



In the Name of God

Signals and Systems

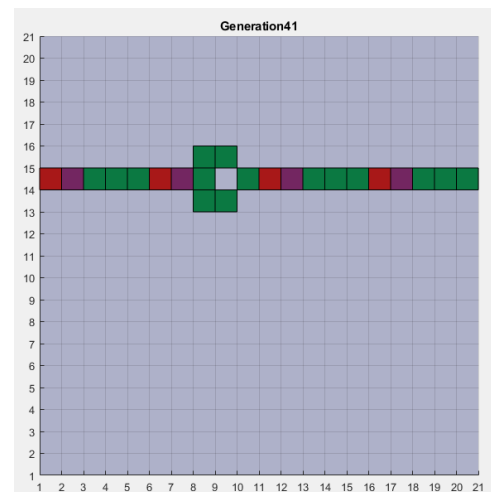
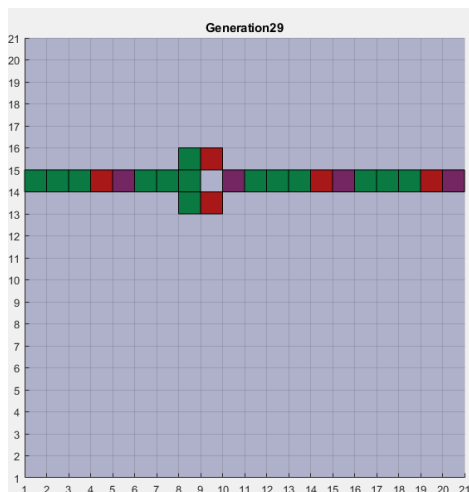
Matlab_Homework_1 Report

Armin Panjehpour – 98101288

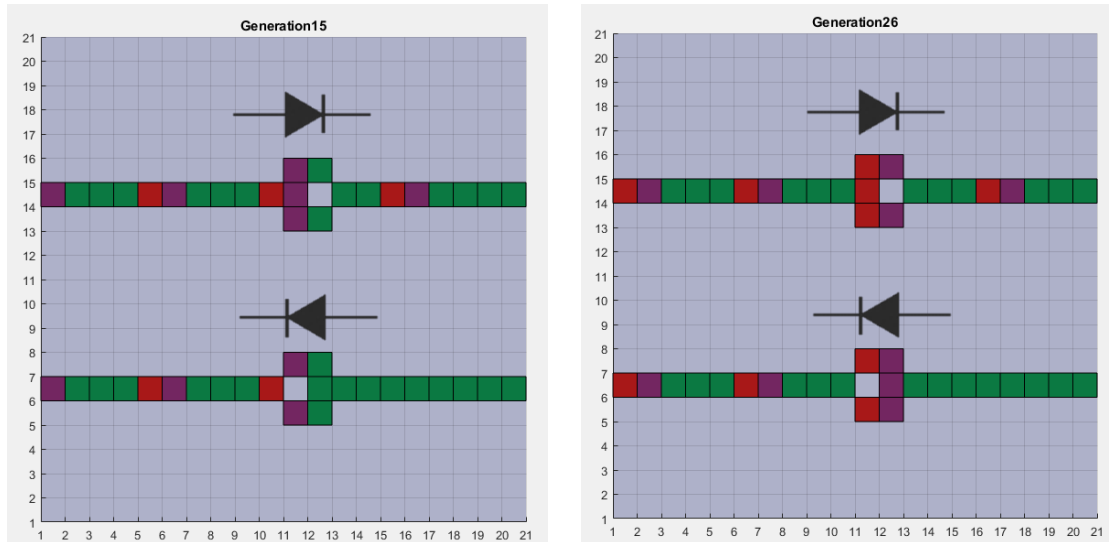
1- Wireworld:

1.1 –

I have made a function named cells that user can set the initial state of the table him/herself. At the first, all the table is empty(0), by clicking ones on a cell, it will be a conductor(1) with the color, **green**. If you click another time, it will be an electron head(2) with the color, **purple** and if you click one more time, it will be an electron tail(3) with the color, **red** and by another clicking it will be again a empty cell with no color. So, in this way you can change the table and set initial state of the table. After you're done, you just have to click for example ENTER key to run the program. If you want to change the cells in the program and not use this system, you just comment the parts I have declared in the code and change the matrix, CurrentCells. About the algorithm, We just have to check neighbors of each cell and be careful about the cells in the borders of the table that we don't get the error, ARRAY INDEX OUT OF RANGE. We need 2 matrixes to save current status of the cells and the next status of the cells and each time we put the value of the next status matrix to the current one. Periodic behavior puts an electron head from left after each 4 moves. Be careful that the



periodic behavior has designed just for the test case that you have put on the PDF and comments the 3 lines which I have declared for other test cases! "line 131 to 133"



You're test case is showing **2 diodes** which in the diode in at the top, the currents can go through the diode but the on in the bottom, the diode don't let the current to go through.

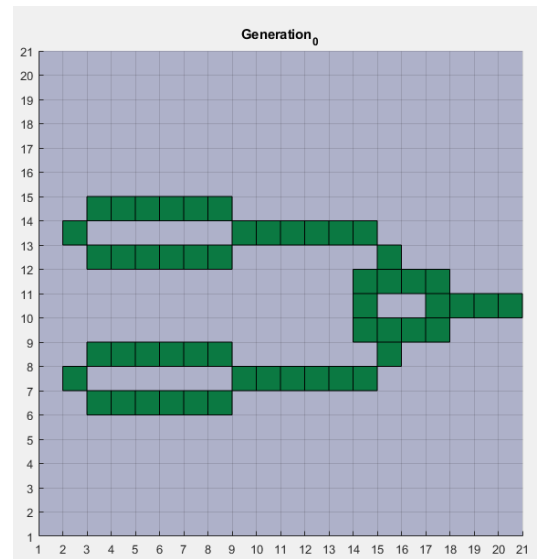
1.2 –

Wireworld is a 4 state cellular automaton which is suited for simulating digital circuits by setting some wires(conductors) in it. By wireworld, we can design gates, memory banks, etc. This cellular atomatons can be 1/2/3 D arrays and each part of these arrays are called **cells**. We can simulate different complicated models by just setting the right initial states in the table (As we did for diode and XOR gate(in the next part)).

1.3 – (remember to comment the periodic behavior for testing - “line 131 to 133”

XOR with wireworld:

This circuit acts like XOR and if we put two electron heads in the inputs it won't let the current pass and if we only put one electron head, it let the current pass.



2.1 – Z Transform: 2.1.1 -

The z transform of $x[n] = \cos(\frac{n\pi}{6})u[n]$ is:

```
Command Window

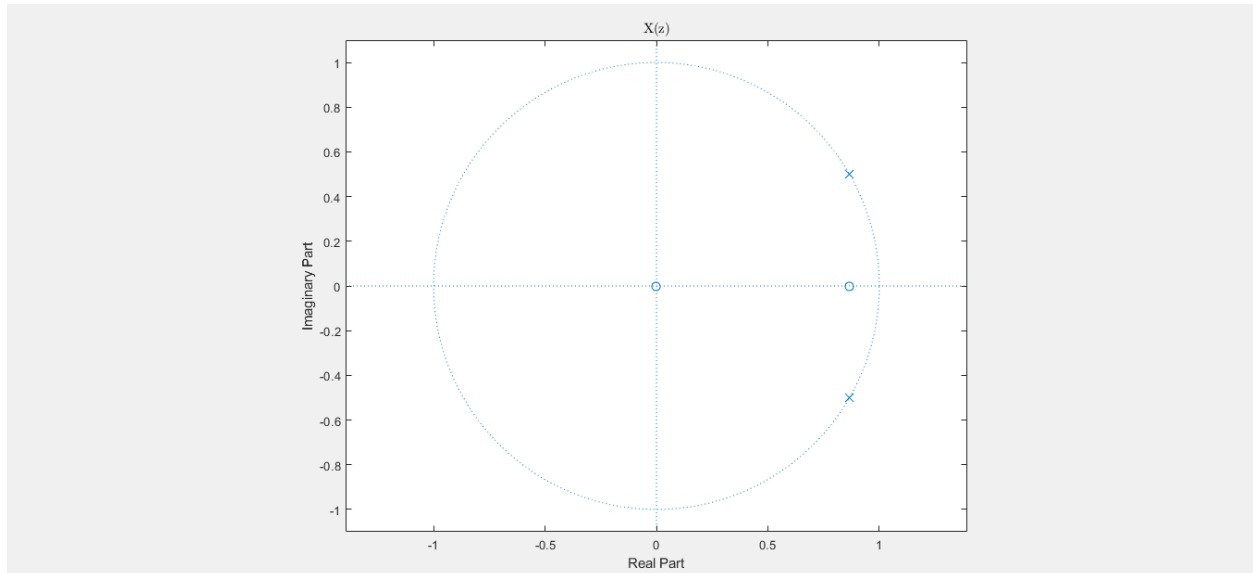
X(z) =
(z*(z - 3^(1/2)/2))/(z^2 - 3^(1/2)*z + 1)
fx >> |
```

$$X(z) = \frac{z(z - \frac{\sqrt{3}}{2})}{z^2 - \sqrt{3}z + 1}$$

For simulating the z transform I've used ztrans function which I passed x(n) as a symbolic function to it.

For plotting the zeros and poles, I have used **numden** function to get the numerator and denominator of the X(z) and then I've used **solve** function, to find the roots of the numerator and denominator which are our zeros and poles. For plotting the zeros and poles, I've used the **zplane** function which gets an array of zeros and poles.

