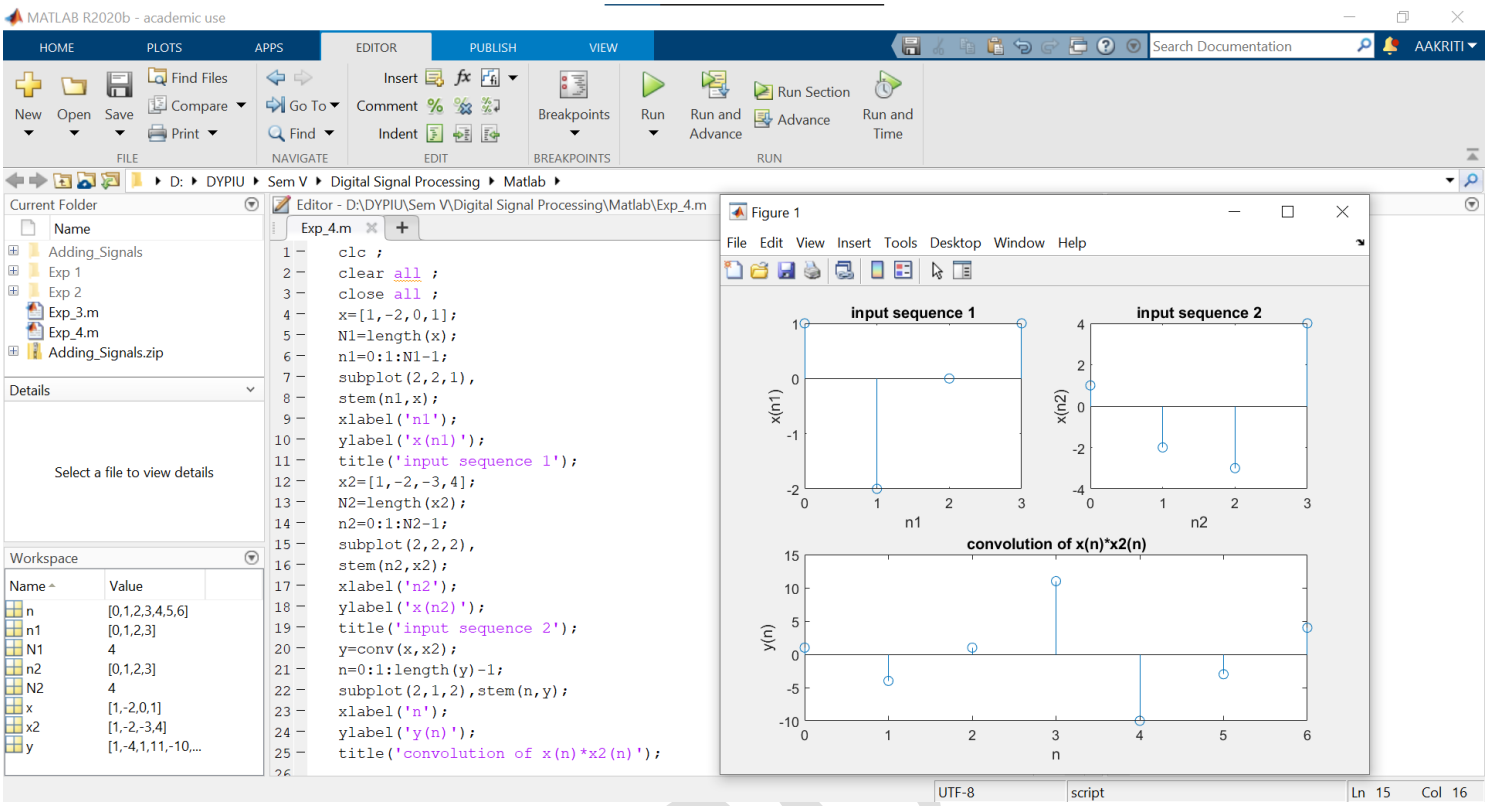
## AIM: To perform convolution of 2 signals

## APPARATUS: MATLAB Version 7.8 (R2009a)

## PROGRAM CODE

```
clc ;
clear all ;
close all ;
x=[1,-2,0,1];
N1=length(x);
n1=0:1:N1-1;
subplot(2,2,1),
stem(n1,x);
xlabel('n1');
ylabel('x(n1)');
title('input sequence 1');
x2=[1,-2,-3,4];
N2=length(x2);
n2=0:1:N2-1;
subplot(2,2,2),
stem(n2,x2);
xlabel('n2');
ylabel('x(n2)');
title('input sequence 2');
y=conv(x,x2);
n=0:1:length(y)-1;
subplot(2,1,2),stem(n,y);
xlabel('n');
ylabel('y(n)');
title('convolution of x(n)*x2(n)');
```

## OUTPUT




**RESULT**:- Thus the convolution of 2 signals was performed using MATLAB.

## VIVA

1. What is convolution signals and systems?
2. Define Linear Convolution.
3. Define Circular Convolution.

## ANSWERS

1. Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Using the strategy of impulse decomposition, systems are described by a signal called the impulse response.

2. Linear convolution is **the basic operation to calculate the output for any linear time invariant system given its input and its impulse response**.

3. In circular convolution, the signals are all periodic. Thus the shifting can be thought of as actually being a rotation. Since the values keep repeating because of the periodicity. Hence, it is known as circular convolution.

# D Y Patil International University, Akurdi, Pune

**Date of Performance:** 5th Oct' 21

## Experiment No: 05

**AIM:** To find Discrete Fourier Transform and Inverse Discrete Fourier Transform of given digital signal. (4 Point)

a. Using function
b. Without Function

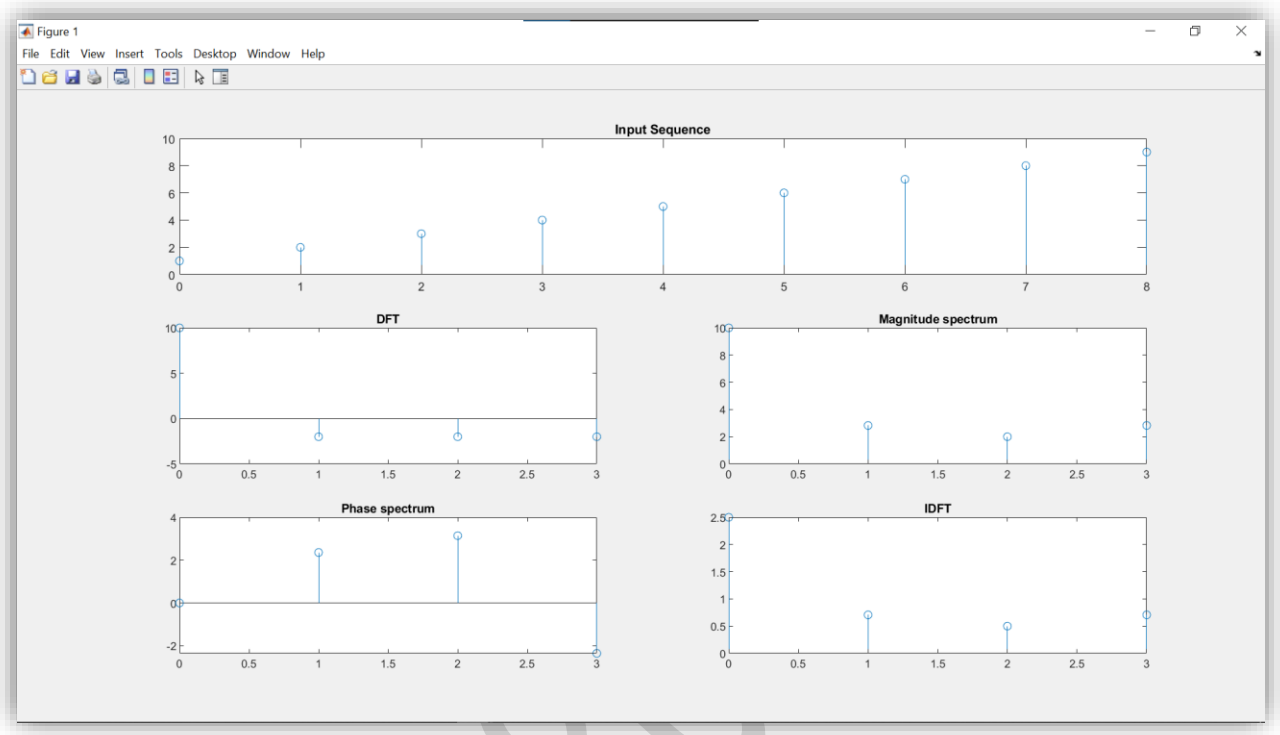
**TOOLS REQUIRED:** PC, MATLAB

**SCRIPT**

**%USING FUNCTION**

```
%DFT and IDFT using matlab functions
clc;
close all;
clear all;
x=input('Please enter the sequence x(n)=');
N=input('Please enter the length of the DFT N=');
X=fft(x,N);
n=0:length(x)-1;
subplot(311);
stem(n,x);
title('Input Sequence');
subplot(323);
n=0:length(X)-1;
stem(n,X);
disp('DFT of input sequence is ');
disp(X);
title('DFT');
subplot(324);
stem(n,abs(X));
title('Magnitude spectrum');
subplot(325);
stem(n,angle(X));
title('Phase spectrum');
xr=ifft(x,N);
subplot(326);
stem(n,abs(xr));
title('IDFT');
disp('IDFT of input sequence is ');
disp(xr);
```

