**LAB # 01**

**GENERATION OF DIFFERENT BASIC SIGNAS USING MATLAB**

**TASK #1: Write a program in MATLAB to generate five basic signal and then plot them.**

Unit impulse:

%unit impulse function

n=-3:1:10;

x=[zeros(1,3) ones(1,1) zeros(1,10)];

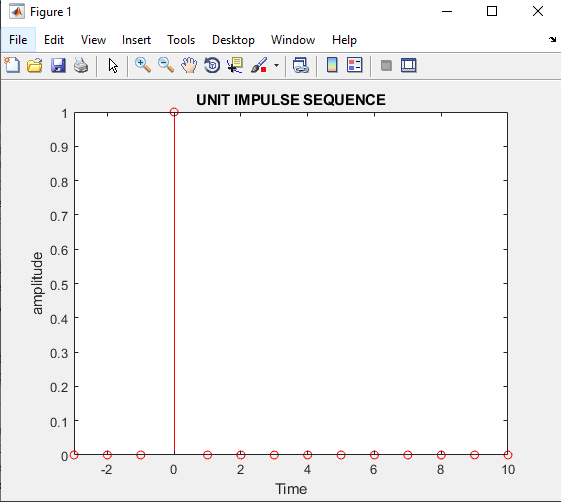
stem (n,x,'r');

ylabel('amplitude')

title('UNIT IMPULSE SEQUENCE')

axis([-3 10 0 1])

**RESULTS:**



**UNIT STEP:**

n=-3:1:10;

x=[zeros(1,3) ones(1,1) ones(1,10)];

stem(n,x, 'r')

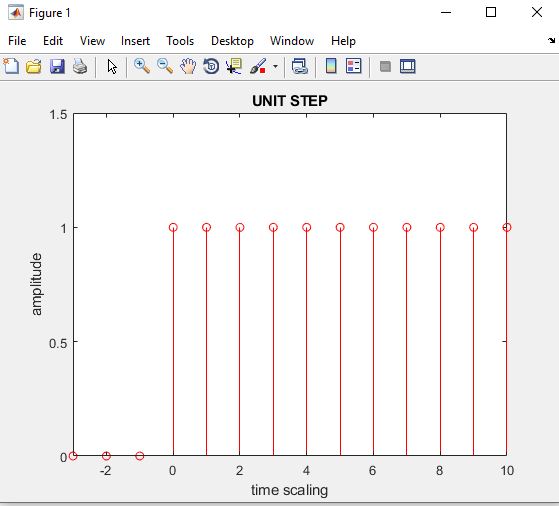
xlabel('time scaling')

ylabel('amplitude')

title('UNIT STEP')

axis([-3 10 0 1.5])

**Results:**



**Unit Ramp:**n=-3:1:10;

y=[zeros(1,3),ones(1,1),ones(1,10)];

x=y.\*n;

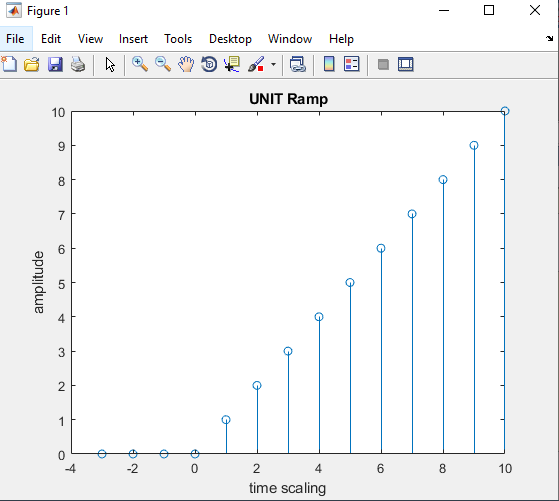
stem(n,x)

xlabel('time scaling')

ylabel('amplitude')

title('UNIT Ramp')

**Results:**



**Sine function:**

n = 0:1:20;

f = 0.1;

phase = 0;

A =10;

arg = 2\*pi\*f\*n - phase;

x = A\*sin(arg);

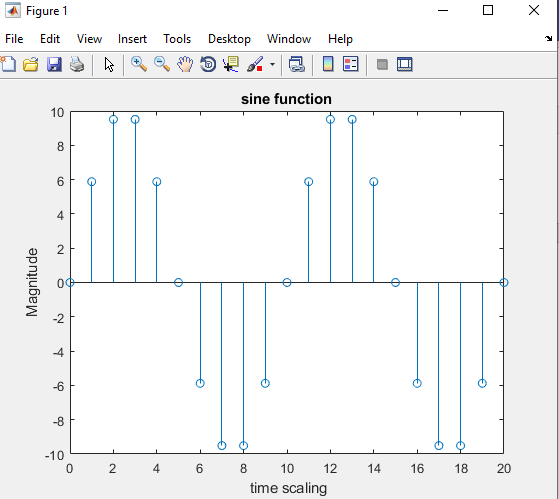
stem(n,x); % Plot the generated sequence

xlabel(' time scaling')

ylabel('Magnitude')

title('sine function')

**Results:**



Exponential function:

n=0:1:20;

%for a>1

a=1.2;

w=(a).^n;

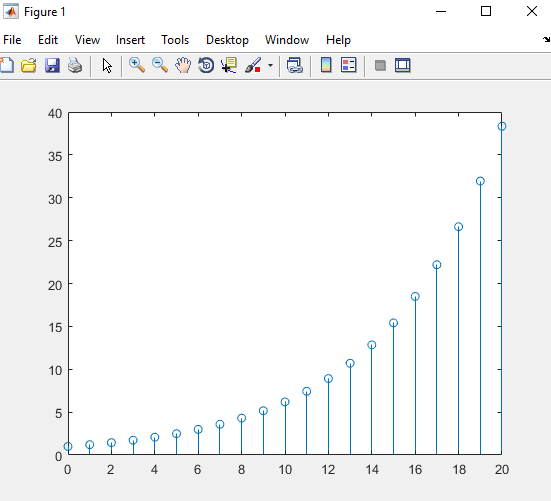
xlabel('time scaling')

ylabel('amplitude')

title('exponential signal for a>1')

stem(n,w)

**Results:**



**Task# 02: Write a Program in MATLAB to generate the unit impulse sequence δ[n] and**

**display it and Modify the above Program to generate a delayed unit impulse sequence δd[n] with a delay of 11 samples. Run the modified program and display the sequence generated.**

clf;

subplot(2,1,1);

L=input('enter lower bound ');

U=input('enter upper bound ');

n=-L:1:U;

x=[zeros(1,L) ones(1,1) zeros(1,U)];

stem (n,x,'r');

ylabel('amplitude')

title('UNIT IMPULSE SEQUENCE')

axis([-L U 0 1])

% Delayed sample unit impulse signal

subplot(2,1,2)

D=input('enter delay samples ');

x1=[zeros(1,L+D) ones(1,1) zeros(1,U-D)];

stem (n,x1,'b');

xlabel('time scaling')

ylabel('amplitude')

title('UNIT IMPULSE DELAYED SEQUENCE')

axis([-L U 0 1])

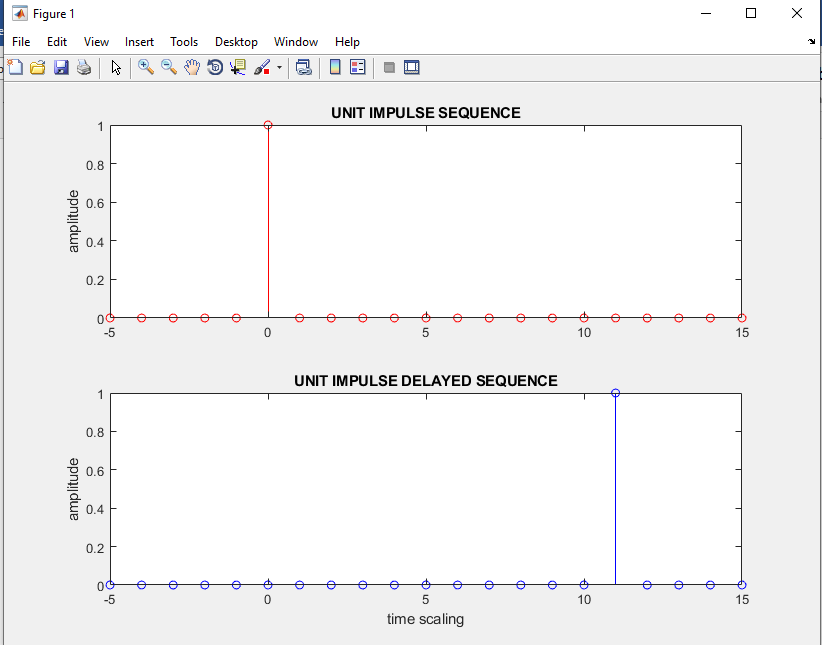
**output:**

enter lower bound 5

enter upper bound 15

enter delay samples 11

**Result:**



**Task#03: Write a Program in MATLAB to generate the unit step sequence u[n] and display it.**

**And also modify the program for delaying the signal.**

subplot(2,1,1);

l=input('enter lower bound ');

u=input('enter upper bound ');

n=-l:1:u;

x=[zeros(1,l) ones(1,1) ones(1,u)];

stem(n,x, 'r')

xlabel('time scaling')

ylabel('amplitude')

title('UNIT STEP')

axis([-l u 0 1.5])

subplot(2,1,2);

d=input('enter delayed sample ');

x=[zeros(1,l+d) ones(1,1) ones(1,u-d)];

stem(n,x, 'r')

xlabel('time scaling')

ylabel('amplitude')

title('UNIT STEP')

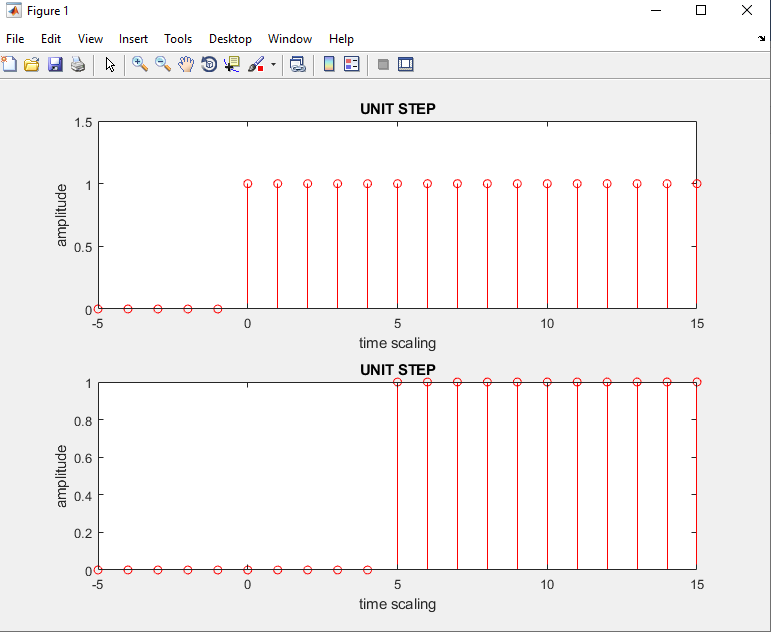
**OUTPUT:**

enter lower bound 5

enter upper bound 15

enter delayed sample 5

**RESULTS:**



**Task #04: Write a Program in MATLAB to generate the unit step ramp r[n] and display it.**

L=input('enter lower bound ');

U=input('enter upper bound');

n=-L:1:U;

y=[zeros(1,L),ones(1,1),ones(1,U)];

x=y.\*n;

stem(n,x)

xlabel('time scaling')

ylabel('amplitude')

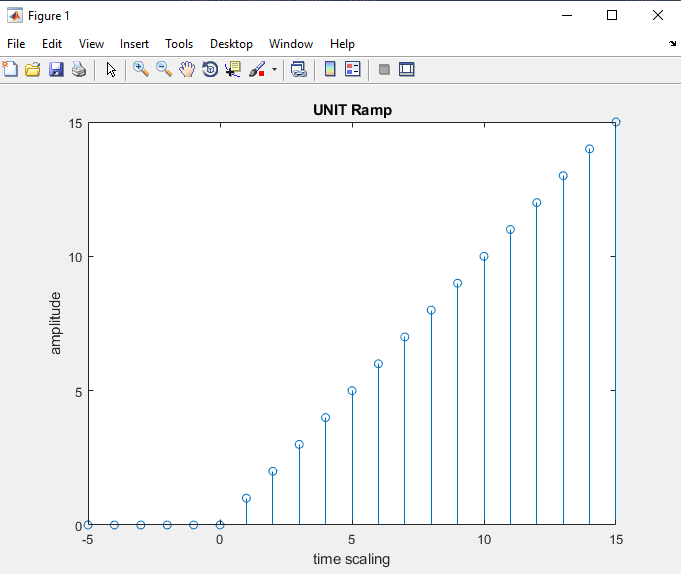
title('UNIT Ramp')

**OUTPUT:**

enter lower bound 5

enter upper bound15

**RESULTS:**



**Task #05: Write a Program in MATLAB to generate a sinusoidal signal**

**by using appropriate value of frequency, input form user and display it.**

L=input('enter lower bound');

U=input('enter upper bound');

n = L:1:U;

f = 0.1;

phase = 0;

A = input('enter amplitude');

arg = 2\*pi\*f\*n - phase;

x = A\*sin(arg);

stem(n,x); % Plot the generated sequence

xlabel(' time scaling')

ylabel('Magnitude')

title('sine function')

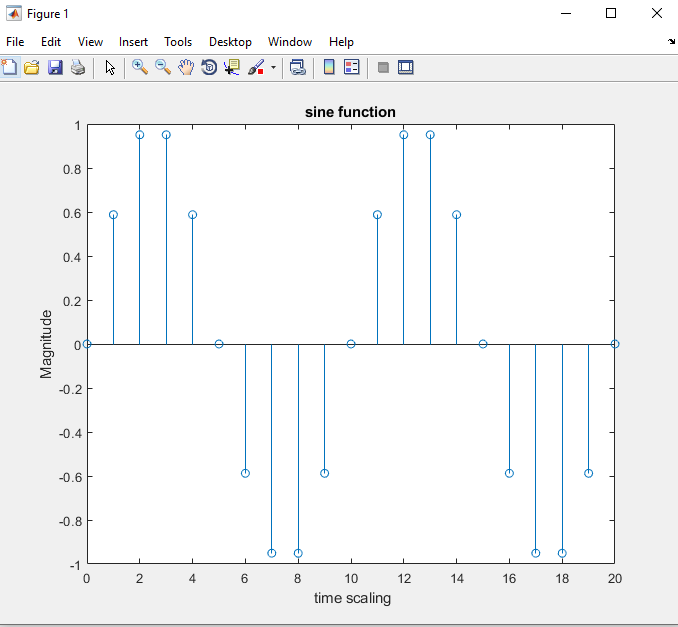
**output:**

enter lower bound0

enter upper bound20

enter amplitude1

**Results:**



**Task #06: Write a Program in MATLAB to generate an exponential sequence**

**a) Real exponential signal**

L=input('enter lower bound ');

U=input('enter upper bound ');

n=L:1:U;

%for a>1

a=input('enter the value of a ');

w=(a).^n;

xlabel('time scaling')

ylabel('amplitude')

title('exponential signal for a>1')

stem(n,w)

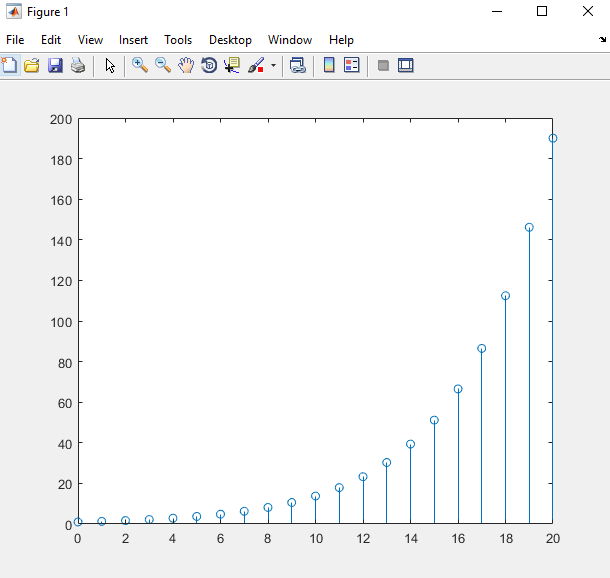
**output:**

enter lower bound 0

enter upper bound 20

enter the value of a 1.3

**Results:**



**b) Complex exponential signal**

L=input('enter lower bound i.e 0 ');

U=input('enter upper bound i.e 30 ');

n=L:1:U;

%a=complex value

a=input('enter the value of a i.e."complex variable and best graph is on a=-(1/12)+(pi/6)\*i"');

C=exp(a\*n);

stem(n,C)

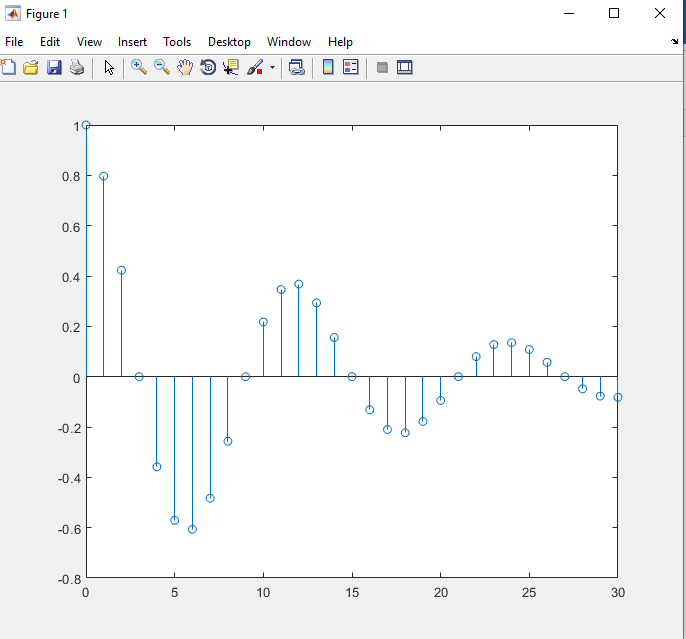
**output:**

enter lower bound i.e 0 0

enter upper bound i.e 30 30

enter the value of a i.e."complex variable and best graph is on a=-(1/12)+(pi/6)\*i"-(1/12)+(pi/6)\*i

**Results:**



**LAB 02**  **SAMPLING AND ALIASING**

**SAMPLING:**

**EXAMPLE 01:**

% in the Time Domain

clf;

t = 0:0.0005:1;

f = 5;

xa = cos(2\*pi\*f\*t);

subplot(2,1,1)

plot(t,xa);grid

xlabel('Time, msec');ylabel('Amplitude');

title('Continuous-time signal x\_{a}(t)');

axis([0 1 -1.2 1.2])

subplot(2,1,2);

**T = 0.02;**

n = 0:T:1;

xs = cos(2\*pi\*f\*n);

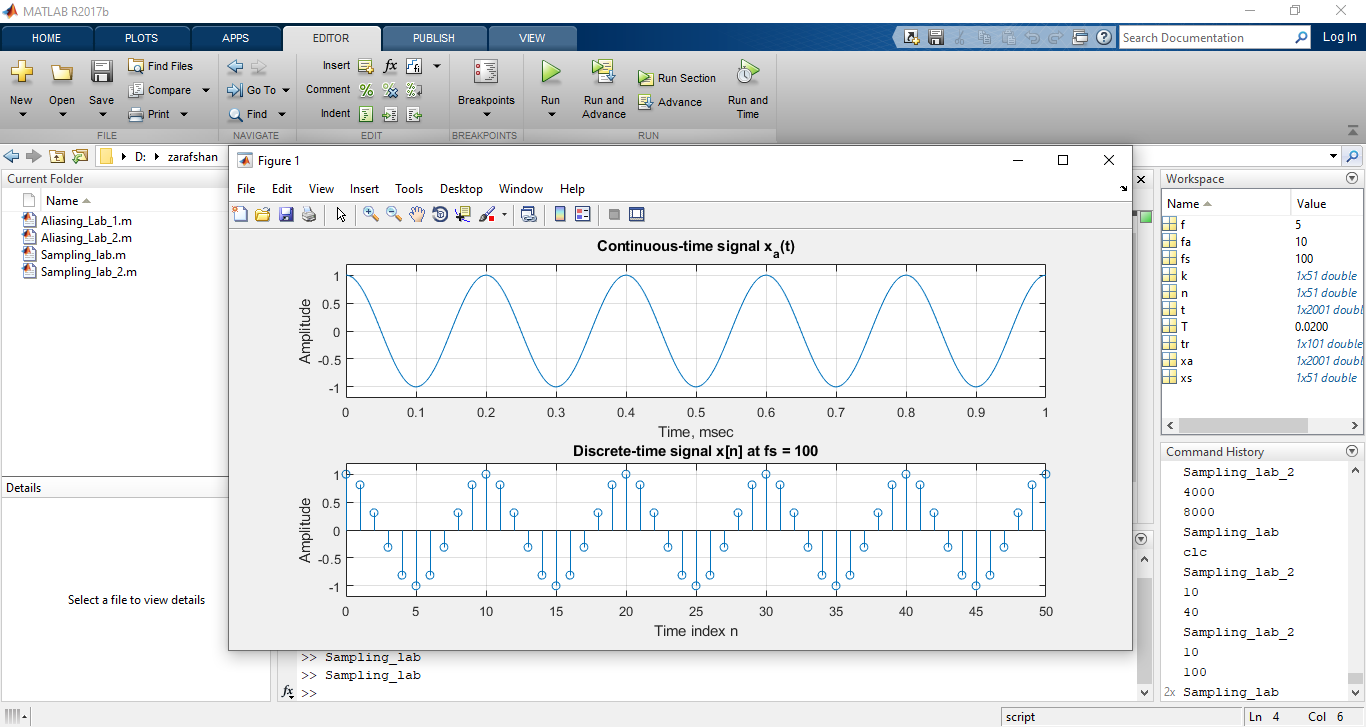
k = 0:length(n)-1; stem(k,xs);

grid

xlabel('Time index n');ylabel('Amplitude');

title('Discrete-time signal x[n] at fs = 100');

axis([0 (length(n)-1) -1.2 1.2])



**EXAMPLE 2:**

% in the Time Domain

clf;

t = 0:0.0005:1;

fa=input('enter analog frequency = ');

fs=input('enter sampling frequency = ');

xa = cos(2\*pi\*fa\*t);

subplot(3,1,1)

plot(t,xa);grid

xlabel('Time, msec');ylabel('Amplitude');

title('Continuous-time signal x\_{a}(t)');

axis([0 1 -1.2 1.2])

subplot(3,1,2);

T = 1/fs;

n = 0:T:1;

xs = cos(2\*pi\*f\*n);

k = 0:length(n)-1; stem(k,xs);

grid

xlabel('Time index n');ylabel('Amplitude');

title('Discrete-time signal x[n] ');

axis([0 (length(n)-1) -1.2 1.2])

subplot(3,1,3);

tr=k.\*T;

plot (tr,xs);

