Objective:

This lab explains how to use MATLAB to compute the output of LTI systems using the function *conv* and some GUI tools.

Basic Theoretical Principles

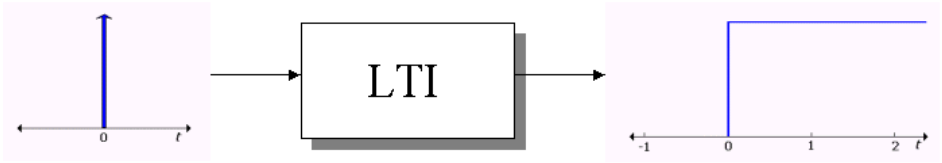
If we have a linear time invariant system and we know the impulse response of it (h(t) we can know the output of any other input using convolution. Convolution is an operation by which the output of an linear time-invariant (LTI) system with a known impulse response can be determined, given an arbitrary input signal. Observe the system to the left, with continuous-time input x(t) and output y(t). Convolution is simply the process that determines the output given the input.

$$x(t) = \delta(t),$$

$$y(t) = \int_{-\infty}^{t} x(\tau)d\tau$$

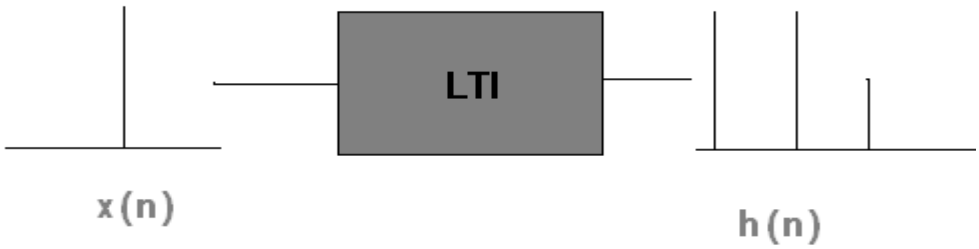
$$= \int_{-\infty}^{t} \delta(\tau)d\tau$$

$$= u(t)$$

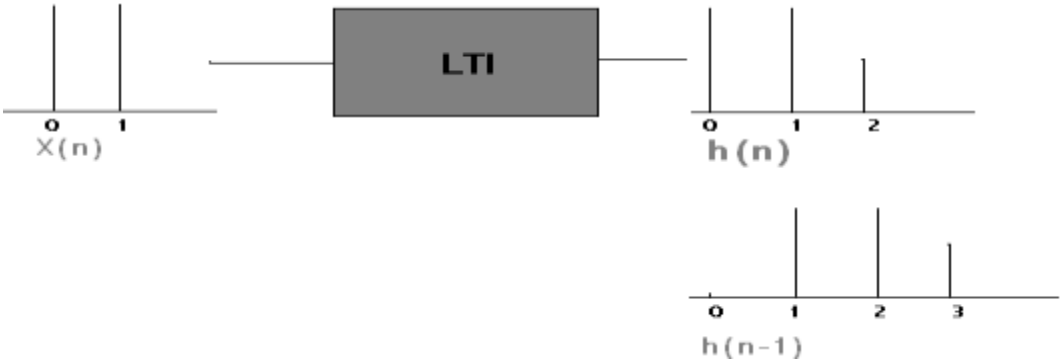


The unit impulse is used as an example input for the system shown above. When the input to any LTI system is a unit impulse, the output is called the impulse response and is denoted by h(t). So, in this example h(t)=u(t) The system in this particular example is known as an integrator because it produces a unit step signal as the output, according to the system's corresponding mathematical relationship shown to the left of it. In general, however, any relationship which is linear and time-invariant, with unit impulse as input qualifies as a valid impulse response for an LTI system.
Because such systems are **time-invariant**, if the impulse is shifted to a new location, the output is simply a shifted version of the impulse response. **Time-invariance** is a very helpful and important property of continuous-time LTI systems. In the case of the integrator, the output of a shifted unit impulse is a shifted unit-step function as shown to the right.