## OUTPUT

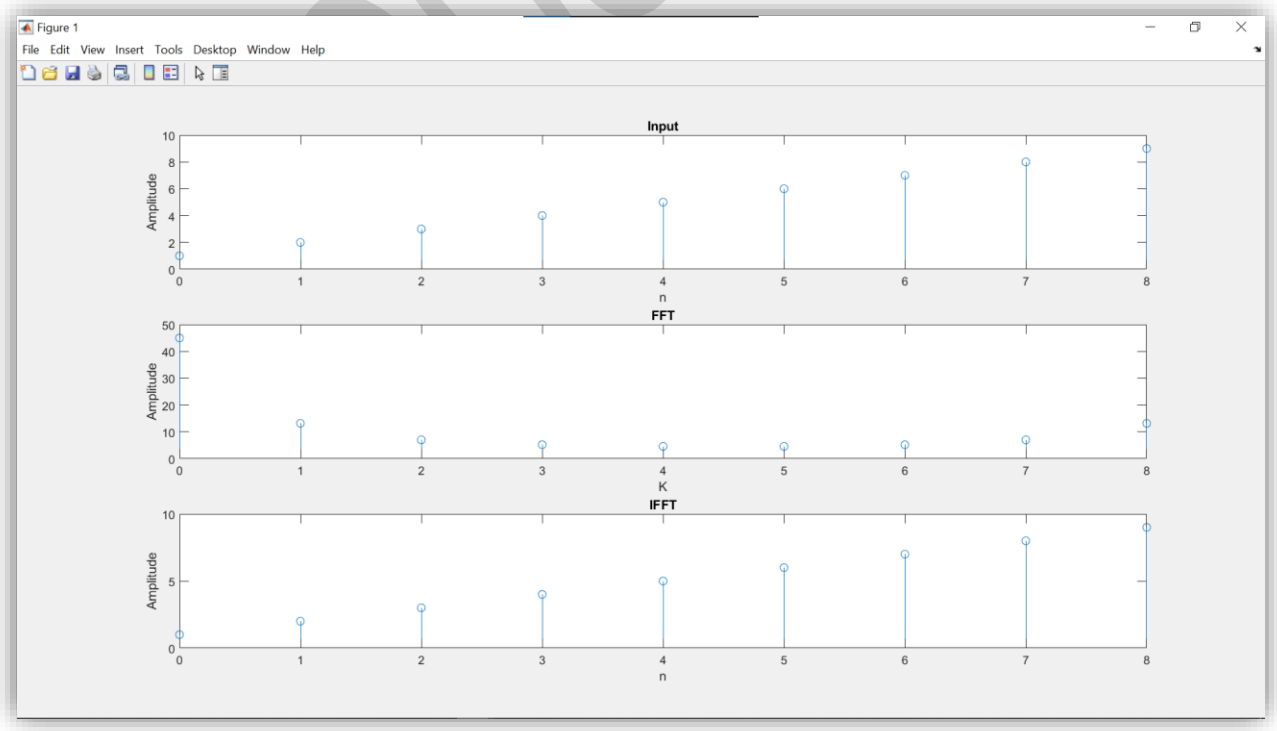



## %WITHOUT FUNCTION

```
clc;
close all;
clear all;
x = input('Enter the sequence for FFT = ');
N = length(x); %finding the length of entered sequence for N point FFT
for k = 1:N %for every value of k
y(k) = 0;
for n = 1:N %for every value of n
y(k) = y(k)+x(n)*exp(-1i*2*pi*(k-1)*(n-1)/N);
%as per FFT formula 0 to n-1
end
end
t = 0:N-1;
subplot(3,1,1);
stem(t,x); % for discrete plotting
%plotting input sequence
ylabel('Amplitude');
xlabel('n');
title('Input');
magnitude = abs(y); % absolute values of individual FFT complex
disp('FFT : ');
```

```
disp(magnitude);
t = 0:N-1;
subplot(3,1,2);
stem(t,magnitude);
%plotting FFT sequence
ylabel('Amplitude');
xlabel('K');
title('FFT');
R = length(y); %length of the fft array
for n = 1:R
x1(n) = 0;
for k = 1:R %loop as per the IFFT formula
    x1(n) = x1(n)+(1/R)*y(k)*exp(1i*2*pi*(k-1)*(n-1)/R);
end
end
t = 0:R-1;
subplot(3,1,3);
stem(t,x1);
%ploting IFFT sequence
disp('IFFT :');
disp(x1);
ylabel('Amplitude');
xlabel('n');
title('IFFT');
```

## OUTPUT



**RESULT**: Thus, the Discrete Fourier Transform and Inverse Discrete Fourier Transform is simulated using MATLAB using function and without function.

**VIVA**

1. Why discrete Fourier transform is used?
2. Define inverse discrete Fourier transformation
3. State the properties of DFT

**ANSWERS:**

1. The DFT is one of the most powerful tools in digital signal processing which enables us to find the spectrum of a finite-duration signal.

There are many circumstances in which we need to determine the frequency content of a time-domain signal. For example, we may have to analyze the spectrum of the output of an LC oscillator to see how much noise is present in the produced sine wave. This can be achieved by the discrete Fourier transform (DFT).

2. The inverse Fourier transform maps the signal back from the frequency domain into the time domain. A time domain signal will usually consist of a set of real values, where each value has an associated time. The inverse Fourier transform takes the frequency series of complex values and maps them back into the original time series. Assuming that the original time series consisted of real values, the result of the IDFT will be complex numbers where the imaginary part is zero.

3
i. Periodicity
ii. Linearity and symmetry
iii. Multiplication of two DFTs
iv. Circular convolution
v. Time reversal
vi. Circular time shift and frequency shift
vii. Complex conjugate
viii. Circular correlation