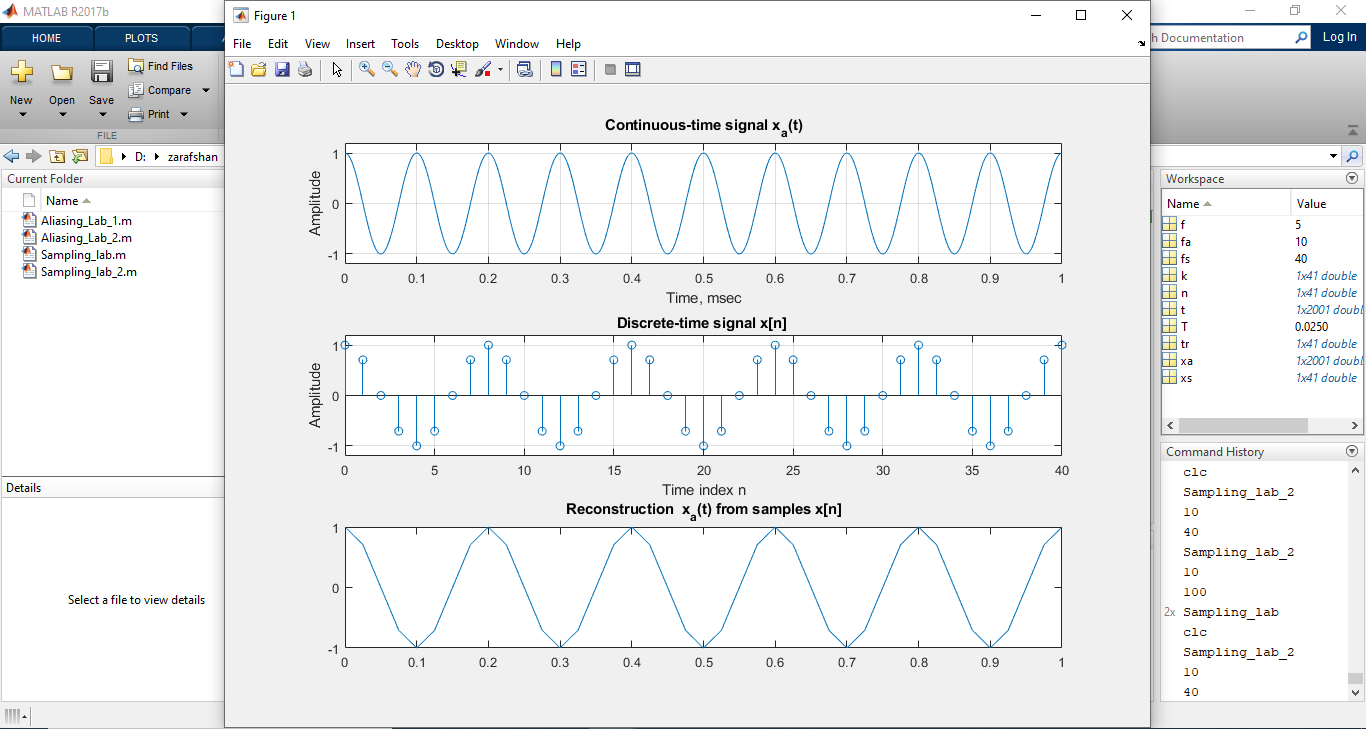
title('Reconstruction x\_{a}(t) from samples x[n]');

**OUTPUT 1:**

Sampling\_lab\_2

enter analog frequency = 10

enter sampling frequency = 40

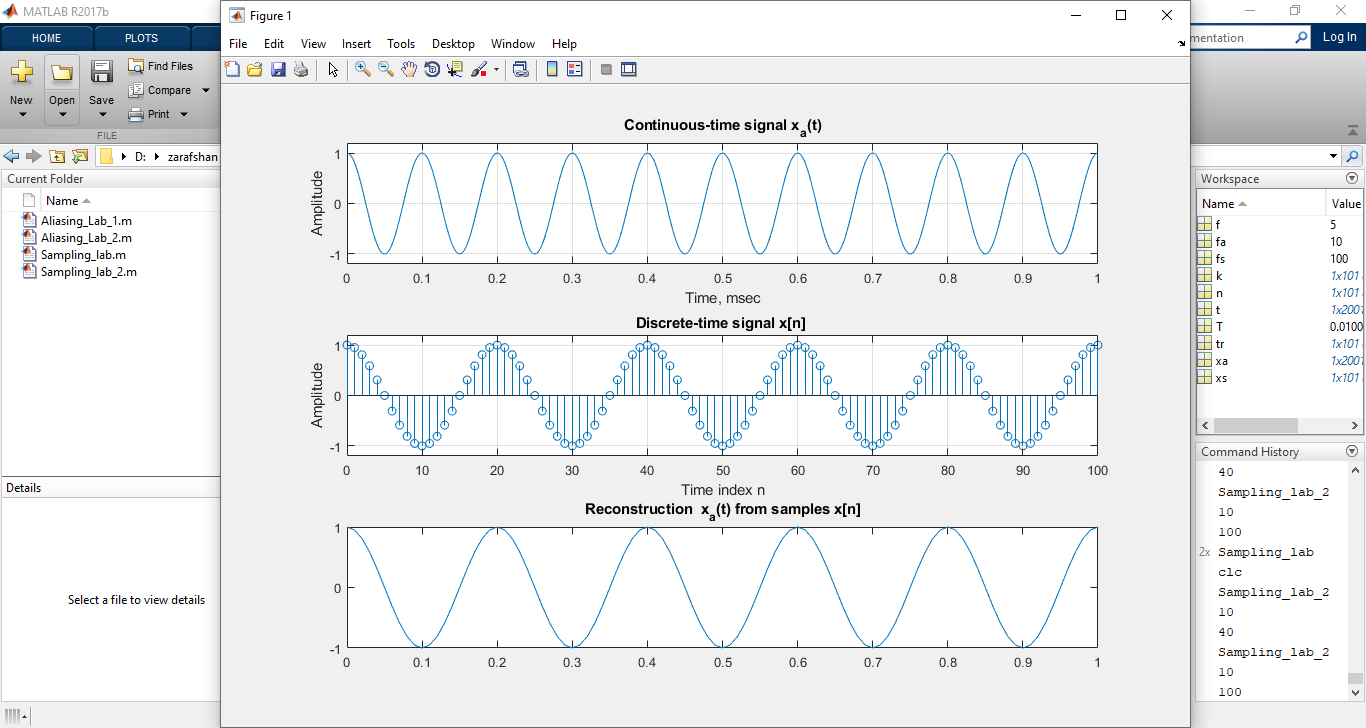


**OUTPUT 2:**

>> Sampling\_lab\_2

enter analog frequency = 10

enter sampling frequency = 100



**ALIASING**

**EXAMPLE 3:**

n = 0:1:10;

wd=input('enter digital frequency = ');

w\_alias = wd+2\*pi;

X = ['The aliase digital frequency will be ',num2str(w\_alias)];

disp(X)

subplot(2,1,1)

x = sin(wd\*n);

xlabel(' time scaling')

ylabel('Magnitude')

title('sin function')

stem(n,x); % Plot the generated sequence

%plot(n,x)

subplot(2,1,2)

xa = sin(w\_alias\*n);

% xlabel(' time scaling')

% ylabel('Magnitude')

% title('sin function')

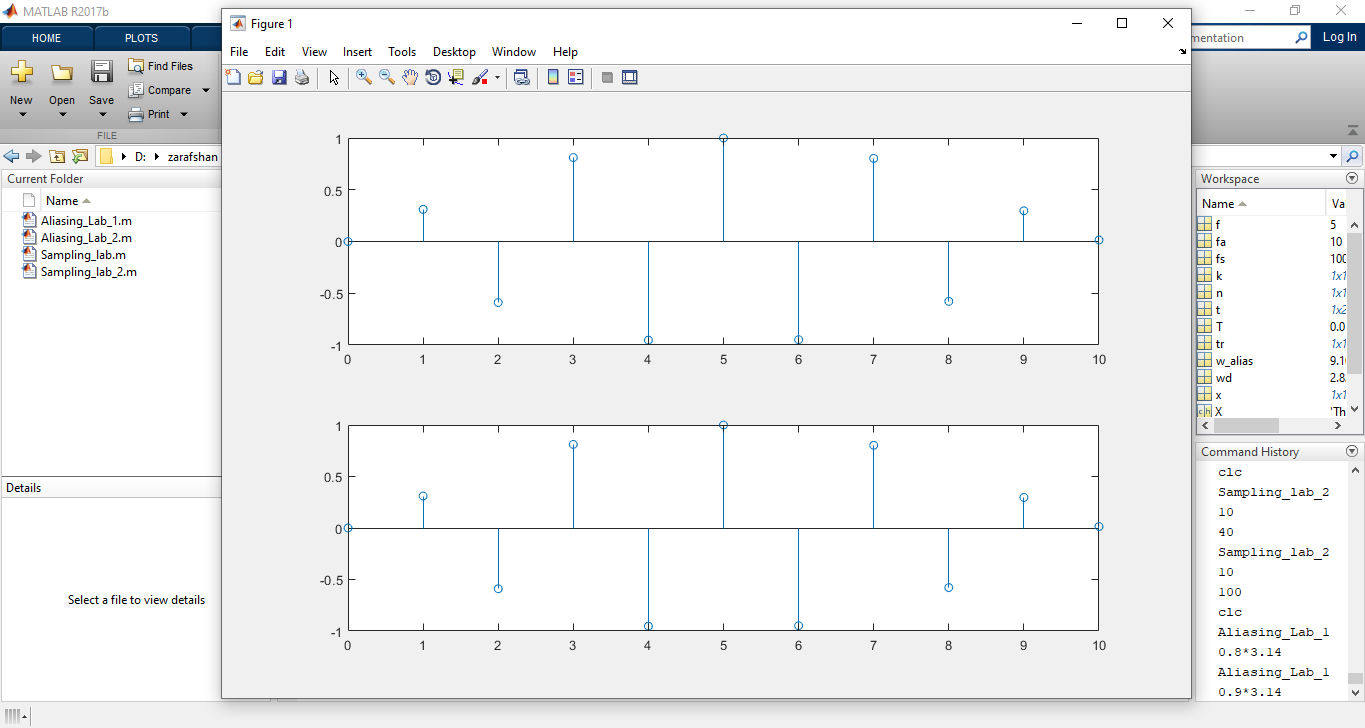
stem(n,xa); % Plot the generated sequence

**OUTPUT:**

>> Aliasing\_Lab\_1

enter digital frequency = 0.9\*3.14

The aliase digital frequency will be 9.1092



**EXAMPLE 4:**

fs=10; %10Hz

fa=80; % 1Hz

f\_aliase=fa+fs

t=0:0.001:1;

f1=2\*cos(2\*pi\*fa\*t);

f2=2\*cos(2\*pi\*f\_aliase\*t);

T=1/fs;

n=0:T:1;

y1=2\*cos(2\*pi\*1\*n);

y2=2\*cos(2\*pi\*11\*n);

k1=0:length(y1)-1;

k2=0:length(y2)-1;

subplot(2,2,1)

plot(t,f1);

xlabel('Time');

ylabel('Amplitude');

title('Continous time wave of frequency 80 Hz');

grid;

subplot(2,2,2)

plot(t,f2);

xlabel('Time');

ylabel('Amplitude');

title('Continous time wave of frequency 90 Hz');

grid;

subplot(2,2,3)

stem(k1,y1);

xlabel('sample number');

ylabel('Amplitude');

title('Sampling 80 Hz signal at 10 Hz');

grid;

subplot(2,2,4)

stem(k2,y2);

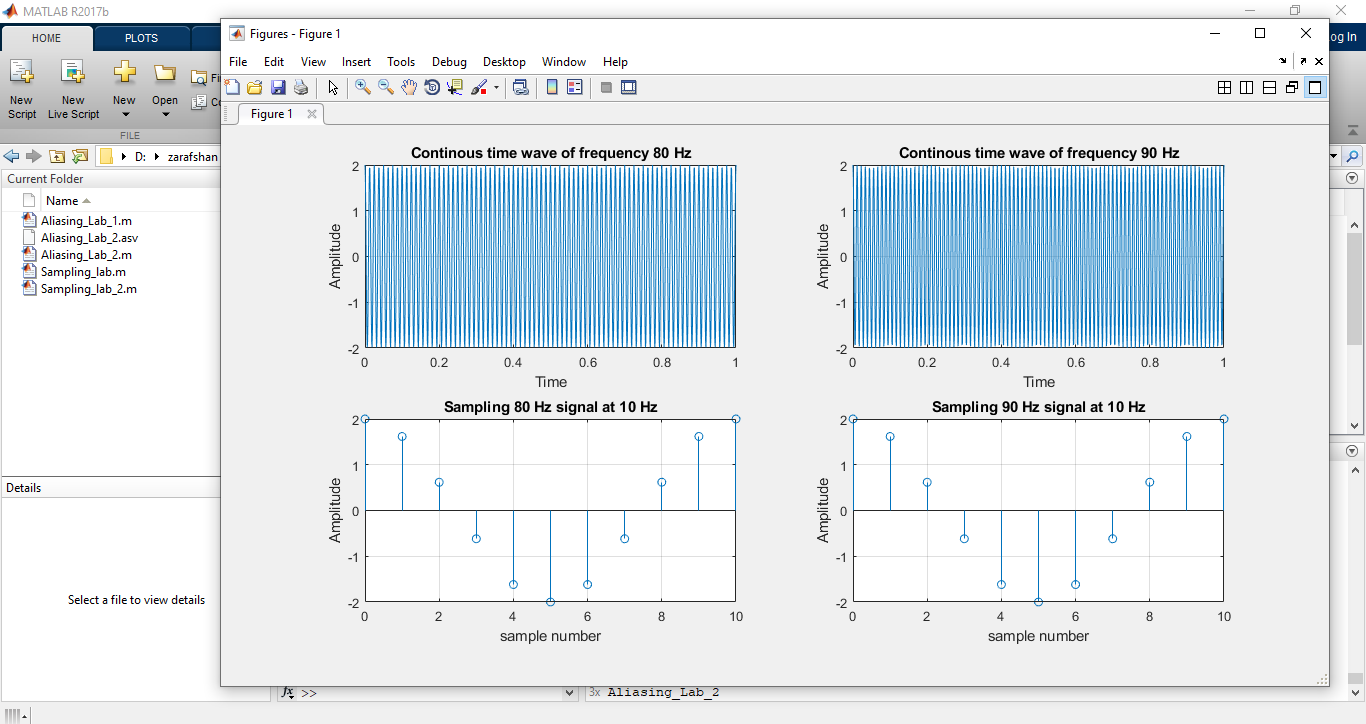
xlabel('sample number');

ylabel('Amplitude');

title('Sampling 90 Hz signal at 10 Hz');

grid;

**OUTPUT:**



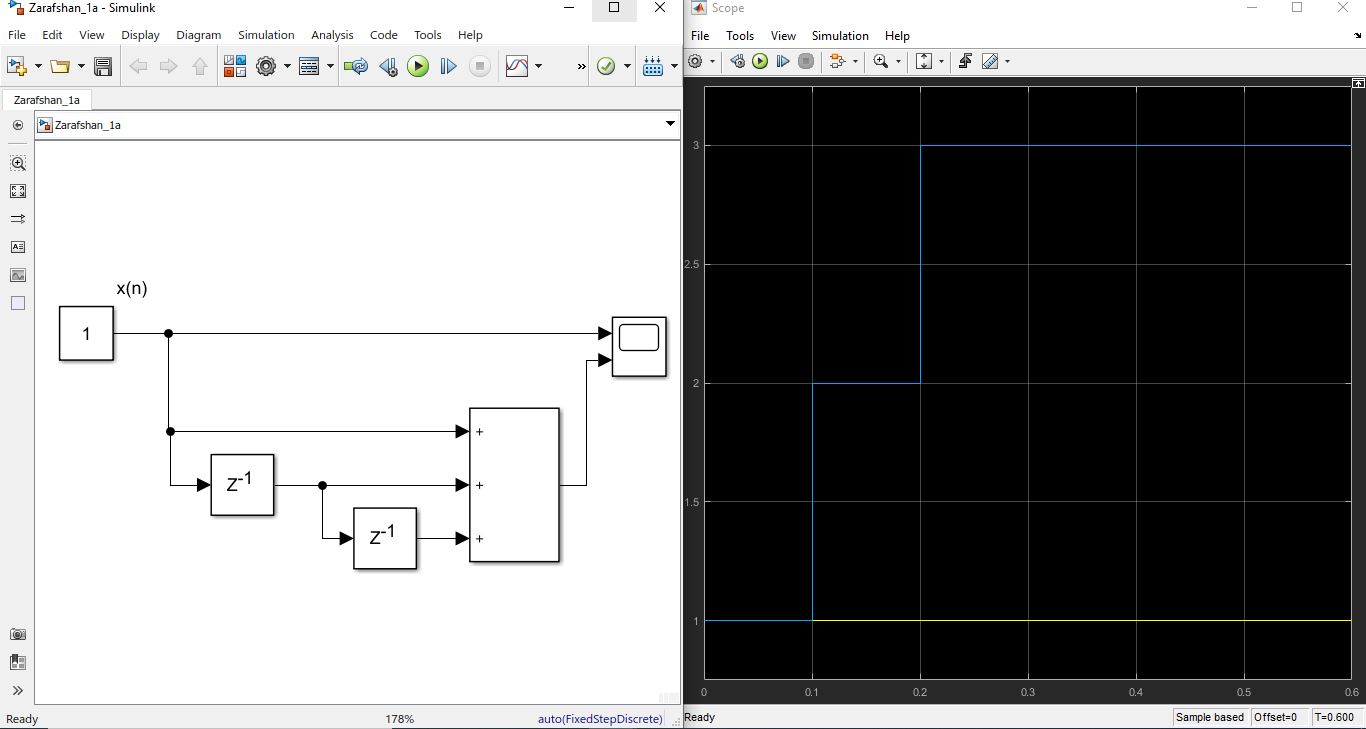
**LAB 03 Discrete Time System in Simulink**

Q1. Implement DT system described by difference equation in Simulink

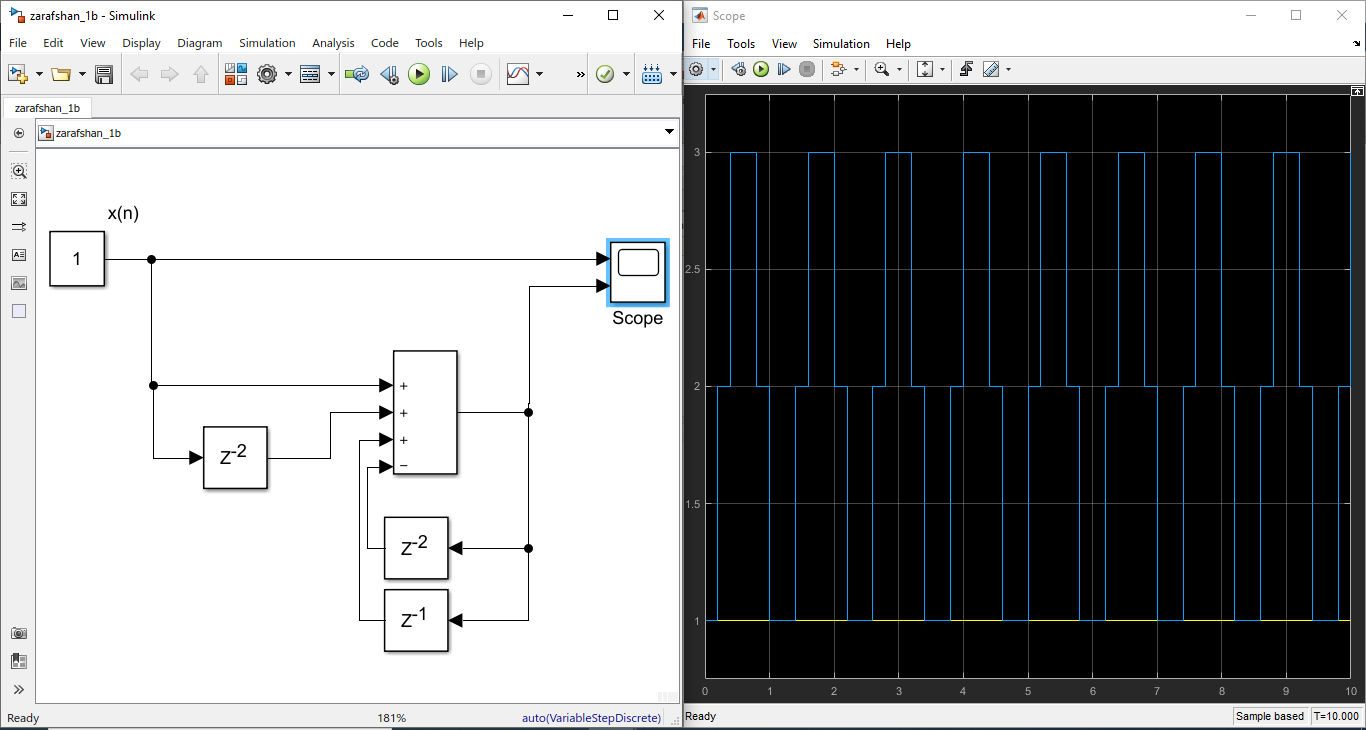
a). y(n)=x(n)+x(n-1)+x(n-2)

b).y(n)=y(n-1)+y(n-2)-x(n)+x(n-2)

**TASK 01 (a):**

****

**TASK 01 (b):**

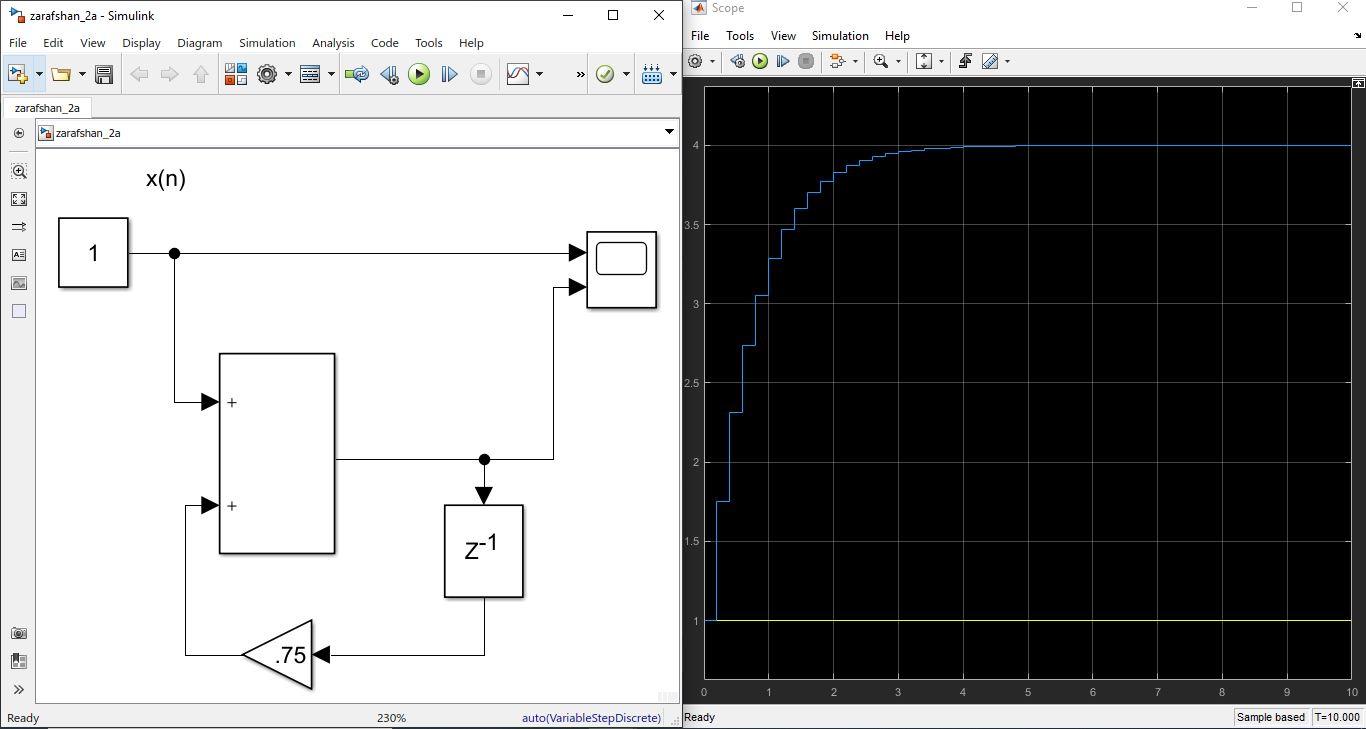


Q2. Draw o/p y(n) for 10 samples,

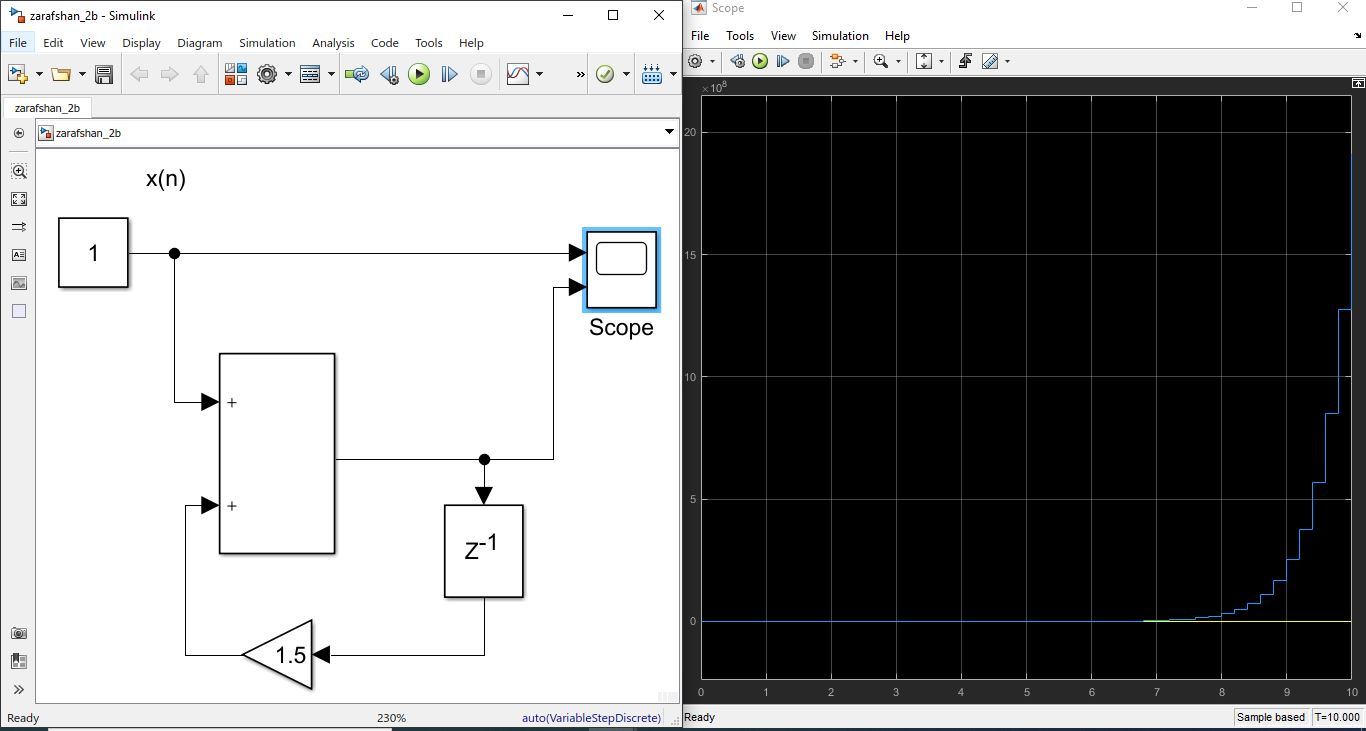
a). y(n)=0.75y(n-1)+x(n) consider y(-1)=0;