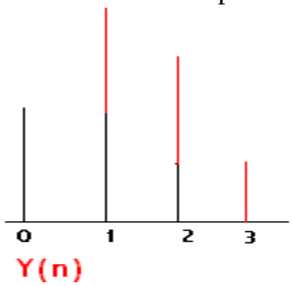
For simplicity let us start with the discrete convolution, and assume the following LTI system whit the following impulse output.



Now if we add another delayed impulse at n=1, since the system is LTI the output of the other impulse will be h(n-1)



And so the final output to this input will be the sum of the output to the two impulse as shown below:



So we can use this method to compute the output of the simple input however the convolution method can be used to compute the output to any input:

1. **MATLAB Implementation:**

```
n=6;
x=[1 2 2 3 0 0];
h=[2 -1 3 0 0 0];
y=zeros(1,n);
for i=1:n
for j=1:i
y(i)=y(i)+x(j)*h(i-j+1)
end
end
```

```
n=6;
x=[1 2 2 3];
h=[2 -1 3];
x=[x zeros(1,6-length(x))];
h=[h zeros(1,6-length(h))];
y=zeros(1,n);
for n=1:6
for k=1:5
y(i)=y(i)+x(j)*h(i-j+1)
end
end
```

If arbitrary sequences are of infinite duration, then MATLAB cannot be used directly to compute the convolution. MAT LAB does provide a built-in function called **conv** that computes the convolution between two finite-duration sequences. The conv function assumes that the two sequences begin at n = 0 and is invoked by :

» y = conv(x,h);

For example, given the following two sequences:

x(n) = [3,1l,7, 0,-1,4,2], -3≤n≤3; h(n) = [2,3,0,-5,2,1], -1≤n≤4
↑ ↑
determine the convolution y(n) = x(n) * h(n).

%%%% solution %%%%%%
»x = [3, 11, 7, 0, -1,4,2].
» h = [2, 3, 0, -5, 2, 1];
» y = conv(x,h)
y =
6 31 47 6 -51 -5 41 18 -22 -3 8 2

However, the conv function neither provides nor accepts any timing information if the sequences have arbitrary support. What is needed is a beginning point and an end point of y(n). Given finite duration x(n) and h(n), it is easy to determine these points. Let

{ x(n); $n_{xb} \le n \le n_{xe}$} and { h(n); $n_{hb} \le n \le n_{he}$}

be two finite-duration sequences. Then the beginning and end points of y(n) are

$n_{yb} = n_{xb} + n_{hb}$ and $n_{ye} = n_{xe} + n_{he}$

respectively. A simple extension of the conv function, called conv_m, which performs the convolution of arbitrary support sequences can now be designed.

```
function [y,ny] = conv_m(x,nx,h,nh)
% Modified convoluction routine for signal processing
% ------- DSP LAB / Computer Engineering / IUG -------
% [y,ny] = conv_m(x,nx,h,nh)
% [y,ny] = convolution result
% [x,nx]. first signal
```

% [h,nh] = second signal
%
nyb = nx(1)+nh(1); nye = nx(length(x)) + nh(length(h)).
ny = [nyb:nye];
y = conv(x,h).

now resolve the previous convolution using the conv_m function.

2. **Solution**:

» x = [3, 11, 7, 0, -1, 4, 2]; nx = [-3:3];
» h = [2, 3, 0, -5, 2, 1]; Nh = [-1:4];
» [y,ny] . conv_m(x,nx,h,nh)
y 6 31 47 6 -51 -5 41 18 -22 -3 8 2
ny. -4 -3 -2 -1 0 1 2 3 4 5 6 7

Hence
y(n) = {6,31,47,6, -51, -5, 41, 18, -22, -3, 8, 2}
↑

3.
```
% Creating our sine-signal
clear
t = -50:50;
Fs = 25;
x=sin(2*pi*(1/Fs)*t);

% Creating our impulse responses. They're unit steps, going from all-
zeroes to all-ones.
h1 = [zeros(1,50) ones(1,50)];
h2 = [zeros(1,40) ones(1,60)];

% Calculating the convolusion result of the square signals above with
our original sine-signal.
convPlot1 = conv(x,h1);
convPlot2 = conv(x,h2);

% Plot and check our result.
plot(convPlot1)
plot(convPlot2)
```