# D Y Patil International University, Akurdi, Pune

**Date of Performance:** 15th Nov' 21

## Experiment No: 06

**AIM: WRITE A PROGRAM FOR FILTER DESIGN USING WINDOW FUNCTION**

1) **RECTANGULAR WINDOW**
2) **TRIANGULAR WINDOW**
3) **KAISER WINDOW**

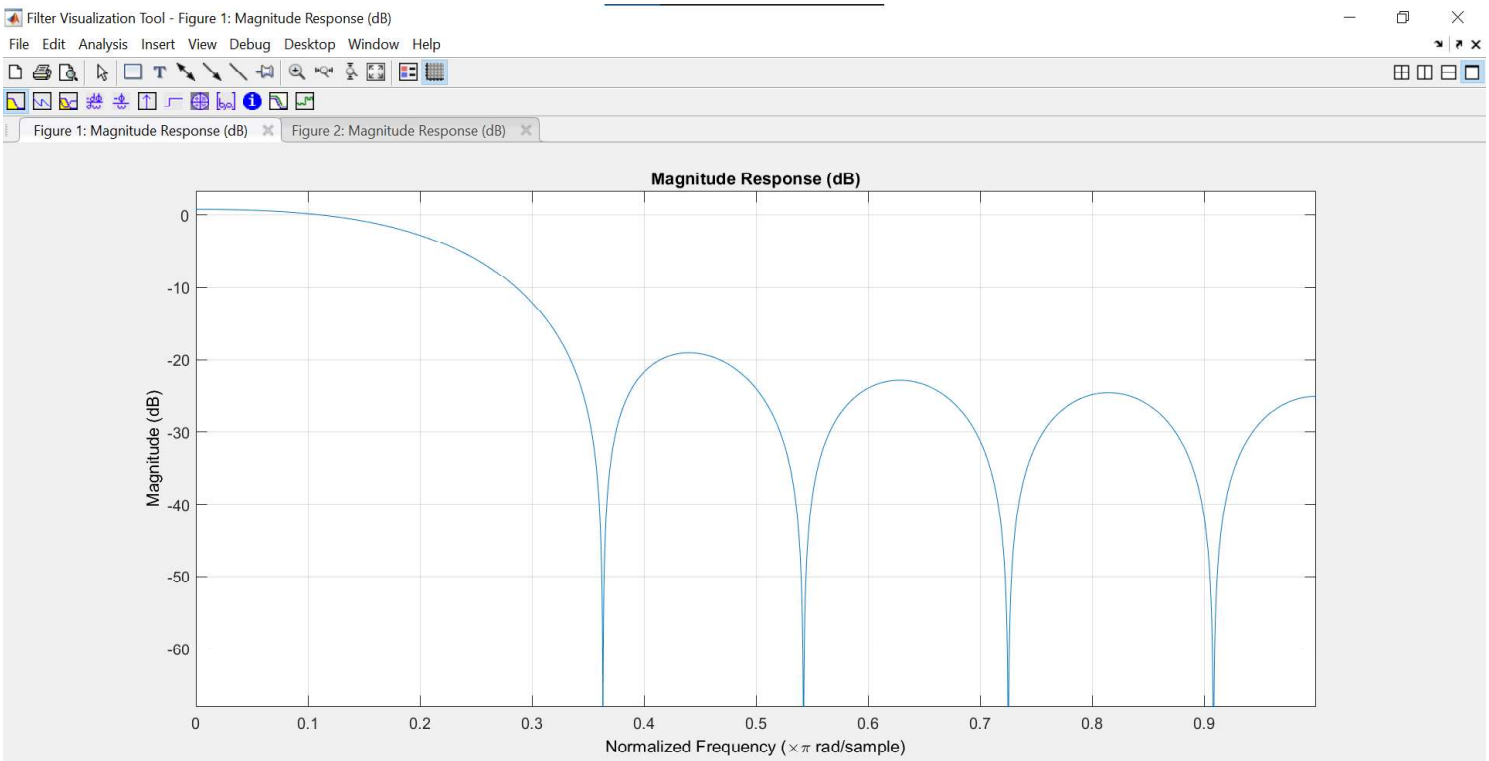
**TOOLS REQUIRED: PC, MATLAB;**

**SCRIPT**

```
%RECTANGULAR WINDOW
m=input('Enter the value of M=');
w=input('Enter the normalised cutoff frequency=')'
a=(m-1)/2;
n=[0:(a-1)*(a+1):(m-1)];
f=w*sinc(w*(-a:a));
lp=f.*rectwin(m)'
fvtool(lp);
```

**OUTPUT**

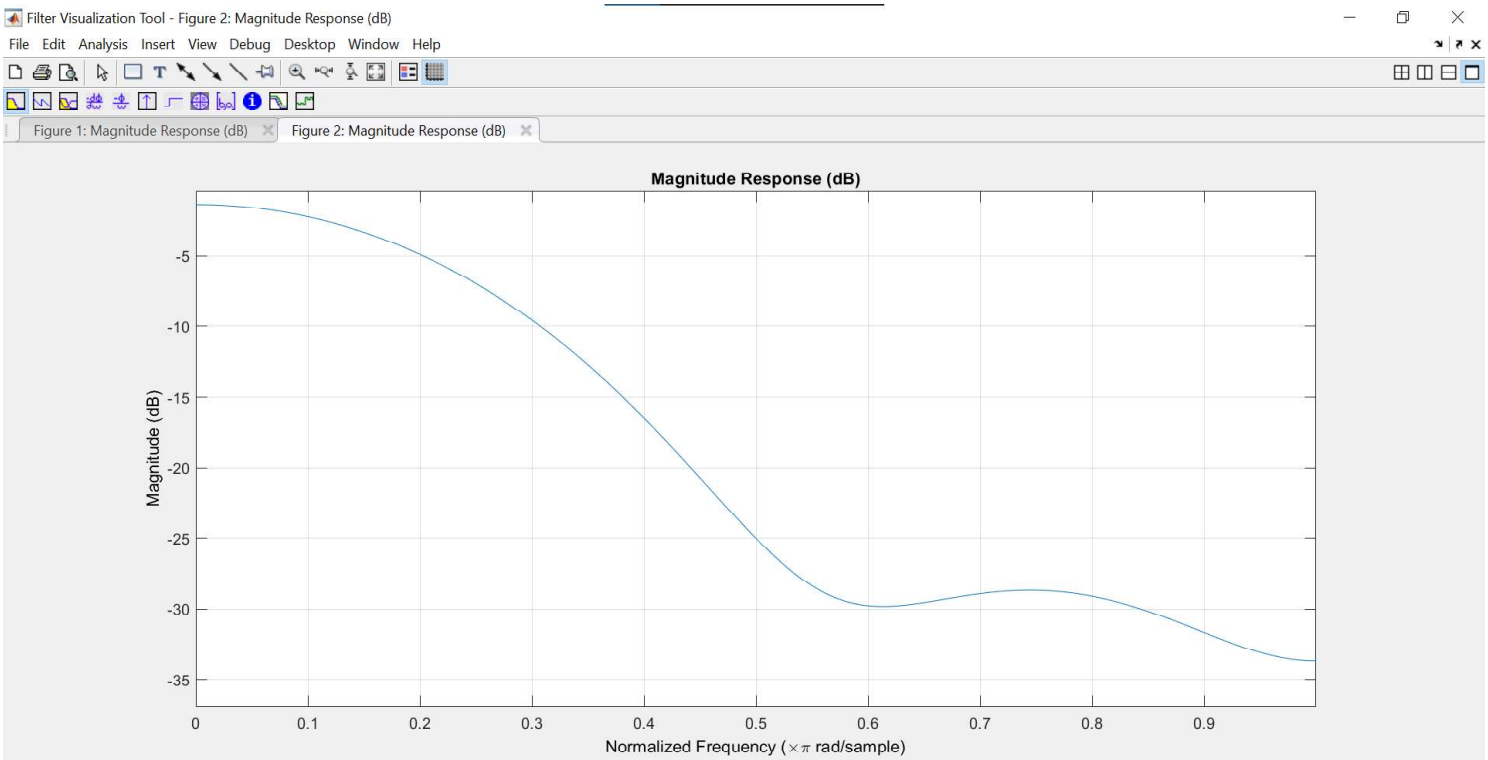
```
Enter the value of M=11
Enter the normalised cutoff frequency=0.245

w =

    0.2450


lp =

   -0.0413    0.0050    0.0785    0.1591    0.2215    0.2450    0.2215    0.1591    0.0785    0.0050   -0.0413
```




```
%TRAINGULAR WINDOW
m=input('Enter the value of M=');
w=input('Enter the normalised cutoff frequency=')'
a=(m-1)/2;
n=[0:(a-1)*(a+1):(m-1)];
f=w*sinc(w*(-a:a));
lp=f.*bartlett(m)'
fvtool(lp);
```