b).y(n)=1.5y(n-1)+x(n) consider y(-1)=0

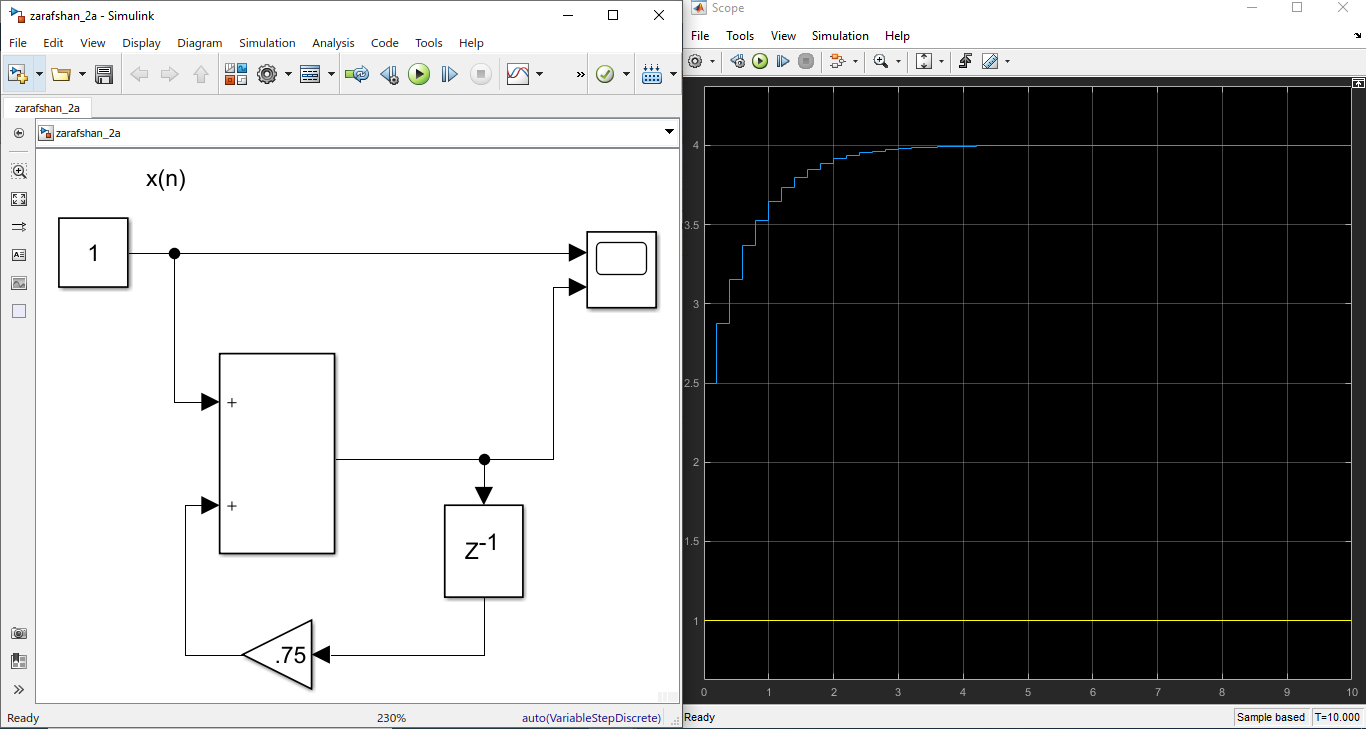
**TASK 02 (a): when y(-1)=0**

****

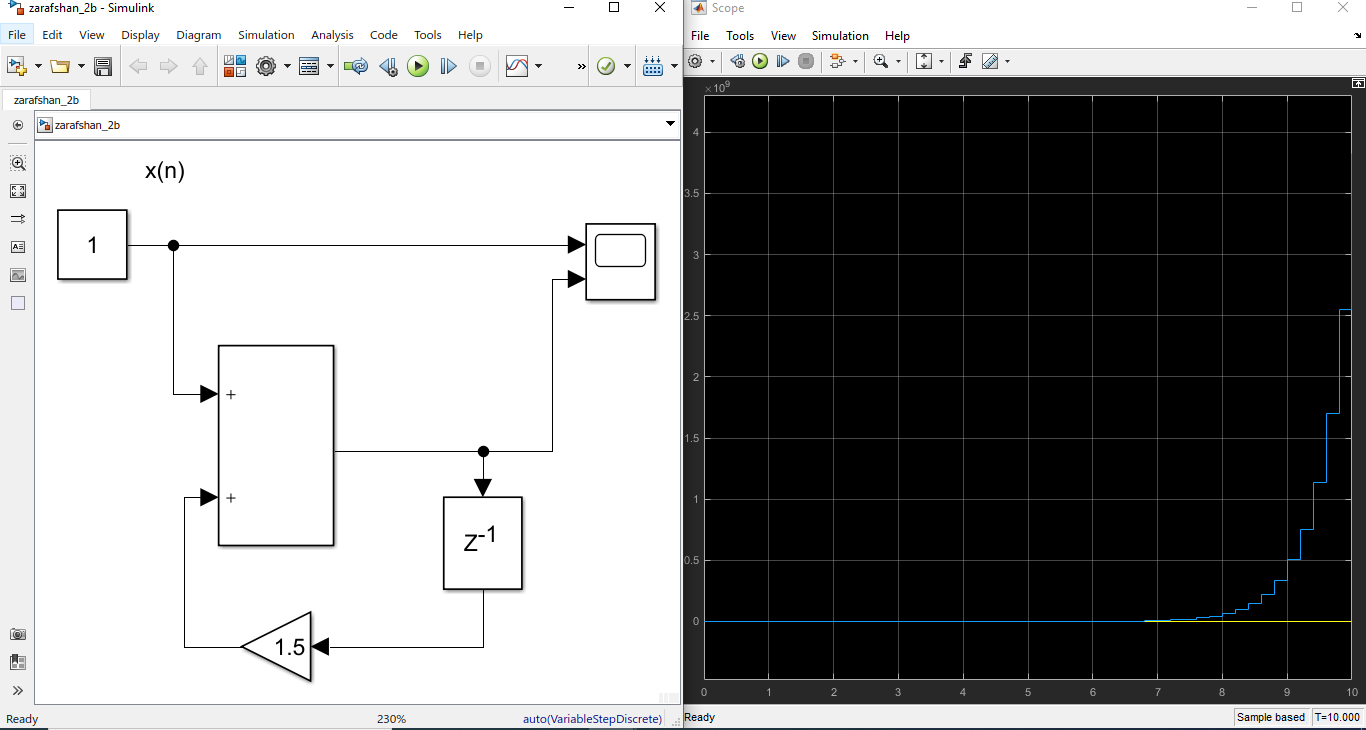
**TASK 02 (b): when y(-1)=0**

****

**TASK 02 (a): when y(-1)=2**

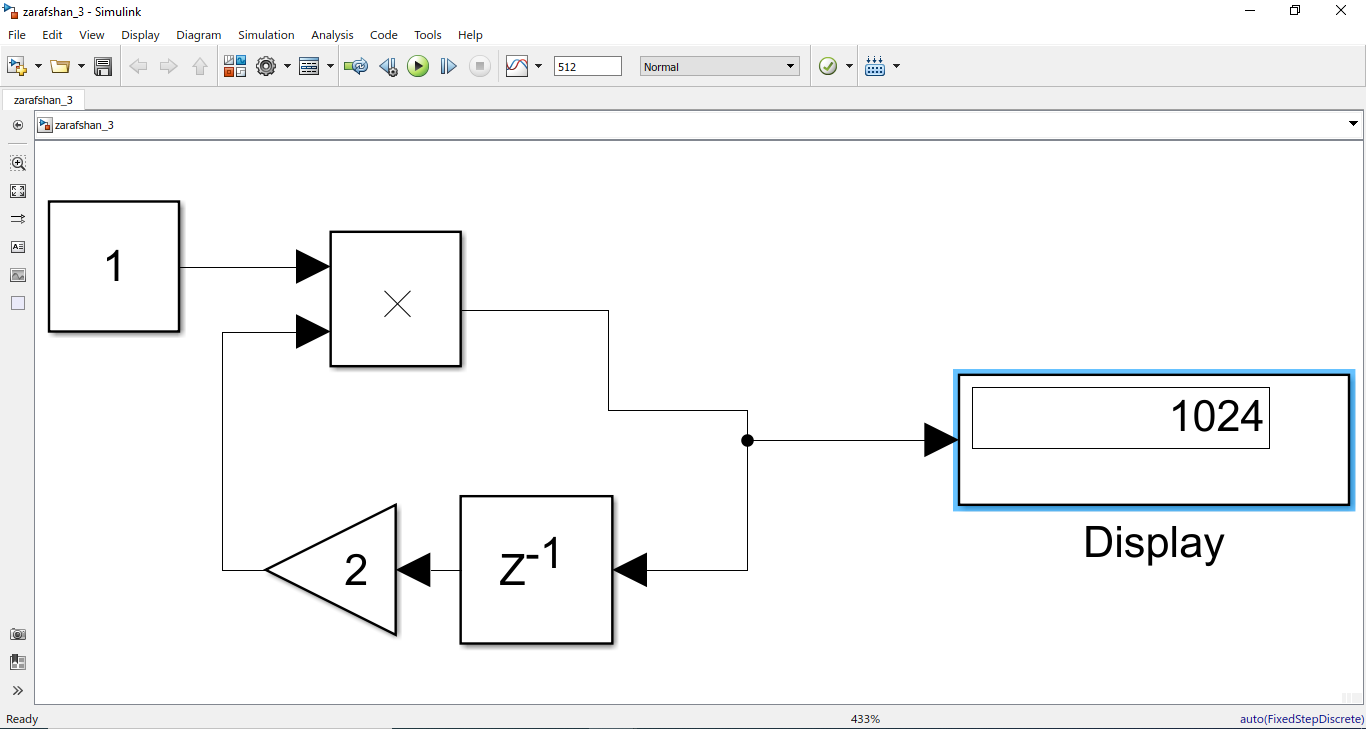


**TASK 02 (b): when y(-1)=2**



Q3. Make model to generate power series of 2, 1,2,4,8,16,-------1024.

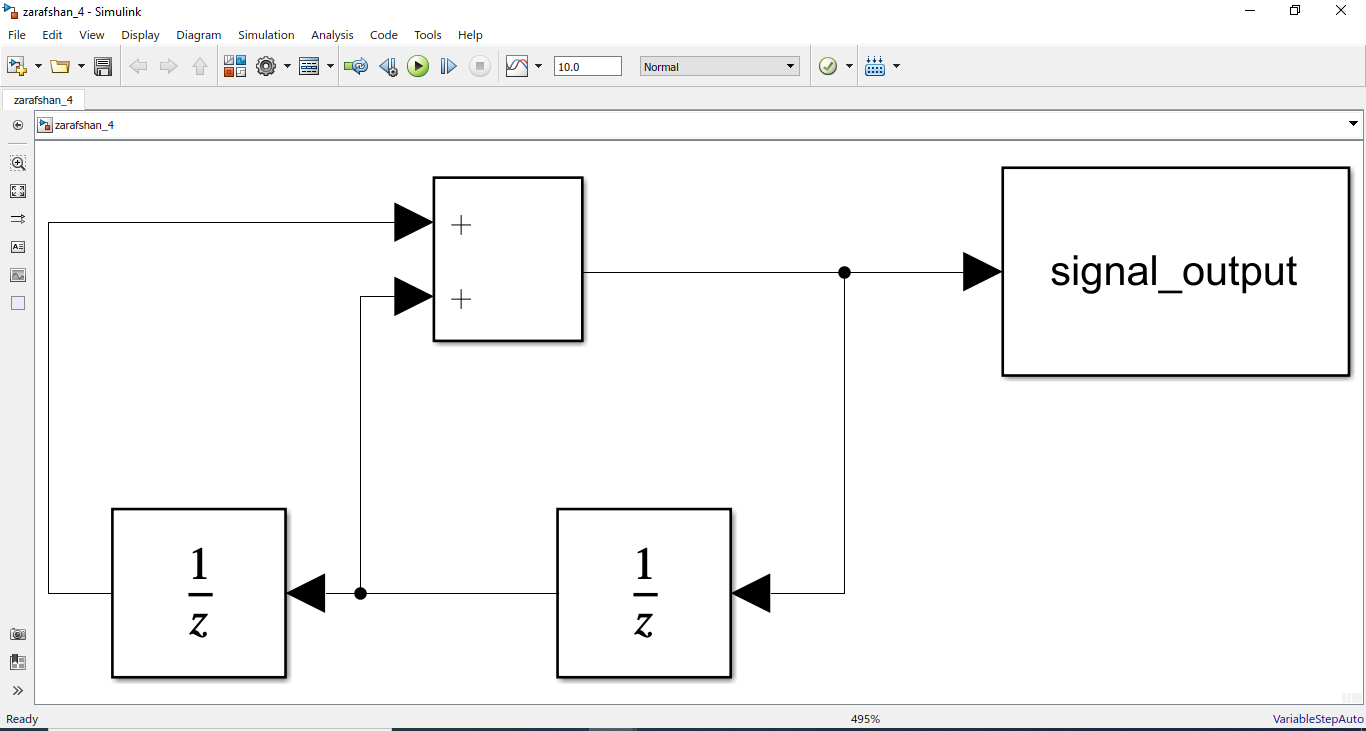
**TASK 3:**



Q4. Implement DT system for Fibonacci series generation

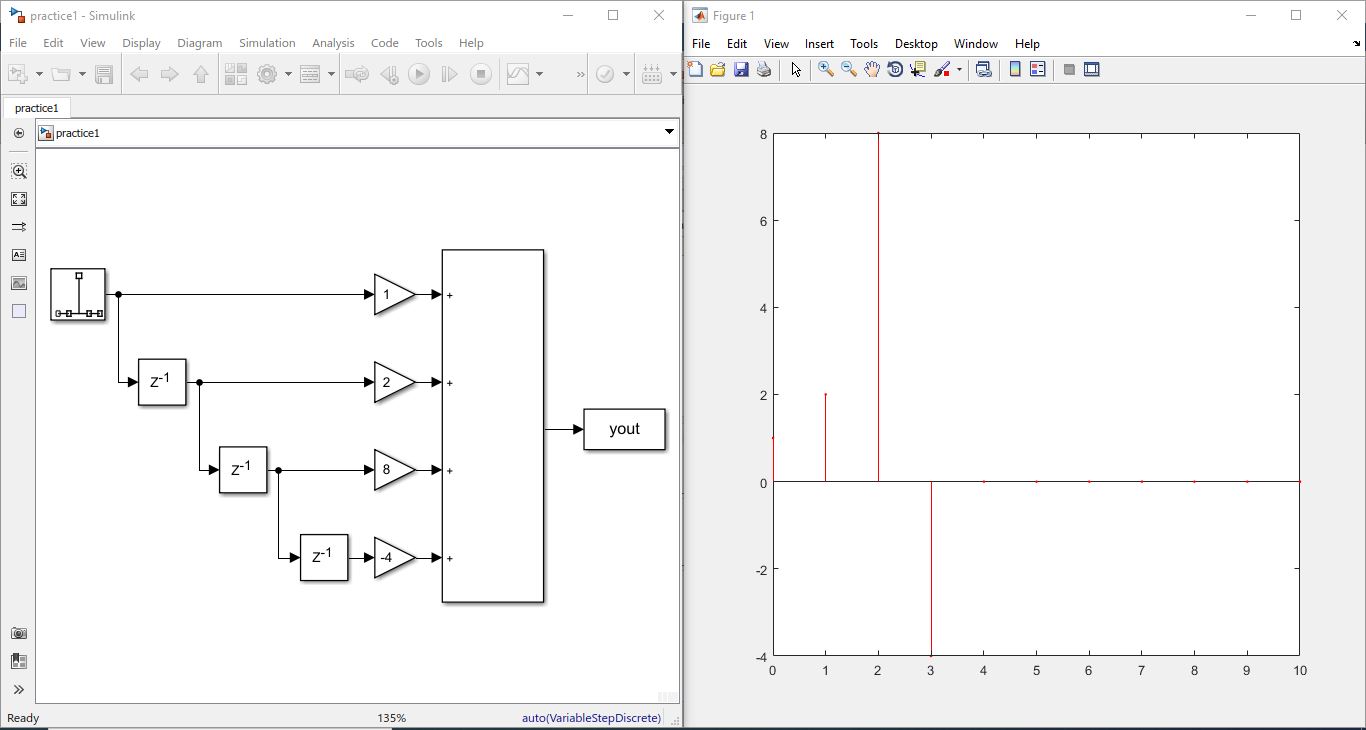
Y(n)=y(n-1)+y(n-2)

**TASK 04:**



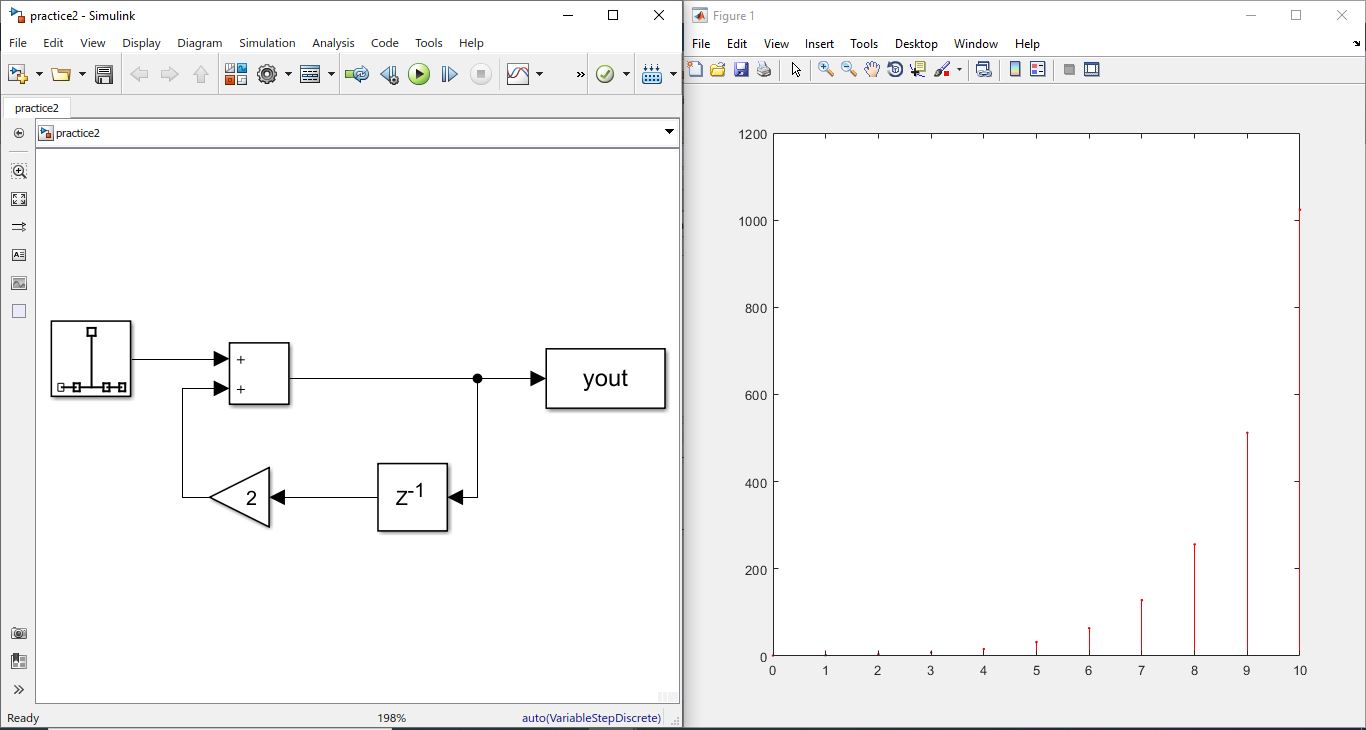
**LAB # 04 IMPULSE RESPONSE OF DT SYSTEM**

**Question1** Implementation of FIR system in Simulink.

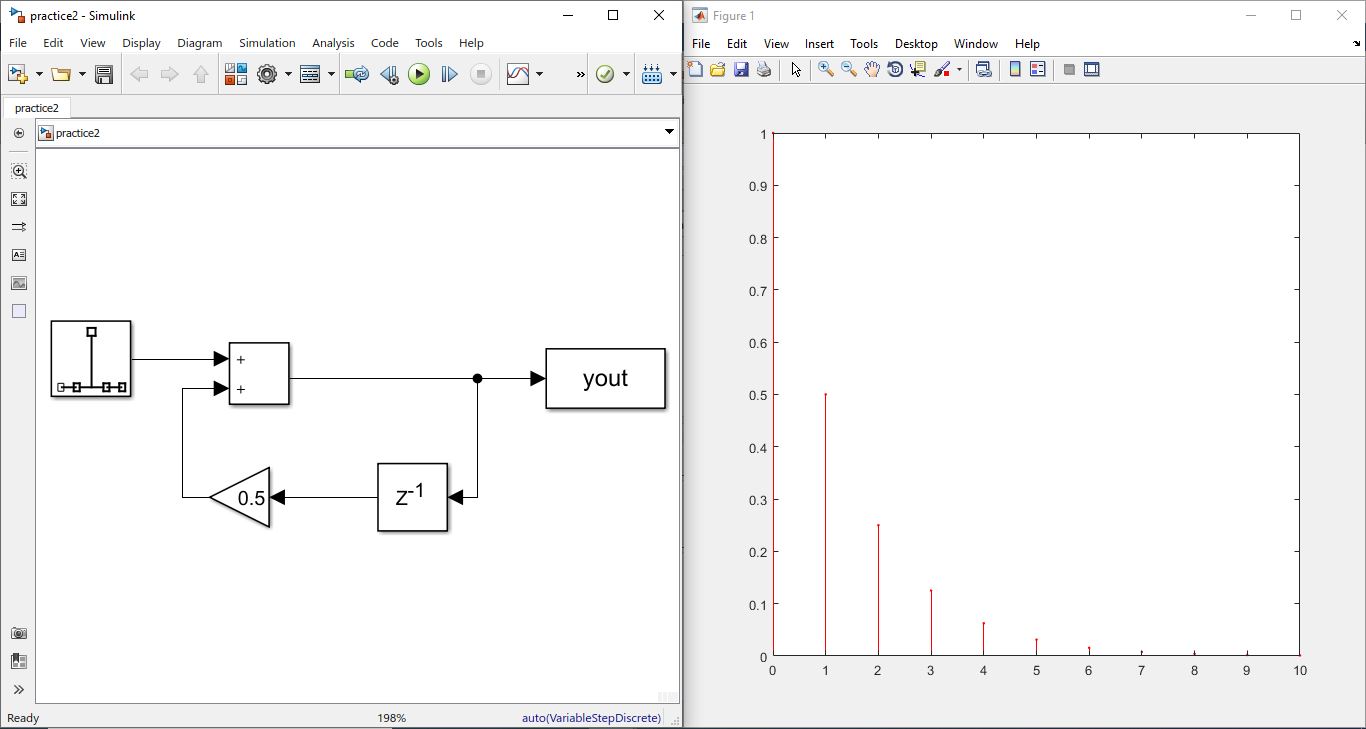


**Question2: Impulse response with feedback**

When Value of a=2



When Value of a=0.5;



**Lab # 05 DT Convolution and Co-relation**

**Objectives:**

1. Finding convolution and correlation of DT System using MATLAB

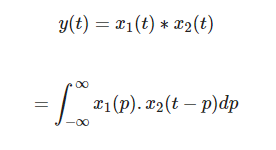
**Theory:**

**Convolution:**

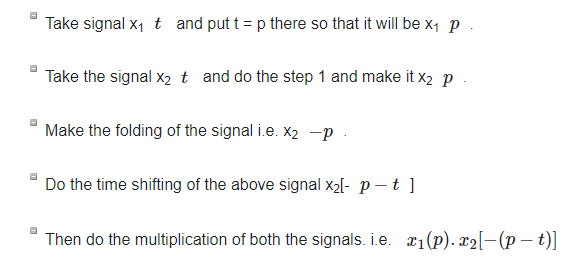
Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Using the strategy of impulse decomposition, systems are described by a signal called the impulse response. Convolution is important because it relates the three signals of interest: the input signal, the output signal, and the impulse response.