The plot of zeros and poles:



Since the signal is right handed, the ROC of this z transform is:

$$|z| \geq 1$$

2.1.2 –

```
Command Window

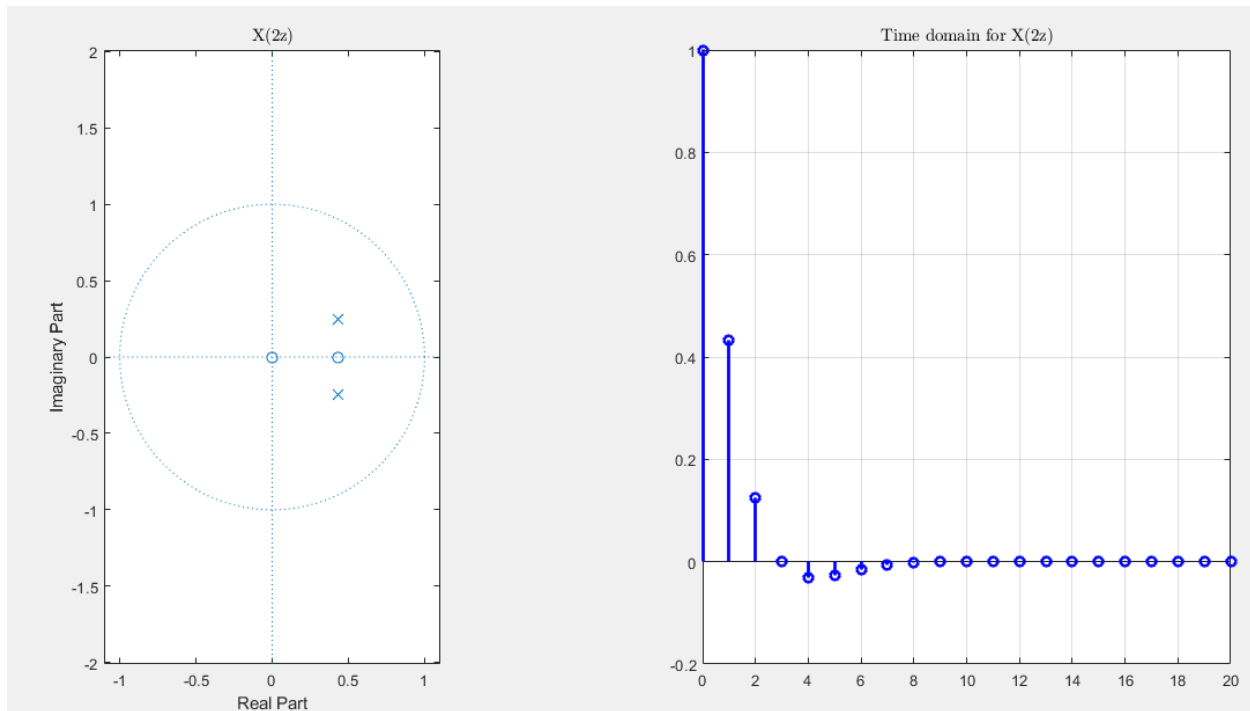
X2 (z) =
(2*z*(2*z - 3^(1/2)/2))/(4*z^2 - 2*3^(1/2)*z + 1)

fx >>
```

$$X_{\gamma}(z) = \frac{(2z)(2z - \frac{\sqrt{3}}{2})}{(2z)^2 - \sqrt{3}(2z) + 1}$$

For calculating the time domain form of $X_{\gamma}(z)$, I have used **iztrans** function which calculate inverse z transform.

The zeros and poles plot and time domain plot of $X_{\gamma}(z)$ is in the next page:



As we expected, our time domain has been multiplied by $\frac{1}{r^n}$, because of the scaling in the z -domain and ROC of $X_r(z)$ is:

$$|z| \geq 0.5$$

As we expected from theories :

$$z_0^n x[n] \xleftrightarrow{Z} X\left(\frac{z}{z_0}\right)$$

with $\text{ROC} = |z_0|R$

۲,۱.۳ —

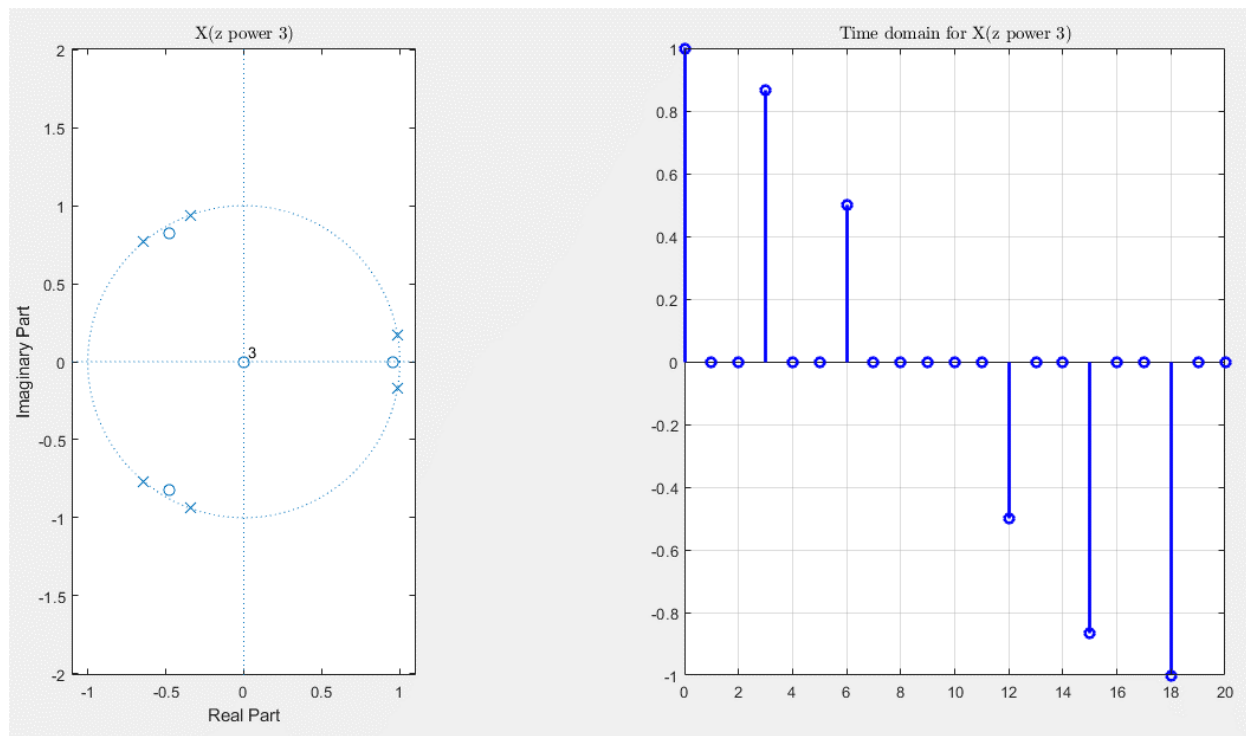
```
Command Window

X3 =
-(z^3*(- z^3 + 3^(1/2)/2))/(z^6 - 3^(1/2)*z^3 + 1)

fx >>
```

$$X_r(z) = \frac{(z^3)(z^3 - \frac{\sqrt{3}}{2})}{(z^3)^2 - \sqrt{3}(z^3) + 1}$$

The zeros and poles plot and time domain plot of $X_r(z)$ is in the next page:



As we expected a time expansion has happened and between each two of samples of $x[n] = \cos(\frac{n\pi}{6})u[n]$ two zeros has been added.

And Roc is: $|z| \geq 1$

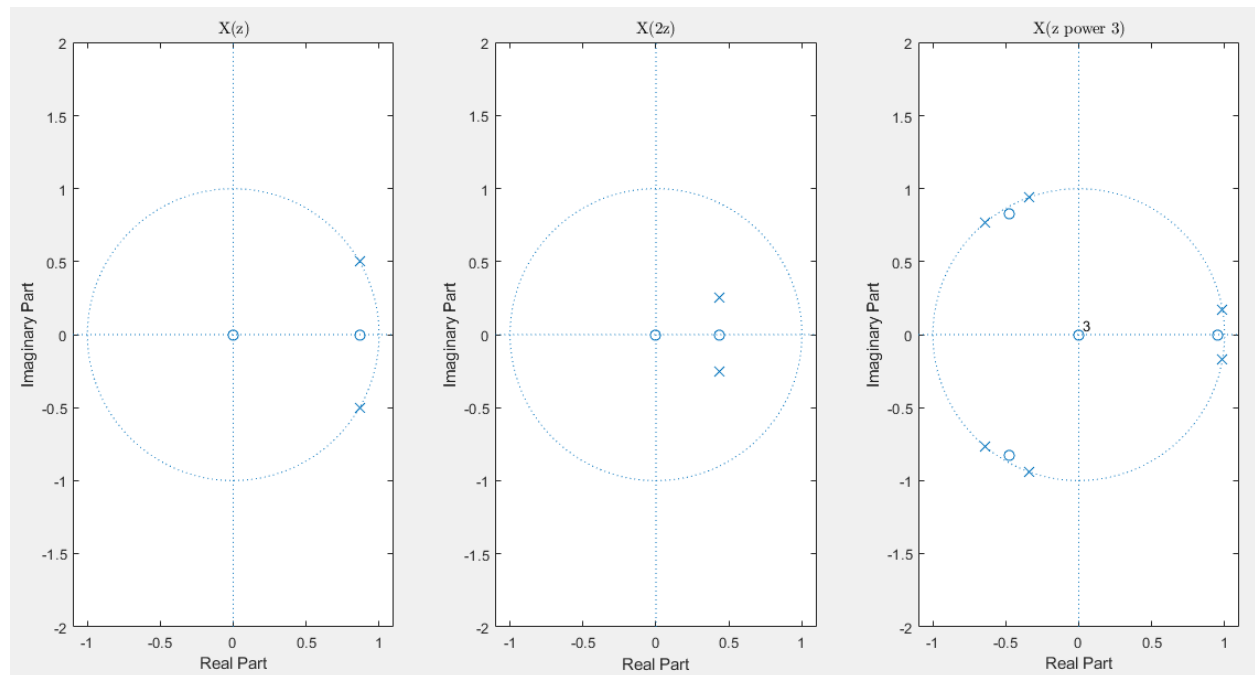
As we expected from the theories:

$$x_{(k)}[n] \xleftrightarrow{Z} X(z^k)$$

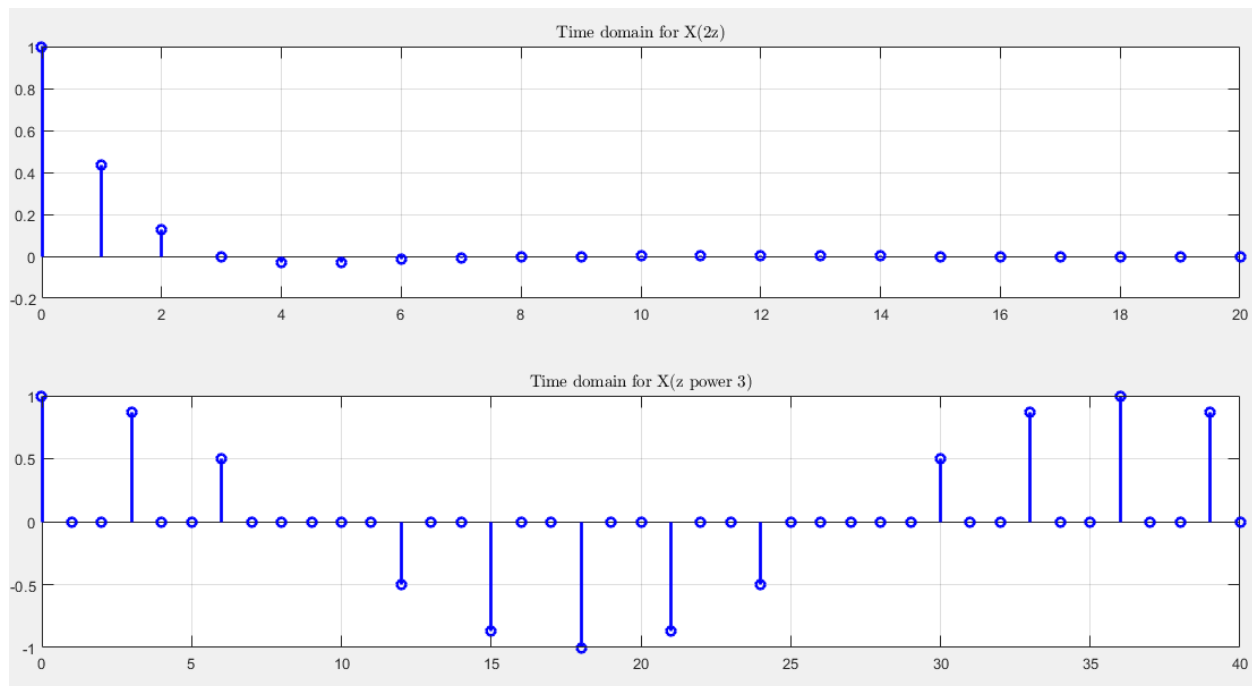
with $\text{ROC} = R^{1/k}$

Let`s look at the plots again together:

Z_planes of $X(z)$, $X(2z)$ and $X(z^3)$:

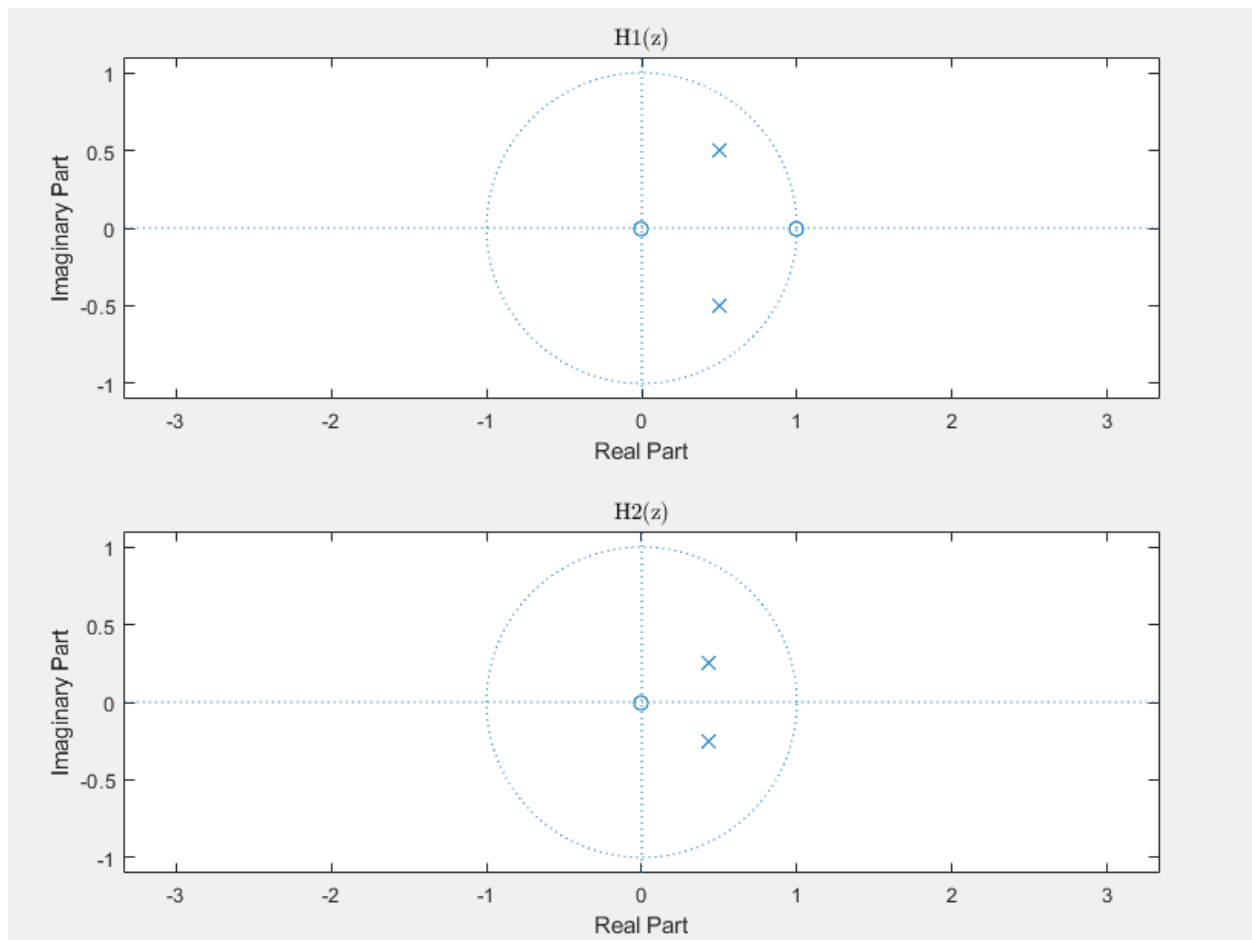


Time domains of $X(2z)$ and $X(z^3)$:



۲.۲ – inverse Z Transform: 2,2,1

Z_Planes of H1 and H2:



Since our systems are casual, so the ROC of H1 and H2 are:

ROC of H1: $|z| \geq \frac{\sqrt{r}}{r}$ so the system is **stable** because it includes the unit cycle $|z| = 1$

ROC of H2: $|z| \geq 0.5$ so the system is **stable** because it includes the unit cycle $|z| = 1$

2.2.1 – partial fraction:(casual system)

At first we get the numerator and denominator coefficients with **tfdata** function for our transfer functions. Then we just need to pass the numerator and denominator coefficients to **residuez** function and it will gives us the partial fraction of our transfer functions which are:

$H_1(z)$

```
ro1 =  
  
0.5000 + 0.5000i  
0.5000 - 0.5000i
```

```
po1 =  
  
0.5000 + 0.5000i  
0.5000 - 0.5000i
```

```
ko1 =
```

```
[]
```

$H_2(z)$

```
ro2 =  
  
0.0000 - 1.0000i  
0.0000 + 1.0000i
```

```
po2 =  
  
0.4330 + 0.2500i  
0.4330 - 0.2500i
```

```
ko2 =
```

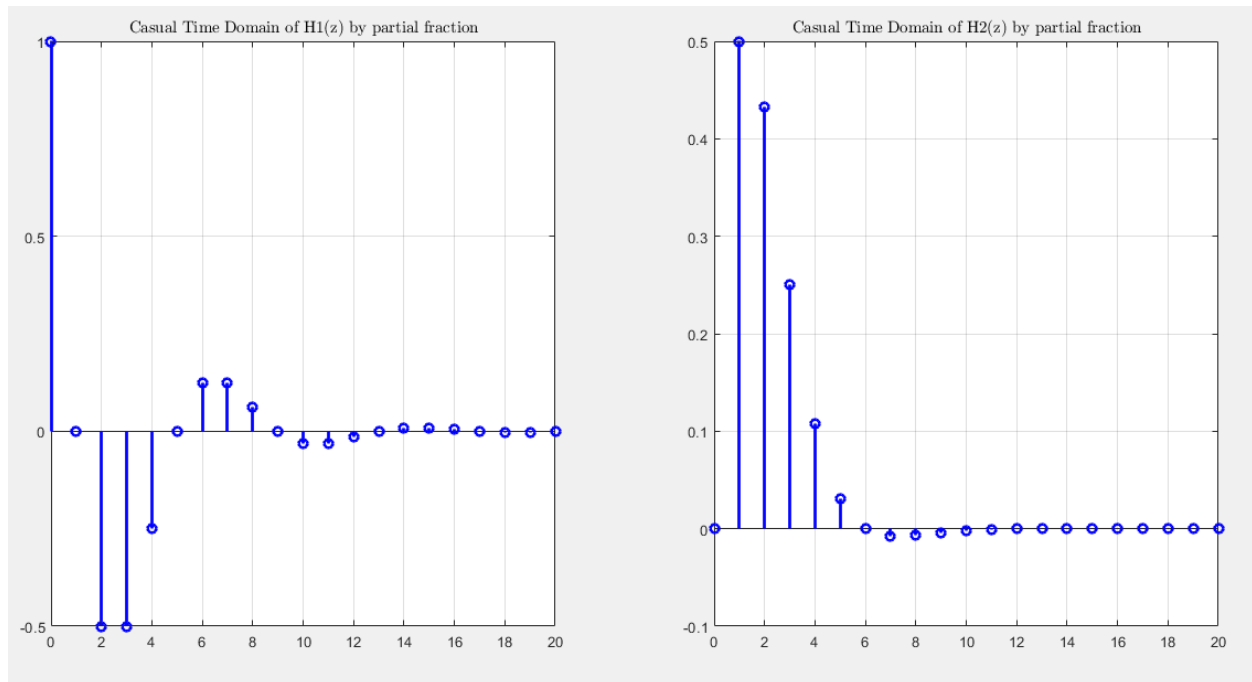
```
[]
```

Now that we have the coefficients, time domain forms of our systems are:(casual systems)