## OUTPUT

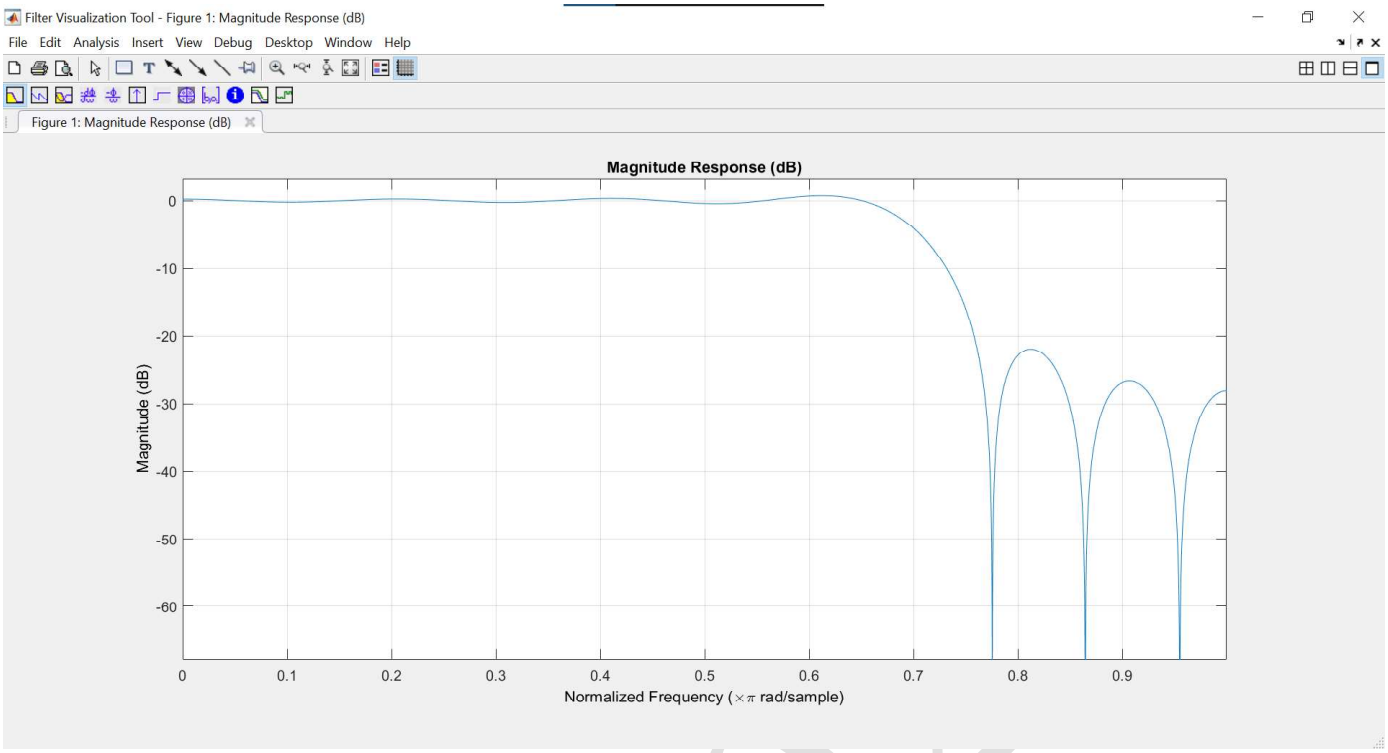
```
>> triangular
Enter the value of M=11
Enter the normalised cutoff frequency=0.245

w =

    0.2450


lp =

         0    0.0010    0.0314    0.0954    0.1772    0.2450    0.1772    0.0954    0.0314    0.0010
```




```
%KAISER WINDOW
m=input('Enter the value of M=');
w=input('Enter the normalised cutoff frequency=')'
a=(m-1)/2;
n=[0:(a-1)*(a+1):(m-1)];
f=w*sinc(w*(-a:a));
lp=f.*kaiser(m)'
fvtool(lp);
```

## OUTPUT

```
Enter the value of M=19
Enter the normalised cutoff frequency=0.713

w =

    0.7130


lp =

  Columns 1 through 11

    0.0321   -0.0304    0.0012    0.0396   -0.0612    0.0353    0.0446   -0.1544    0.2495    0.7130    0.2495

  Columns 12 through 19

   -0.1544    0.0446    0.0353   -0.0612    0.0396    0.0012   -0.0304    0.0321
```




## RESULT

The filter design was implemented using different window functions.

## VIVA QUESTIONS

1. Define FIR filter.
   **Ans.** An FIR filter is designed by finding the coefficients and filter order that meet certain specifications, which can be in the time domain (e.g. a matched filter) and/or the frequency domain (most common).

2. Expand FIR.
   **Ans.** Finite Impulse Response

3. Why do we need filter?
   **Ans.** Filters are used to achieve a kind of frequency selectivity on the spectrum of the input signal.

4. Explain rectangular window.
   **Ans.** Rectangular window is the simplest window, equivalent to replacing all but N values of a data sequence by zeroes, making it appear as though the waveform suddenly turns on and off.
   1. W(n) = 1
   2. W(n) = 1 for |n| <= M-1/2
      0, otherwise

**Date of Performance:** 8th Nov' 21

# Experiment No: 07

**PROGRAM-7 ECG SIGNAL GENERATION**

**AIM**: To design ECG Signal using MATLAB

**APPARATUS**: MATLAB Version 7.8 (R2009a)

## Introduction:

The aim of the ECG simulator is to produce the typical ECG waveforms of different leads and as many arrhythmias as possible. My ECG simulator is a matlab based simulator and is able to produce normal lead II ECG waveform.

The use of a simulator has many advantages in the simulation of ECG waveforms. First one is saving of time and another one is removing the difficulties of taking real ECG signals with invasive and non-invasive methods. The ECG simulator enables us to analyze and study normal and abnormal ECG waveforms without actually using the ECG machine. One can simulate any given ECG waveform using the ECG simulator.

**Main features of this simulator**:

- Any value of heart beat can be set
- Any value of intervals between the peaks (ex-PR interval) can be set
- Any value of amplitude can be set for each of the peaks
- Fibrillation can be simulated
- Noise due to the electrodes can be simulated
- Heart pulse of the particular ECG wave form can be represented in a separate graph

**PROGRAM** :

**%PROGRAM for the generation of ECG signal**

```
% complete.m

x=0.01:0.01:2;
default=input('Press 1 if u want default ecg signal else press 2:\n');
if(default==1)
      li=30/72;
      a_pwav=0.25;
```

```
      d_pwav=0.09;
      t_pwav=0.16;

      a_qwav=0.025;
      d_qwav=0.066;
      t_qwav=0.166;

      a_qrswav=1.6;
      d_qrswav=0.11;

      a_swav=0.25;
      d_swav=0.066;
      t_swav=0.09;

      a_twav=0.35;
      d_twav=0.142;
      t_twav=0.2;

      a_uwav=0.035;
      d_uwav=0.0476;
      t_uwav=0.433;
else
    rate=input('\n\nenter the heart beat rate :');
    li=30/rate;

    %p wave specifications
    fprintf('\n\np wave specifications\n');
    d=input('Enter 1 for default specification else press 2: \n');
    if(d==1)
        a_pwav=0.25;
        d_pwav=0.09;
        t_pwav=0.16;
    else
       a_pwav=input('amplitude = ');
       d_pwav=input('duration = ');
       t_pwav=input('p-r interval = ');
       d=0;
    end


    %q wave specifications
    fprintf('\n\nq wave specifications\n');
    d=input('Enter 1 for default specification else press 2: \n');
    if(d==1)
        a_qwav=0.025;
        d_qwav=0.066;
        t_qwav=0.166;
    else
       a_qwav=input('amplitude = ');
       d_qwav=input('duration = ');
       t_qwav=0.166;
```

```
   d=0;
end



%qrs wave specifications
fprintf('\n\nqrs wave specifications\n');
d=input('Enter 1 for default specification else press 2: \n');
if(d==1)
    a_qrswav=1.6;
    d_qrswav=0.11;
else
   a_qrswav=input('amplitude = ');
   d_qrswav=input('duration = ');
   d=0;
end



%s wave specifications
fprintf('\n\ns wave specifications\n');
d=input('Enter 1 for default specification else press 2: \n');
if(d==1)
    a_swav=0.25;
    d_swav=0.066;
    t_swav=0.09;
else
   a_swav=input('amplitude = ');
   d_swav=input('duration = ');
   t_swav=0.09;
   d=0;
end


%t wave specifications
fprintf('\n\nt wave specifications\n');
d=input('Enter 1 for default specification else press 2: \n');
if(d==1)
    a_twav=0.35;
    d_twav=0.142;
    t_twav=0.2;
else
   a_twav=input('amplitude = ');
   d_twav=input('duration = ');
   t_twav=input('s-t interval = ');
   d=0;
end


%u wave specifications
fprintf('\n\nu wave specifications\n');
```

```
    d=input('Enter 1 for default specification else press 2: \n');
    if(d==1)
        a_uwav=0.035;
        d_uwav=0.0476;
        t_uwav=0.433;
    else
       a_uwav=input('amplitude = ');
       d_uwav=input('duration = ');
       t_uwav=0.433;
       d=0;
    end



end

 pwav=p_wav(x,a_pwav,d_pwav,t_pwav,li);


 %qwav output
 qwav=q_wav(x,a_qwav,d_qwav,t_qwav,li);


 %qrswav output
 qrswav=qrs_wav(x,a_qrswav,d_qrswav,li);

 %swav output
 swav=s_wav(x,a_swav,d_swav,t_swav,li);


 %twav output
 twav=t_wav(x,a_twav,d_twav,t_twav,li);


 %uwav output
 uwav=u_wav(x,a_uwav,d_uwav,t_uwav,li);

 %ecg output
 ecg=pwav+qrswav+twav+swav+qwav+uwav;
 figure(1)
 plot(x,ecg);


% p wav.m

function [pwav]=p_wav(x,a_pwav,d_pwav,t_pwav,li)
l=li;
a=a_pwav;
x=x+t_pwav;
b=(2*l)/d_pwav;
```

```
n=100;
p1=1/l;
p2=0;
for i = 1:n
    harm1=(((sin((pi/(2*b))*(b-(2*i))))/(b-
(2*i))+(sin((pi/(2*b))*(b+(2*i))))/(b+(2*i)))*(2/pi))*cos((i*pi*x)/l);
    p2=p2+harm1;
end
pwav1=p1+p2;
pwav=a*pwav1;
```