**Operation Definition:**

The convolution of two signals in the time domain is equivalent to the multiplication of their representation in frequency domain. Mathematically, we can write the convolution of two signals as



Steps for convolution



**Properties of convolution:**

**Commutative**

It states that order of convolution does not matter, which can be shown mathematically as



**Associative**

It states that order of convolution involving three signals, can be anything. Mathematically, it can be shown as;



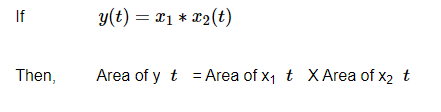
**Distributive**

Two signals can be added first, and then their convolution can be made to the third signal. This is equivalent to convolution of two signals individually with the third signal and added finally. Mathematically, this can be written as;



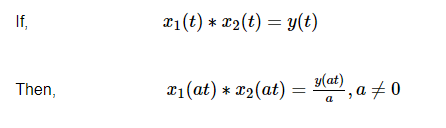
**Area**

If a signal is the result of convolution of two signals then the area of the signal is the multiplication of those individual signals. Mathematically this can be written



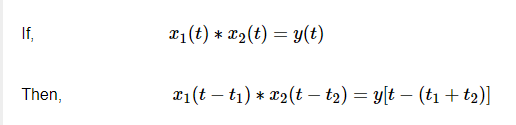
**Scaling**

If two signals are scaled to some unknown constant “a” and convolution is done then resultant signal will also be convoluted to same constant “a” and will be divided by that quantity as shown below.



**Delay**

Suppose a signal yt is a result from the convolution of two signals x1t and x2t. If the two signals are delayed by time t1 and t2 respectively, then the resultant signal yt will be delayed by t1+t2. Mathematically, it can be written as −



**Correlation**:

correlation describes the mutual relationship which exists between two or more things. The same definition holds good even in the case of signals. That is, correlation between signals indicates the measure up to which the given signal resembles another signal.

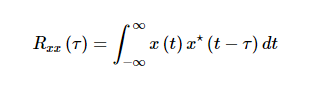
In other words, if we want to know how much similarity exists between the signals 1 and 2, then we need to find out the correlation of Signal 1 with respect to Signal 2 or vice versa.

1. **Types of Correlation**

Depending on whether the signals considered for correlation are same or different, we have two kinds of correlation: autocorrelation and cross-correlation.

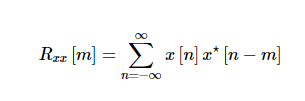
**Autocorrelation**

This is a type of correlation in which the given signal is correlated with itself, usually the time-shifted version of itself. Mathematical expression for the autocorrelation of continuous time signal *x* (*t*) is given by



where ⋆⋆ denotes the complex conjugate.

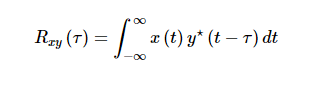
Similarly, the autocorrelation of the discrete time signal *x*[*n*] is expressed as



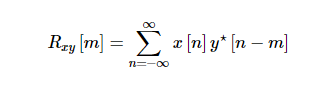
Next, the autocorrelation of any given signal can also be computed by resorting to graphical technique. The procedure involves sliding the time-shifted version of the given signal upon itself while computing the samples at every interval. That is, if the given signal is digital, then we shift the given signal by one sample every time and overlap it with the original signal. While doing so, for every shift and overlap, we perform multiply and add.

**Cross-Correlation**

This is a kind of correlation, in which the signal in-hand is correlated with another signal so as to know how much resemblance exists between them. Mathematical expression for the cross-correlation of continuous time signals *x* (*t*) and *y* (*t*) is given by



Similarly, the cross-correlation of the discrete time signals x [n] and y [n] is expressed as



**LAB TASKS**

**Example 1: convolution**

**MATLAB CODE:**

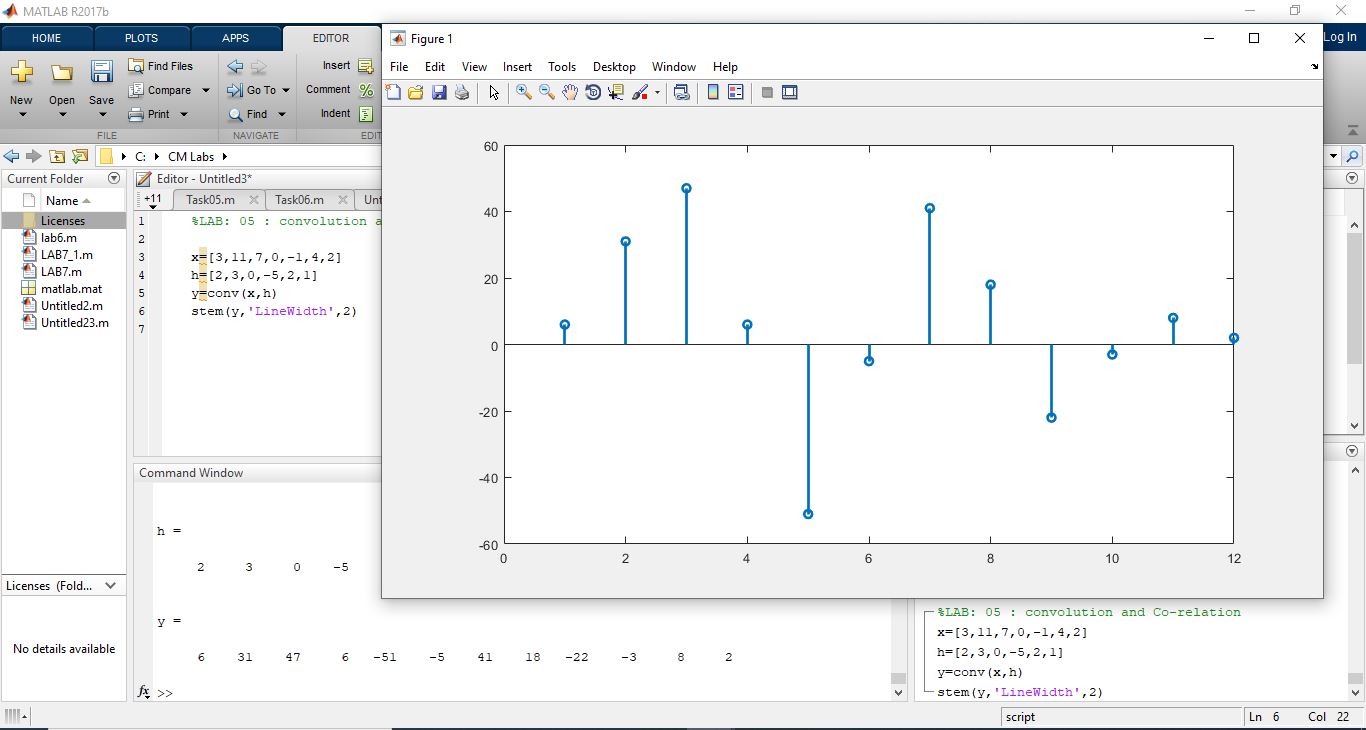
x=[3,11,7,0,-1,4,2]

h=[2,3,0,-5,2,1]

y=conv(x,h)

stem(y,'LineWidth',2)

**OUTPUT:**



**Example 2:**

**MATLAB CODE:**

clear all

close all

h = [1 2 3 4 5 4 3 2 1];

x = sin(0.2\*pi\*[0:20]);

y = conv(h, x);

figure(1);

stem (x);

title('Discrete Filter Input x[n]');

xlabel('index, n');

ylabel('Value, x[n]');

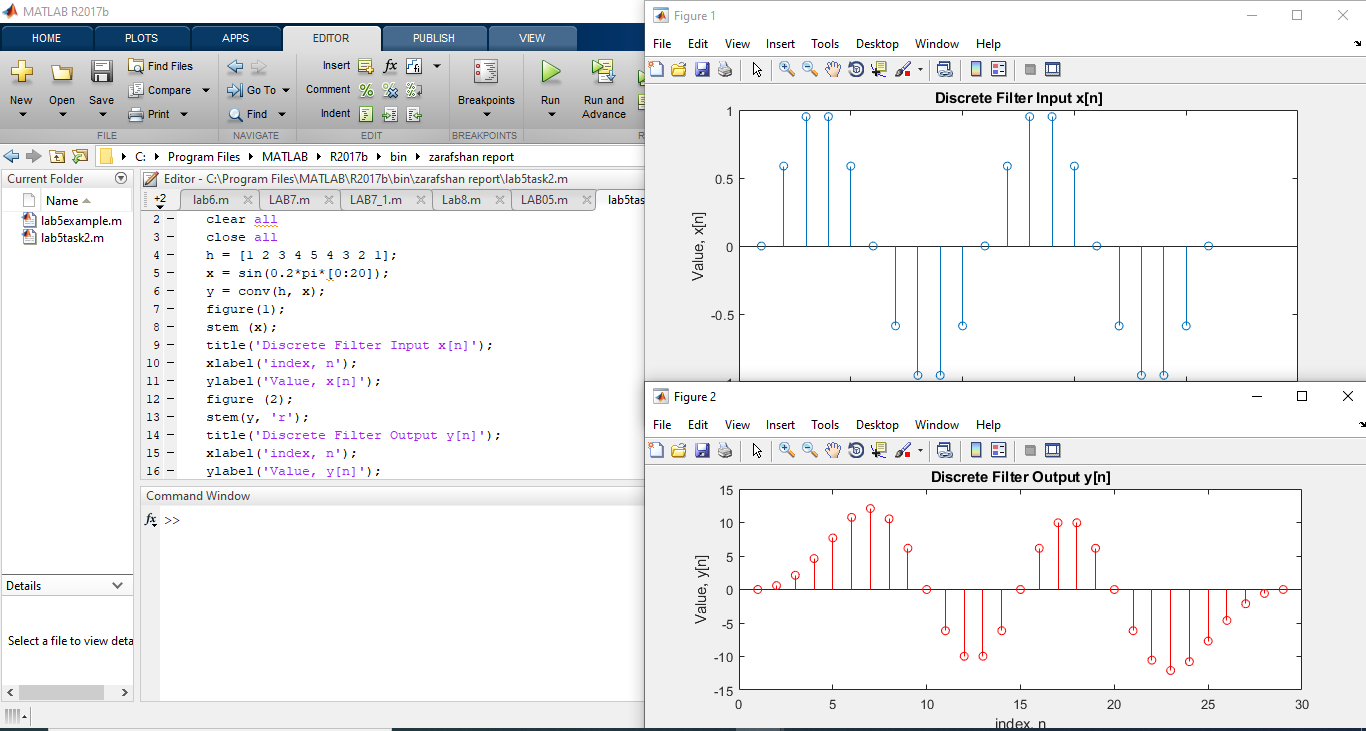
figure (2);

stem(y, 'r');

title('Discrete Filter Output y[n]');

xlabel('index, n');

ylabel('Value, y[n]');



**Example 03:**

**MATLAB CODE:**

% n= 0 1 2 3 4 5 6 7 8

% x (n) =-4 2 -1 3 -2 -6 -5 4 5

% h(x) =-4 1 3 7 4 -2 -8 -2 -1

h=input('Enter value of h ');

x=input('Enter value oh x ');

c=conv(h,x);

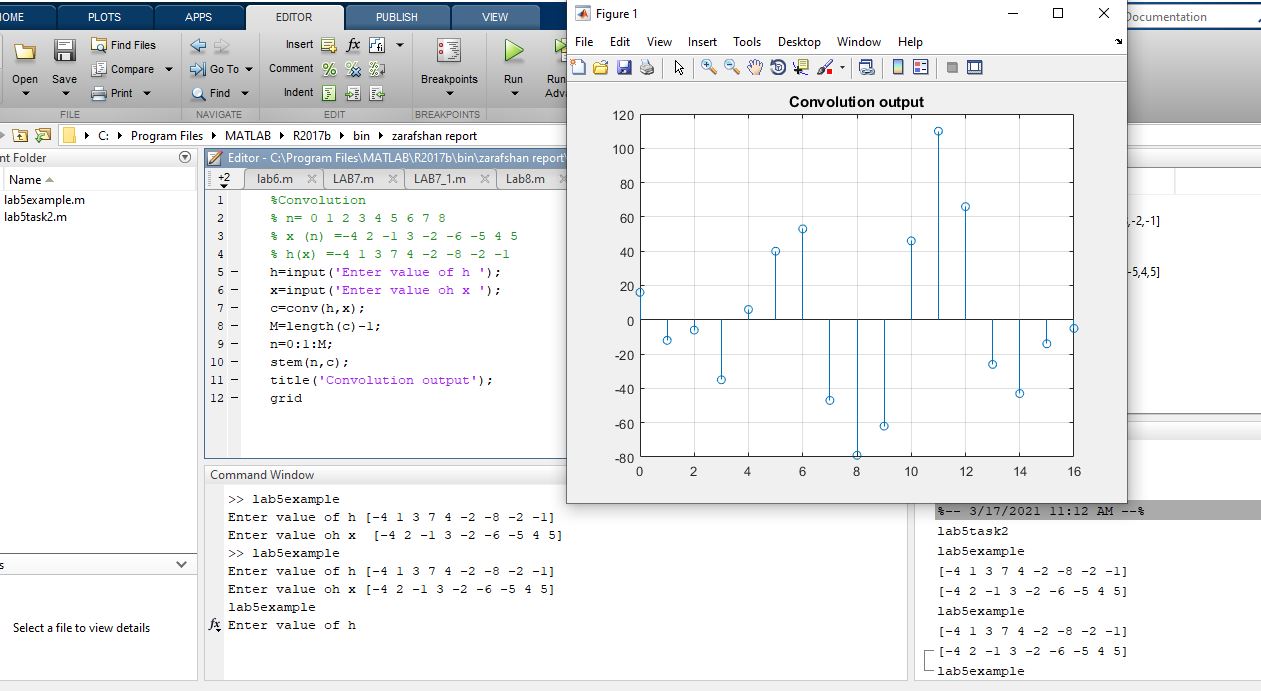
M=length(c)-1;

n=0:1:M;

stem(n,c);

title('Convolution output');

grid

****

**Lab # 06 Solution of Difference Equations**

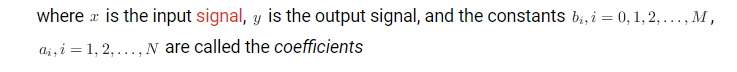
**Objective:**

* Solve Ordinary Differential Equations (ODEs).
* Appreciate the importance of numerical methods in solving ODEs.
* Assess the reliability of the different techniques.
* Select the appropriate method for any particular problem.

**Theory:**

The difference equation is a formula for computing an output sample at time $ n$ based on past and present input samples and past output samples in the time domain.[6.1](https://www.dsprelated.com/freebooks/filters/footnode.html" \l "foot9508)We may write the general, [causal](http://www.dsprelated.com/dspbooks/filters/Causal_Recursive_Filters.html), [LTI](http://www.dsprelated.com/dspbooks/filters/Linear_Time_Invariant_Digital_Filters.html) difference equation as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| $\displaystyle y(n)$ | $\displaystyle =$ | $\displaystyle b_0 \,x(n) + b_1 \,x(n - 1) + \cdots + b_M \,x(n - M)$ |  |
|  |  | $\displaystyle \qquad\quad\; - a_1 \,y(n - 1) - \cdots - a_N \,y(n - N)$ |  |
|  | $\displaystyle =$ | $\displaystyle \sum_{i=0}^M b_i \,x(n-i) - \sum_{j=1}^N a_j \,y(n-j) \protect$ | (6.1) |



As a specific example, the difference equation

$\displaystyle y(n) = 0.01\, x(n) + 0.002\, x(n - 1) + 0.99\, y(n - 1)
$

specifies a [digital filtering](http://www.dsprelated.com/dspbooks/filters/) operation, and the coefficient sets $ (0.01, 0.002)$ and $ (0.99)$ fully characterize the [filter](http://www.dsprelated.com/dspbooks/filters/What_Filter.html). In this example, we have $ M = N = 1$. When the coefficients are [real numbers](http://mathworld.wolfram.com/RealNumber.html), as in the above example, the filter is said to be real. Otherwise, it may be complex. Notice that a filter of the form of Eq.$ \,$([5.1](https://www.dsprelated.com/freebooks/filters/Difference_Equation_I.html#eq:tpnine)) can use ``past'' output samples (such as $ y(n-1)$) in the calculation of the ``present'' output $ y(n)$. This use of past output samples is called feedback. Any filter having one or more feedback paths ($ N>0$) is called recursive. (By the way, the minus signs for the feedback in Eq.$ \,$([5.1](https://www.dsprelated.com/freebooks/filters/Difference_Equation_I.html#eq:tpnine)) will be explained when we get to [transfer functions](http://www.dsprelated.com/dspbooks/filters/Transfer_Function_Analysis.html) in §[6.1](https://www.dsprelated.com/freebooks/filters/Z_Transform.html#sec:zt).) More specifically, the $ b_i$ coefficients are called the feedforward coefficients and the $ a_i$ coefficients are called the feedback coefficients. A filter is said to be recursive if and only if $ a_i\neq 0$ for some $ i>0$. Recursive filters are also called infinite-[impulse-response](http://www.dsprelated.com/dspbooks/filters/Impulse_Response_Representation.html) (IIR) filters. When there is no feedback ( $ a_i=0, \forall i>0$), the filter is said to be a non-recursive or finite-[impulse](http://www.dsprelated.com/dspbooks/filters/Impulse_Response_Representation.html)-response (FIR) digital filter. When used for discrete-time [physical modeling](http://en.wikipedia.org/wiki/Model_(physical)), the difference equation may be referred to as an explicit [finite difference scheme](http://www.dsprelated.com/dspbooks/pasp/Finite_Difference_Schemes.html).[6.2](https://www.dsprelated.com/freebooks/filters/footnode.html" \l "foot9509) Showing that a recursive filter is LTI (Chapter [4](https://www.dsprelated.com/freebooks/filters/Linear_Time_Invariant_Digital_Filters.html#chap:theory)) is easy by considering its impulse-response representation (discussed in §[5.6](https://www.dsprelated.com/freebooks/filters/Impulse_Response_Representation.html#sec:imprep)). For example, the recursive filter.

**LAB TASKS:**

**EXAMPLE 01;**

**MATLAB CODE:**

% Example 1

% y(n) +2y(n-1)+3y(n-2) = X(n)+3X(n-1) + X(n-2)

x=[1,2,3,2,1]

a=[1,2,3]

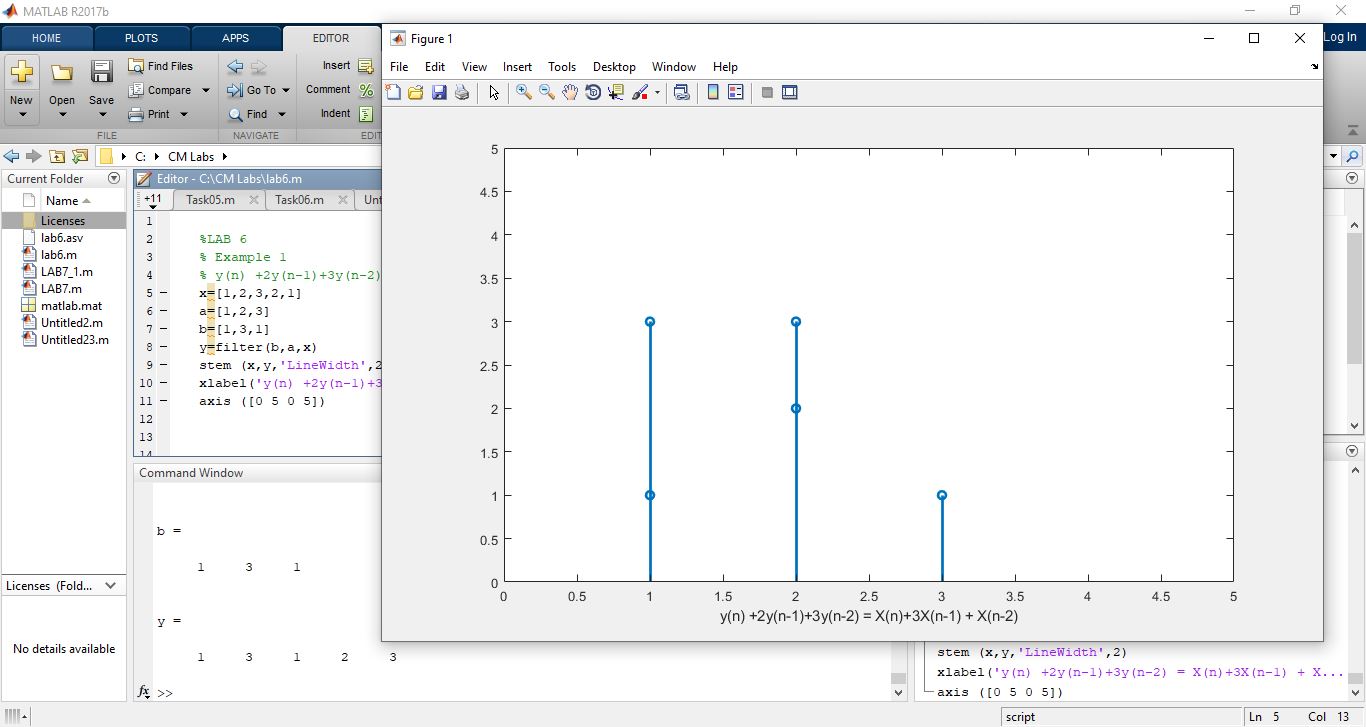
b=[1,3,1]

y=filter(b,a,x)

stem (x,y,'LineWidth',2)

xlabel('y(n) +2y(n-1)+3y(n-2) = X(n)+3X(n-1) + X(n-2)');

axis ([0 5 0 5])

****

**EXAMPLE 02;**

%Example 2

% y(n)=1/7 [x(n)+x(n-2)+x(n-3)+x(n-4)+x(n-5)+x(n-6)]

%

x=[0,2,4,8,4,2]

a=[1]