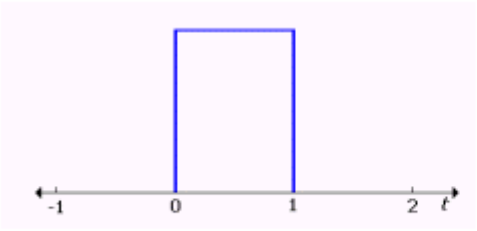
Continuous Convolution
If both linearity and time-invariance hold, the output of the system can be found through a relation known as the Convolution Integral:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$
$$= x(t) * h(t)$$

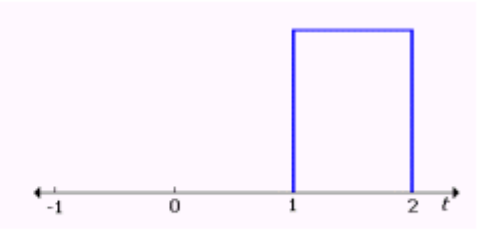
The above relationship means " x(t) convolved with h(t) " where h(t) is the impulse response of the LTI system. The variable t is taken to be a constant for the integration, which is done over the dummy variable τ for all nonzero values of the input function. After integration the τ variable disappears, leaving a function of time which is in the output of the system.

Let us look at the argument of the impulse response function in the integral. Note that it is negated as a function of τ, and shifted by an amount t. This effectively flips the function over the vertical axis and shifts it along forward by an amount equal to t. Depending upon the value of t, the shifted and flipped impulse response and input function will have a particular product and thus integrate into different piecewise functions. Continuous convolutions therefore often result in functions with multiple distinct regions.

Consider an example using the following system and input:

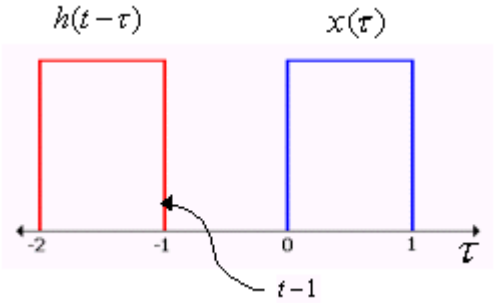


$$x(t) = u(t) - u(t-1)$$



$$h(t) = u(t-1) - u(t-2)$$

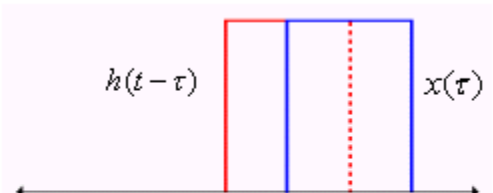
The input, shown on the left, is a pulse of length one; the impulse response as seen on the right is a pulse of length one delayed by one. Taking these two functions and converting them into the forms used in the convolution integral gives the following:



As explained above, the impulse response has been flipped over the vertical axis and is shown in red. This particular instance of the impulse response is displayed for t=0; In this case, the leading edge of the pulse is t-1. The input function x(t) is simply expressed in terms of the variable tau instead of t, which is also the axis variable of integration.

If you visualize *h(t)* sliding along the tau axis and through x(t), four distinct regions of integration become apparent. The first is for $t < 1$, which is the same as the initial situation shown above. No overlap exists between the two pulses *x(t)* and *h(t)* for this region and the integral simply goes to zero. The second region is for $1 < t < 2$, shown to the right. The pulses overlap in this region in the interval from $t = 0$ to $t = t\text{-}1$. The corresponding integral is evaluated below:

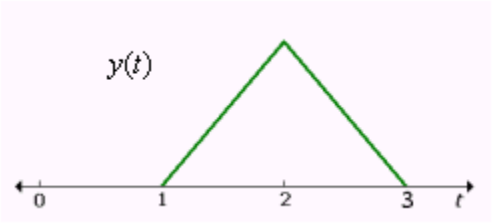
$$\begin{aligned} y(t) &= \int_0^{t-1} (u(\tau) - u(\tau - 1))(u(t - \tau - 1) - u(t - \tau - 2))d\tau \\ &= \int_0^{t-1} (1)(1)d\tau \\ &= \int_0^{t-1} d\tau \\ &= t - 1 \quad \text{for} \quad 1 < t < 2 \end{aligned}$$