**% q_wav.m**

```
function [qwav]=q_wav(x,a_qwav,d_qwav,t_qwav,li)
l=li;
x=x+t_qwav;
a=a_qwav;
b=(2*l)/d_qwav;
n=100;
q1=(a/(2*b))*(2-b);
q2=0;
for i = 1:n
    harm5=(((2*b*a)/(i*i*pi*pi))*(1-cos((i*pi)/b)))*cos((i*pi*x)/l);
    q2=q2+harm5;
end
qwav=-1*(q1+q2);
```

**% qrs_wav.m**

```
function [qrswav]=qrs_wav(x,a_qrswav,d_qrswav,li)
l=li;
a=a_qrswav;
b=(2*l)/d_qrswav;
n=100;
qrs1=(a/(2*b))*(2-b);
qrs2=0;
for i = 1:n
    harm=(((2*b*a)/(i*i*pi*pi))*(1-cos((i*pi)/b)))*cos((i*pi*x)/l);
    qrs2=qrs2+harm;
end
qrswav=qrs1+qrs2;
```

**% s_wav**

```
function [swav]=s_wav(x,a_swav,d_swav,t_swav,li)
l=li;
x=x-t_swav;
```

```
a=a_swav;
b=(2*l)/d_swav;
n=100;
s1=(a/(2*b))*(2-b);
s2=0;
for i = 1:n
    harm3=(((2*b*a)/(i*i*pi*pi))*(1-cos((i*pi)/b)))*cos((i*pi*x)/l);
    s2=s2+harm3;
end
swav=-1*(s1+s2);
```

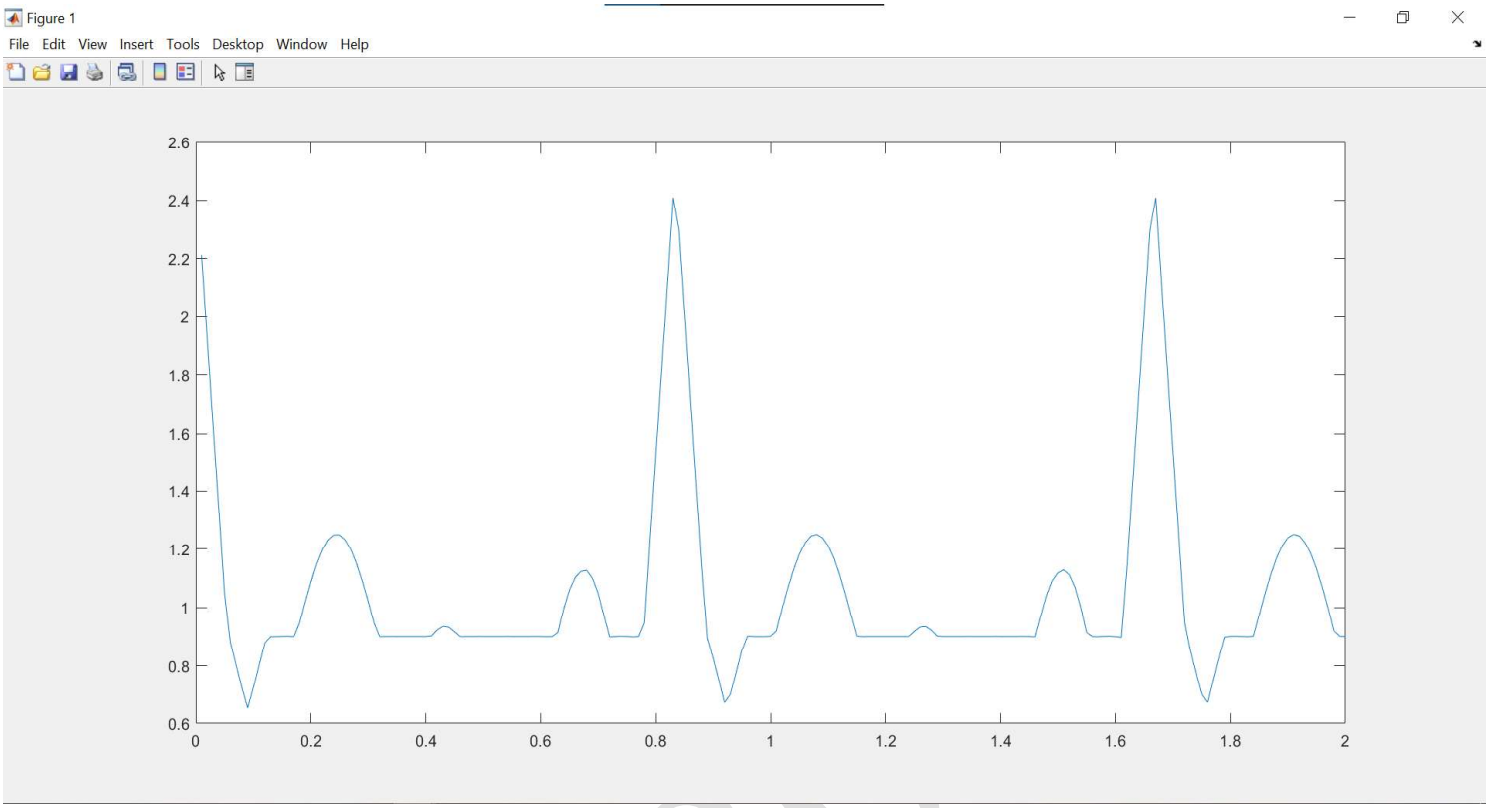
**% t wav**

```
function [twav]=t_wav(x,a_twav,d_twav,t_twav,li)
l=li;
a=a_twav;
x=x-t_twav-0.045;
b=(2*l)/d_twav;
n=100;
t1=1/l;
t2=0;
for i = 1:n
    harm2=(((sin((pi/(2*b))*(b-(2*i))))/(b-
(2*i))+(sin((pi/(2*b))*(b+(2*i))))/(b+(2*i)))*(2/pi))*cos((i*pi*x)/l);
    t2=t2+harm2;
end
twav1=t1+t2;
twav=a*twav1;
```

**% u wav.m**

```
function [uwav]=u_wav(x,a_uwav,d_uwav,t_uwav,li)
l=li;
a=a_uwav
x=x-t_uwav;
b=(2*l)/d_uwav;
n=100;
u1=1/l
u2=0
for i = 1:n
    harm4=(((sin((pi/(2*b))*(b-(2*i))))/(b-
(2*i))+(sin((pi/(2*b))*(b+(2*i))))/(b+(2*i)))*(2/pi))*cos((i*pi*x)/l);
    u2=u2+harm4;
end
uwav1=u1+u2;
uwav=a*uwav1;
```

## OUTPUT




**RESULT:** ECG signal was generated using MATLAB.

## VIVA QUESTIONS

1. What is ECG?
   **Ans.** Electrical activity of a heart is called as electrocardiogram (ECG).

2. State the advantages of ECG simulation.
   Ans.
   - saving of time
   - removing the difficulties of taking real ECG signals with invasive and non-invasive methods.
   - enables us to analyze and study normal and abnormal ECG waveforms without actually using the ECG machine.

# D Y Patil International University, Akurdi, Pune

**Date of Performance:** 1st Nov' 21

## Experiment No: 08

## AIM: DETERMINATION OF POWER SPECTRUM OF A GIVEN SIGNAL

## APPARATUS: MATLAB Version 7.8 (R2009a)

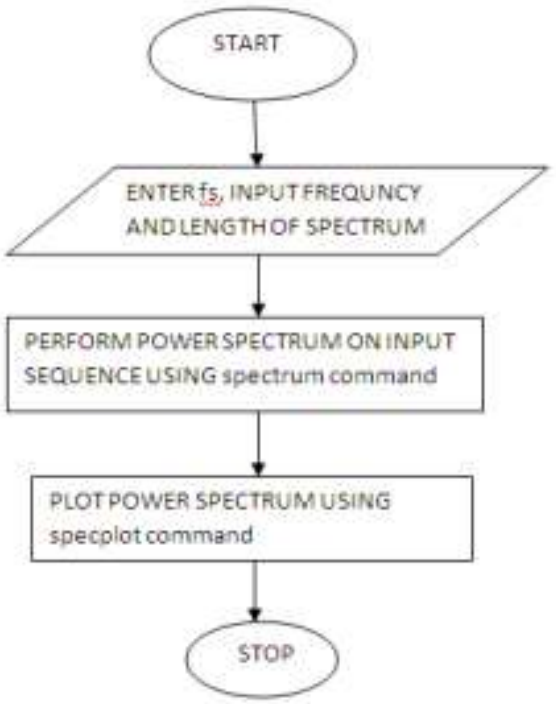
## THEORY:

The power spectrum describes the distribution of signal power over a frequency spectrum. The most common way of generating a power spectrum is by using a discrete Fourier transform, but other techniques such as the maximum entropy method can also be used. The power spectrum can also be defined as the Fourier transform of auto correlation function.