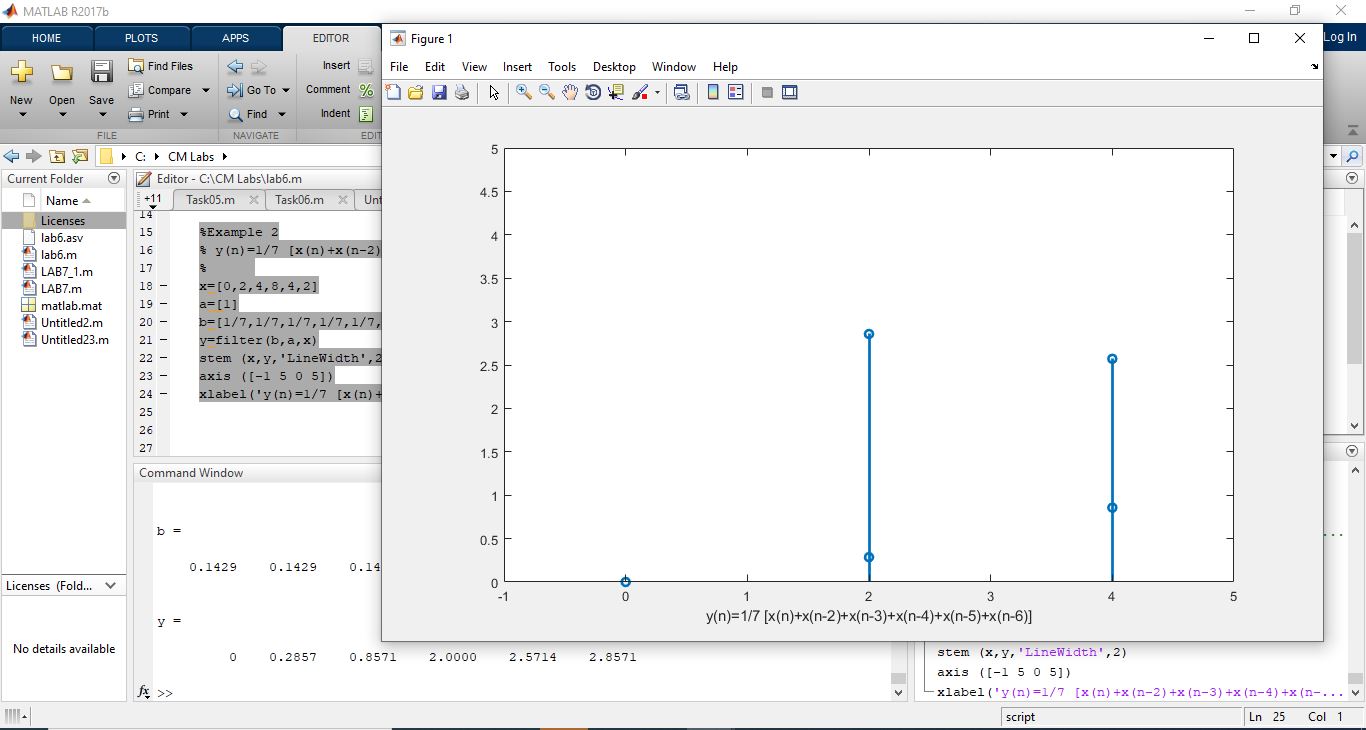
b=[1/7,1/7,1/7,1/7,1/7,1/7,1/7]

y=filter(b,a,x)

stem (x,y,'LineWidth',2)

axis ([-1 5 0 5])

xlabel('y(n)=1/7 [x(n)+x(n-2)+x(n-3)+x(n-4)+x(n-5)+x(n-6)]');

****

**EXAMPLE 03;**

**MATLAB CODE;**

%2y(n) +2y(n-1)+3y(n-2) = 3X(n)+1X(n-1) - 3X(n-2)

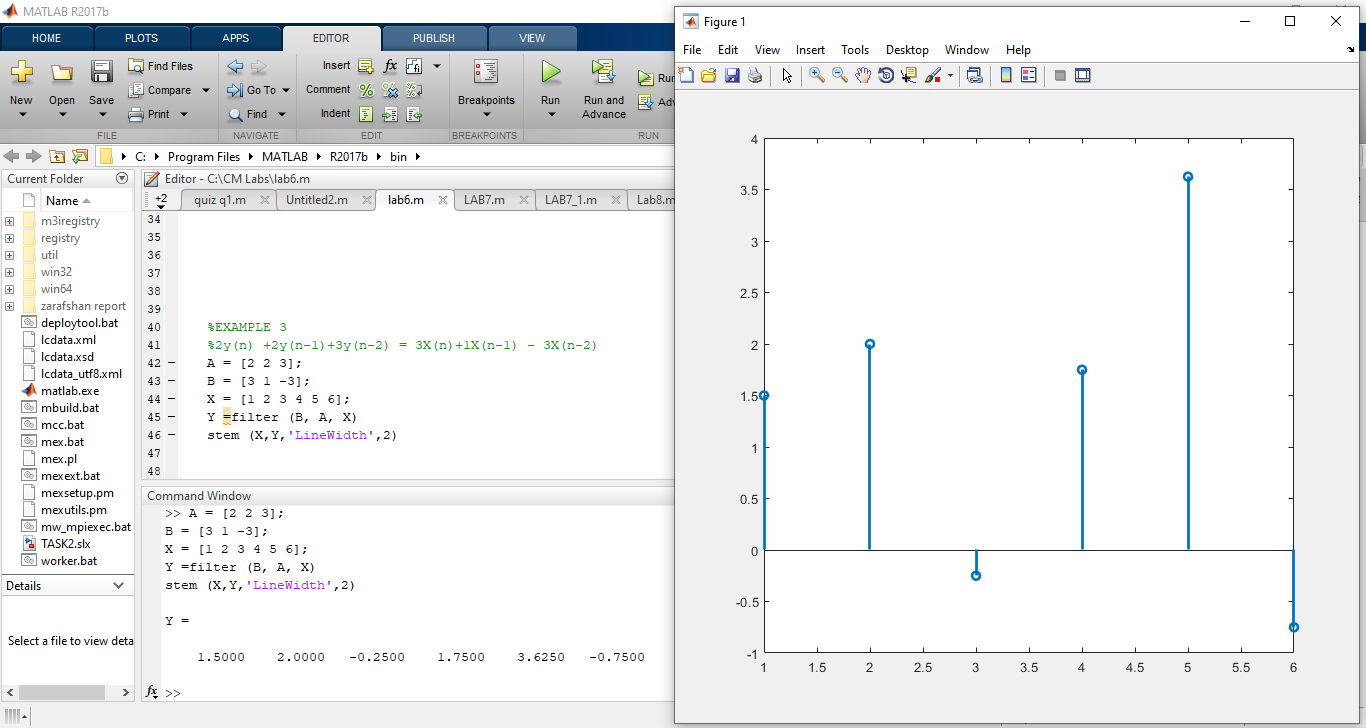
A = [2 2 3];

B = [3 1 -3];

X = [1 2 3 4 5 6];

Y =filter (B, A, X)

stem (X,Y,'LineWidth',2)



**Lab # 07 Z Transform in MATLAB**

**Objective:**

* To understand the properties of Z-Transform and associating the knowledge of properties of ROC in response to different operations on discrete signals.

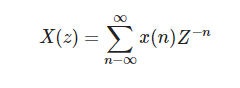
**Theory:**

**Z-transform:**

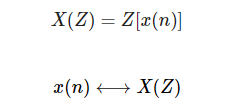
the Z-transform converts a discrete-time signal, which is a sequence of real or complex numbers, into a complex frequency-domain representation.

Discrete Time Fourier Transform DTFT  exists for energy and power signals. Z-transform also exists for neither energy nor Power NENPNENP type signal, up to a certain extent only. The replacement  is used for Z-transform to DTFT conversion only for absolutely summable signal.

So, the Z-transform of the discrete time signal  in a power series can be written as −



The above equation represents a two-sided Z-transform equation. Generally, when a signal is Z-transformed, it can be represented as –



If it is a continuous time signal, then Z-transforms are not needed because Laplace transformations are used. However, Discrete time signals can be analyzed through Z-transforms only.

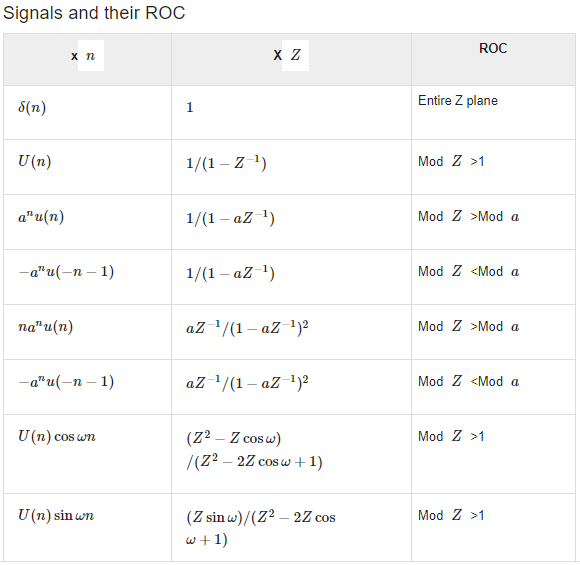
* **Region of Convergence**

Region of Convergence is the range of complex variable Z in the Z-plane. The Z- transformation of the signal is finite or convergent. So, ROC represents those set of values of Z, for which XZZ has a finite value.

* **Properties of ROC**
* ROC does not include any pole.
* For right-sided signal, ROC will be outside the circle in Z-plane.
* For left sided signal, ROC will be inside the circle in Z-plane.
* For stability, ROC includes unit circle in Z-plane.
* For Both sided signal, ROC is a ring in Z-plane.
* For finite-duration signal, ROC is entire Z-plane.

The Z-transform is uniquely characterized by −

* Expression of XZZ
* ROC of XZ



**LAB TASKS:**

**EXAMPLE 01**

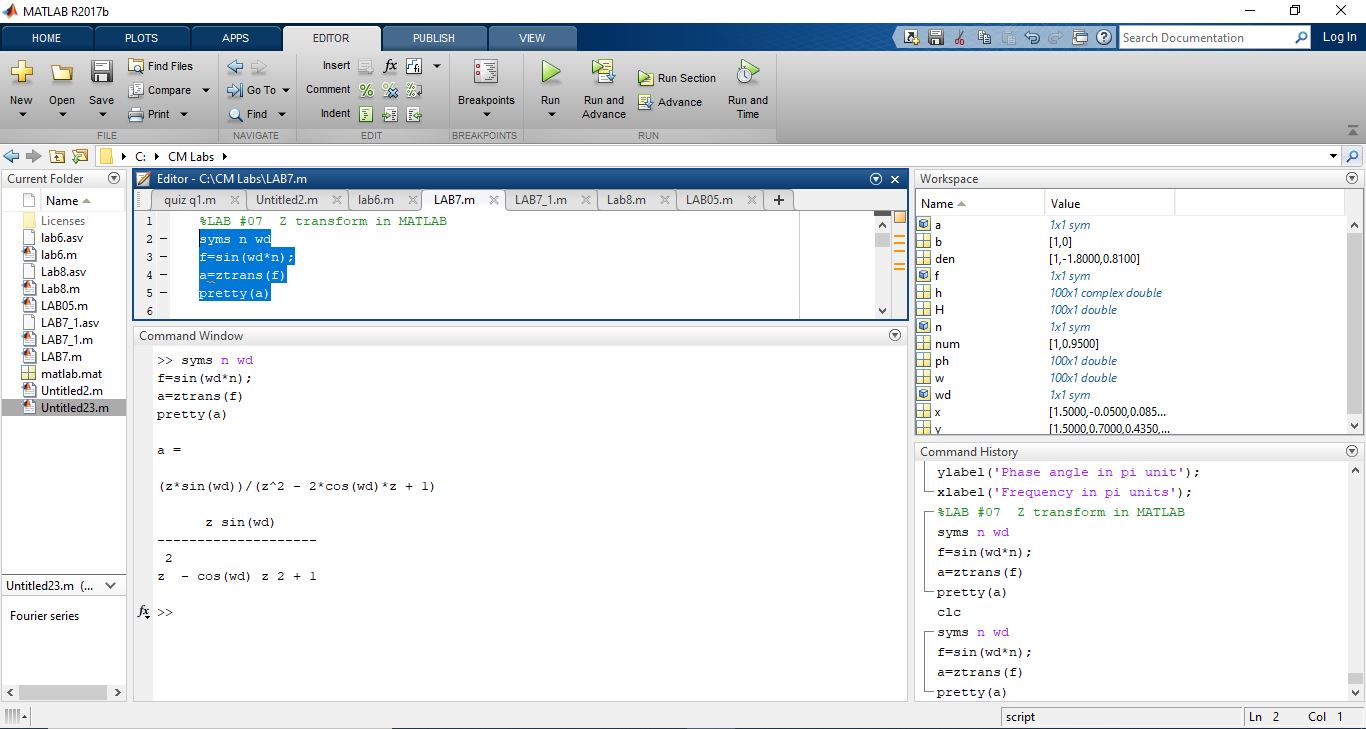
**MATLAB CODE;**

syms n wd

f=sin(wd\*n);

a=ztrans(f)

pretty(a)



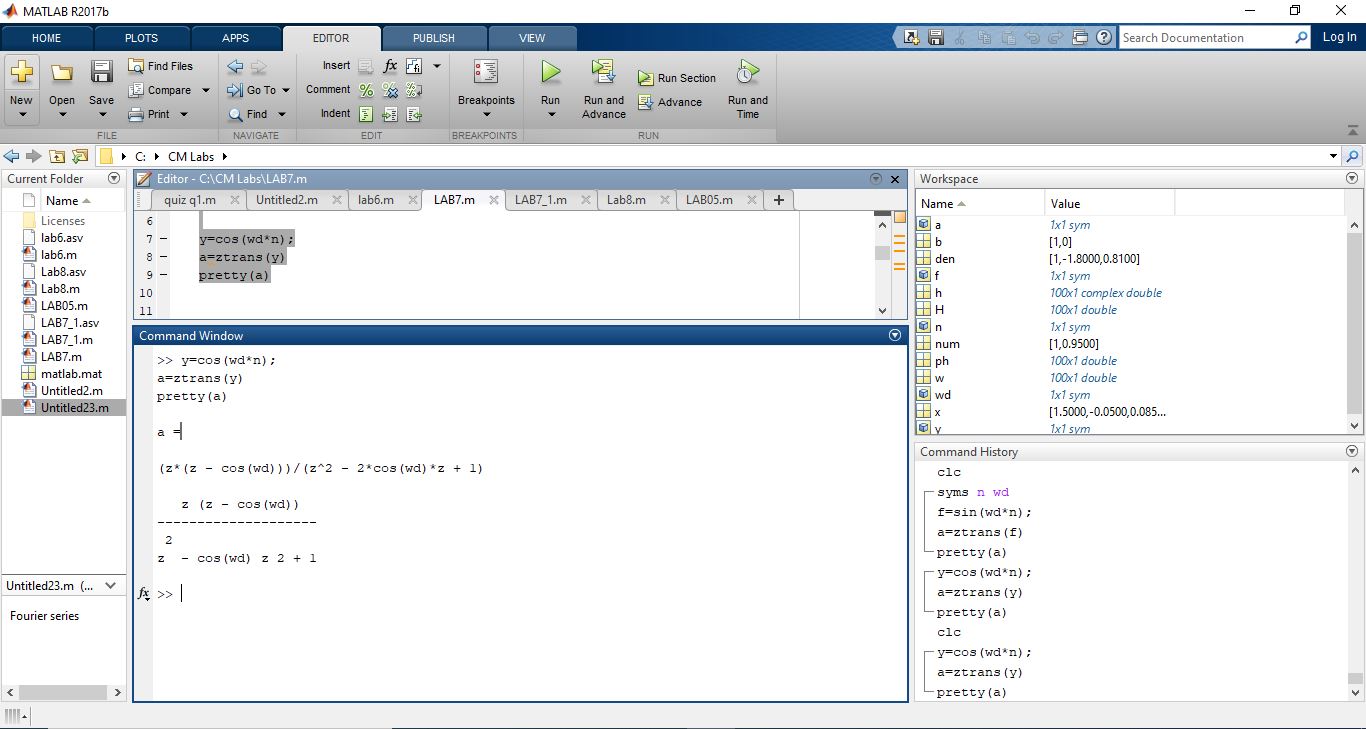
**Example 02**

**MATLAB CODE;**

y=cos(wd\*n);

a=ztrans(y)

pretty(a)



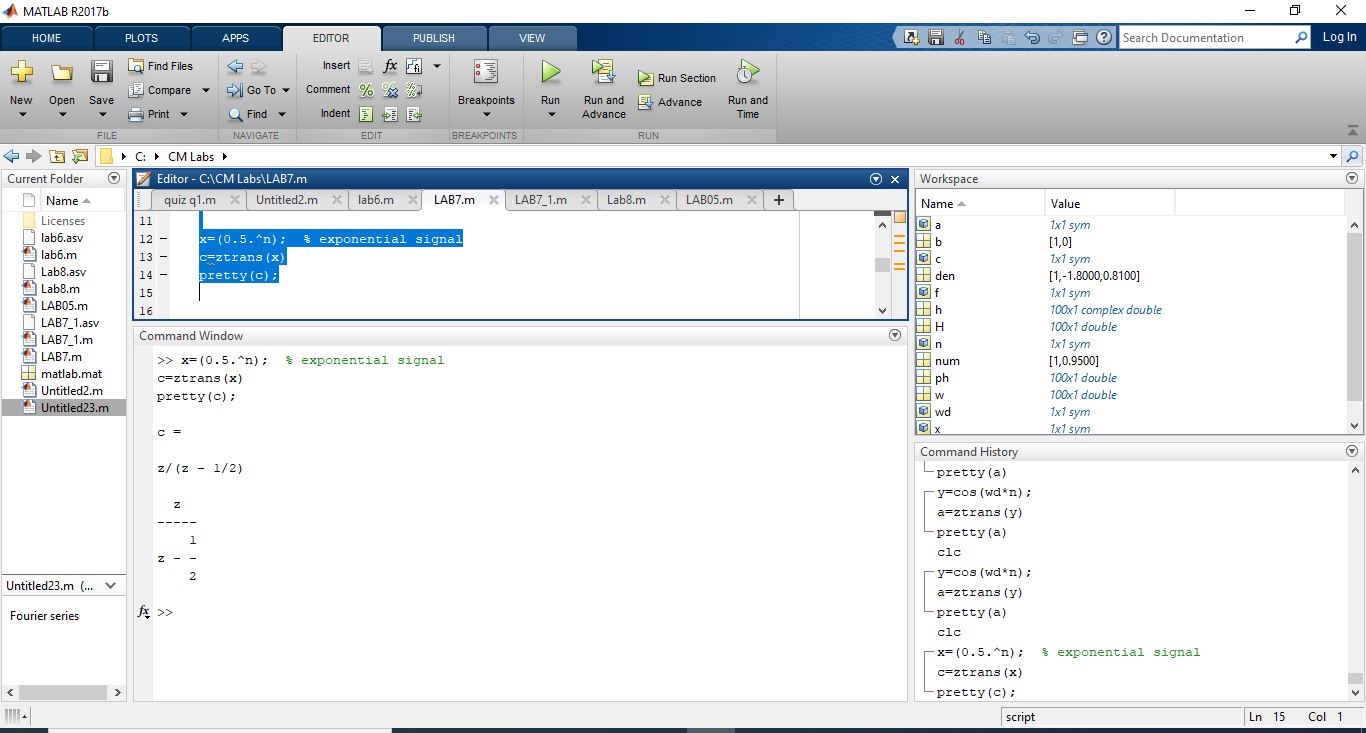
**Example 03**

**MATLAB CODE;**

x=(0.5.^n); % exponential signal

c=ztrans(x)

pretty(c);

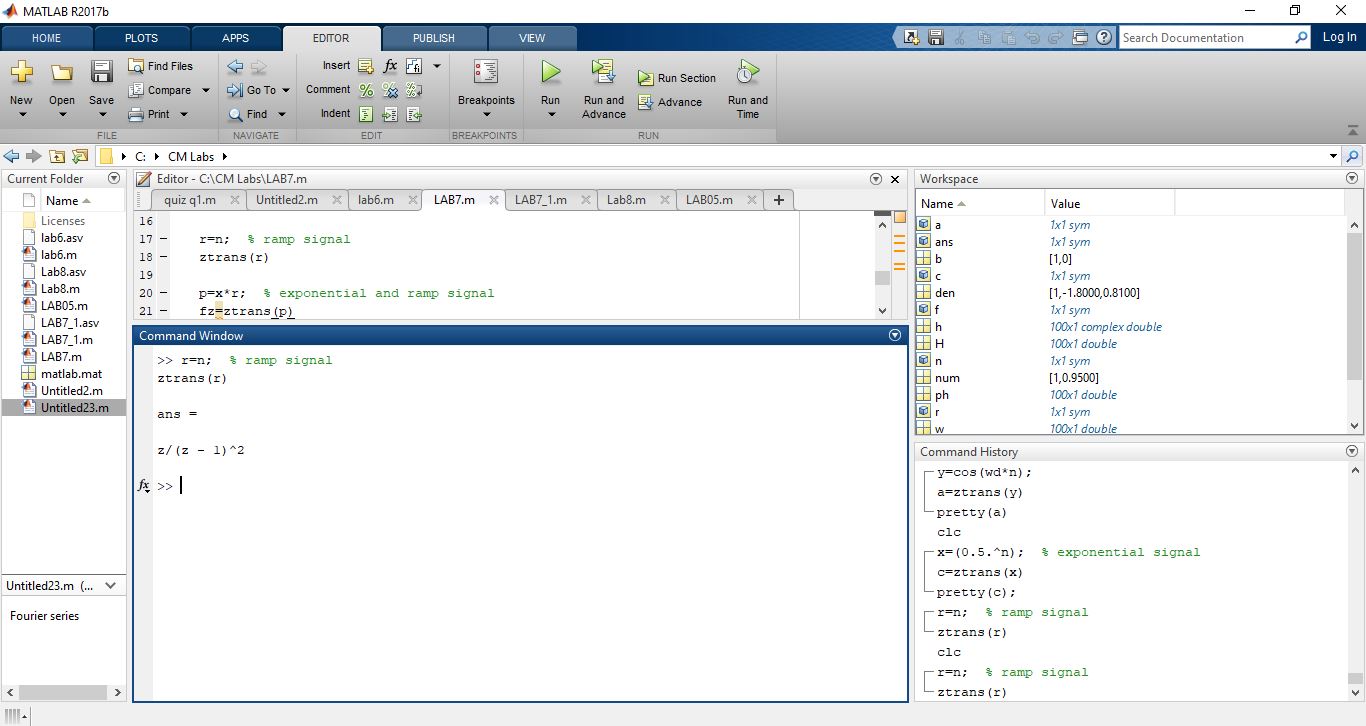


**Example 04**

**MATLAB CODE;**

r=n; % ramp signal

ztrans(r)



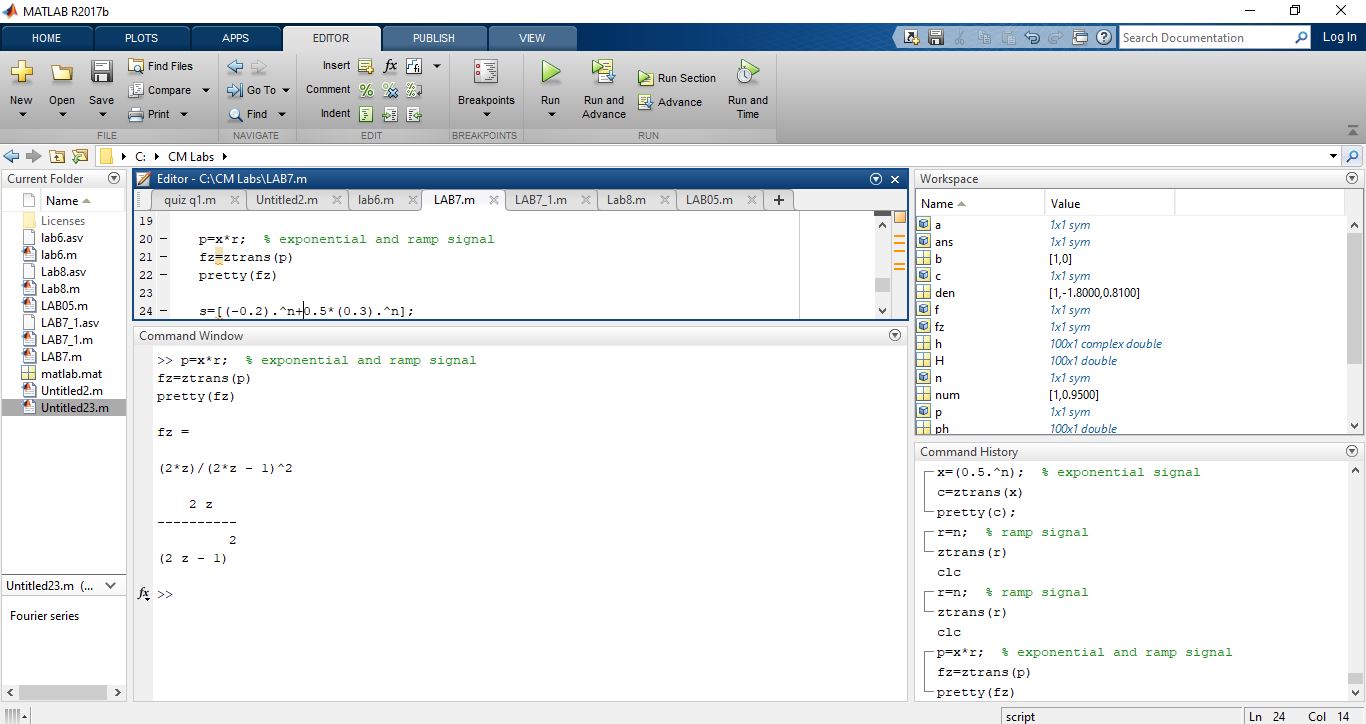
**Example 05**

**MATLAB CODE;**

p=x\*r; % exponential and ramp signal

fz=ztrans(p)

pretty(fz)



**Example 05**

**MATLAB CODE;**

% X(z)= z

% ------------

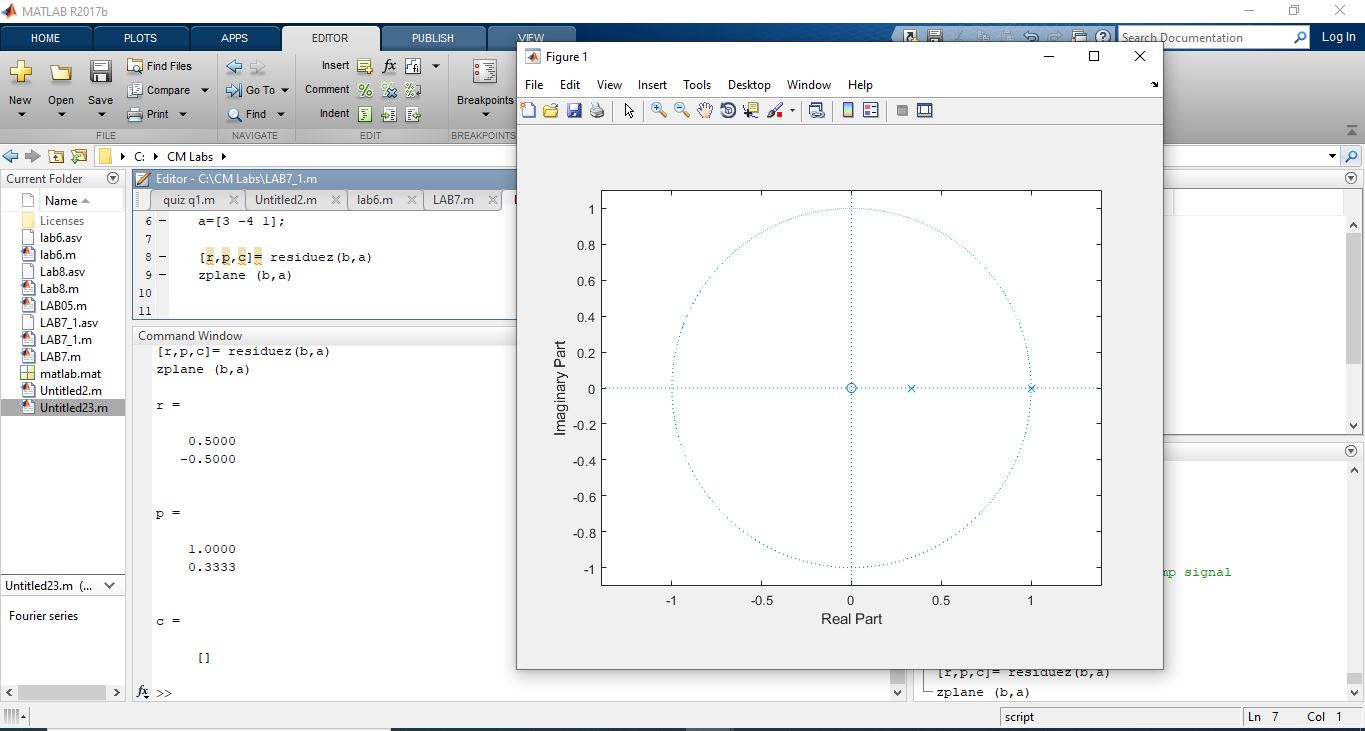
% 3 z^2 -4z + 1

b=[0 1];

a=[3 -4 1];

[r,p,c]= residuez(b,a)

zplane (b,a)



**Example 06**

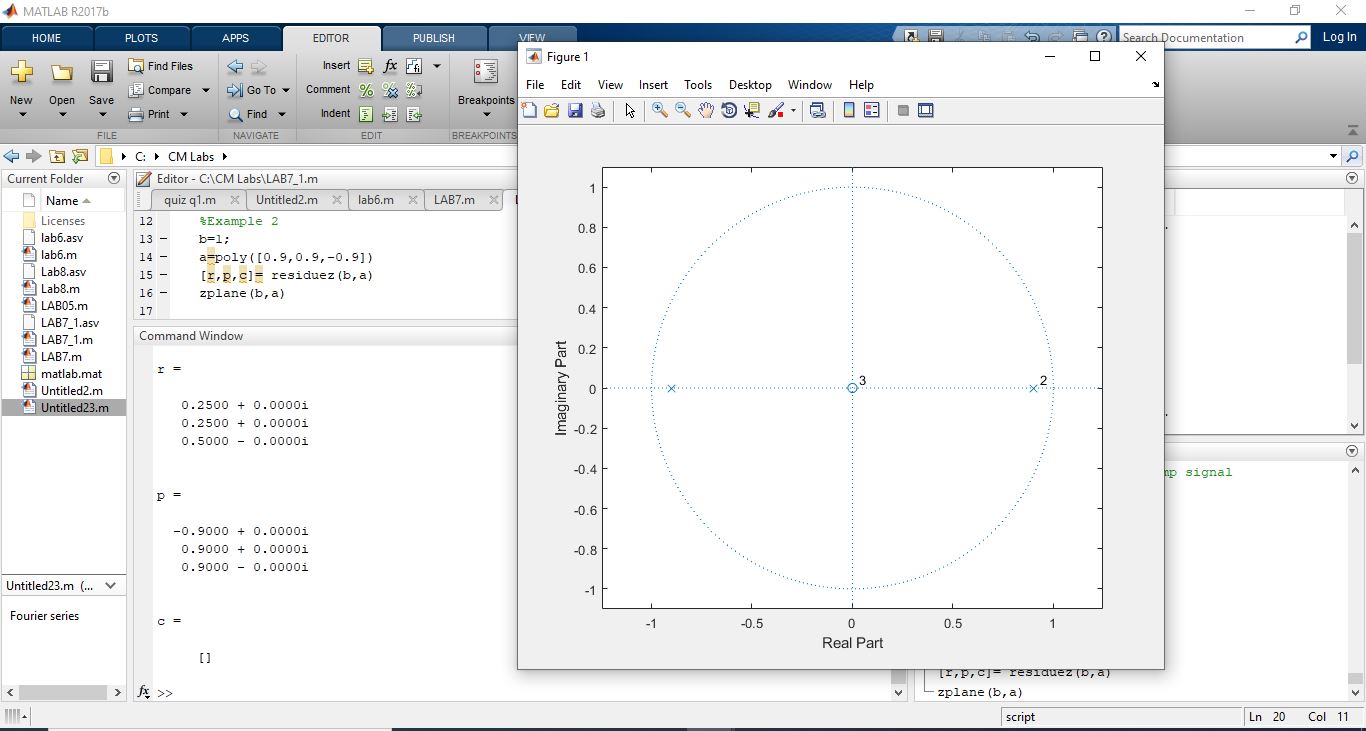
**MATLAB CODE;**

b=1;

a=poly([0.9,0.9,-0.9])

[r,p,c]= residuez(b,a)

zplane(b,a)



**Example 07**

**MATLAB CODE;**

b=[1 0.4\*sqrt(2)];

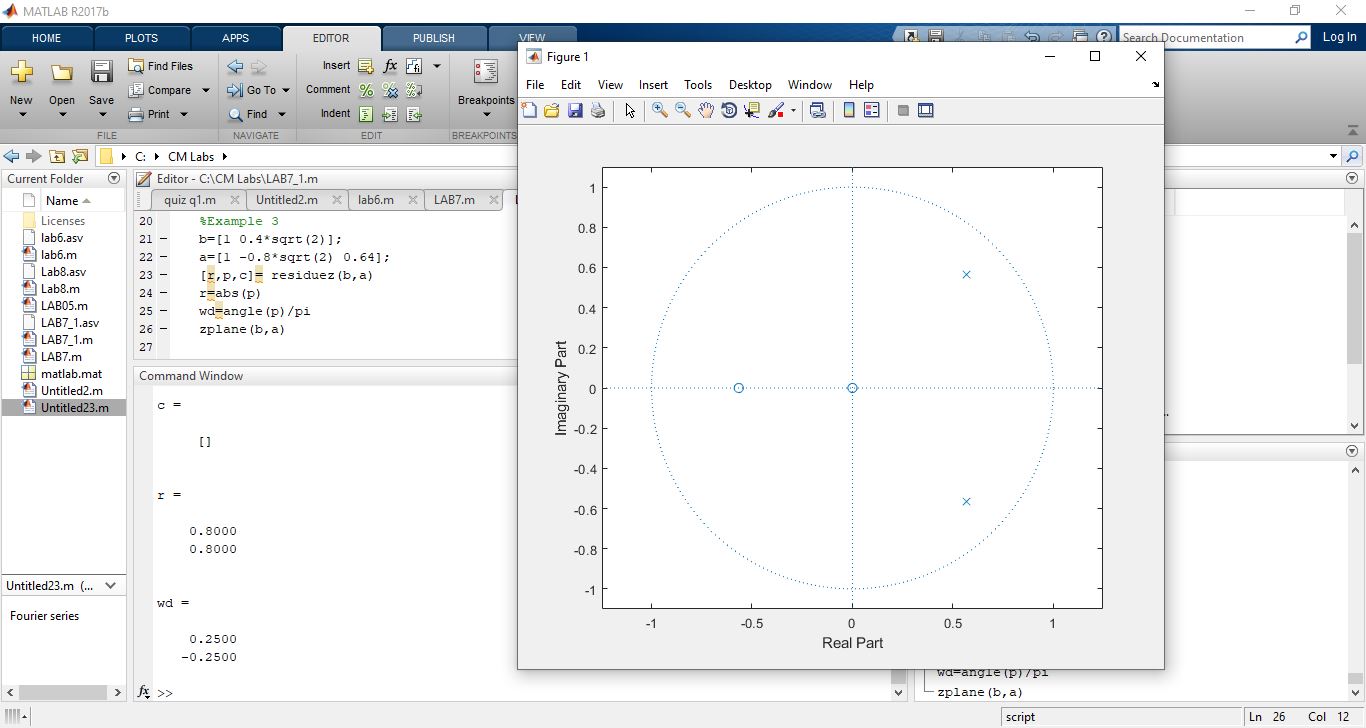
a=[1 -0.8\*sqrt(2) 0.64];

[r,p,c]= residuez(b,a)

r=abs(p)

wd=angle(p)/pi

zplane(b,a)



**Example 08**

**MATLAB CODE;**

n=(0:9)

x=((-0.2).^n +0.5\*(0.3).^n)

% y(n)=x(n)+0.5y(n-1)

% H(z)= 1

% ------

% 1-0.5z^-1

b=1;

a=[1 -0.5];

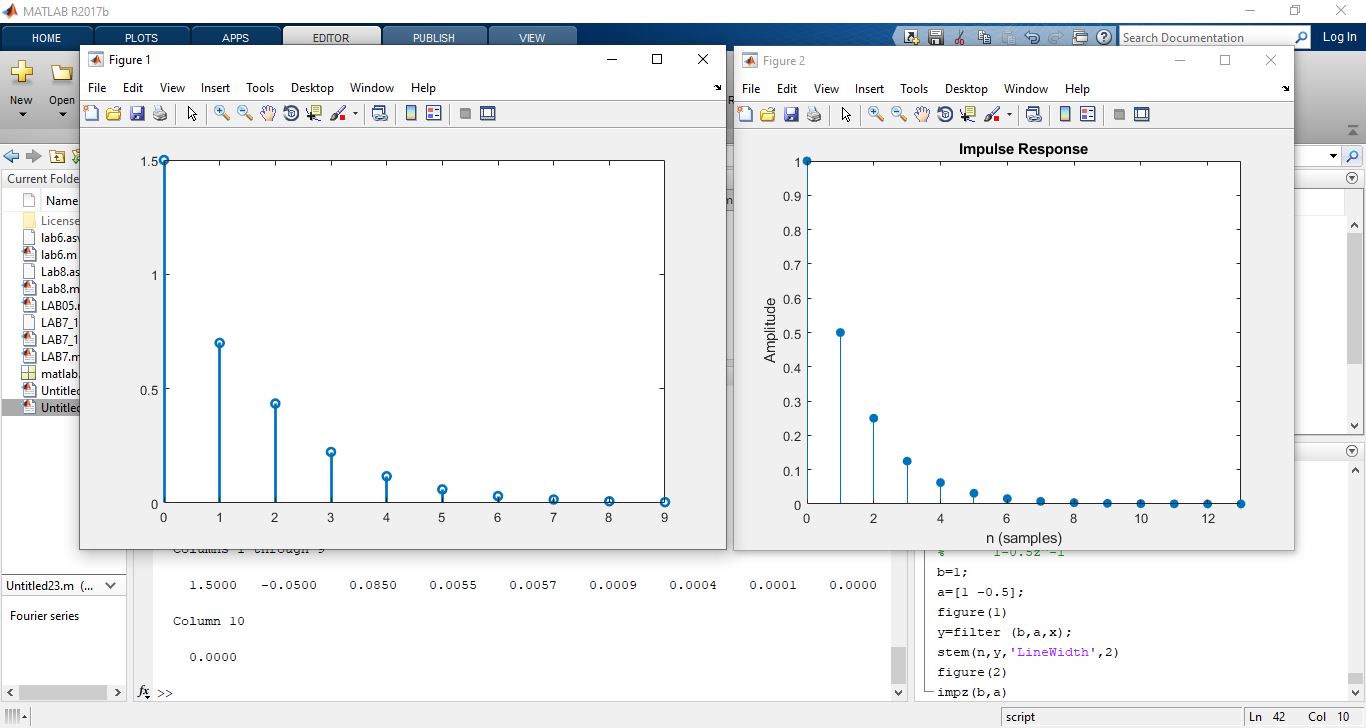
figure(1)

y=filter (b,a,x);

stem(n,y,'LineWidth',2)

figure(2)

impz(b,a)



**Lab # 08 Frequency Response of LTI System**

**Objective:**

* To check frequency response of LTI System
* Find magnitude and phase of the given system

**Theory:**

**FREQUENCY RESPONSE OF A SYSTEM:**

The response of a system can be partitioned into both the transient response and the steady state response. We can find the transient response by using Fourier integrals. The steady state response of a system for an input sinusoidal signal is known as the **frequency response**. In this chapter, we will focus only on the steady state response.

If a sinusoidal signal is applied as an input to a Linear Time-Invariant (LTI) system, then it produces the steady state output, which is also a sinusoidal signal. The input and output sinusoidal signals have the same frequency, but different amplitudes and phase angles.

Let the input signal be −

r(t)=Asin(ω0t)

The open loop transfer function will be −

G(s)=G(jω)G(s)=G(jω)

We can represent G(jω)G(jω) in terms of magnitude and phase as shown below.

G(jω)=|G(jω)|∠G(jω)G(jω)=|G(jω)|∠G(jω)

Substitute, ω=ω0ω=ω0 in the above equation.

G(jω0)=|G(jω0)|∠G(jω0)G(jω0)=|G(jω0)|∠G(jω0)

The output signal is

c(t)=A|G(jω0)|sin(ω0t+∠G(jω0))c(t)=A|G(jω0)|sin⁡(ω0t+∠G(jω0))

* The **amplitude** of the output sinusoidal signal is obtained by multiplying the amplitude of the input sinusoidal signal and the magnitude of G(jω)G(jω) at ω=ω0ω=ω0.
* The **phase** of the output sinusoidal signal is obtained by adding the phase of the input sinusoidal signal and the phase of G(jω)G(jω) at ω=ω0ω=ω0.

Where,

* **A** is the amplitude of the input sinusoidal signal.
* **ω0** is angular frequency of the input sinusoidal signal.

We can write, angular frequency ω0ω0 as shown below.

ω0=2πf0ω0=2πf0

Here, f0f0 is the frequency of the input sinusoidal signal. Similarly, you can follow the same procedure for closed loop control system.

**Complex Exponential:**

The relation between complex exponentials and sinusoids is captured by Euler’s famous identity: ejφ = cosφ + j sinφ .

where j = −1. ejφ represents a complex number(or a point in the complex plane) that has a real component of cosφ and an imaginary component of sinφ. It therefore has magnitude 1 (because cos2 φ + sin2 φ = 1), and makes an angle of φ with the positive real axis. In other words, ejφ is the point on the unit circle in the complex plane (i.e., at radius 1 from the origin) and at an angle of φ relative to the positive real axis. A short refresher on complex numbers may be worthwhile. The complex number c = a + jb can be thought of as the point (a, b) in the plane, √ and accordingly has magnitude |c| = a2 + b2 and angle with the positive real axis of ∠c = arctan(b/a). Note that a = |c| cos(∠c) and b = |c|sin(∠c). Hence, in view of Euler’s identity, we can also write the complex number in so-called polar form, c = |c|.ej∠c; this represents a point at distance |c| from the origin, at an angle of ∠c.

**LAB TASKS:**

**EXAMPLE 01;**

**MATLAB CODE:**

% H(z)=(1+0.95z-1)/(1-1.8z-1+0.81z-2)

num=[1 0.95];

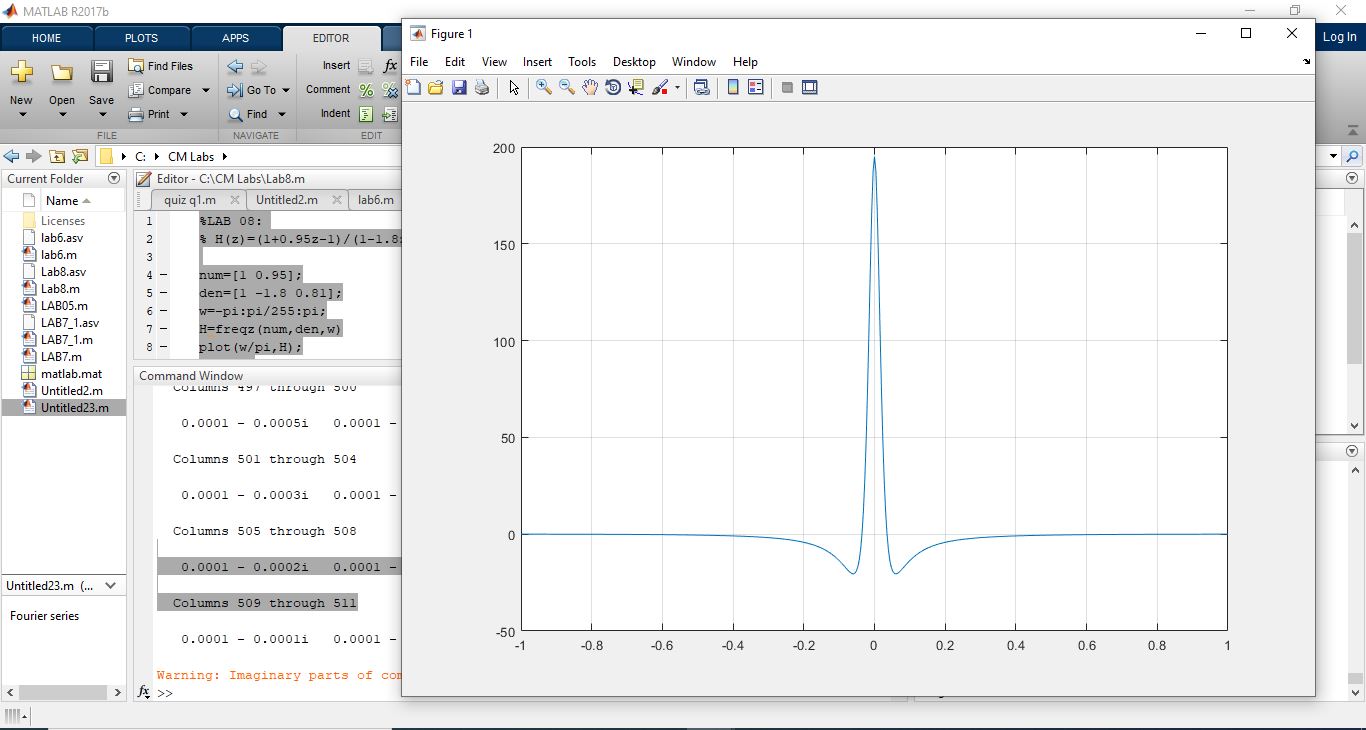
den=[1 -1.8 0.81];

w=-pi:pi/255:pi;

H=freqz(num,den,w)

plot(w/pi,H);

grid on

****

**EXAMPLE 02;**

**MATLAB CODE;**

%H(z)= 1

% -------

% 1-0.9z^1

b=[1 0];

a=[1 -0.9];

[h w]=freqz(b,a,100);

H=abs(h);

ph=angle(h);

subplot(2,1,1)

plot(w/pi,H); grid

title('Magnitude response of the filter');

ylabel('Magnitude')

xlabel('frequecny in pi units')

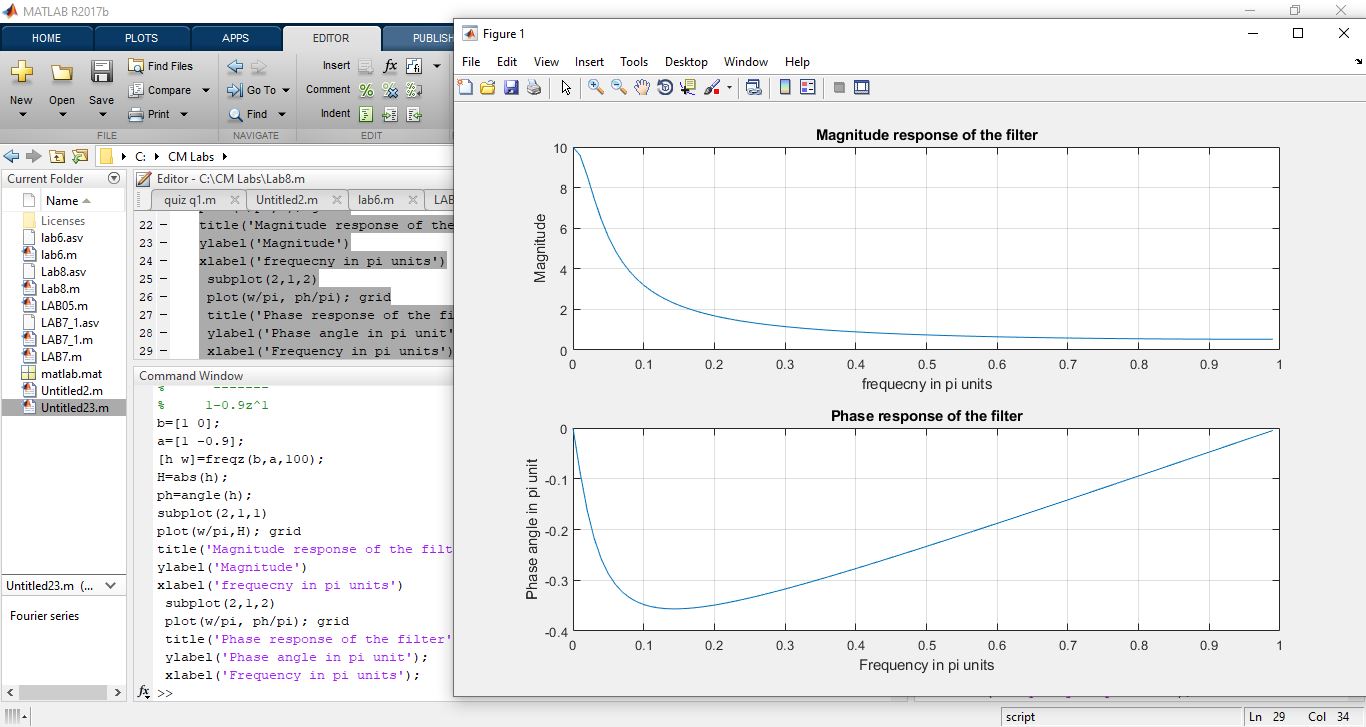
subplot(2,1,2)

plot(w/pi, ph/pi); grid

title('Phase response of the filter')

ylabel('Phase angle in pi unit');

xlabel('Frequency in pi units');

****

**EXAMPLE 03;**

**MATLAB CODE**

%EXAMPLE 3

clc

close all

b = [1 2 1];

a = 1;

w = -3\*pi: 1/100:3\*pi;

H = freqz(b, a, w);

subplot(3,1,1);

plot(w, abs(H), 'LineWidth', 2);

title('Magnitude of Frequency Response of filter with coefficients bk={1, 2, 1}');

xlabel('\omega');

ylabel('H(\omega)');

grid;

subplot(3,1,2);

plot(w, abs(H), 'r ', 'LineWidth', 2);

title('Zoomed view of the above graph from ?\pi to \pi');

xlabel('\omega');

ylabel('H(\omega)');

%axis([-pi pi min(H) max(H)+0.5]);

grid;

subplot(3,1,3);

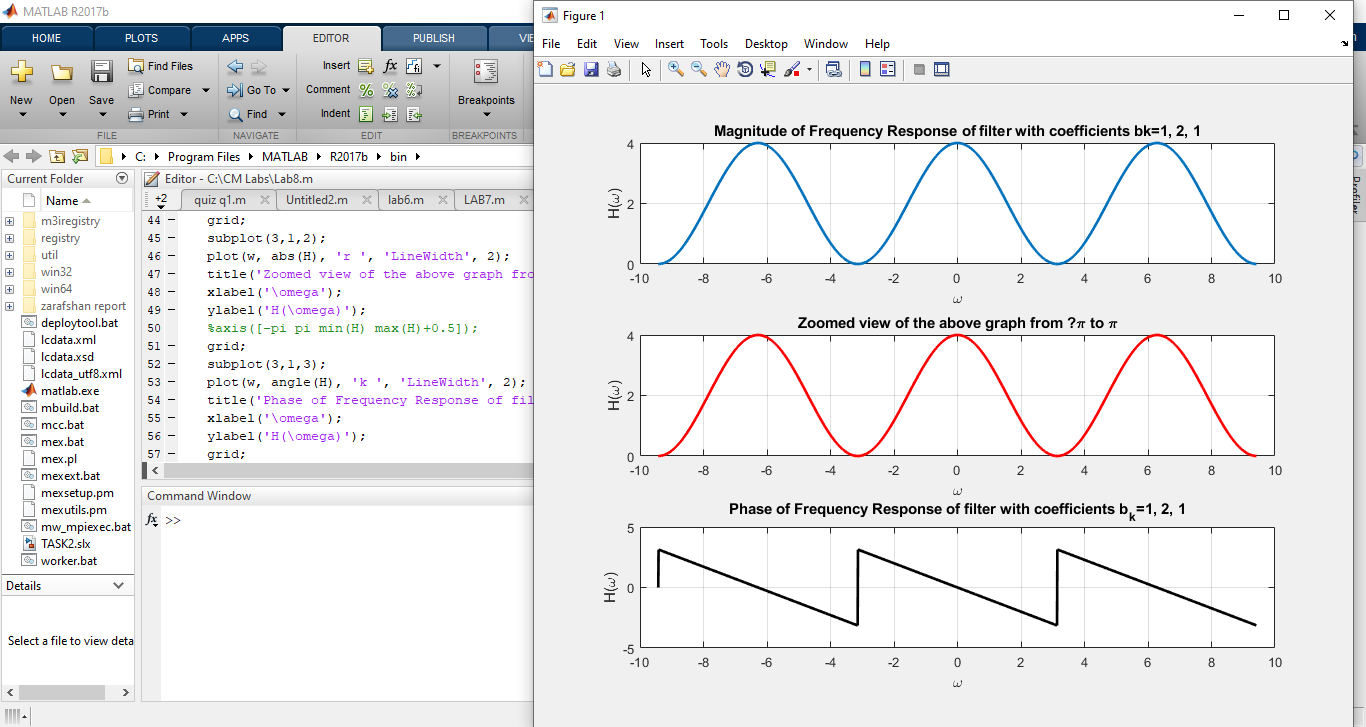
plot(w, angle(H), 'k ', 'LineWidth', 2);

title('Phase of Frequency Response of filter with coefficients b\_k={1, 2, 1}');

xlabel('\omega');

ylabel('H(\omega)');

grid;



**Lab # 09 DTFS and DTFT**

**Objective:**

* Write a program to calculate the DTFS and DTFT of a discrete-time signal.
* Use the Matlab commands fft and ifft to find the DTFS and DTFT coefficients X[k] and IDTFS and DTFT of a discrete-time signal x[n].

**Theory:**

**DTFS:**

Discrete time signals are fundamentally different from countinuous time signals in that they only exist at discrete instances of time and are undefined elsewhere. This introduces some quirks that are present in analysis of CT signals.

But first let's see why expressing signals as linear combinations of exponentials is useful. The explanation below also holds for CT signals & systems , but is particularly easy to understand for DT systems.

Consider an linear shift-invariant (LSI) system with impulse response h[n]h[n].

The output of the system to any input s[n]s[n] can be written as

y[n]=∑k=−∞∞h[k]s[n−k]y[n]=∑k=−∞∞h[k]s[n−k]

Let s[n]=ejωns[n]=ejωn -- a complex exponential

We can write:

y[n]=∑k=−∞∞h[k]ejω(n−k)=ejωn∑k=−∞∞h[k]e−jωk=H(ω)ejωny[n]=∑k=−∞∞h[k]ejω(n−k)=ejωn∑k=−∞∞h[k]e−jωk=H(ω)ejωn

In other words an input ejωnejωn simply comes out as exactly itself, only it is scaled by H(ω)H(ω). H(ω)H(ω) is not a function of n.

In other words ejωnejωn is an EIGEN input to an LSI system which comes out as a scaled version of itself.

More generally, if s[n]=zns[n]=zn where zz is any complex term

y[n]=∑k=−∞∞h[k]zn−k=zn∑k=−∞∞h[k]z−k=H(z)zny[n]=∑k=−∞∞h[k]zn−k=zn∑k=−∞∞h[k]z−k=H(z)zn

Any exponential signal is an eigen input to an LSI system. We have used the symbol "z" here because we will talk about H(z)H(z) later when we discuss the *z-transform*.

What this means, however, is that if we can write s[n]s[n] as a summation of exponentials:

s[n]=∑akejωkn for some ωks[n]=∑akejωkn for some ωk

Then:

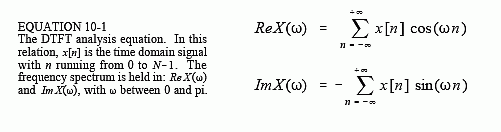
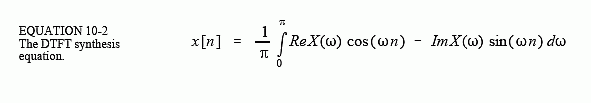
y[n]=∑[akH(ωk)]ejωkny[n]=∑[akH(ωk)]ejωkn

The output is easily obtained. To analyze the output we only need to analyze periodic discrete-time complex exponentials.

**DTFT:**

The Discrete Time Fourier Transform (DTFT) is the member of the Fourier transform family that operates on aperiodic, discrete signals. The best way to understand the DTFT is how it relates to the DFT. To start, imagine that you acquire an N sample signal, and want to find its frequency spectrum. By using the DFT, the signal can be decomposed into sine and cosine waves, with frequencies equally spaced between zero and one-half of the sampling rate. As discussed in the last chapter, padding the time domain signal with zeros makes the period of the time domain longer, as well as making the spacing between samples in the frequency domain narrower. As N approaches infinity, the time domain becomes aperiodic, and the frequency domain becomes a continuous signal. This is the DTFT, the Fourier transform that relates an aperiodic, discrete signal, with a periodic, continuous frequency spectrum.

The mathematics of the DTFT can be understood by starting with the synthesis and analysis equations for the DFT (Eqs. 8-2, 8-3 and 8-4), and taking N to infinity:

There are many subtle details in these relations. First, the time domain signal, *x*[*n*], is still discrete, and therefore is represented by *brackets*. In comparison, the frequency domain signals, *ReX*(ω) & *ImX*(ω), are continuous, and are thus written with *parentheses*. Since the frequency domain is continuous, the synthesis equation must be written as an integral, rather than a summation.

**LAB TASKS:**

**EXAMPLE 01;**

t=0:0.01:8;

x=t.\*exp(-t);

subplot(2,1,1);plot(t,x);title('x(t)=t\*exp(-t)');grid

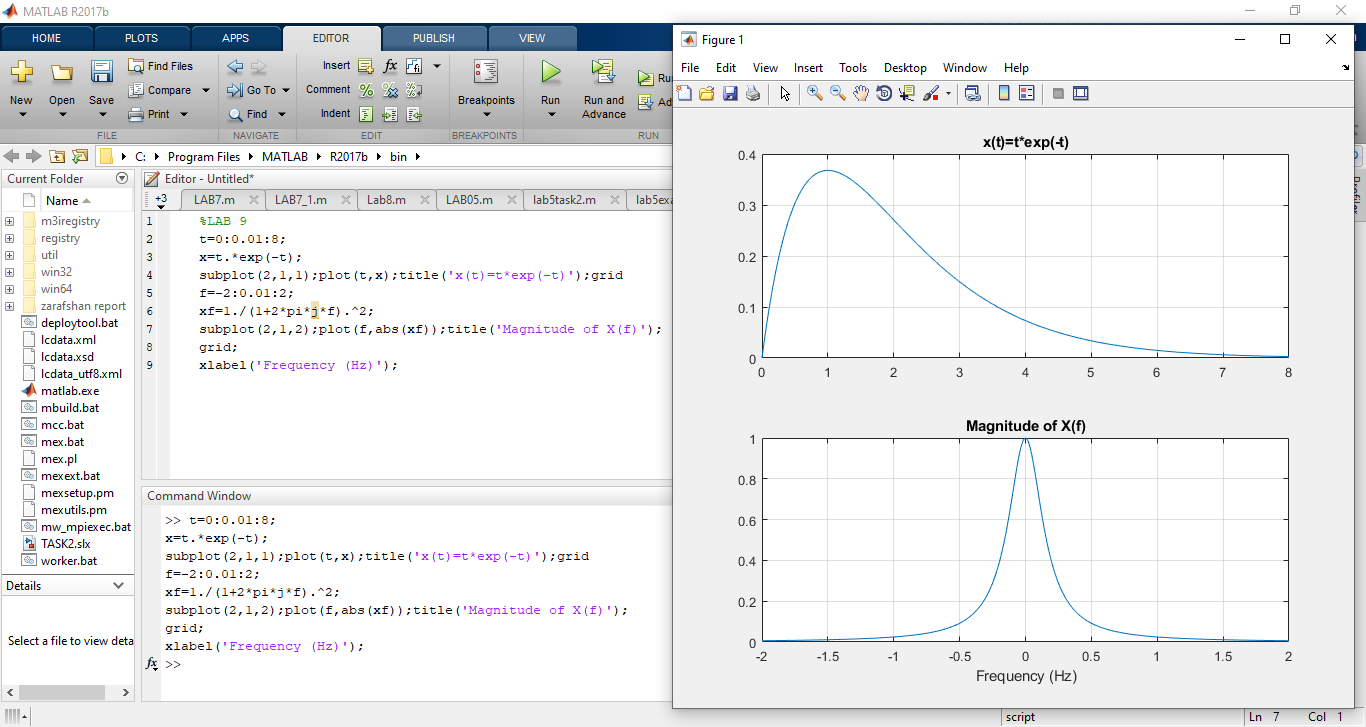
f=-2:0.01:2;

xf=1./(1+2\*pi\*j\*f).^2;

subplot(2,1,2);plot(f,abs(xf));title('Magnitude of X(f)');

grid;

xlabel('Frequency (Hz)');



**Example 02**

**Matlab code;**

%EXAMPLE 02

Sf=2; D=8; N=16;

t=0:1/Sf: (N-1)\*1/Sf; x=t.\*exp(-t);

ya=fft(x,N); ya=fftshift(ya);

fo=1/D; %Spectral Resolution;

fa=-(N/2)\*fo:fo:(N/2-1)\*fo;

subplot(2,1,1);stem(fa,1/Sf\*abs(ya));title('1/Sf\*DFT: Sf=2Hz and D=8 sec');grid; hold on; plot(f,abs(xf)); hold off;

% (b) Sf=16 Hz; D= 8 sec; N=128 samples

Sf=16; D=8; N=128;

t=0:1/Sf: (N-1)\*1/Sf; x=t.\*exp(-t);

yb=fft(x,N); yb=fftshift(yb);

fo=1/D; %Spectral Resolution;

fb=-(N/2)\*fo:fo:(N/2-1)\*fo;

% Truncate the frequency range to 2 Hz.

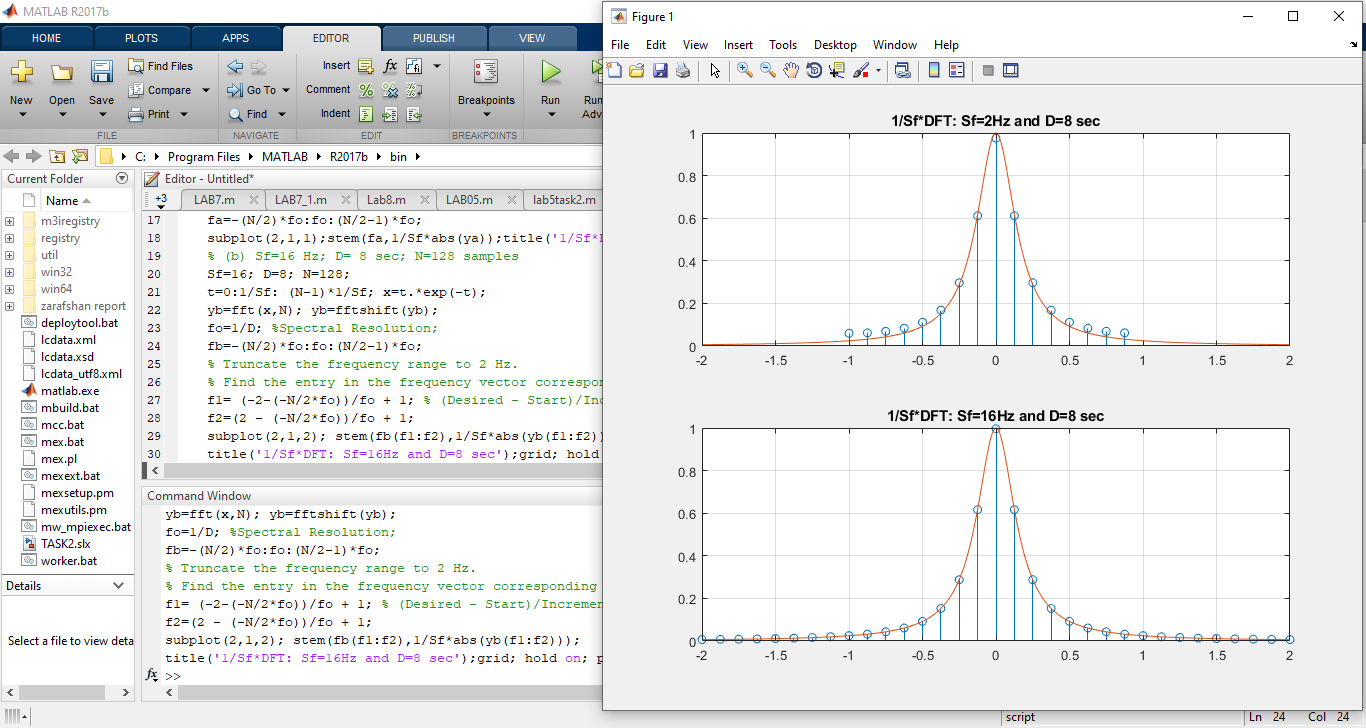
% Find the entry in the frequency vector corresponding to -2 and +2 Hz:

f1= (-2-(-N/2\*fo))/fo + 1; % (Desired - Start)/Increment + 1

f2=(2 - (-N/2\*fo))/fo + 1;

subplot(2,1,2); stem(fb(f1:f2),1/Sf\*abs(yb(f1:f2)));

title('1/Sf\*DFT: Sf=16Hz and D=8 sec');grid; hold on; plot(f,abs(xf)); hold off;



**Lab # 10 Design of FIR Filter**

**Objective:**

To learn how to design FIR (finite impulse response) filters with the given frequency specifications.

* Fourier Transform Method and Window method
* Optimal Design Method

**Theory:**

**What is FIR filter?**

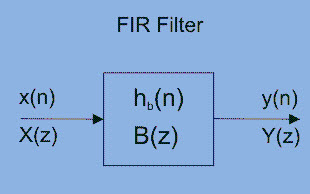
The term FIR abbreviation is “Finite Impulse Response” and it is one of two main types of

digital filters used in DSP applications. Filters are signal conditioners and function of each

filter is, it allows an AC components and blocks DC components. The best example of the

filter is a phone line, which acts as a filter. Because, it limits frequencies to a rage

significantly smaller than the range of human beings can hear frequencies.



### FIR Filters for Digital Signal Processing

There are various kinds of filters, namely LPF, HPF, BPF, BSF. A LPF allows only low frequency signals through tom its o/p, so this filter is used to eliminate high frequencies. A LPF is convenient for controlling the highest range of frequencies in an audio signal. An HPF is quite opposite to LPF. Because, it rejects only frequency components below some threshold. The best example of the HPF is, cutting out the 60Hz audible AC power, which can be selected up as noise associated almost any signal in the USA.

The alternative of IR filter is a DSP filter which can also be IIR. IIR filters uses feedback, so when you i/p an impulse the o/p theoretically rings forever. The terms used for describing IR filters are Tap, impulse response, MAC (multiply accumulate), delay line, transition band and circular buffer.

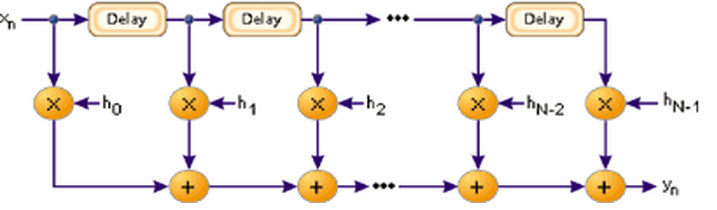
### Design Methods of FIR Filter

The design methods of FIR filter based on approximation of ideal filter. The ensuing filter approaches the perfect characteristic because the order of the filter will increase, so creating the filter and its implementation additional complicated.

The design process starts with necessities and specifications the FIR filter. The method used in the design process of the filter depends upon the implementation and specifications. There are many advantages and disadvantages of the design methods. Thus, it is very significant to elect the right method for FIR filter design. Due to efficiency and simplicity of the FIR filter, most commonly window method is used. The other method sampling frequency method is also very simple to use, but there is a small attenuation in the stopband.

#### **Logical Structure of FIR Filter**

A FIR filter is used to implement almost any type of digital frequency response. Usually these filters are designed with a multiplier, adders and a series of delays to create the output of the filter. The following figure shows the basic FIR filter diagram with N length. The result of delays operates on input samples. The values of hk are the coefficients which are used for multiplication. So that the o/p at a time and that is the summation of all the delayed samples multiplied by the appropriate coefficients.



Logical Structure of FIR Filter

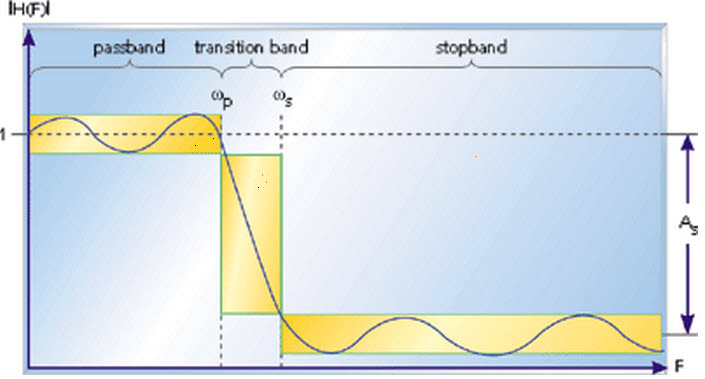
The [filter design can be defined](https://en.wikipedia.org/wiki/Filter_(signal_processing)) as, it is the process of choosing the length and coefficients of the filter. The intention is to set the parameters so that the required parameters like a stop band and pass band will give the result from running the filter. Most of the engineers use MATLAB software to design the filter.

Usually, filters are defined by their responses to the separate frequency [components that found](https://www.elprocus.com/basic-components-used-electronics-electrical/) the i/p signal The responses of the filters a classified into three types based on the frequencies such as stop band, pass band and transition band. The response of the passband is the filter’s effect on frequency components that are delivered through mostly unaffected.

Frequencies in a filter’s stopband are, by difference, highly reduced. The transition band signifies the frequencies in the middle, which may receive some reduction, but are not detached completely from the o/p signal.

#### **Frequency Response of an FIR Filter**

The frequency response plot of the filter is shown below, where ωp is the passband ending frequency, ωs is the stopband beginning frequency, As is the amount of attenuation in the stopband. Frequencies b/n ωp and ωs drop in the transition band and are reduced to some lesser degree.That confirms that the filter meets the preferred specifications includes transition bandwidth, ripple, filter’s length and coefficients. The longer the filter, the more finely the response can be tuned. With the N length and coefficients, float h[N] = {…………}, decided upon, the FIR filter implementation is fairly straightforward.

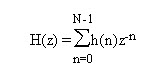


Frequency Response of an FIR Filter

**Z Transform of an FIR Filter is**

For an N-tap FIR filter with h(k) coefficient, then the o/p is defined as  
y(n)=h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + ……… h(N-1)x(n-N-1)

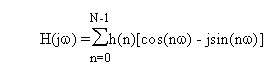
**The Z-transform of the filter is**  
H(z)=h(0)z-0 + h(1)z-1 + h(2)z-2 + ……… h(N-1)z-(N-1) or



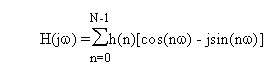
**Transfer Function of FIR Filter**



**The Frequency Response Formula for an FIR Filter**



**DC Gain of an FIR filter is**



The applications of FIR filters mainly involve in digital communications in the intermediate frequency stages of the receiver. For instance, a digital radio receives and [converts the analog signal to the intermediate frequency and then converts it to digital](https://www.elprocus.com/analog-to-digital-adc-converter/) using with a digital to analog converter. Then uses the finite impulse response to choose the preferred frequency. It is used in software radio, that permits easily adaptable filters with good rejection and without changing hardware.

**LAB TASKS:**

**Example 01;**

**Matlab code;**

%example 01

%y(n)= x(n) + 2x(n-1) +8X(n-2)-4x(n-3)

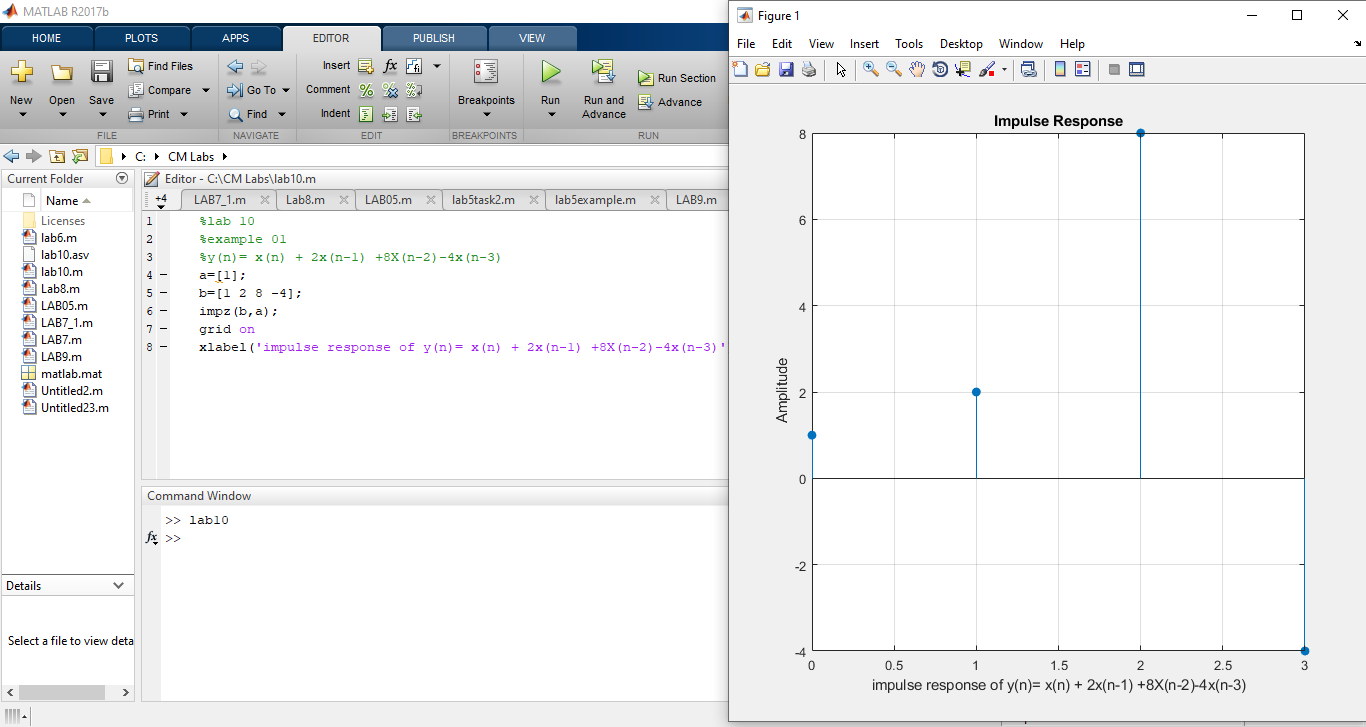
a=[1];

b=[1 2 8 -4];

impz(b,a);

grid on

xlabel('impulse response of y(n)= x(n) + 2x(n-1) +8X(n-2)-4x(n-3)');



**Exampe 02**

**Matlab code:**

%Example 02

clear

n=-3:7;

%input signal

x=[0 0 0 2 4 6 4 2 0 0 0];

%fifilter coefficients

b=[1/3 1/3 1/3]; %feedforward filter coefficient

a=1; %feedback filter coefficients

%output signal

y=filter(b,a,x);

figure

subplot(2,1,1);

stem (n,x,'filled');

xlabel ('Sample number');

ylabel('Signal amplitude');

title('Original signal');

grid

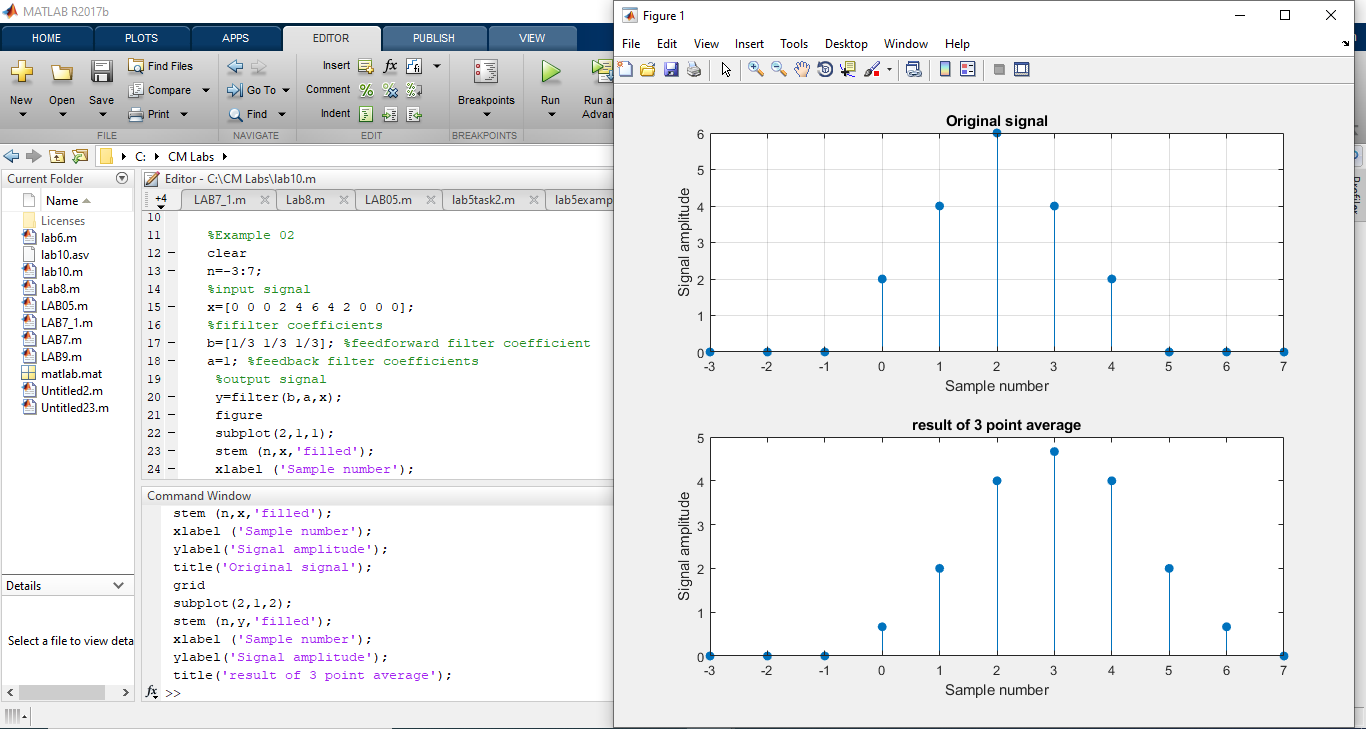
subplot(2,1,2);

stem (n,y,'filled');

xlabel ('Sample number');

ylabel('Signal amplitude');

title('result of 3 point average');



**Example 03**

**Matlab code;**

%example 03

n=0:40;

%input signal

x=(1.02).^n +0.5\*cos(2\*pi\*n/8 + pi/4);

%filter coefficients

b1=[1/3 1/3 1/3];

a1=[1];

%plot of original signal

figure;

subplot (3,1,1);

stem(n,x,'filled');

title('input signal');

xlabel('sample number');

ylabel('signal amplitude');

grid;

%output of 37 point filtering

y1=filter(b1,a1,x);

subplot(3,1,2);

stem(n,y1,'filled');

title('output of 3 pont filtering');

xlabel('sample number');

ylabel('signal amplitude');

grid;

b2=[1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7];

y2=filter(b2,a1,x);

subplot(3,1,3);

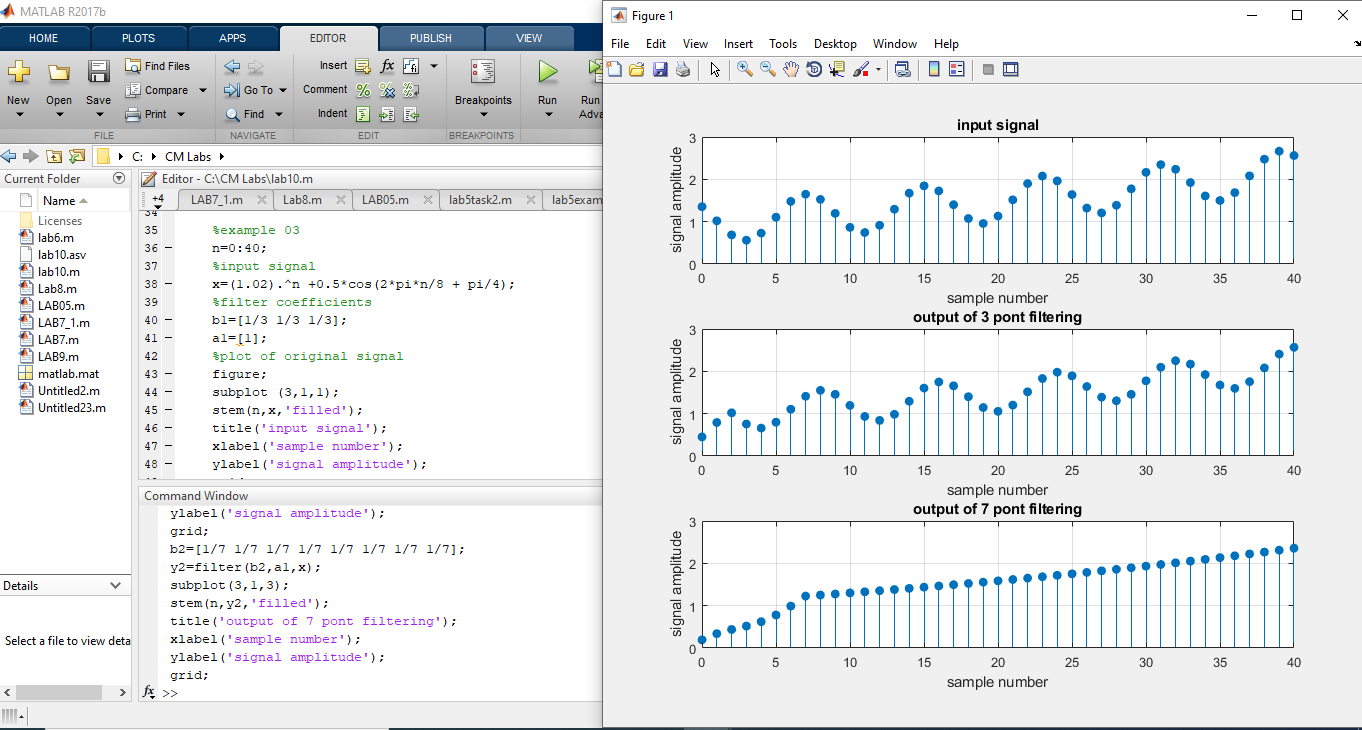
stem(n,y2,'filled');

title('output of 7 pont filtering');

xlabel('sample number');

ylabel('signal amplitude');

grid;



**Example 04;**

**Matlab code;**

%example 04

n=0:50;

%input signal

x=(1.02).^n +2\*sin(2\*pi\*n/8 + pi/4);

%filter coefficients

b1=[1/3 1/3 1/3];

a1=[1];

%plot of original signal

figure;

subplot (3,1,1);

stem(n,x,'filled');

title('input signal, x=(1.02)^n +2\*sin(2\*pi\*n/8 + pi/4);');

xlabel('sample number');

ylabel('signal amplitude');

grid;

%output of 37 point filtering

y1=filter(b1,a1,x);

subplot(3,1,2);

stem(n,y1,'filled');

title('output of 3 pont filtering');

xlabel('sample number');

ylabel('signal amplitude');

grid;

b2=[1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7];

y2=filter(b2,a1,x);

subplot(3,1,3);

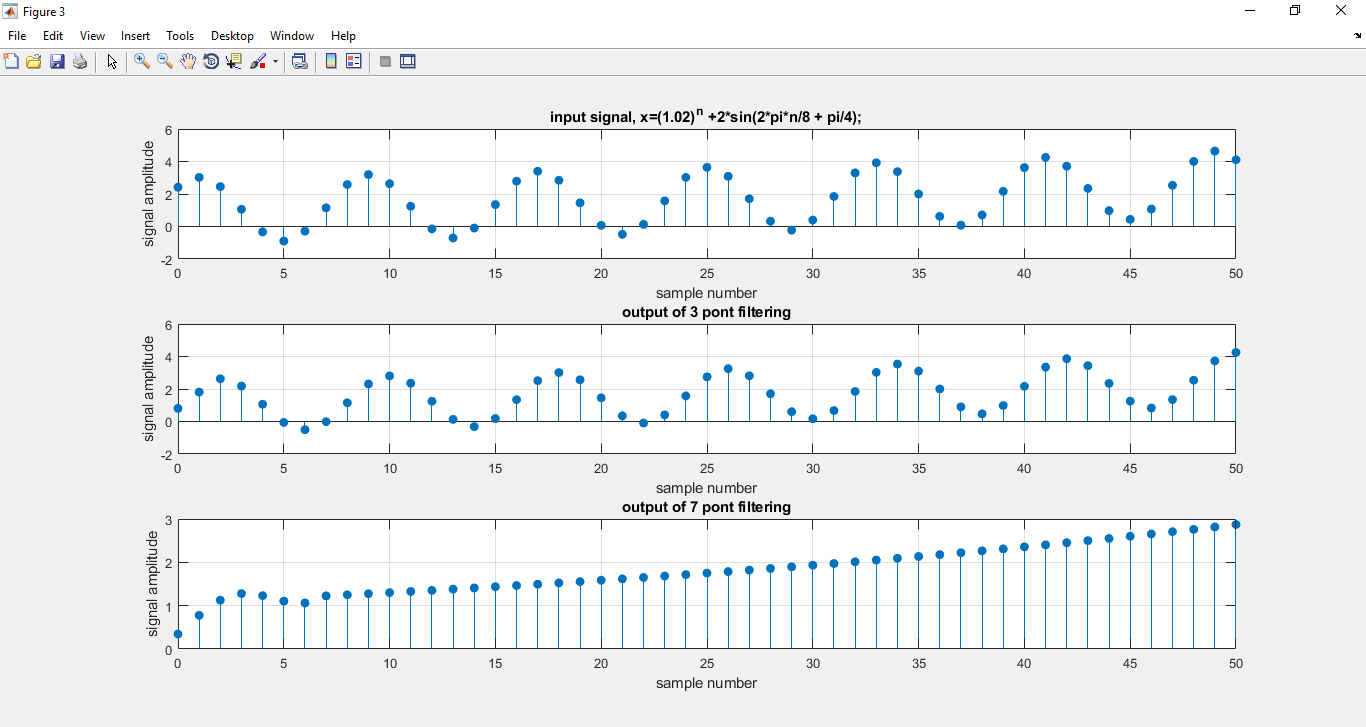
stem(n,y2,'filled');

title('output of 7 pont filtering');

xlabel('sample number');

ylabel('signal amplitude');

grid;



**Lab # 11 IIR Filter Design**

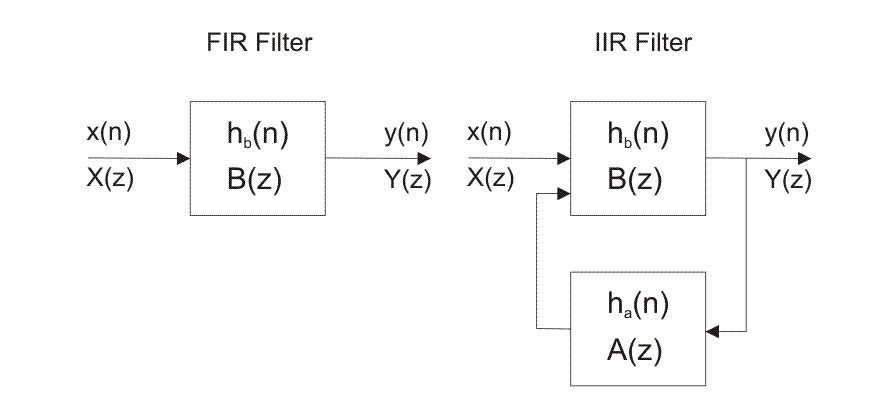
**Objective:**

* Understanding IIR, filter.
* Designing IIR and implementing in MATLAB

**Theory:**

**IIR Filters**

IIR filters are digital filters with infinite impulse response. Unlike FIR filters, they have the feedback (a recursive part of a filter) and are known as recursive digital filters therefore.



For this reason, IIR filters have much better frequency response than FIR filters of the same order. Unlike FIR filters, their phase characteristic is not linear which can cause a problem to the systems which need phase linearity. For this reason, it is not preferable to use IIR filters in digital signal processing when the phase is of the essence.  
  
Otherwise, when the linear phase characteristic is not important, the use of IIR filters is an excellent solution.  
  
There is one problem known as a potential instability that is typical of IIR filters only. FIR filters do not have such a problem as they do not have the feedback. For this reason, it is always necessary to check after the design process whether the resulting IIR filter is stable or not.  
  
IIR filters can be designed using different methods. One of the most commonly used is via the reference analog prototype filter. This method is the best for designing all standard types of filters such as low-pass, high-pass, band-pass and band-stop filters.

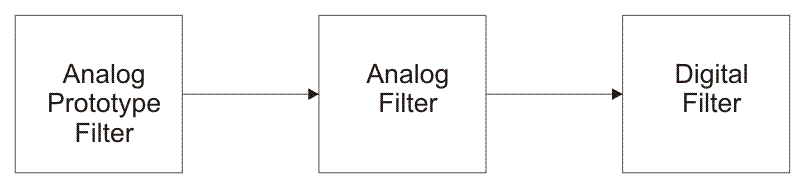


Figure 1:Block diagram of design method using reference analog prototype filter

IR filters can have linear phase characteristic, which is not typical of IIR filters. When it is necessary to have linear phase characteristic, FIR filters are the only available solution. In other cases when linear phase characteristic is not necessary, such as speech signal processing, FIR filters are not good solution. IIR filters should be used instead. The resulting filter order is considerably lower for the same frequency response.  
  
The filter order determines the number of filter delay lines, i.e. number of input and output samples that should be saved in order that the next output sample can be computed. For instance, if the filter order is 10, it means that it is necessary to save 10 input samples plus 10 output samples preceeding the current sample. All these 21 samples will affect the next output sample.  
  
The IIR filter transfer function is a ratio of two polynomials of complex variable z-1. The numerator defines location of zeros, whereas the denominator defines location of poles of the resulting IIR filter transfer function.

**1.1 What are IIR filters? What does “IIR” mean?**

IIR filters are one of two primary types of digital filters used in Digital Signal Processing (DSP) applications (the other type being FIR). “IIR” means “Infinite Impulse Response.”

**1.2 Why is the impulse response “infinite?”**

The impulse response is “infinite” because there is feedback in the filter; if you put in an impulse (a single “1” sample followed by many “0” samples), an infinite number of non-zero values will come out (theoretically.)

**1.3 What is the alternative to IIR filters?**

DSP filters can also be “[Finite Impulse Response](http://scopeplot.com/dg/dsp/faqs/fir)” (FIR). FIR filters do not use feedback, so for a FIR filter with *N* coefficients, the output always becomes zero after putting in *N* samples of an impulse response.

**1.4 What are the advantages of IIR filters (compared to FIR filters)?**

IIR filters can achieve a given filtering characteristic using less memory and calculations than a similar FIR filter.

**1**.**5 What are the disadvantages of IIR filters (compared to FIR filters)?**

* They are more susceptible to problems of finite-length arithmetic, such as noise generated by calculations, and limit cycles. (This is a direct consequence of feedback: when the output isn’t computed perfectly and is fed back, the imperfection can compound.)
* They are harder (slower) to implement using fixed-point arithmetic.
* They don’t offer the computational advantages of FIR filters for [multirate](https://dspguru.com/dsp/faqs/multirate) (decimation and interpolation) applications.

**Lab task:**

**Example 01**

**Matlab code;**

%example 02

b = [0.0798 0.0798 0.0798 0.0798];

a = [1 -1.55 1.272 -0.398];

% FREQUENCY RESPONSE OF THE SYSTEM

w = -pi:pi/500:pi;

[h,w] = freqz(b,a,w);

H=abs(h);

ph=angle(h);

subplot(2,1,1)

plot(w/pi,H,'LineWidth',2); grid;

title ('Magnitude response of the filter');

ylabel('Magnitude')

xlabel('frequency in pi units')

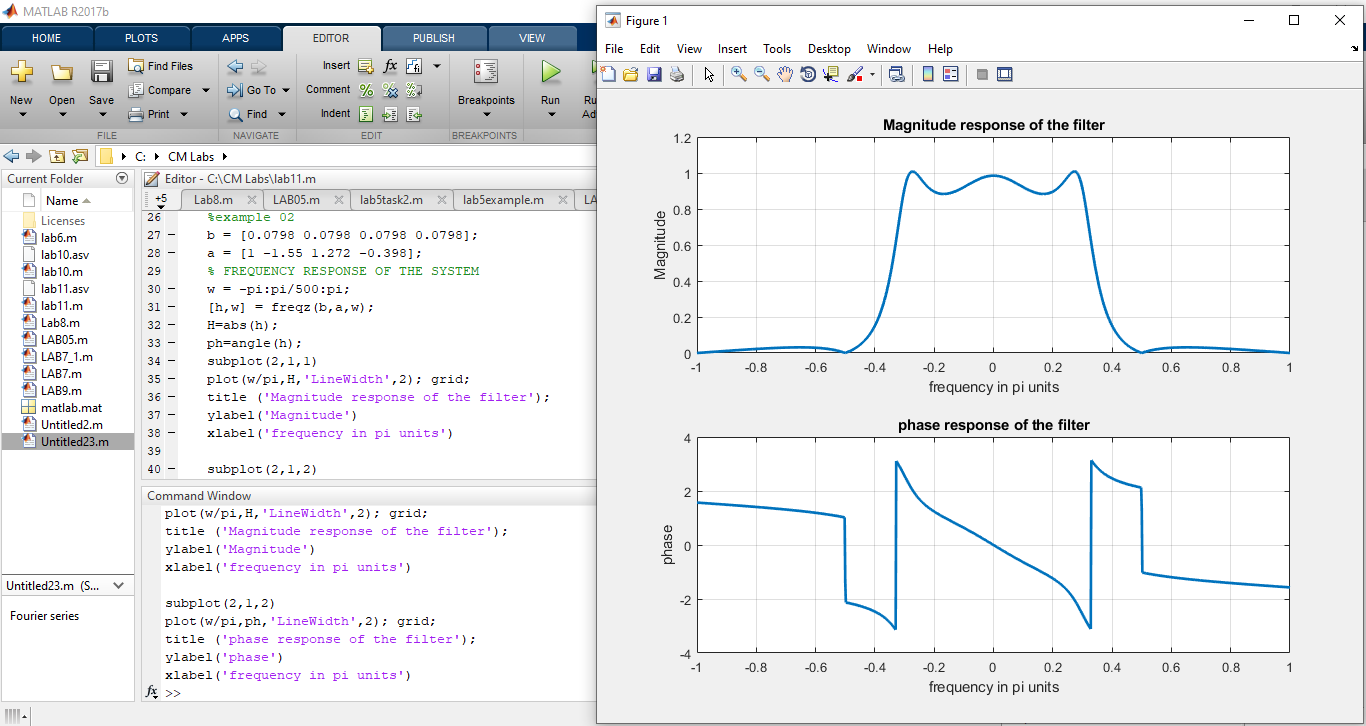
subplot(2,1,2)

plot(w/pi,ph,'LineWidth',2); grid;

title ('phase response of the filter');

ylabel('phase')

xlabel('frequency in pi units')



**Lab tasks;**

**Example 01;**

%Example 01

%y(n)=0.5y(n-1)+x(n)

%H(z)= 1

% ---------

% 1 -0.5 z^-1

G=0.5;

b=G\*[1 0];

a=[1 -0.5];

[h,w]=freqz(b,a,100);

H=abs(h);

ph=angle(h);

subplot(2,1,1)

plot(w/pi,H); grid;

title ('Magnitude response of the filter');

ylabel('Magnitude')

xlabel('frequency in pi units')

subplot(2,1,2)

plot(w/pi,ph); grid;

title ('phase response of the filter');

ylabel('phase')

xlabel('frequency in pi units')