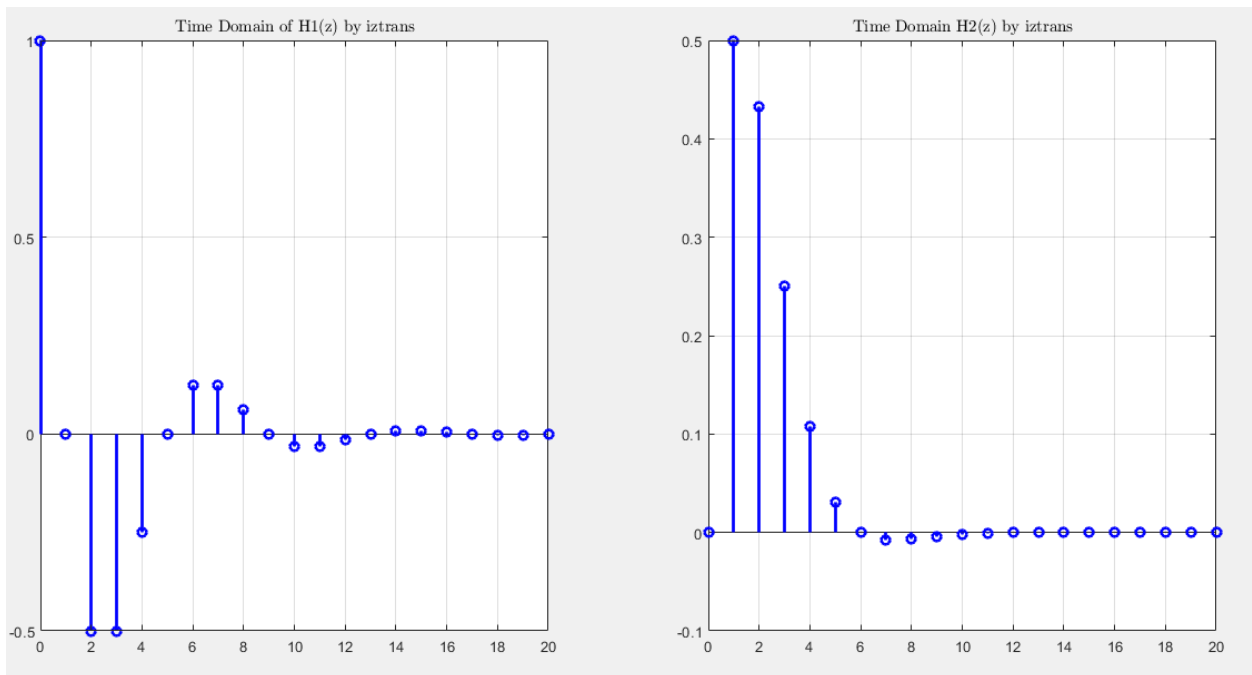
$$h_1[n] = [(0.5 + 0.5i)(0.5 + 0.5i)^n + (0.5 - 0.5i)(0.5 - 0.5i)^n]u[n]$$

$$h_2[n] = [(-i)(0.433 + 0.25i)^n + (i)(0.433 - 0.25i)^n]u[n]$$

Time domains of H1 and H2:



2.2.3 – iztrans:



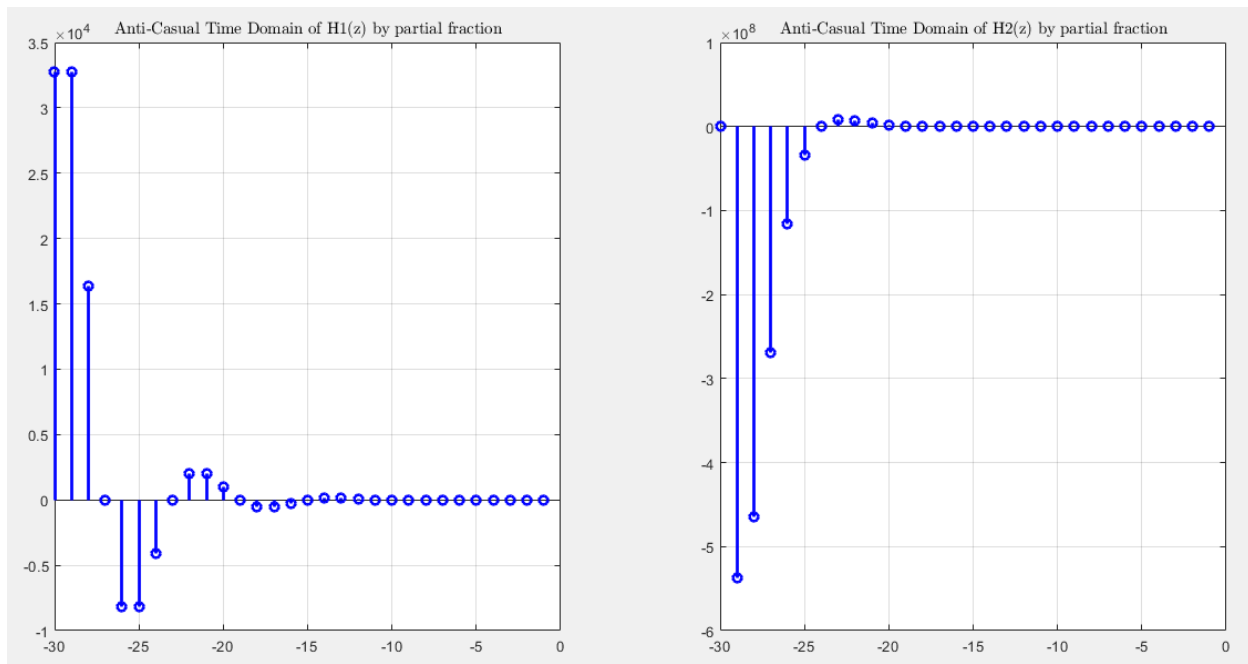
As we can see, the results with iztrans is just like when we calculate partial fractions in the previous part.

2.2.4 – partial fraction(anti – casual system):

This time our system is anti casual so, our time domain functions are:(we have calculate the partial fraction in part 2,2,2 and now we just have to declare our function)

$$h1[n] = [-(0.5 + 0.5i)(0.5 + 0.5i)^n + (-0.5 + 0.5i)(0.5 - 0.5i)^n]u[-n-1]$$

$$h2[n] = [(i)(0.433 + 0.25i)^n - (i)(0.433 - 0.25i)^n]u[-n-1]$$



We can't calculate a time domain form of an anti casual systems because they're left handed and if iztrans considers the system is right handed. So the result would be wrong. So the answer is NO!

2.3 – Difference equations: $y[n] - 0.7y[n - 1] + 0.49y[n - 2] = 2x[n] - x[n - 1]$

2.3.1 –

At first we get z transform of each side of the equation on the paper and we have the transfer function now:

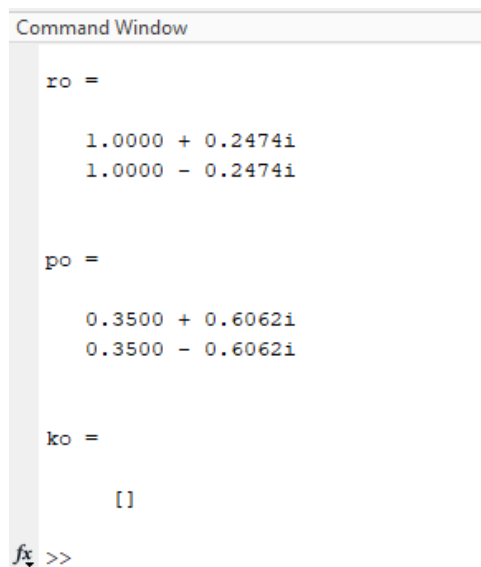
$$Y(z) - 0.7z^{-1}Y(z) + 0.49z^{-2}Y(z) = 2X(z) - z^{-1}X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{2 - z^{-1}}{1 - 0.7z^{-1} + 0.49z^{-2}}$$

After this we can calculate every thing with matlab:

Then we do partial fraction by the function, **residuez**, and because of the system that is casual, our impulse response is:

❖ Coefficients of the partial fraction:



```
Command Window

ro =

    1.0000 + 0.2474i
    1.0000 - 0.2474i

po =

    0.3500 + 0.6062i
    0.3500 - 0.6062i

ko =

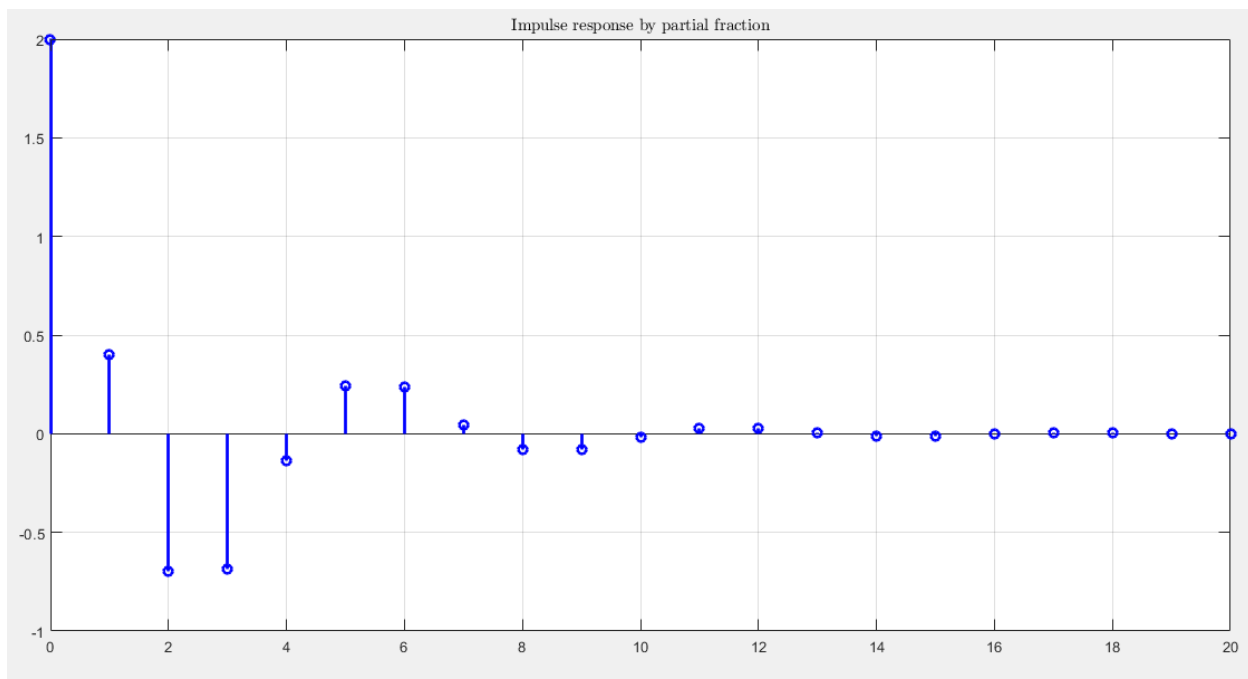
    []

fx >>
```

So our impulse response is:

$$h[n] = [(1 + 0.2474i)(0.35 + 0.6062i)^n + (1 - 0.2474i)(0.35 - 0.6062i)^n]u[n]$$

Impulse response plot:



2.3.2 –

We now we can write $h[n]$ like this form: $h[n] = \sum_i (\alpha_i p_i^n) u[n]$

Which p_i 's are the poles of our systems. As we calculate in the first part our transfer function is: $H(z) = \frac{Y(z)}{X(z)} = \frac{2 - z^{-1}}{1 - 0.7z^{-1} + 0.49z^{-2}}$

As we see our systems has 2 poles so $h[n]$ includes of 2 parts.