## Algorithm:

Step I : Give input sequence x
Step II : Give sampling frequency, input frequency and length of the spectrum.
Step III : Find power spectrum of input sequence using matlab command spectrum.
Step IV : Plot power spectrum using specplot.

## Flow Chart:



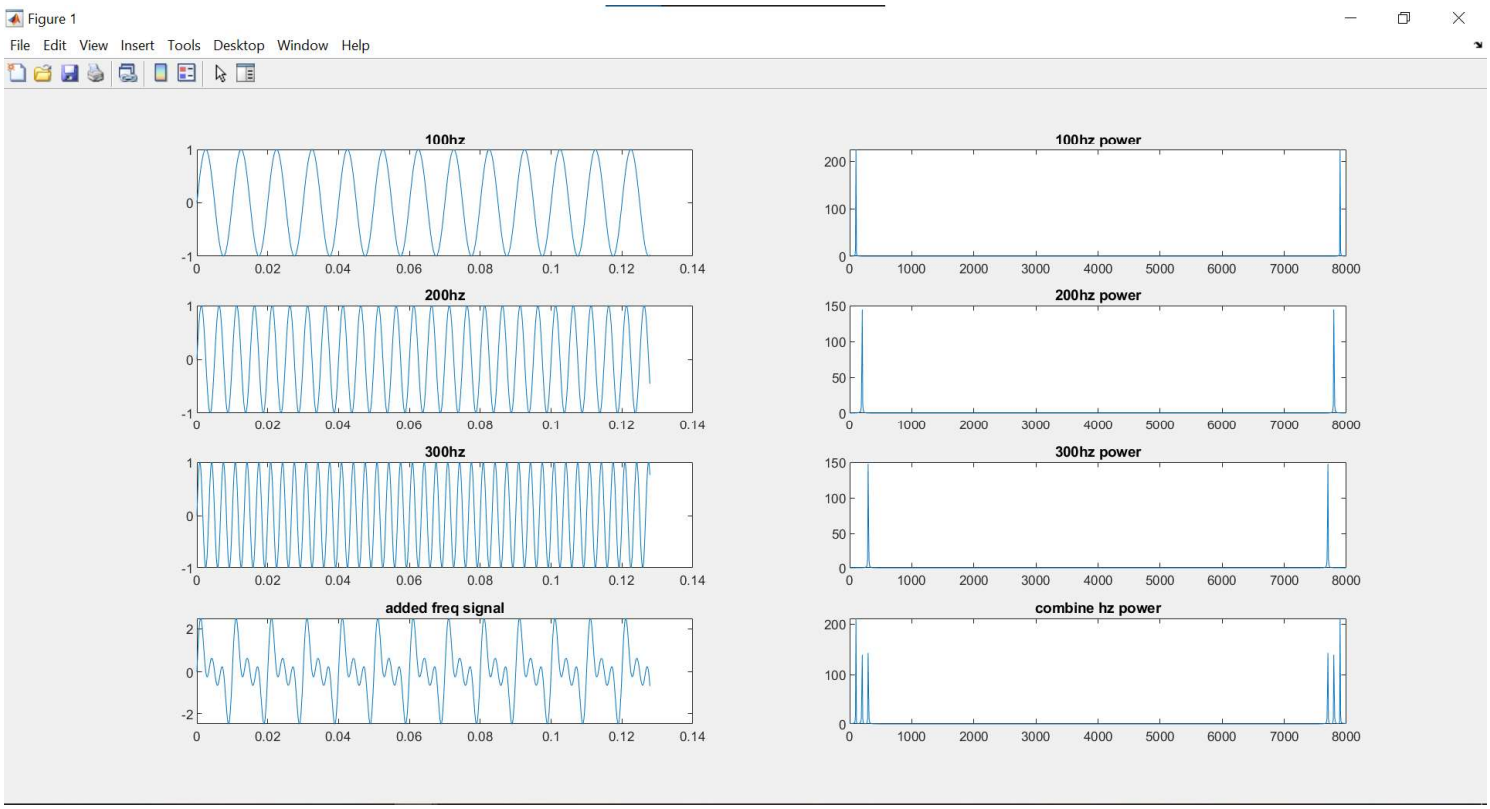
**PROGRAM:**

```
N=1024;
fs=8000;

% signal 1
f=input('enter the frequency[1 to 5000]:');;
ts=1/fs;
t = ts*(0:N-1);
x=sin(2*pi*f*t);
subplot(4,2,1),plot(t,x),title('100hz');
% power spectrum of 1
y = fft(x);
N = length(x);           % number of samples
f = (0:N-1)*(fs/N);      % frequency range
pow = abs(y).^2/N;     % power of the DFT
subplot(4,2,2),plot(f,pow), title('100hz power');

% signal 2
f1=input('enter the frequency[1 to 5000]:');;
x1=sin(2*pi*f1*t);
subplot(4,2,3),plot(t,x1),title('200hz');
% power spectrum of 2
y = fft(x1);
N = length(x);           % number of samples
f = (0:N-1)*(fs/N);      % frequency range
pow = abs(y).^2/N;     % power of the DFT
subplot(4,2,4),plot(f,pow), title('200hz power');

% signal 3
f2=input('enter the frequency[1 to 5000]:');;
x2=sin(2*pi*f2*t);
subplot(4,2,5),plot(t,x2),title('300hz');
% power spectrum of 3
y = fft(x2);
N = length(x);           % number of samples
f = (0:N-1)*(fs/N);      % frequency range
pow = abs(y).^2/N;     % power of the DFT
subplot(4,2,6),plot(f,pow), title('300hz power');

% signal combined Hz
x3=x+x1+x2;
subplot(4,2,7),plot(t,x3),title('added freq signal');
% power spectrum of combined Hz
y = fft(x3);
N = length(x);           % number of samples
f = (0:N-1)*(fs/N);      % frequency range
pow = abs(y).^2/N;     % power of the DFT
subplot(4,2,8),plot(f,pow), title('combine hz power');
```

**INPUT:**

Enter the frequency [1 to 5000]: 1000
Enter the frequency [1 to 5000]: 2000
Enter the frequency [1 to 5000]: 3000

**Output Waveform:**




**RESULT:**

**VIVA QUESTIONS:**

1. Define power signal.

**Ans.**

A signal is said to be power type signal, if and only if, normalized average power is finite and non-zero i.e. 0<p<∞0<p<∞. For power type signal, normalized average power is finite and non-zero. Almost all the periodic signals are power signals and their average power is finite and non-zero.

2. Define energy signal.

**Ans.**

A signal is said to be an Energy signal, if and only if, the total energy contained is finite and nonzero 0<E<∞0<E<∞. Therefore, for any energy type signal, the total normalized signal is finite and nonzero.

3. Define power spectral density of a signal.

**Ans.**

The *power spectral density* (PSD) of the signal describes the power present in the signal as a function of frequency, per unit frequency.

4. How the energy of a signal can be calculated?

5. Explain difference between energy spectral density and power spectral density.

**Ans.**

The power spectral density of a signal describes its frequency distribution of power. The energy spectral density is the same thing, but integrated over a finite time interval.

6. Explain the PSD plot.

**Ans.**

The unit of PSD is energy (variance) per frequency(width) and you can obtain energy within a specific frequency range by integrating PSD within that frequency range. Looking at PSD is like looking at simple time series plot except that we look at time series as a function of frequency instead of a function of time.

7. What is the importance of PSD?

**Ans.**

The PSD of a material can be important in **understanding its physical and chemical properties**. It affects the strength and load-bearing properties of rocks and soils.

8. What are the applications of PSD?

**Ans.**

- In electronic communication systems, including radio communications, radars, and related systems, plus passive remote sensing technology.
- Cosmology

9. Explain MATLAB function randn(size(n)).

**Ans.**

The randn function generates arrays of random numbers whose elements are normally distributed with mean 0, variance , and standard deviation . **Y = randn(n) returns** an n -by- n matrix of random entries. An error message appears if n is not a scalar.

10. What is the need to represent the signal in frequency domain?

**Ans.**

Frequency domain representations are particularly useful when analyzing linear systems. **EMC and signal integrity engineers** must be able to work with signals represented in both the time and frequency domains. Signal sources and interference are often defined in the time domain.