A third region consists of $2 < t < 3$, when the leading edge of the impulse response slides passed the leading edge of the input signal. This is shown in the plot to the right. In this region the pulses overlap from $t = t\text{-}2$ to $t = 1$. The integral for this region is evaluated as shown below:

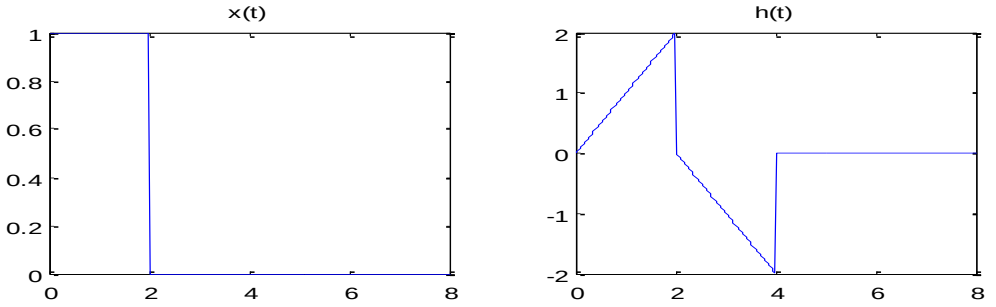
After $t=3$, no overlap occurs and the integral goes to zero. Note that at $t = 2$ between the second and third regions, both pulses completely overlap and the integral evaluates to the greatest value of the output signal. Combining the above results, our output signal is a piecewise linear function defined and plotted to the right.

$$y(t) = \begin{cases} t-1 & 1 < t < 2 \\ 3-t & 2 < t < 3 \\ 0 & \text{otherwise} \end{cases}$$



MATLAB Implementation:

We can use the function conv to perform the continuous convolution as shown in the following example :



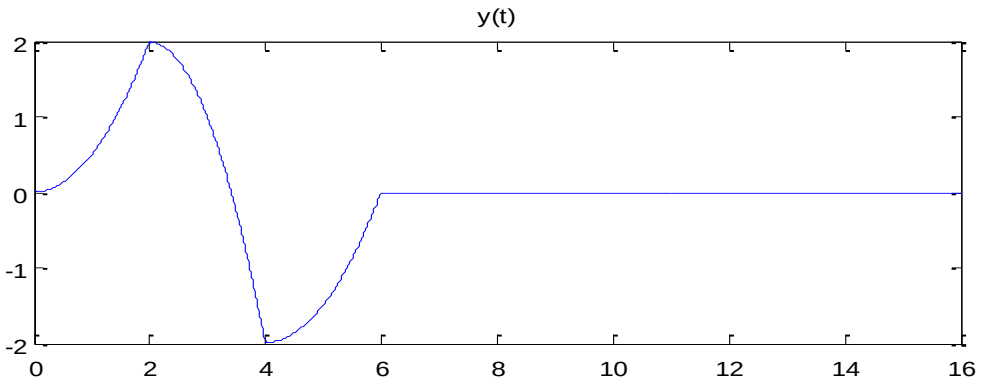
We can compute the convolution using this Matlab code :

4.

```
t=0:.01:8;
T=0:.01:16;
x=u(t)-u(t-2);
subplot(2,2,1)
plot(t,x)
title('x(t)')
h=r(t)-2*r(t-2)-2*u(t-2)+2*u(t-4)+r(t-4);
subplot(2,2,2)
plot(t,h)
```

```
title('h(t)')
y=conv(x,h).*.01;
subplot(2,1,2)
plot(T,y)
title('y(t)')
```

the result is shown in the following figure :



We can note that we multiply by 0.01 which is the time step since compute the integral by multiply the signal high with the time width which is the time step

Aim of Experiment:- To develop the program for finding the convolution between two sequences.