So, we need 2 initial states to find α_1 and α_2 :

If we calculate, $h[0] = 2$ and $h[1] = 0.4$

So now we now our $h[n]$ is in the form below:

$$h[n] = [\alpha_1(0.35 + 0.6062i)^n + \alpha_2(0.35 - 0.6062i)^n]u[n]$$

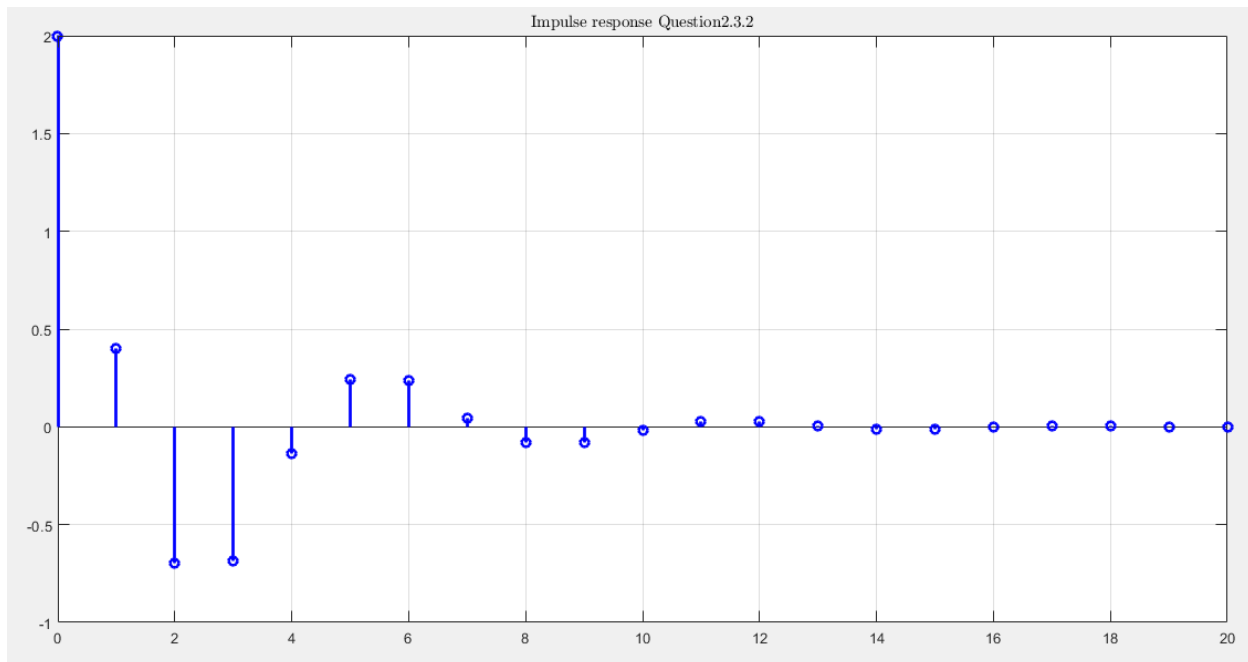
so we have 2 equations and α_1 and α_2 . So we can solve this equation by the **solve** function and it will give us α_1 and α_2 :

```
alpha1 =  
1 + 750i/3031
```

```
alpha2 =  
1 - 750i/3031
```

exactly like the previous part.

Impulse response plot:



✓ So our calculations in part one and our results in here are the same.

2.3.3 –

Now we use **filter** function to find the impulse response of this system. **filter** has three inputs, **filter(a,b,x)**. This function filters the input **x** using a rational transfer function defined by its numerator and denominator coefficients that are **a** and **b**. **filter** output is a vector or matrix with the size of the input vector or matrix. Again we can calculate numerator and denominator like all previous questions with **tfdata**. We can make an impulse input:

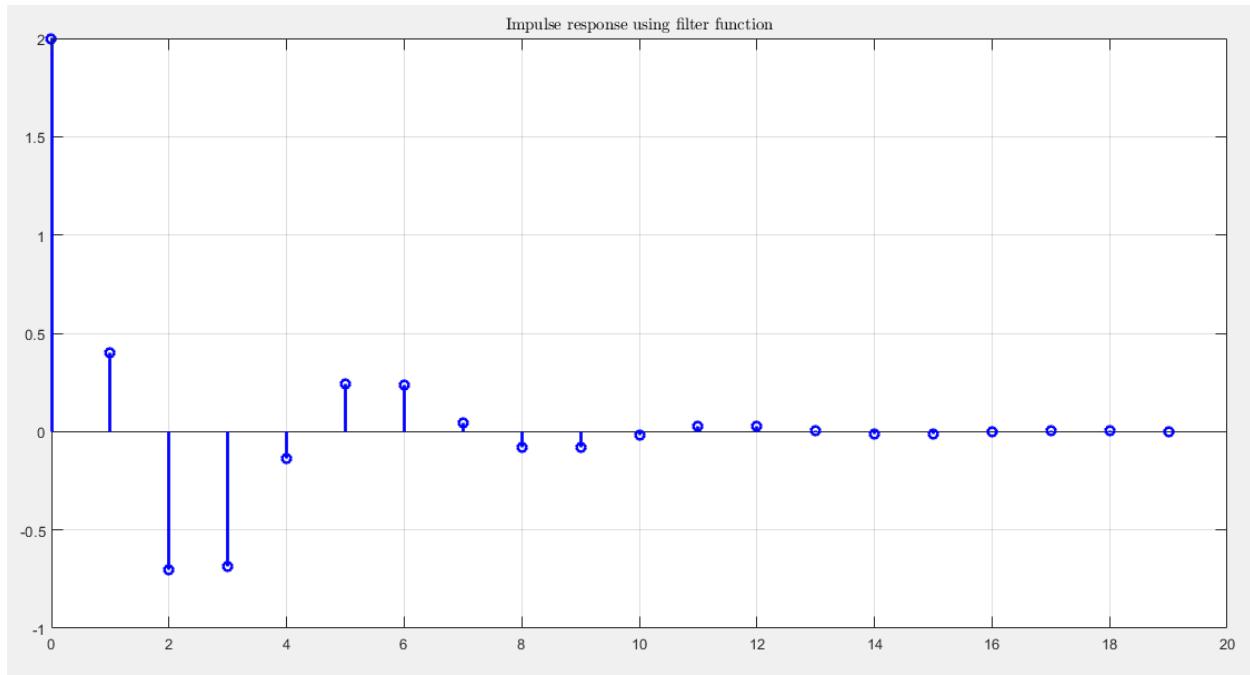
```
x = zeros(1,30);  
x(1) = 1;
```

and then we just have to use **filter** function:

```
h = filter(numerator,denominator,x);
```

and **h** is a vector of impulse response values.

Impulse response plot:

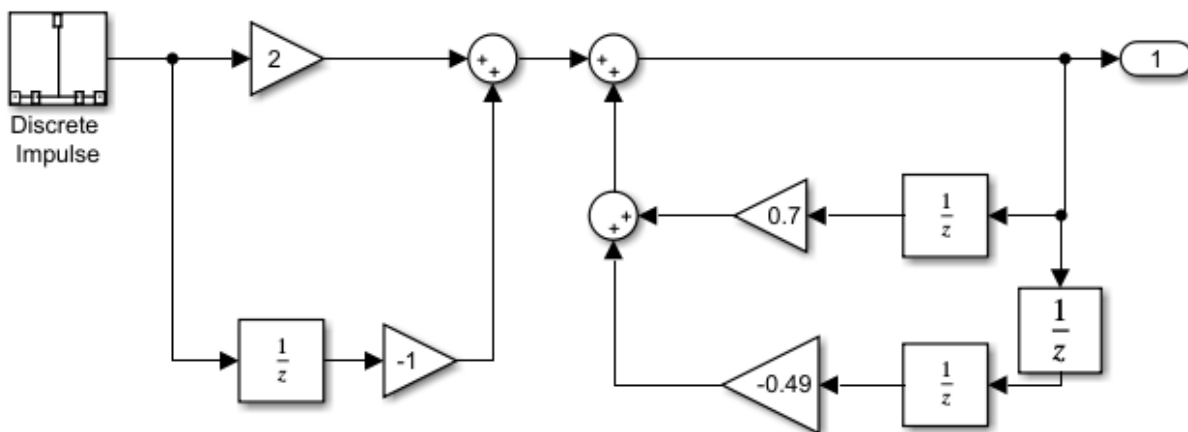


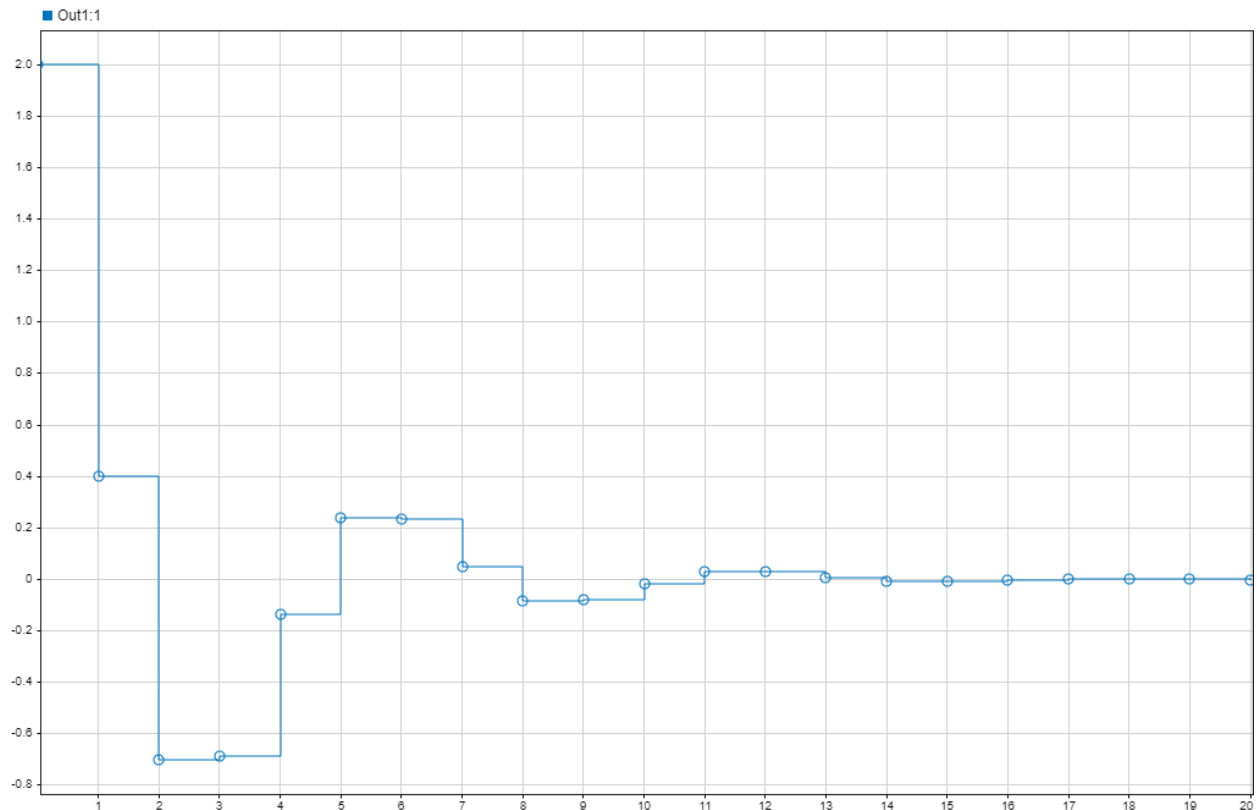
✓ So our calculations in part one and two and our results in here are the same.