# D Y Patil International University, Akurdi, Pune

**Date of Performance:** 1st Nov' 21

## Experiment No: 09

**AIM:** To generate DTMF Signals using MATLAB Software.

**Software: MATLAB**

**THEORY:**

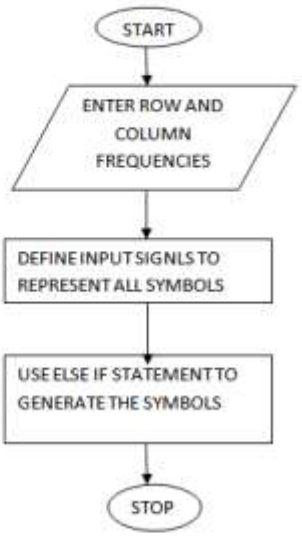
The DTMF stands for "*D*ual *T*one *M*ulti *F*requency", and is a method of representing digits with tone frequencies, in order to transmit them over an analog communications network, for example a telephone line. In telephone networks, DTMF signals are used to encode dial trains and other information.

Dual-tone Multi-Frequency (DTMF) signaling is the basis for voice communications control and is widely used worldwide in modern telephony to dial numbers and configure switchboards. It is also used in systems such as in voice mail, electronic mail and telephone banking.

A DTMF signal consists of the sum of two sinusoids - or tones - with frequencies taken from two mutually exclusive groups. These frequencies were chosen to prevent any harmonics from being incorrectly detected by the receiver as some other DTMF frequency. Each pair of tones contains one frequency of the low group (697 Hz, 770 Hz, 852 Hz, 941 Hz) and one frequency of the high group (1209 Hz, 1336 Hz, 1477Hz) and represents a unique symbol.

| Frequency | 1209 Hz | 1336 Hz | 1477 Hz |
|---|---|---|---|
| 697 Hz | 1 | 2 | 3 |
| 770 Hz | 4 | 5 | 6 |
| 852 Hz | 7 | 8 | 9 |
| 941 Hz | * | 0 | # |

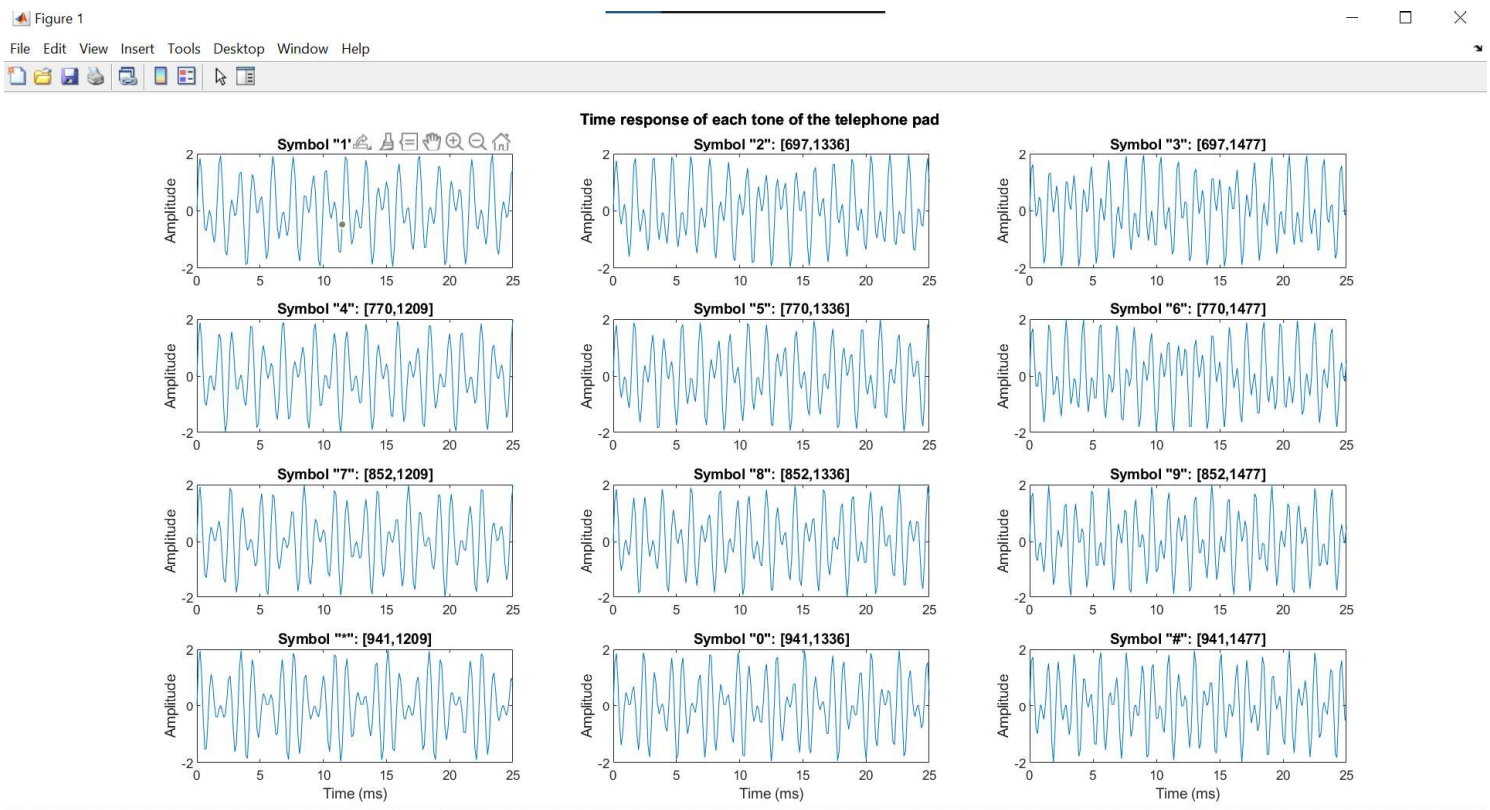
**Flow Chart:**




**PROGRAM:**

```
symbol = {'1','2','3','4','5','6','7','8','9','*','0','#'};
lfg = [697 770 852 941]; % Low frequency group
hfg = [1209 1336 1477];  % High frequency group
f  = [];
for c=1:4,
    for r=1:3,
        f = [ f [lfg(c);hfg(r)] ];
    end
end
Fs  = 8000;        % Sampling frequency 8 kHz
N = 800;           % Tones of 100 ms
t   = (0:N-1)/Fs; % 800 samples at Fs
pit = 2*pi*t;

tones = zeros(N,size(f,2));
for toneChoice=1:12,
    % Generate tone
    tones(:,toneChoice) = sum(sin(f(:,toneChoice)*pit))';
    % Plot tone
    subplot(4,3,toneChoice),plot(t*1e3,tones(:,toneChoice));
    title(['Symbol "', symbol{toneChoice},'":
[',num2str(f(1,toneChoice)),',',num2str(f(2,toneChoice)),']'])
    set(gca, 'Xlim', [0 25]);
    ylabel('Amplitude');
    if toneChoice>9, xlabel('Time (ms)'); end
end
set(gcf, 'Color', [1 1 1], 'Position', [1 1 1280 1024])
annotation(gcf,'textbox', 'Position',[0.38 0.96 0.45 0.026],...
    'EdgeColor',[1 1 1],...
    'String', '\bf Time response of each tone of the telephone pad', ...
    'FitBoxToText','on');
```

**Output Waveform:**




**RESULT:** DTMF waveform looks as a random signal.

**VIVA QUESTIONS:**

1. Expand DTMF?

**Ans.**

Dual Tone Multi Frequency

2. Define frequency groups for all symbols?

3. Define DTMF?

**Ans.**

Dual-tone multi-frequency signaling is a telecommunication signaling system using the voice-frequency band over telephone lines between telephone equipment and other communications devices and switching centers.

4. Give the applications of DTMF?

**Ans.**

- The DTMF tones are mainly used at the telephone switching centers, to detect the dialed numbers
- These are used in the terrestrial stations to switch on and switch off the remote transmitters
- DTMF is also used in the call centers, IVR systems, and security systems
- This system can be used in the industrial applications

5. Define low frequency group and high frequency group?

**Exercise:**

1. Generate DTMF signal for the symbols 5, 8, 3 and #.

**Date of Performance:** 15th Nov' 21

# Experiment No: 10

## AIM: To study CODE COMPOSER STUDIO

## INTRODUCTION TO CODE COMPOSER STUDIO

Code Composer Studio™ (CCS or CCStudio) is the integrated development environment for TI's DSPs, microcontrollers and application processors. CCStudio includes a suite of tools used to develop and debug embedded applications. It includes compilers for each of TI's device families, source code editor, project build environment, debugger, profiler, simulators and many other features.

CCStudio provides a single user interface taking users through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before and add functionality to their application thanks to sophisticated productivity tools.

CCStudio version 4 (CCSv4) is based on the Eclipse open source software framework. CCSv4 is based on Eclipse because it offers an excellent software framework for development environments a standard framework many embedded software vendors. CCSv4 combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature rich development environment for embedded developers

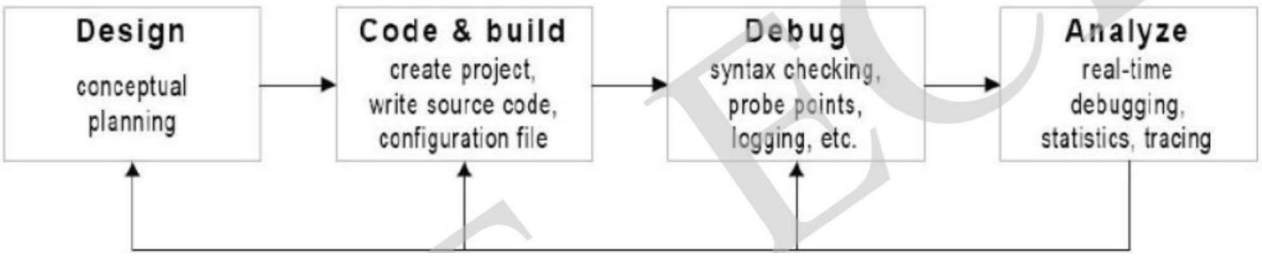
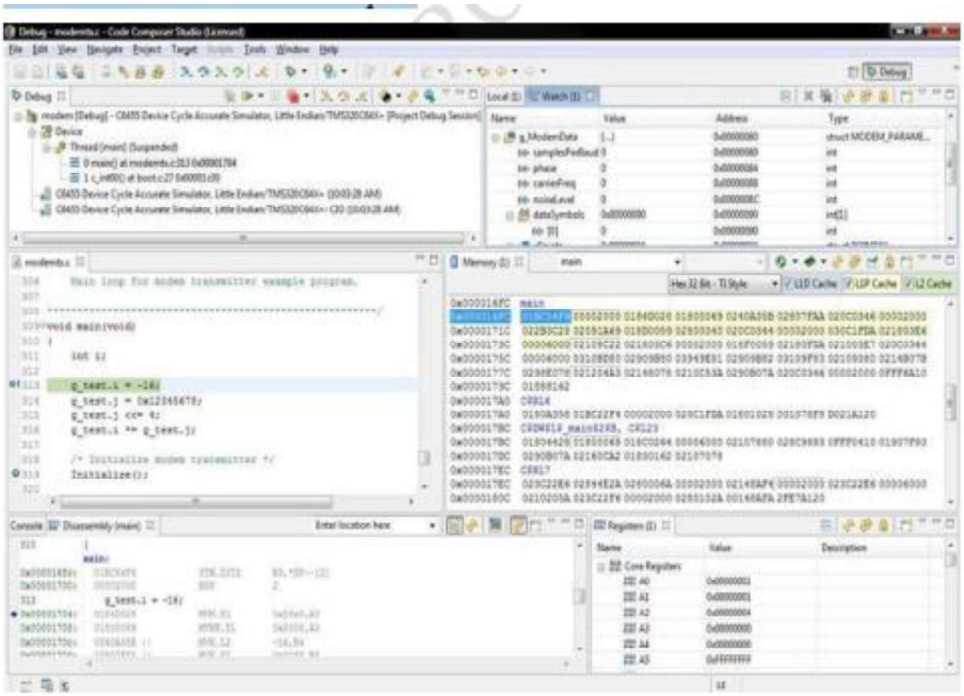



Fig.: Simplified Code Composer Studio IDE Development Flow.



## Features

### Debugger

CCStudio's integrated debugger has several capabilities and advanced breakpoints to simplify development. Conditional or hardware breakpoints are based on full C expressions, local variables or registers. The advanced memory window allows you to inspect each level of memory so that you can debug complex cache coherency issues. CCStudio supports the development of complex systems with multiple processors or cores. Global breakpoints and synchronous operations provide control over multiple processors and cores.

### Profiling

CCStudio's interactive profiler makes it easy to quickly measure code performance and ensure the efficient use of the target's resources during debug and development sessions. The profiler allows developers to easily profile all C/C++ functions in their application for instruction cycles or other events such as cache misses/hits, pipeline stalls and branches. Profile ranges can be used to concentrate efforts on high-usage areas of code during optimization, helping developers produce finely-tuned code. Profiling is available for ranges of assembly, C++ or C code in any combination. To increase productivity, all profiling facilities are available throughout the development cycle.

### Scripting

Some tasks such as testing need to run for hours or days without user interaction. To accomplish such a task, the IDE should be able to automate common tasks. CCStudio has a complete scripting environment allowing for the automation of repetitive tasks such as testing and performance benchmarking. A separate scripting console allows you to type commands or to execute scripts within the IDE.

## Image Analysis and Visualization

CCStudio has many image analysis and graphic visualization. It includes the ability to graphically view variables and data on displays which can be automatically refreshed. CCStudio can also look at images and video data in the native format (YUV, RGB) both in the host PC or loaded in the target board.

- Real-time mode which provides for the debugging of code that interacts with interrupts that must not be disabled. Real-time mode allows you to suspend background code at break events while continuing to execute time-critical interrupt service routines.

- Multi-core operations such as synchronous run, step, and halt. This includes crosscore triggering, which provides the ability to have one core trigger other cores to halt. Advanced Event Triggering (AET) which is available on selected devices, allows a user to halt the CPU or trigger other events based on complex events or sequences such as invalid data or program memory accesses. It can non-intrusively measure performance and count system events (for example, cache events).

CCStudio provides Processor Trace on selected devices to help customers find previously "invisible" complex real-time bugs. Trace can detect the really hard to find bugs – race conditions between events, intermittent real-time glitches, crashes from stack overflows, runaway code and false interrupts without stopping the processor. Trace is a completely nonintrusive debug method that relies on a debug unit inside the processor so it does not interfere or change the application's real-time behavior. Trace can fine tune code performance and cache optimization of complex switch intensive multi-channel applications. Processor Trace supports the export of program, data, timing and selected processor and system events/interrupts. Processor Trace can be exported either to an XDS560 Trace external JTAG emulator, or on selected devices, to an on chip buffer Embedded Trace Buffer (ETB).

**Real time operating system support**

CCSv4 comes with two versions of TI's real time operating system:

- DSP/BIOS 5.4x is a real-time operating system that provides pre-emptive multitasking services for DSP devices. Its services include ISR dispatching, software interrupts, semaphores, messages, device I/O, memory management, and power management. In addition, DSP/BIOS 5.x also includes debug instrumentation and tooling, including low overhead print and statistics gathering.

- BIOS 6.x is an advanced, extensible real-time operating system that supports ARM926, ARM Cortex M3, C674x, C64x+, C672x, and 28x-based devices. It offers numerous kernel and debugging enhancements not available in DSP/BIOS 5.x, including faster, more flexible memory management, events, and priority-inheritance mutexes.
  Note: BIOS 6.x includes a DSP/BIOS 5.x compatibility layer to support easy migration of application source code.

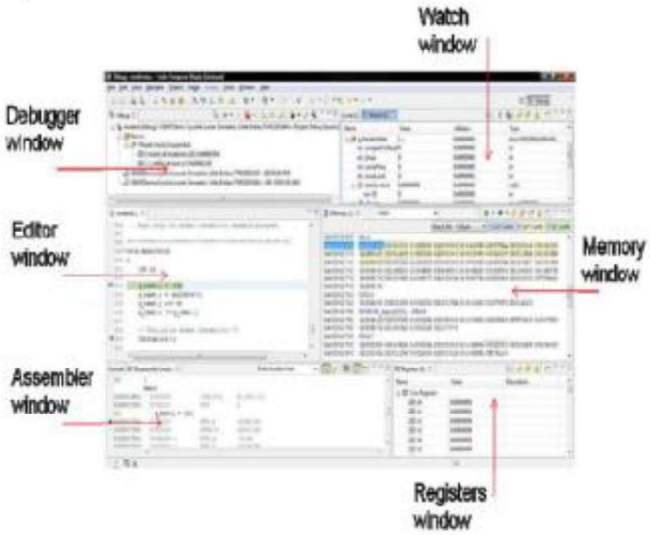
**Step 1:**

Open the code composer studio (CCSV4) and give a name to workspace and store it in the default path itself.

Note: don't assign other than default path unless you are familiar with eclipse frame work based CCSV4


**Step 2:**

Project windows overview




**RESULT:** Code Composer Studio software tools are studied.