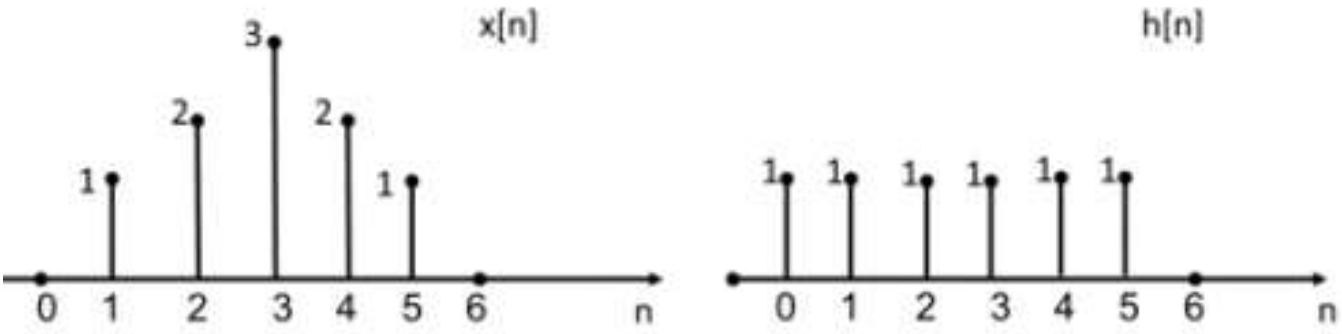
Appartus:- PC installed with Matlab Software.

5. **Program:-**

```
x=input('enter the first sequence')
enter the first sequence[1 2 3 4]
h=input('enter the second sequence')
enter the second sequence[1 1 1 1]
y=conv(x,h);
subplot(2,2,1);
stem(x);
xlabel('a');
ylabel('Input Sequence');
subplot(2,2,2);
stem(h);
xlabel('b');
ylabel('Impulse Sequence');
subplot(2,2,3);
stem(y);
xlabel('c');
ylabel('output sequence');
```

title('Convolution between two Sequences');

Consider an LTI system with the impulse response h[n] and input x[n]. Find the convolution sum, y[n] & sketch the output for this system.



1. Prove your answer using Graphical Method.
2. Use MATLAB to compute & plot y[n].

6.

```
x=[0 1 2 3 2 1];
h=[1 1 1 1 1 1];
nx=(0:1:5);
nh=(0:1:5);
subplot(3,1,1);
stem(nx,x); xlabel('n');ylabel('x[n]');
subplot(3,1,2);
stem(nh,h); xlabel('n');ylabel('h[n]');
n=(0:1:length(x)+length(h)-2);
subplot(3,1,3);
c=conv(x,h);
stem(n,c,'k'); xlabel('n');ylabel('y[n]');
```

Z-Transform

Z-transform is defined as:

$$X(z)=\sum_{n=0}^{\infty}x(n)z^{-n}$$

or

$$X(z)=Z[x(n)]$$

The inverse z-transform is denoted by

$$x(n)=Z^{-1}[X(z)]$$

MatLab Symbolic Toolbox gives the z-transform of a function

7. %ztransform of finite duration sequence

```
clc;
close all;
clear all;
syms 'z';
disp('If you input a finite duration sequence x(n), we
will give you its z-transform');
nf=input('Please input the initial value of n = ');
nl=input('Please input the final value of n = ');
x= input('Please input the sequence x(n)= ');
syms 'm';
syms 'y';
f(y,m)=(y*(z^(-m)));
disp('Z-transform of the input sequence is displayed
below');
k=1;
for n=nf:1:nl
   answer(k)=(f((x(k)),n));
  k=k+1;
end
disp(sum(answer));
```

Example of Output

If you input a finite duration sequence x(n), we will give you its z-transform
Please input the initial value of n = 0
Please input the final value of n = 4
Please input the sequence x(n)= [1 0 3 -1 2]
Z-transform of the input sequence is displayed below

3/z^2 - 1/z^3 + 2/z^4 + 1

syms 'z';

This statement creates symbolic variable z. syms function is used to creates symbolic variable.

ztrans(signal, index, point);

8.

```
syms n
>> f=0.5^n
f =(1/2)^n
>> ztrans(f)
```

9.