2.3.4 –

our system in the Simulink:

$$y[n] - 0.7y[n - 1] + 0.49y[n - 2] = 2x[n] - x[n - 1]$$

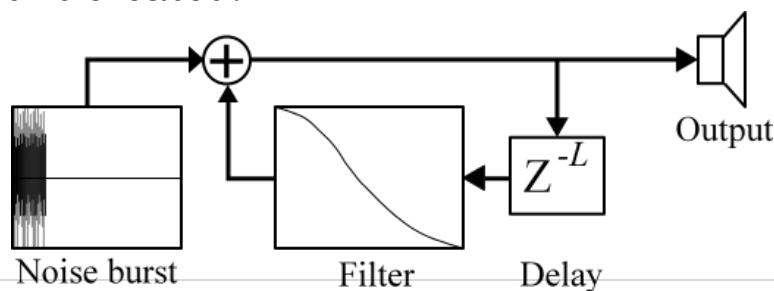




✓ So our calculations in the first three parts and our results in here are the same.

2.4 – Karplus-Strong:

Karplus–Strong string synthesis loops a wave through a filtered delay and it will simulate the sound of string. The input can be an excitation waveform, a white noise for example. This input pass through a delay and then pass through a filter with the gain lower than 1 to maintain a stable loop, which it actually will do averaging between 2 samples and then the output of this filter will be a feedback for the system and in this way, the output sound is created.



- To design this system in Matlab, at first we need the transfer function of system's difference equation which the coefficients of the numerator and denominator are defined in the code. We just need to pass these coefficients and the input x to the **filter** function and it will calculate the y for us. There is a for which does it for every note and create the y and at the end of each loop play the sound using **sound** function.

Length of the y should be duration of each note \times FS / 1000.

I just have defined 12 notes in the code which some of the are from the octave 4 and some of the are from octave 5. This a list of them:

A , A# , C , C# , B , E , D , D# , F , F# , G , G#

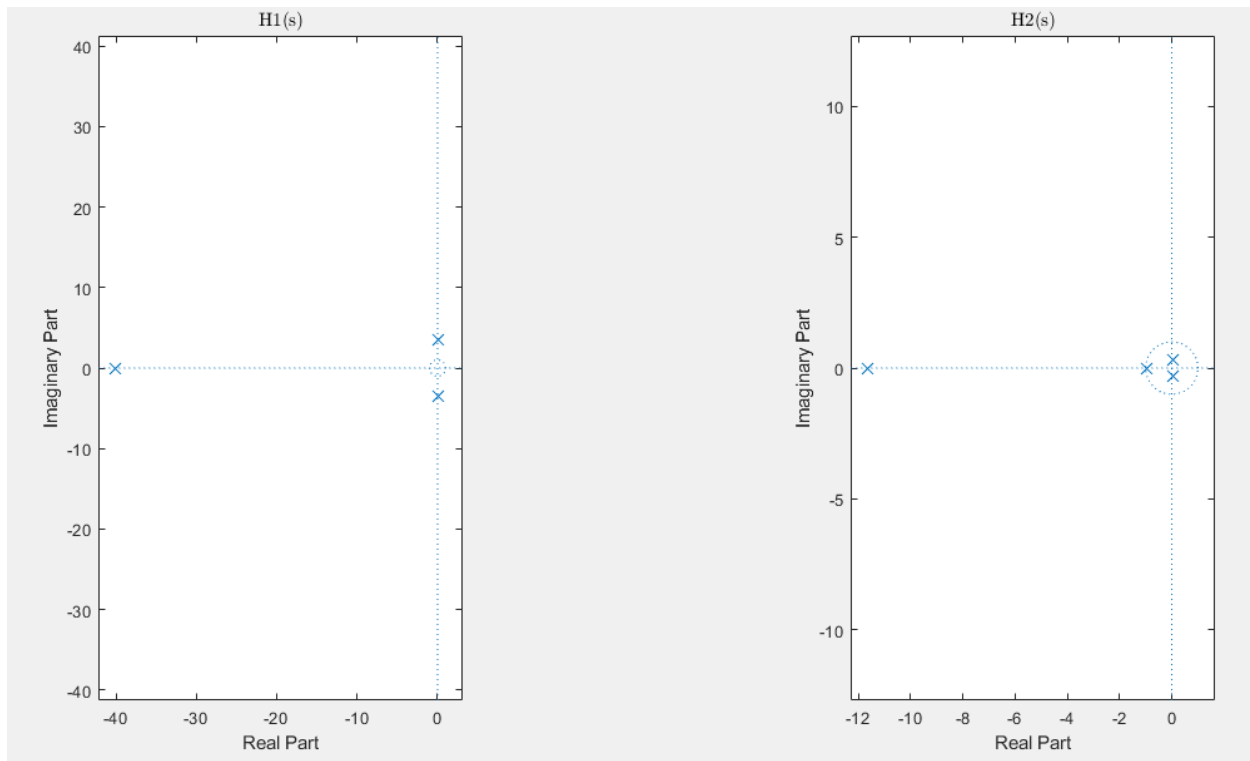
Nothing else go and play it ☺ (Imagine Dragons – Demons!)

3.1 – Laplace Transform: 3.1.1:

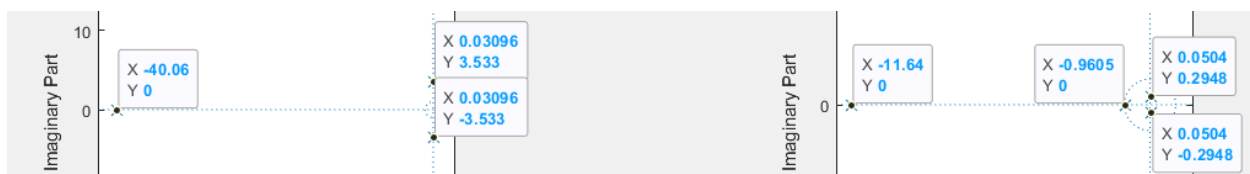
We can calculate the numerator and denominator of each transfer function with the **numden** function and then find their roots and that would be a our zeros and poles. (but systems in here doesn't have any zeros). Then we use **zplane** to plot the poles:

$$H_1(s) = \frac{1}{s^3 + 40s^2 + 10s + 500}$$

$$H_2(s) = \frac{1}{s^4 + 12.5s^3 + 10s^2 + 1}$$

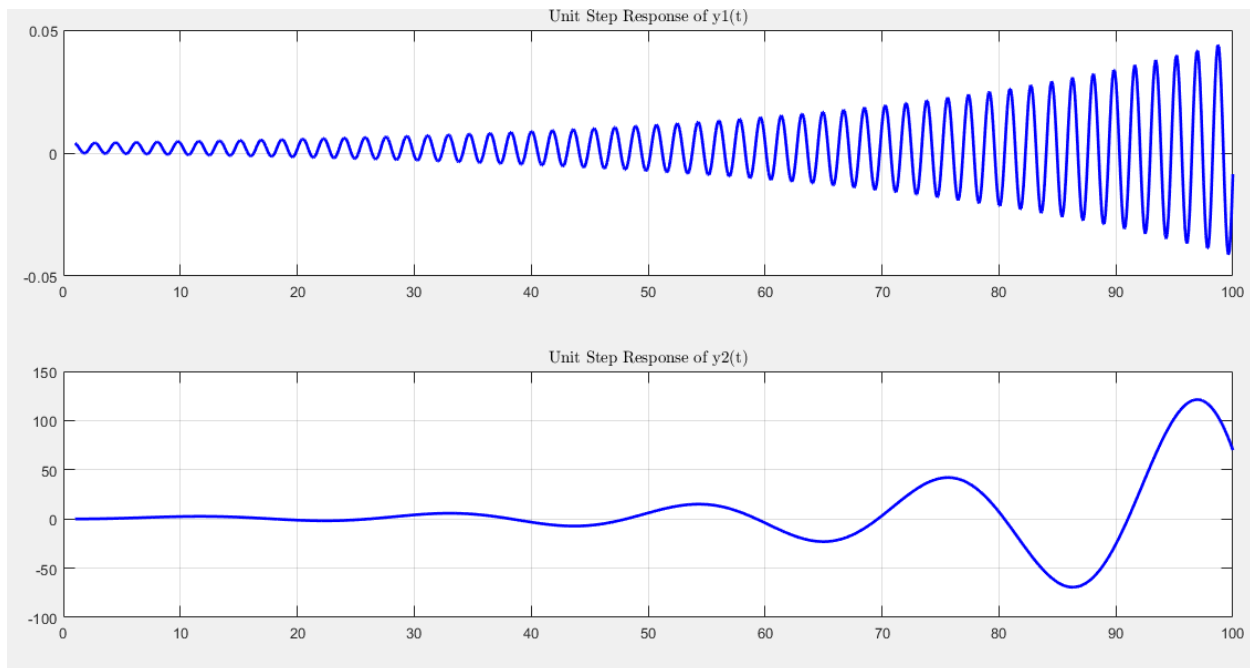


If we assume the systems are right handed, if we look at the most right poles, they're after the imaginary axis and so the ROCs don't include the $j\omega$ axis and so they're both **unstable**.