Example:

$x(n)=\frac{1}{4^n}u(n)$

```
>> syms z n
>> ztrans(1/4^n)

ans =

4*z/(4*z-1)
```

$X(z)=\frac{2z}{2z-1}$

```
>> syms z n
>> iztrans(2*z/(2*z-1))

ans =

(1/2)^n
```

Example: Lets consider a difference equation

$$6y(n)-5y(n-1)+y(n-2)=\frac{1}{4^n},\ n\geq 0$$

and

$$y(n-1)=1,\ y(n-2)=0$$

Taking z-transform

$$6Y(z)-5\left\{z^{-1}Y(z)+y(-1)\right\}+\left\{z^{-2}Y(z)+z^{-1}y(-1)+y(-2)\right\}=\frac{4z}{4z-1}$$

$$\left(6-5z^{-1}+z^{-2}\right)Y(z)=\frac{4z}{4z-1}-z^{-1}+5z^{-1}$$

$$Y(z)=\left(\frac{1}{6-5z^{-1}+z^{-2}}\right)\left(\frac{4z}{4z-1}-z^{-1}+5z^{-1}\right)$$

```
>> syms z n
>> iztrans((4*z/(4*z-1)-z^-1+5)/(6-5*z^-1+z^-2))
```

ans =

1/2*(1/4)^n+5/2*(1/2)^n-2*(1/3)^n

Therefore

$$y(n)=\frac{1}{2}\left(\frac{1}{4}\right)^n+\frac{5}{2}\left(\frac{1}{2}\right)^n-2\left(\frac{1}{3}\right)^n$$

In Matlab "deconv" command is used to compute the inverse z transform. (Determine the values of x(n) for few samples)

deconv

Deconvolution and polynomial division

Syntax

```
[q,r] = deconv(x,h)
```

Description

[q,r] = deconv(x,h) deconvolves vector x out of vector h, using long division. The quotient is returned in vector q and the remainder in vector r such that v = conv(u,q)+r.

If x and h are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials, and deconvolution is polynomial division. The result of dividing h by x is quotient q and remainder r.

Examples

If

```
x = [1  2  3  4]
h = [10  20  30]
```

the convolution is

```
c = conv(x,h)
c =
10   40   100   160   170   120
```

Use deconvolution to recover h:

```
[q,r] = deconv(c,x)
q =
        10   20   30
r =
         0    0    0    0    0    0
```

This gives a quotient equal to v and a zero remainder.

10. Example:

Find the first five term of the inverse z-transform of

$$X(z)=\frac{1+2z^{-1}+z^{-2}}{1-2z^{-1}+4z^{-2}}$$

```
>> b=[1 2 1];
>> a=[1 -2 4];
>> n=5;
 >> b=[b zeros(1,n-1)];
>> [x,r]=deconv(b,a);
 >> disp(x)
1 4 5 -6 -32
```

11.

Z Plan

Pole-zero Diagram The MatLab function “zplane” can display the pole-zero diagram

$$H(z)=\frac{z^{-1}+\frac{1}{2}z^{-2}}{1+\frac{3}{5}z^{-1}+\frac{2}{25}z^{-2}}$$

```
>> b=[0 1 1/2];
>> a=[1 3/5 2/25];
>> zplane(b,a)
```

MATLAB program to plot zeros and poles of z-transform

Program Code

```
%Plotting zeros and poles of z-transform
clc;
close all;
clear all;
disp('For plotting poles and zeros');
b=input('Input the numerator polynomial coefficients');
a=input('Input the denominator polynomial coefficients');
[b,a]=eqtflength(b,a);
[z,p,k]=tf2zp(b,a);
zplane(z,p);
disp('zeros');
disp(z);
disp('poles');
disp(p);
disp('k');
disp(k);
```

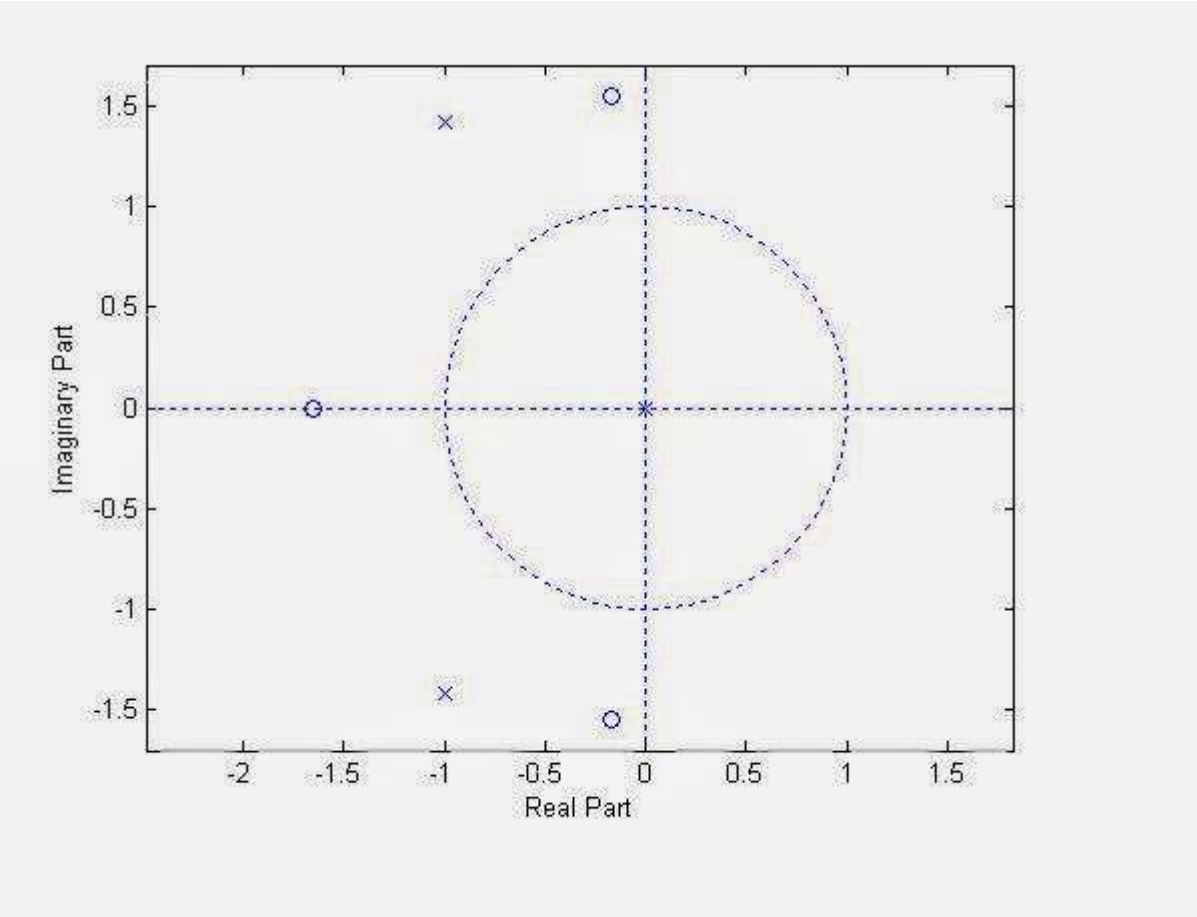
Example of Output

```
For plotting poles and zeros
Input the numerator polynomial coefficients[1 2 3 4]
Input the denominator polynomial coefficients[1 2 3]
zeros
  -1.6506
  -0.1747 + 1.5469i
  -0.1747 - 1.5469i

poles
        0
  -1.0000 + 1.4142i
  -1.0000 - 1.4142i

k

     1
```



13. Question: x[n] = (1/2) ^ n * u[n] + (-1/3) ^ n * u[n]

1. Find it's *z*-transform.
2. Plot it's poles and zeros

The birst bit of code you gave uses symbolic math to solve for the *z*-transform. You'll need to convert the output to a discrete-time model supported by the Control System toolbox.

```
syms n;
f = (1/2)^n + (-1/3)^n;
F = ztrans(f)
```

returns `z/(z - 1/2) + z/(z + 1/3)`. You can optionally use `collect` to convert this

```
z/(z - 1/2) + z/(z + 1/3)
```

```
F2 = collect(F)
```

to `(12*z^2 - z)/(6*z^2 - z - 1)`. Then you'll want to find the coefficients of the polynomials in the numerator and denominator and create a discrete-time transfer function object with `tf` for a particular sampling period:

```
[num,den] = numden(F2);
Ts = 0.1; % Sampling period
H = tf(sym2poly(num),sym2poly(den),Ts)
pzmap(H)
```