3.1.2: unit step responses of H1 and H2 –

(the code of this part may take a little bit longer to run because of number of the samples)



As we can see, the unit step responses both diverge and it shows that the systems are both **unstable** because the inputs are limited and the outputs diverge.

3.1.3 –

We want to calculate impulse and unit step responses of $G(s)$ for $a = 4/5/6$:

$$G(s) = \frac{2s+1}{s^2+as+7}$$

We use **ilaplace** and it will gives us the inverse Laplace transform. The ilaplace of $G(s)$ gives us the impulse response of the system and the ilaplace of $G(s)/s$ gives us the unit step response of the sysem:

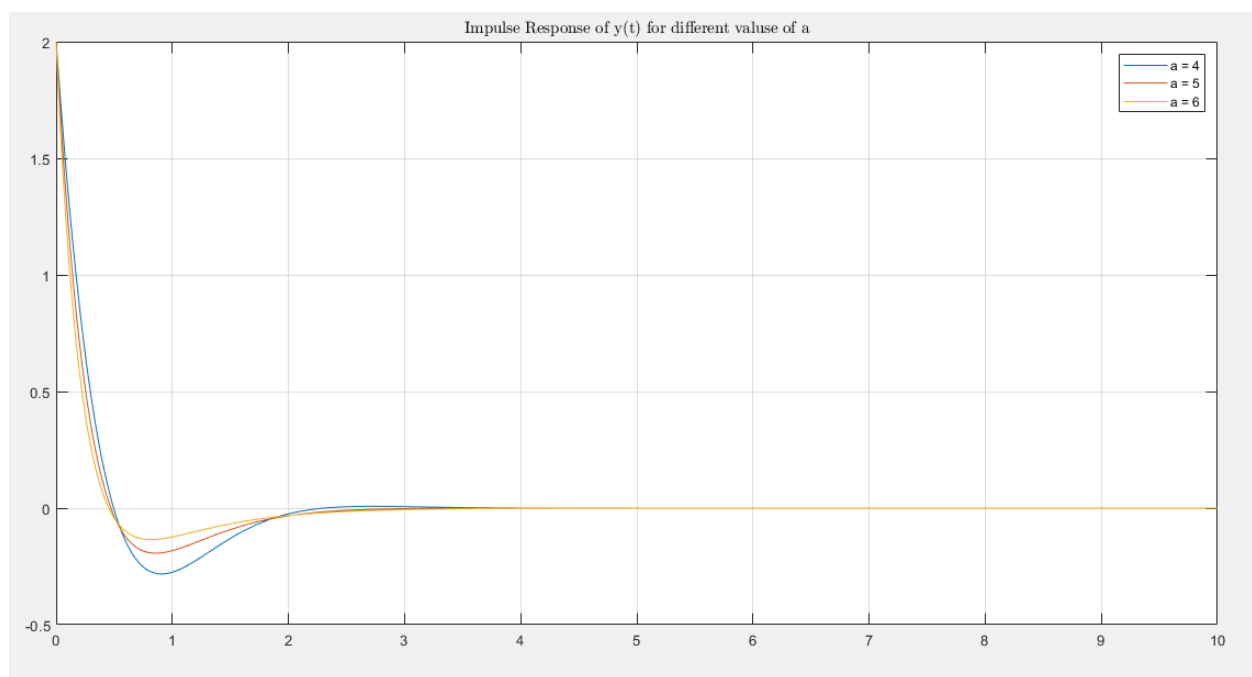
```
%impulse responses of the 3 systems
h1_inv(n) = ilaplace(H1,s,n)
h2_inv(n) = ilaplace(H2,s,n)
h3_inv(n) = ilaplace(H3,s,n)

%unit step response responses of the 3 systems
h1_inv(n) = ilaplace(H1/s,s,n);
h2_inv(n) = ilaplace(H2/s,s,n);
h3_inv(n) = ilaplace(H3/s,s,n);
```

Impulse responses:(a = 4/5/6)

```
h1_inv(n) =  
  
2*exp(-2*n)*(cos(3^(1/2)*n) - (3^(1/2)*sin(3^(1/2)*n))/2)  
  
h2_inv(n) =  
  
2*exp(-(5*n)/2)*(cos((3^(1/2)*n)/2) - (4*3^(1/2)*sin((3^(1/2)*n)/2))/3)  
  
h3_inv(n) =  
  
2*exp(-3*n)*(cosh(2^(1/2)*n) - (5*2^(1/2)*sinh(2^(1/2)*n))/4)
```

Impulse response plots:



Unit step responses:(a = 4/5/6)

Command Window

```
h1_inv(n) =

1/7 - (exp(-2*n)*(cos(3^(1/2)*n) - 4*3^(1/2)*sin(3^(1/2)*n)))/7

h2_inv(n) =

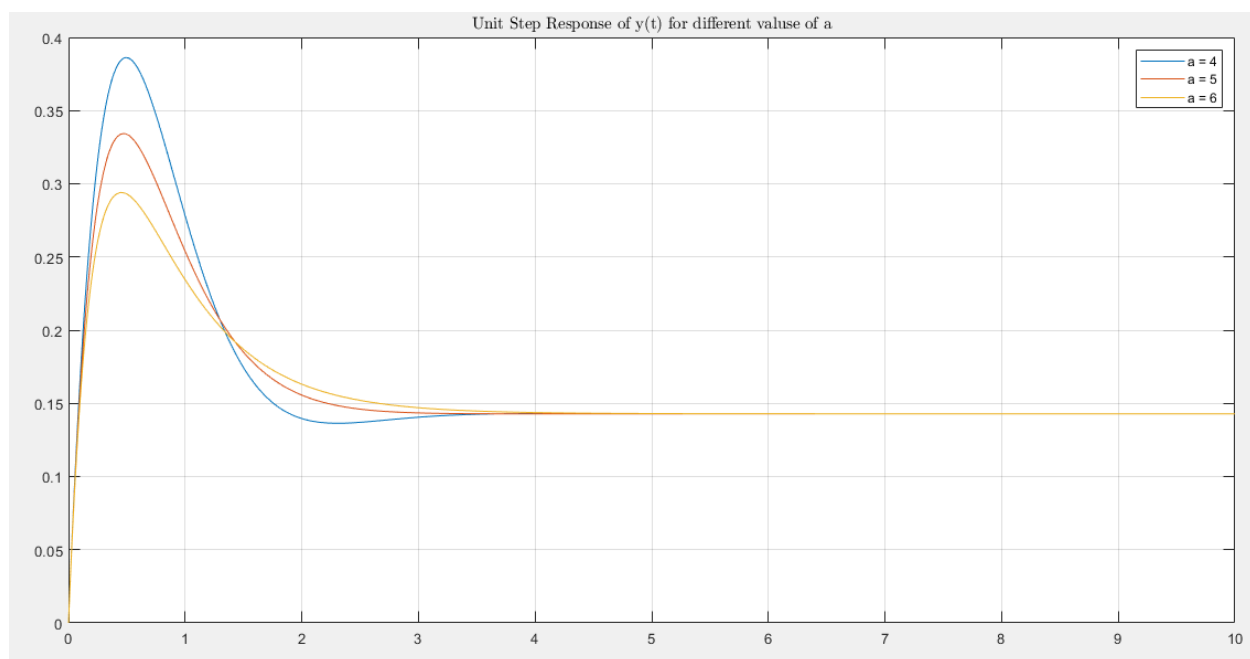
1/7 - (exp(-(5*n)/2)*(cos((3^(1/2)*n)/2) - (23*3^(1/2)*sin((3^(1/2)*n)/2))/3))/7

h3_inv(n) =

1/7 - (exp(-3*n)*(cosh(2^(1/2)*n) - (11*2^(1/2)*sinh(2^(1/2)*n))/2))/7
```

fx >> |

Unit step response plots:



a(Unit step response)	4	5	6
Steady State response	0,142857143	0,142857143	0,142857143
Maximum value	0.3862	0.3343	0,294
time of Maximum value	0,49-0.5	0,47	0.45
first time the response gets to half of the steady state response	0,037	0,038	0,039

By increasing a, steady state values increase a little bit and by increasing a, systems goes to it's steady state faster and time constants are lower.

And by increasing a, the maximum value of unit step responses gets lower.

3.1.4 – feedback:

$$a) H_1(s) = \frac{s+1}{s^2+3s+4}$$

$$b) H_2(s) = \frac{s+1}{s^3+3s^2+4s}$$

$$c) H_2(s) = \frac{s+1}{s^4+3s^3+4s^2}$$

We've got these systems and for creating a negative feedback, we use feedback function in the way below:

```
% use feedback function to create a feedback
H1_new = feedback(H1,1);
H2_new = feedback(H2,1);
H3_new = feedback(H3,1);
```

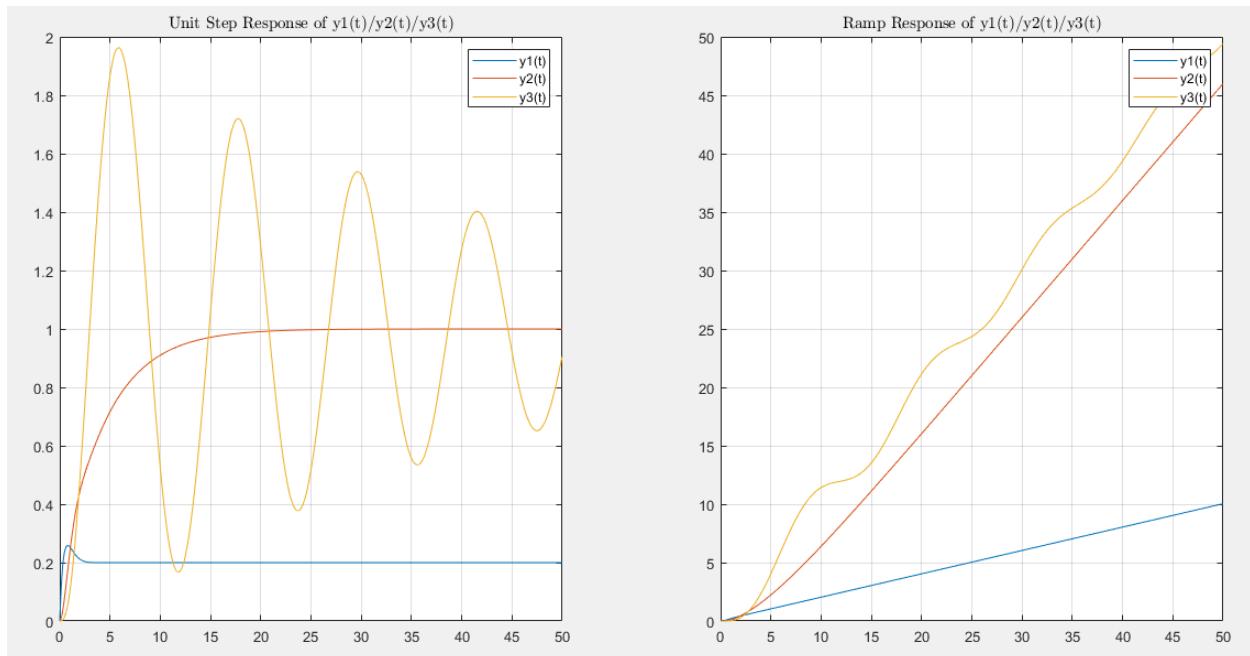
Now we have new transfer functions, we use **ilaplace** to find inverse Laplace transform of H1_new/s, H2_new/s, H3_new/s which gives us the unit step responses and we **lsim** function to find the ramp responses. **lsim** gets the transfer function, input and the domain and its output is the response.