14. How to matlab z-transform of x(n)=((4/3)^n) u(1-n)

```
clearvars; clc; close all;
% u(t) : 0 for n<0 and 1 for n>=0
oldparam = sympref('HeavisideAtOrigin',1);
syms n z
assume(n, 'integer');
%%time series
x(n)=((4/3)^n)*heaviside(1-n);
% break into two time series
x1(n)=( (4/3)^n )* heaviside(-n)
x2(n)=( (4/3)^n )* (heaviside(n-1)-heaviside(n-2))
% plot time series
ntest=-8:1:8;
subplot(2,1,1); stem(ntest, x(ntest), 'b^');
title(texlabel(x(n))); xlabel('n'), ylabel('x(n)');
subplot(2,1,2); stem(ntest, x1(ntest), 'ro'); hold on;
subplot(2,1,2); stem(ntest, x2(ntest), 'g*','Color', [0 .5 0]);
legend('x1', 'x2'); title('decomposition to x1(n), x2(n)');
%%z-transform of x1(n)
% create "time reversed" version
x1r(n)=subs(x1(n),n,-n);
% z-transform of x1r(n)
X1r(z)=simplifyFraction( ztrans(x1r(n),n,z) );
% z-transform of x1(n)
X1(z)=simplifyFraction( subs(X1r(z),z,z^-1) );
%%z-transform of x2(n)
X2(z)=simplifyFraction( ztrans(x2(n)) );
%%calculation of z-transform of x(n)
X(z)=X1(z)+X2(z)
```

15. How can I get the z-transform for the sequence a^|n| (a>0)?

decompose it into 2 sequences, $a^{n}u[n]$ and $a^{(-n)}u[-n-1]$ (or should I have it as $a^{(-n-1)}u[-n-1]$), is this correct?

$a^{|n|} = a^{n}u[n] + a^{-n}u[-n-1]$

or

$a^{|n|} = a^{n}u[n] + a^{-n-1}u[-n-1]$

Thanks

ps: with Matlab codes

```
syms a n
y = ztrans(a^n+a^(-n));
```

it gives (a*z)/(a*z - 1) - z/(a - z)

```
  a z          z
-------   -   -----
a z - 1      a - z ,
```

and y = ztrans(a^n+a^(-n - 1));

we have z/(a*z - 1) - z/(a - z)

```
>> pretty(y)
    z            z
-------     -    -----
a z - 1         a - z
```

$a^{|n|} = a^{n}u[n] + a^{-n}u[-n-1]$

$a^{n}u[n]$ <----------> z/(z-a)

$a^{-n}u[-n]$ <----------> (1/z)/((1/z)-a) = 1/(1-az)

$a^{-n-1}u[-n-1]$ <----------> z/(1-az)

a. $a^{-n-1}u[-n-1]$ <----------> az/(1-az) = - az/(az-1) (-ve sign again)

Another way to find the second term:

$a^{-n}u[-n-1]$ <----------> Sum(n=-inf to +inf) $[a^{-n}u[-n-1]\ z^{-n}]$

= Sum(n=-inf to -1) $[(az)^{-n}]$

= Sum(n=1 to inf) $[(az)^{n}]$

= az/(1-az)

= - az/(az-1) (Again -ve sign)

Remember that

Sum $(n=n_1$ to $n=n_2)$ $[(a)^{n}] = (a)^{n_1}-(a)^{n_2}+1/(1-a)$.

Finally, $x[n]=a|n|=a^{n}u[n]+a^{-n}u[-n-1]$ and not $a^{n}u[n]+a^{-n-1}u[-n-1]$

proof: x[-1]=a|-1|=a

using the first decomposition $a^{-1}u[-1]+a^{-(-1)}u[-(-1)-1]=0+a=a$ (correct).

Using the second decomposition $a^{-1}u[-1]+a^{-(-1)-1}u[-(-1)-1]=0+1=1$ (incorrect).