We could use lsim for other previous parts rather than finding inverse Laplace transform and etc. That would make our code shorter.

- ❖ feedback function just takes tf's in its input and ilaplace takes symf so we have to change the H1/H2/H3_new to symf that I've done in the code.

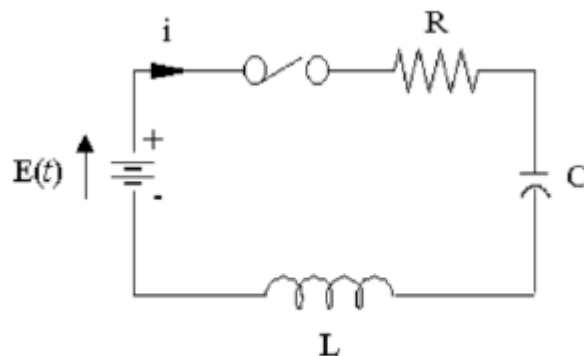
(the code of this part may take a little bit longer to run because of number of the samples)

Unit step/Ramp response Plots:



The difference between H1, H2, H3 is just number of poles that are zero. Addition of poles, zero to the transfer functions has the effect of making the system more unstable as we can see in the plots. It's because this causes the root locus pulls right and this makes the system less stable.

3.2 – analyze a RLC circuit with Laplace transform:



$$L = 3H, R = 16\Omega, C = 0.02F, E = 300V$$

Equations:

$$LC \frac{dv_c}{dt} + RC \frac{dv_c}{dt} + v_c = E(t)$$

$$0.06 \frac{dv_c}{dt} + 0.32 \frac{dv_c}{dt} + v_c = 300u(t)$$

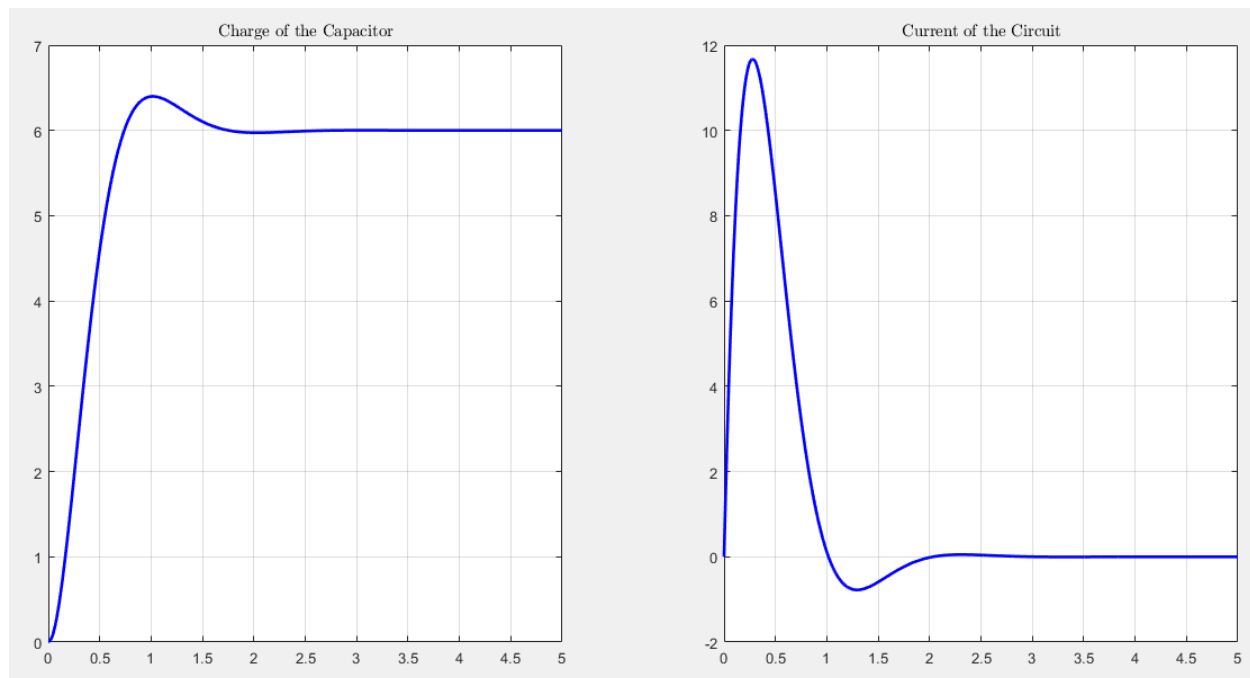
We get Laplace transform from both sides:

$$0.06sV(s) + 0.32sV(s) + V(s) = 300$$

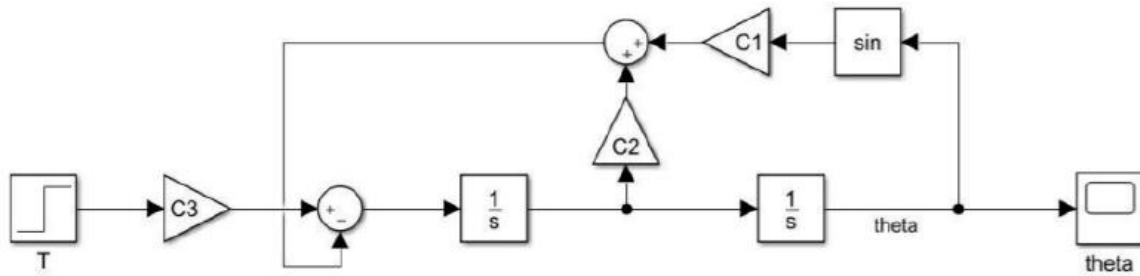
$$V(s) = \frac{300}{s(0.06s^2 + 0.32s + 1)}$$

Although we know $i = C \frac{dv_c}{dt}$. So, the Laplace transform of the current is:

$$I(s) = \frac{300}{s^2(0.06s^2 + 0.32s + 1)}, \quad Q(s) = CV(s) = \frac{6}{s(0.06s^2 + 0.32s + 1)}$$

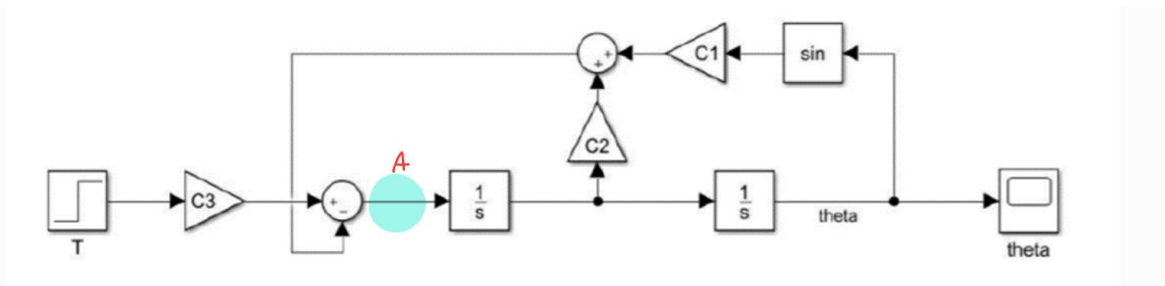


3.4 – Pendulum Model:



To find C1, C2, C3, we just have to find the difference equation of the block diagram and equalize it with $\frac{d^2\theta}{dt^2} + \frac{c}{ml} \frac{d\theta}{dt} + \frac{g}{l} \sin\theta = \frac{T}{ml^2}$

And so we have C1, C2, C3:



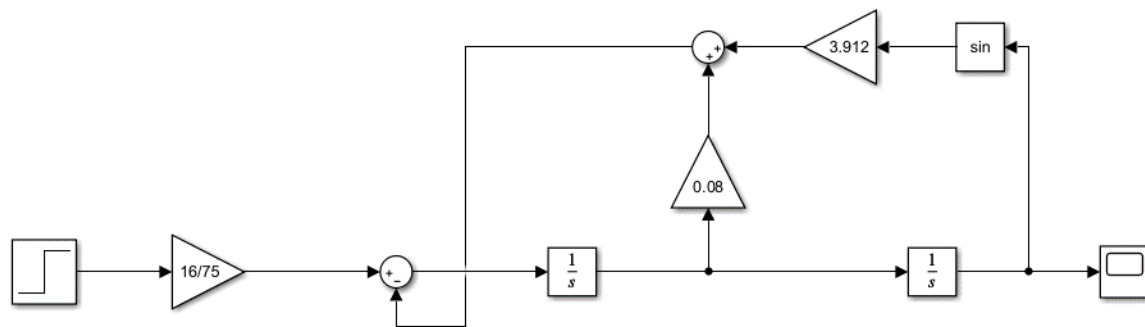
A is $\frac{d^2\theta}{dt^2}$ \leadsto So A is the second derivative

$$\leadsto \frac{d^2\theta}{dt^2} = C_3 T - C_2 \frac{d\theta}{dt} - C_1 \sin\theta = \frac{T}{ml^2} - \frac{c}{ml} \frac{d\theta}{dt} - \frac{g}{l} \sin\theta$$

$$\leadsto C_3 = \frac{1}{ml^2}, \quad C_2 = \frac{c}{ml}, \quad C_1 = \frac{g}{l}$$

$$C1 = \frac{g}{l}, \quad C2 = \frac{c}{ml}, \quad C3 = \frac{1}{ml^2}$$

Designing the block diagram in the Simulink:



Output:

