

# **PROFESSIONAL PORTFOLIO**

of

**ATUL SHROTRIYA**

Master of Science, Mechanical Engineering  
Concentration – Control Systems

**“I shall either find a way or make one.”**  
**-Hannibal Barca**

# **Index**

## **System Identification and Estimation**

Barnsley Fern and MATLAB, Simulink, State Variables	1-23
Discrete Equations and Recursive Estimation	24-59
Maximum Likelihood Estimation and Discrete-Time Kalman Filter	60-83
Extended Kalman Filter Analysis	84-97
System Identification Toolbox (MATLAB)	98-143
Neural Networks	144-198
Fuzzy Systems	199-230
Kalman Filter Bank for Failure Detection	231-251

## **Intelligent Control Systems**

State Variable Systems, Computer Simulation for van der Pol oscillator, Lorenz Attractor chaotic system and Volterra predator-prey system	252-264
Stock Market time Series Analysis	265-273
Digital Speech Processing and Machinery Monitoring using Discrete Fourier Transform Analysis	274-286
Recursive Least Squares System Identification and discrete time system simulation	287-296
Mobile Robot Control and Potential Fields	297-311
Neural Networks	312-323
Adaptive Control and Robust Control	324-334

<u>Independent Research - Comparison of different neural and fuzzy models for recognizing vowels within uncontrolled environments</u>	335-343
---	---------

## Dynamic Systems Modeling and Simulation

Spring Damper suspension system	344-348
Pendulum Motion	349-355
Impact Bumper System with fluids and entrapped air	356-360
Pressurized air tank with load	361-371
Inverted Pendulum Stabilization using Simulink	372-382

## Control System Components

Multipoint System and Hydraulic Actuator driving a load	383-390
Handheld Sprayer Design	391-396
Low Pass Filter Design	397-400
Servo-valve Review	401
Power Steering Derivation and analysis	402-407

## Projects

Object Recognition using Python and OpenCV  (May 20 - Present)	408-409
<u>Group Project - Cast Steel Gate Valve Design and Assembly using SolidWorks and Manufacturing Layout</u>  (Mar 20 - May 20)	410-435

**Hydraulic Cylinder Design and Assembly in SolidWorks**

436

(November 2020)

---

**Optimal Control of Cuff Pressure in Oscillometric**

437-442

**Devices (Mar 20 - May 20)**

## EE5327, Fall 2020

### Homework 1: MATLAB, Simulink, State Variable Review Due September 10, 2020

#### Problem 1) Random Numbers - 10 points

In MATLAB, do the following:

- a) Generate a sequence of 1000 uniformly distributed random numbers and plot them as dots in a figure window. Compute the average (mean) value and variance.
- b) Generate a sequence of 1000 normally distributed random numbers and plot them as dots in a figure window. Compute the average (mean) and variance. Use the data statistics feature in the figure window to confirm that you computed the mean correctly.
- c) Now use the uniformly distributed random numbers block in Simulink to generate 1000 numbers, take that data back into MATLAB, and compute the mean value and variance of that data.
- d) Similarly, use the appropriate normally distributed random number block in Simulink to generate 1000 numbers, take that data back to MATLAB, and computer the mean value and variance.

#### Problem 2) ODE, State Space, and Masked Subsystem - 20 points

Implement the following linear dynamics in Simulink by first converting the second order equation to a system of two first order ODEs (use two integrators)

$$\ddot{x} + b\dot{x} + kx = u$$

Using the standard step input for this problem and a stop time of 10 seconds. Choose a fixed step solver with step size of 0.01 seconds.

- a) Check out the simulation by setting  $b = 0.7$  and  $k = 5$  back at the MATLAB workspace. Plot the results and show your Simulink diagram.
- b) Now group all but the input block and scope into a subsystem. Rerun your simulation to confirm it still operates correctly and show your Simulink diagram.
- c) Use a state space block (use  $C = [1 0]$  and  $D = [0]$ ) to implement the system and compare the output of the state space block to the masked subsystem.
- d) Now put a mask on your subsystem so it takes the values of  $b$  and  $k$  from the mask. Confirm by setting the value of  $b$  in the mask to 0.4 and  $k$  to 4 and rerun. Plot the time history and show a screenshot of your mask.

#### Problem 3) State Space and Transfer Function Equivalence - 20 points

For the system in Problem 2 with  $b$  equal 0.7 and  $k$  equal 5, do the following:

- a) Create the equivalent numerator and denominator representing the transfer function in MATLAB using the ss2tf function.
- b) With these variables created in MATLAB, develop a Simulink model containing both the transfer function and state-space blocks. Compare their responses to a step input to see how they match.

**Problem 4)** Discrete Time Transfer Functions - 20 points

For the system in Problem 2 with b equal 0.7 and k = 5, do the following:

- a) To find the discrete transfer function, use the MATLAB command c2d to convert the continuous time transfer function to a discrete form with a sampling time of 0.1 sec. Show the discrete time transfer function.
- b) Implement the continuous time transfer function and the discrete time transfer function in Simulink to show how the two compare in response to a step input.

**Problem 5)** (Mangled) Barnsley Fern - 30 points

Implement the following equations in a MATLAB script.

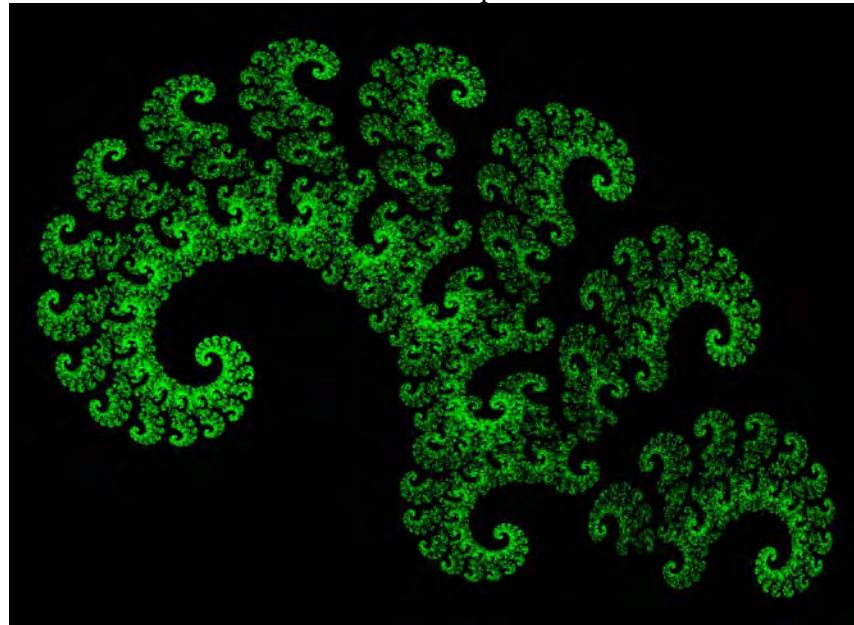
$$f_1 := \begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} 0.7873 & -0.3230 \\ 0.3230 & 0.7873 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix}$$

$$f_2 := \begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} 0.0841 & -0.3286 \\ 0.2930 & 0.0895 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix}$$

$$f_3 := \begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} -0.2458 & 0.1523 \\ 0.1722 & 0.3358 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} 0 \\ 0.44 \end{bmatrix}$$

This problem has you must generate uniform random numbers in the range (0,1) (help rand in MATLAB) and then use if-then rules in a MATLAB script to decide which of the 3 equations above you use for that point. Starting from  $[x, y] = [0, 0]$ , plot 10,000 points choosing equation  $f_1$  80% of the time, equation  $f_2$  10% of the time, and equation  $f_3$  10% of the time. Color each point green. When you're sure you have it working, plot 100,000 green points with MarkerSize of 1 – this may take a few minutes to run. An example of the expected result is shown below.

Problem 5 example result

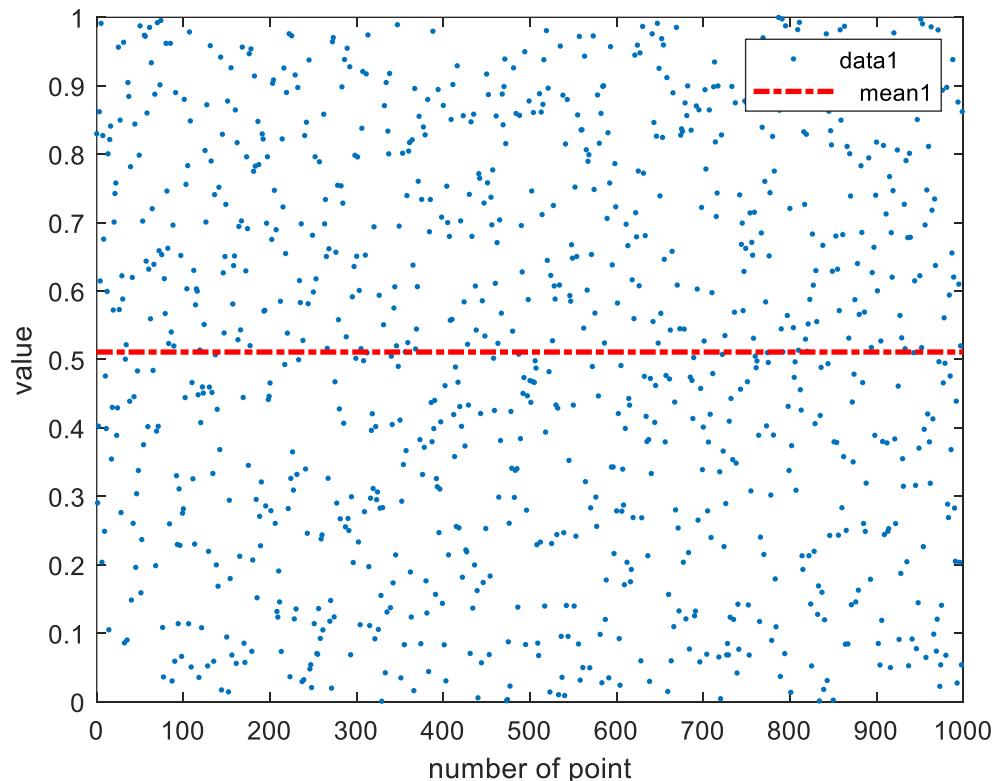


## Problem – 1:

a.

MATLAB code for both a and b parts:

```
function hw1q1
udran=rand(1000,1);
mean1=mean(udran)
var1=var(udran)
figure(1)
plot([0:1:999],udran,'.')
ndran=normrnd(0,1,1,1000);
mean2=mean(ndran)
var2=var(ndran)
figure(2)
plot([0:1:999],ndran,'.')
end
```



```

>> hw1ql1

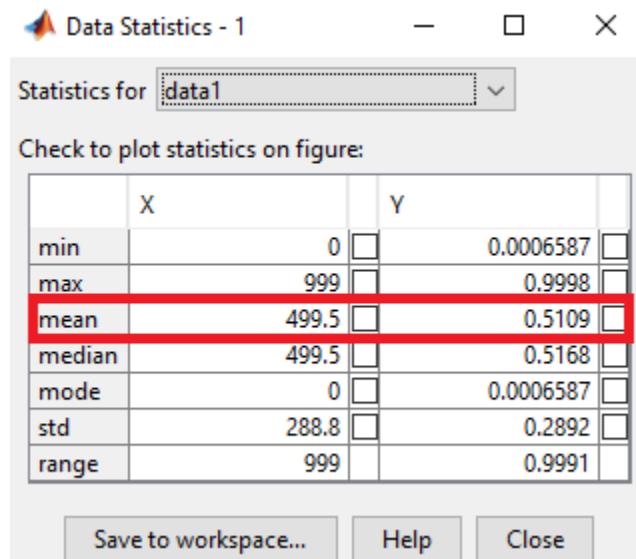
mean1 =
    0.5109

var1 =
    0.0836

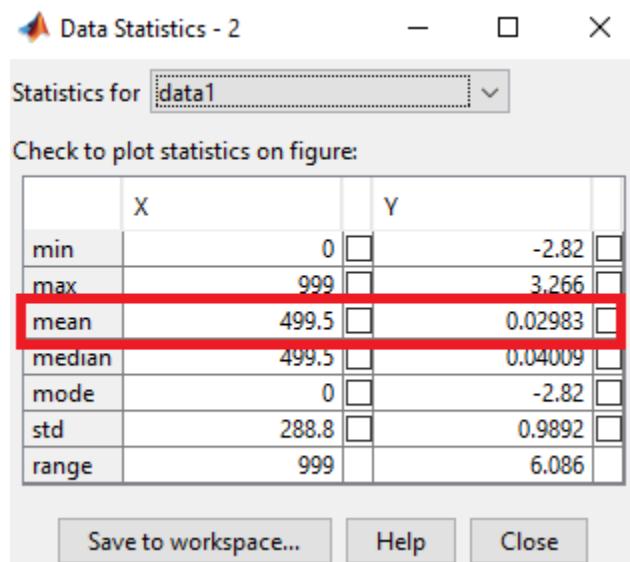
mean2 =
    0.0298

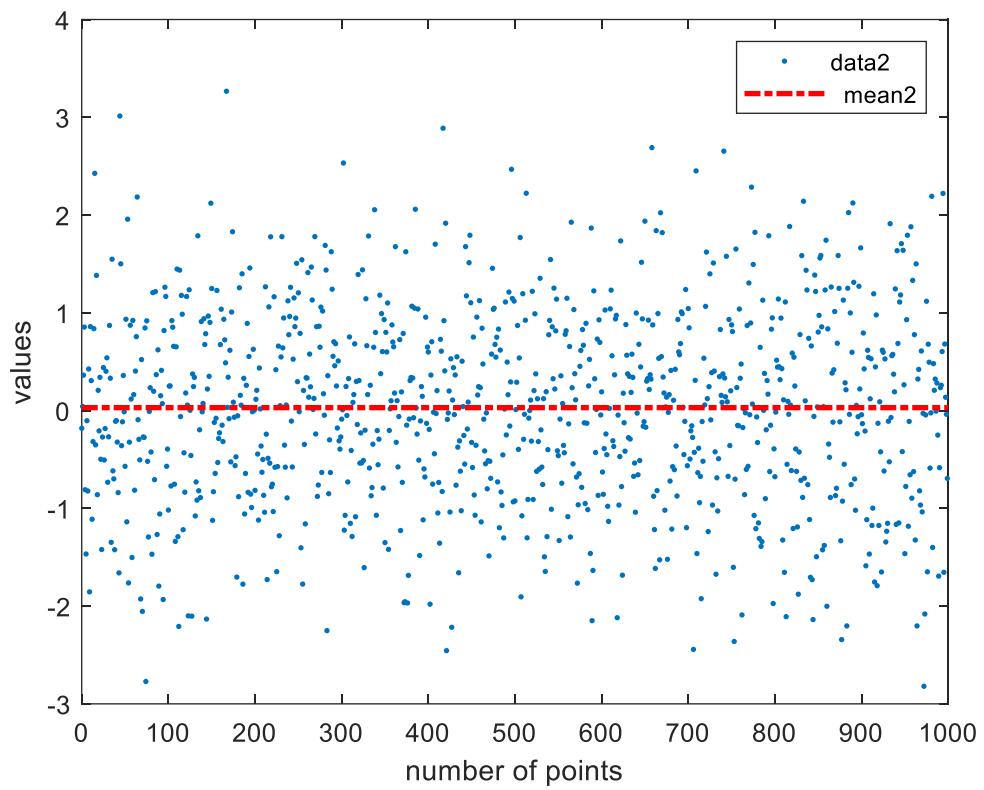
var2 =
    0.9785

```



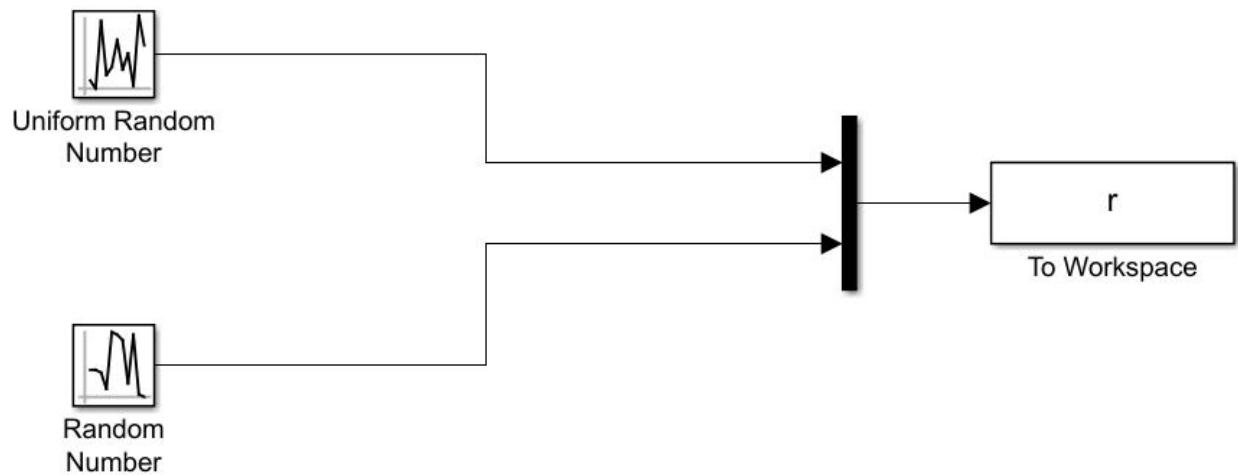
b.





c. Part d is also included through the same SIMULINK diagram.

Note: I could have drawn more than one simulink diagrams for most problems but I have accidentally configured something incorrectly and a lot of properties need to be defined before I can run it. Will try to fix it before next submission



```
>> mean3=mean(r(:,1))          >> mean4=mean(r(:,2))

mean3 =
0.0032

mean4 =
-0.0139

>> var3=var(r(:,1))           >> var4=var(r(:,2))

var3 =
0.3158

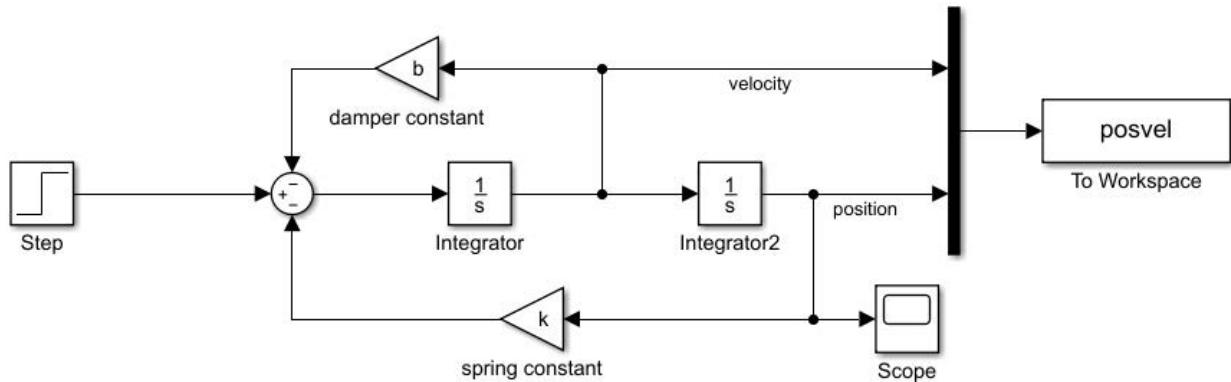
var4 =
1.0621
```

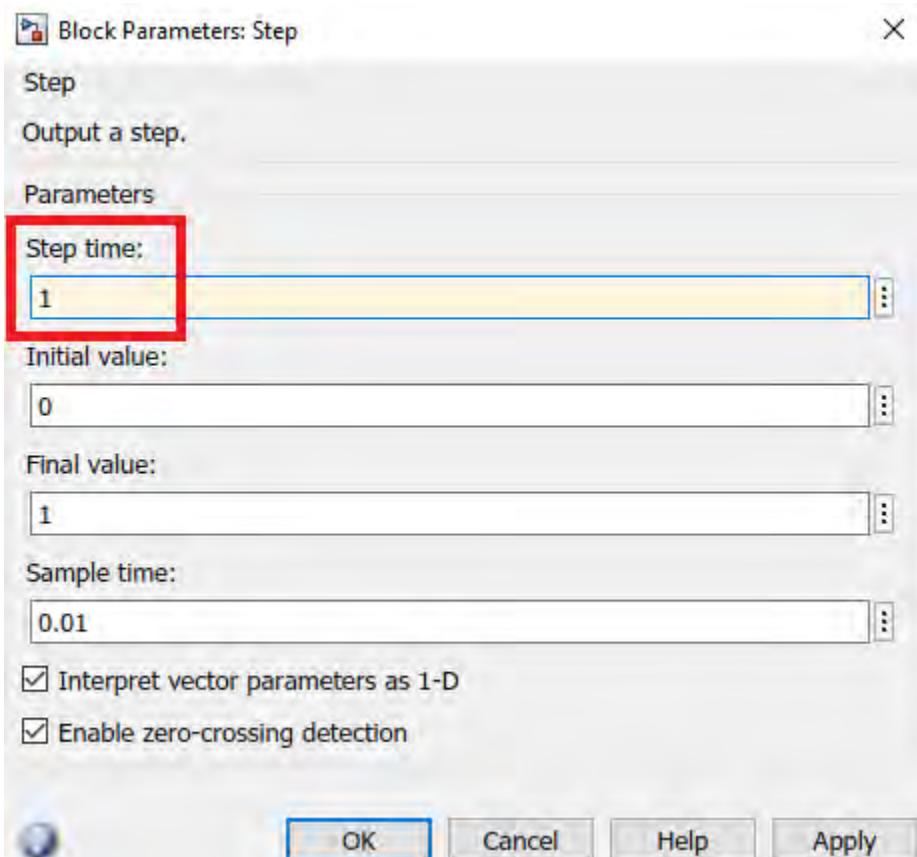
## Problem – 2:

Commands to define fixed parameters:

```
b=0.7;  
k=5;  
t=0:0.01:10;  
A=[0 1;-k -b];  
B=[0;1];  
C=[1 0];  
D=0;
```

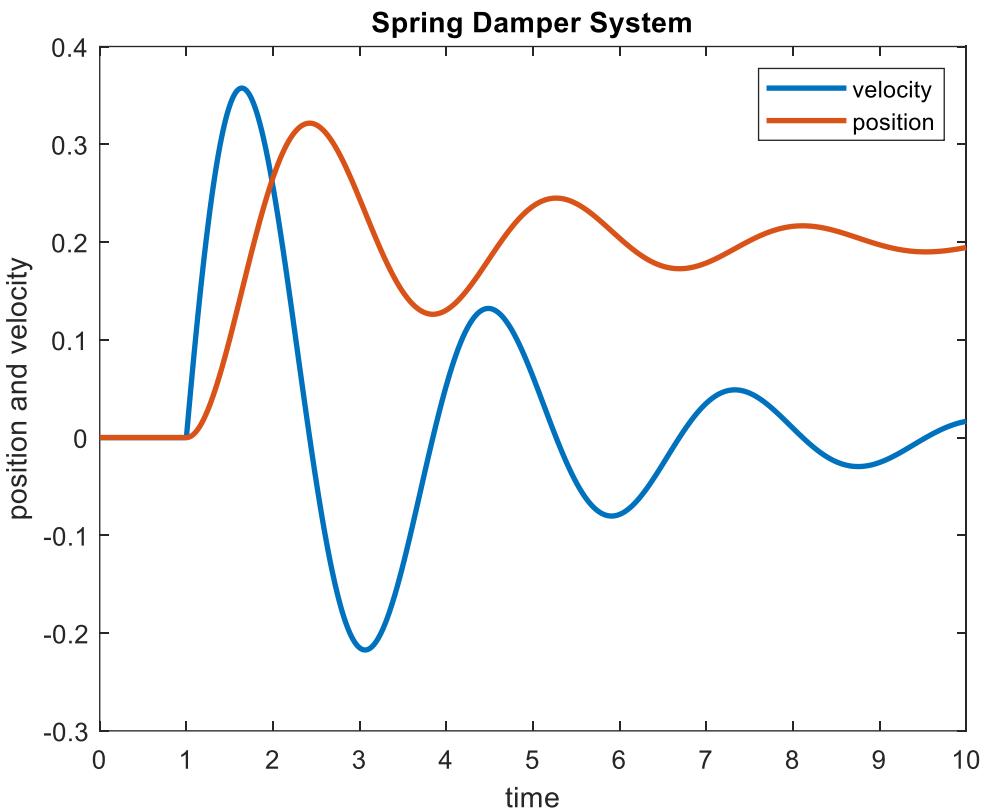
a.



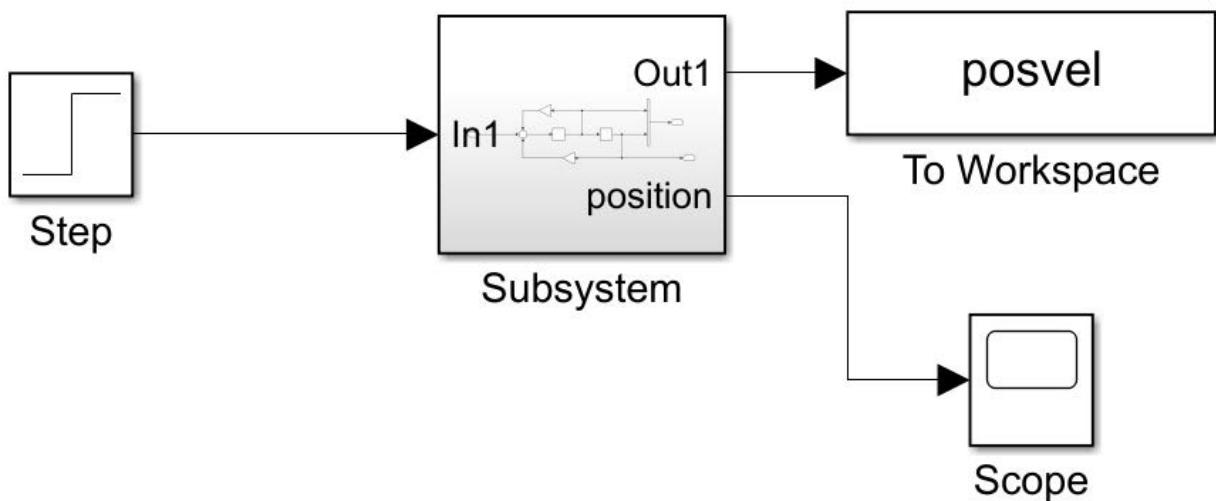


The plots start from 1 second as the step time is set at 1. This can be reset to zero or altered accordingly

**Plot command** - `plot(t,posvel(:,1),t,posvel(:,2))`

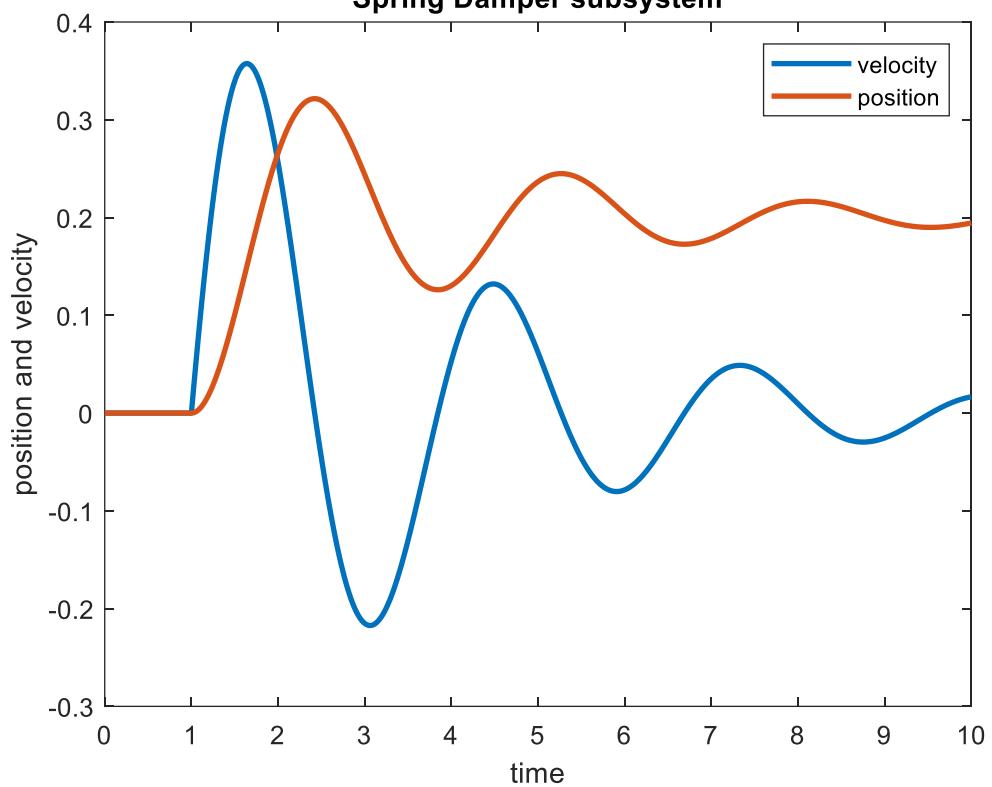


b.

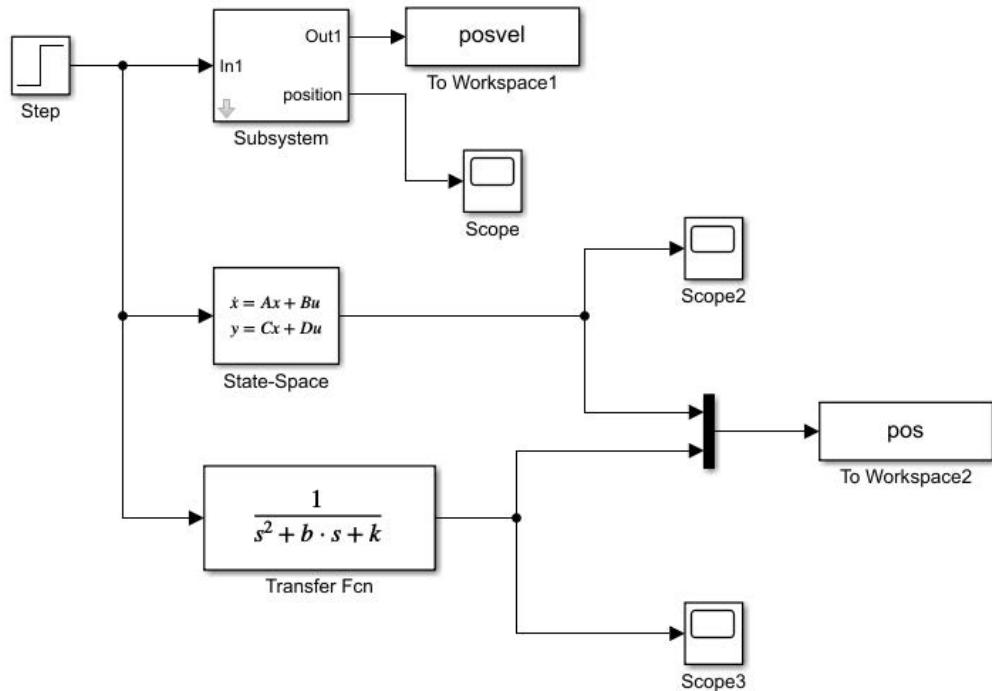


**Plot command** - `plot(t,posvel(:,1),t,posvel(:,2))`

### Spring Damper subsystem

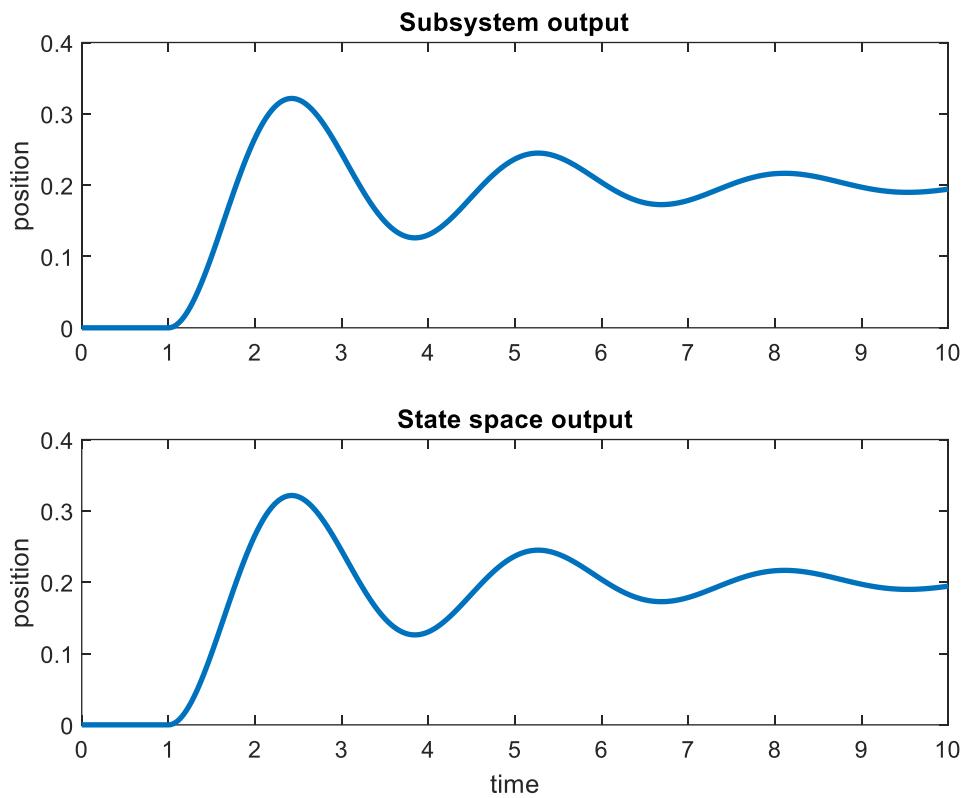


### c. Transfer function added for inclusion of Problem 3 part B



#### Plot commands

```
figure
subplot(2,1,1);
plot(t,posvel(:,2))
subplot(2,1,2);
plot(t,pos(:,1))
```

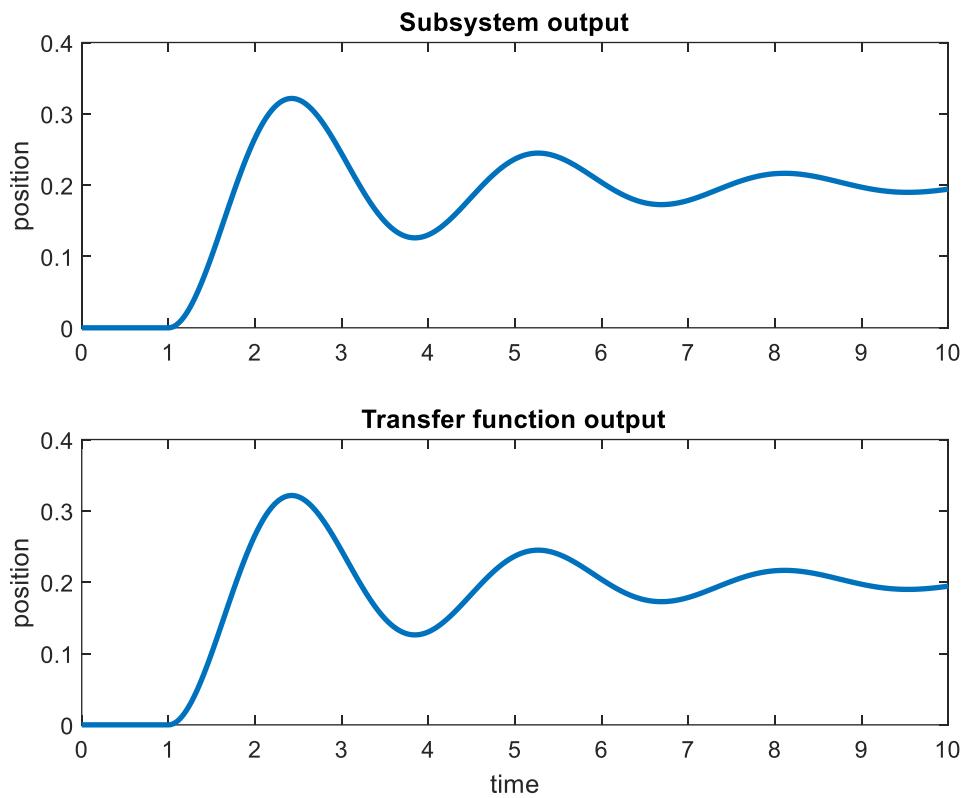


### Plot commands

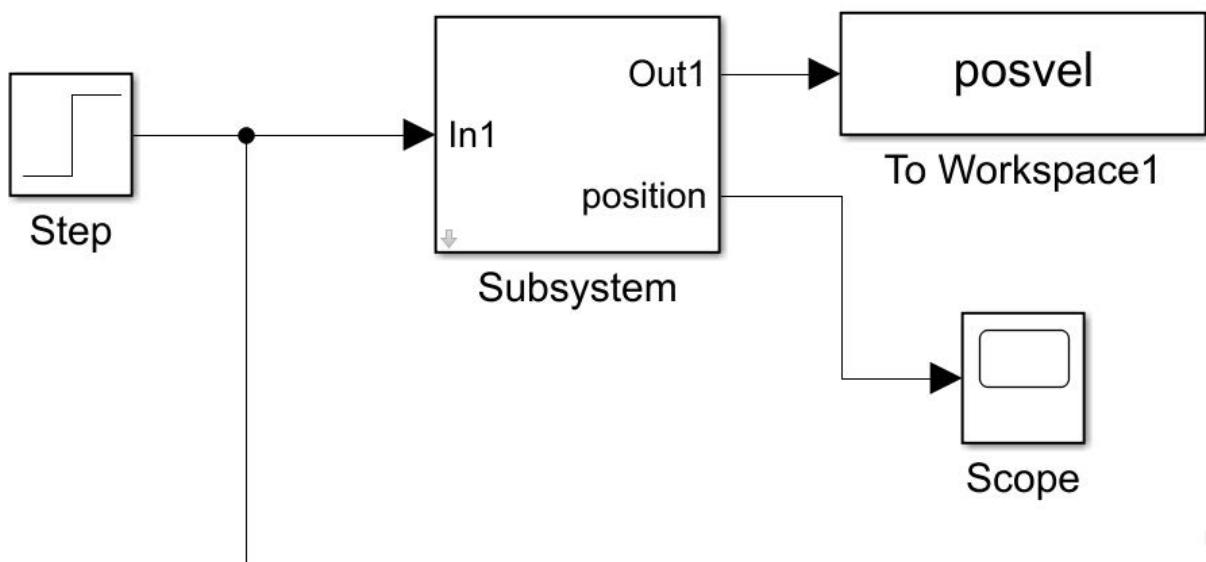
```

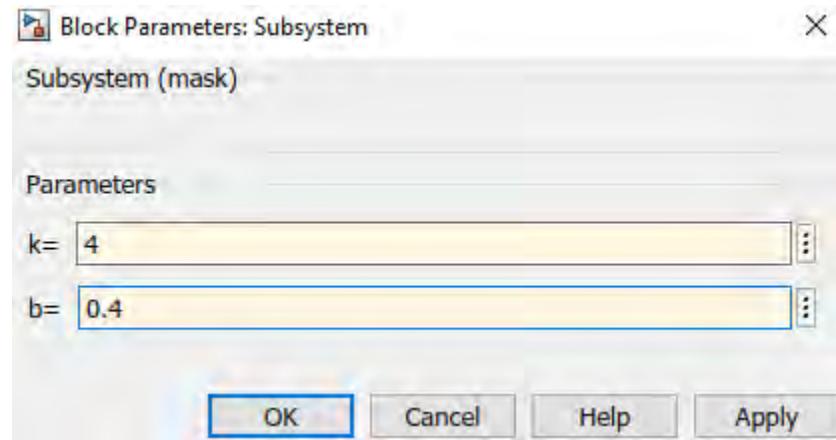
figure
subplot(2,1,1);
plot(t,posvel(:,2))
subplot(2,1,2);
plot(t,pos(:,2))

```

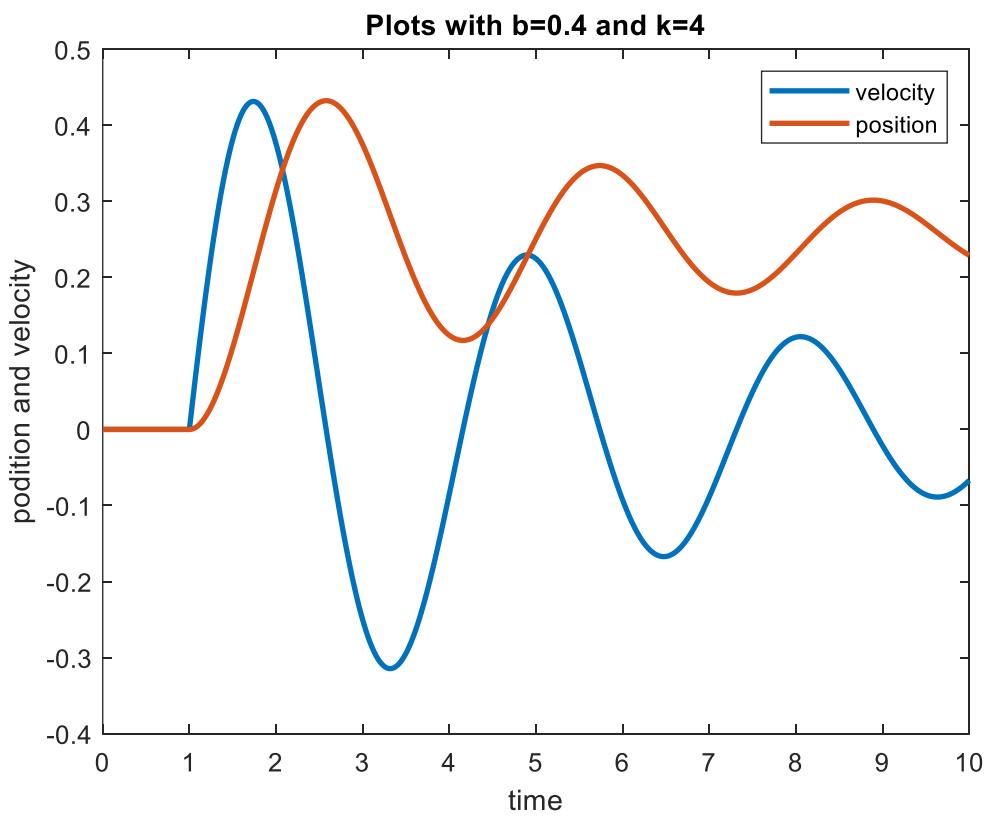


d.





**Plot command -** `plot(t, posvel(:,1), t, posvel(:,2))`



### Problem – 3:

a.

MATLAB commands for getting numerator and denominator

```
[num,den]=ss2tf(A,B,C,D)
```

```
num =
```

```
0 0 1
```

```
den =
```

```
1.0000 0.7000 5.0000
```

```
>> TF=tf(num,den)
```

```
TF =
```

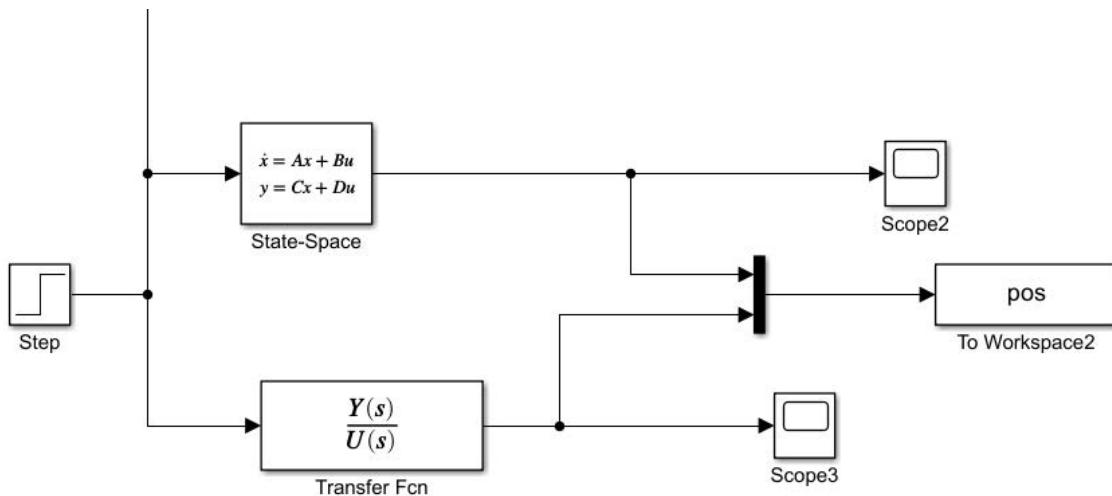
```
1
```

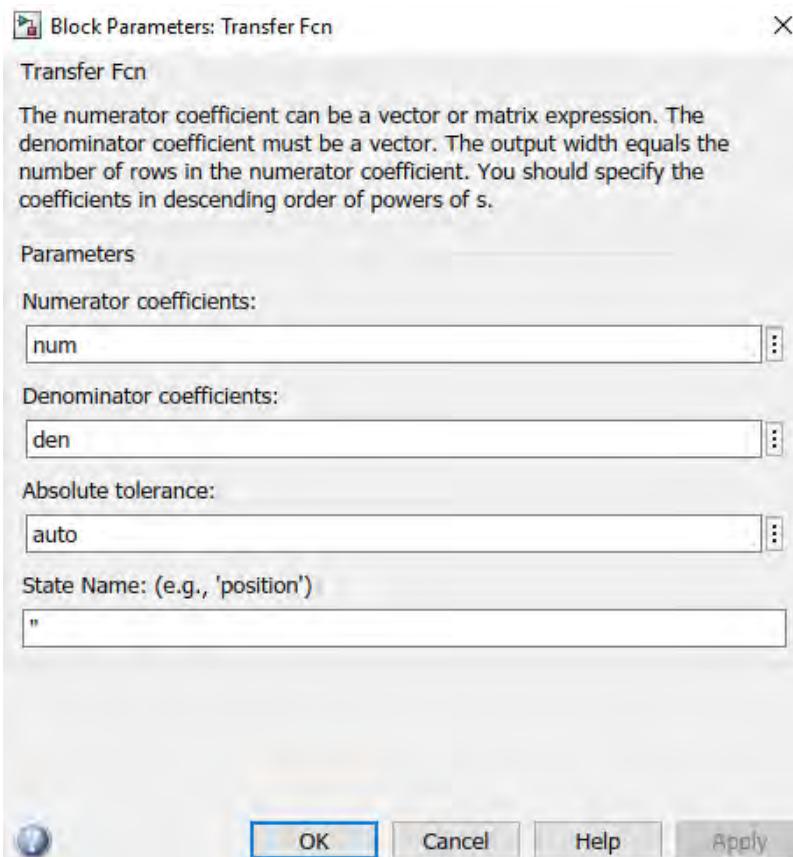
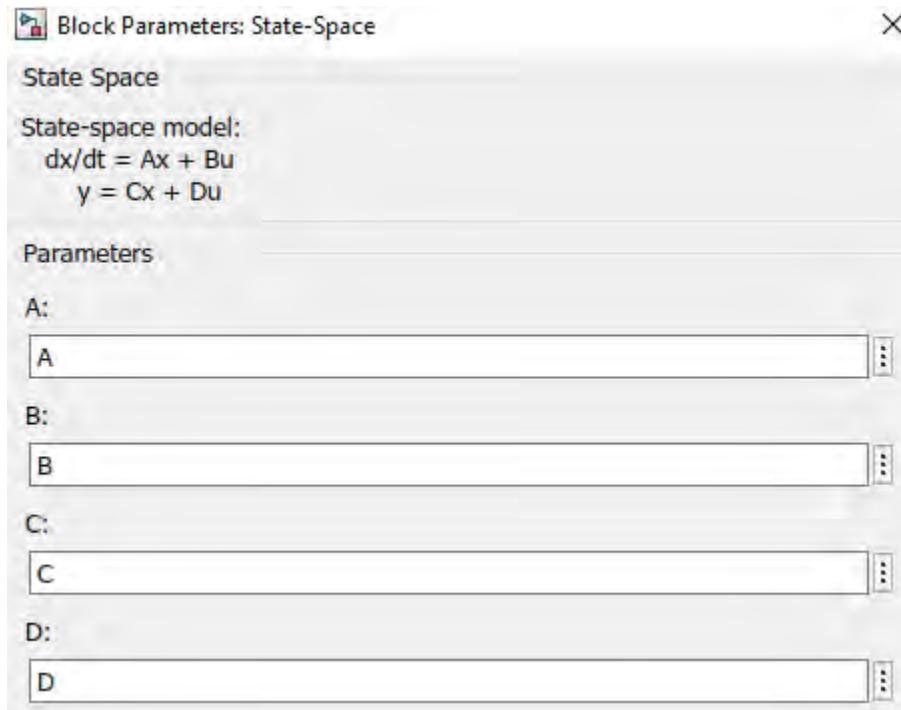
```
-----
```

```
s^2 + 0.7 s + 5
```

Continuous-time transfer function.

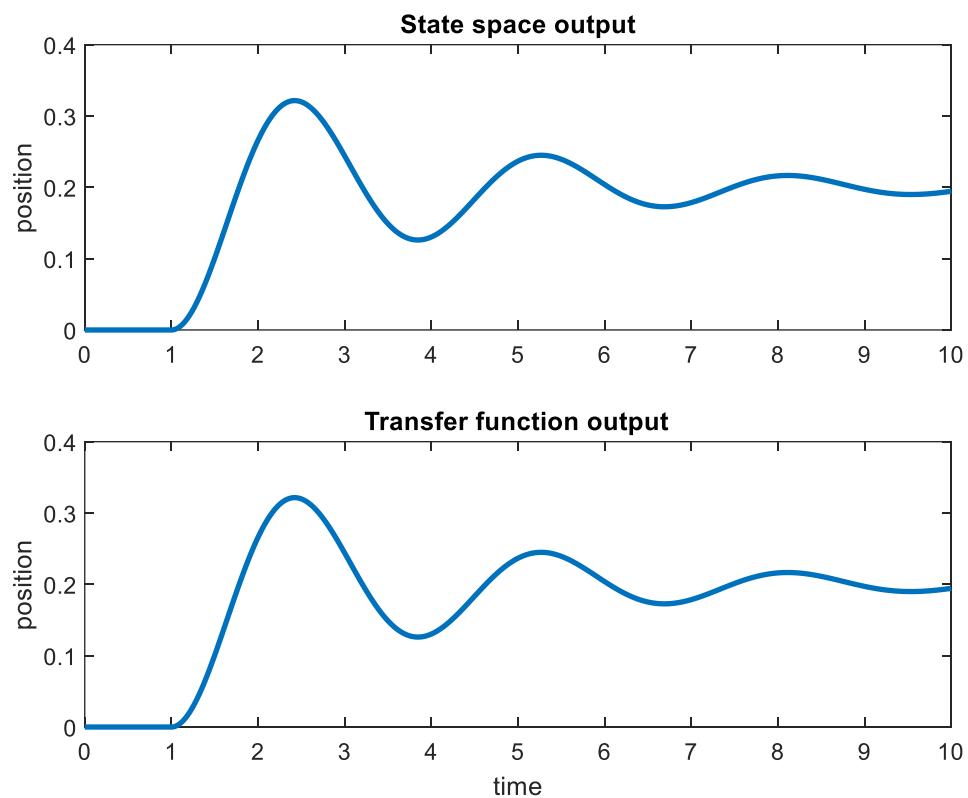
b. Both functions provide the same output





**Plot Commands:**

```
figure  
subplot(2,1,1);  
plot(t,pos(:,1))  
subplot(2,1,2);  
plot(t,pos(:,2))
```



**Problem – 4:**

a.

**Discrete Time function commands:**

DF=c2d(TF,0.1)

DF =

0.004865 z + 0.004753

-----

z^2 - 1.884 z + 0.9324

Sample time: 0.1 seconds

Discrete-time transfer function.

**b. Outputs are very similar. Can be improved by reducing step size**

**Discrete time state space commands:**

SS=ss(A,B,C,D)

SS =

A =

x1	x2	
x1	0	1
x2	-5	-0.7

B =

u1

x1 0

x2 1

C =

x1 x2

y1 1 0

D =

u1

y1 0

Continuous-time state-space model.

```
>> discr=c2d(SS,0.1,'Tustin')
```

```
discr =
```

```
A =
```

```
    x1      x2
```

```
x1  0.9761  0.09547
```

```
x2 -0.4773  0.9093
```

```
B =
```

```
    u1
```

```
x1  0.004773
```

```
x2  0.09547
```

```
C =
```

```
    x1      x2
```

```
y1  0.9881  0.04773
```

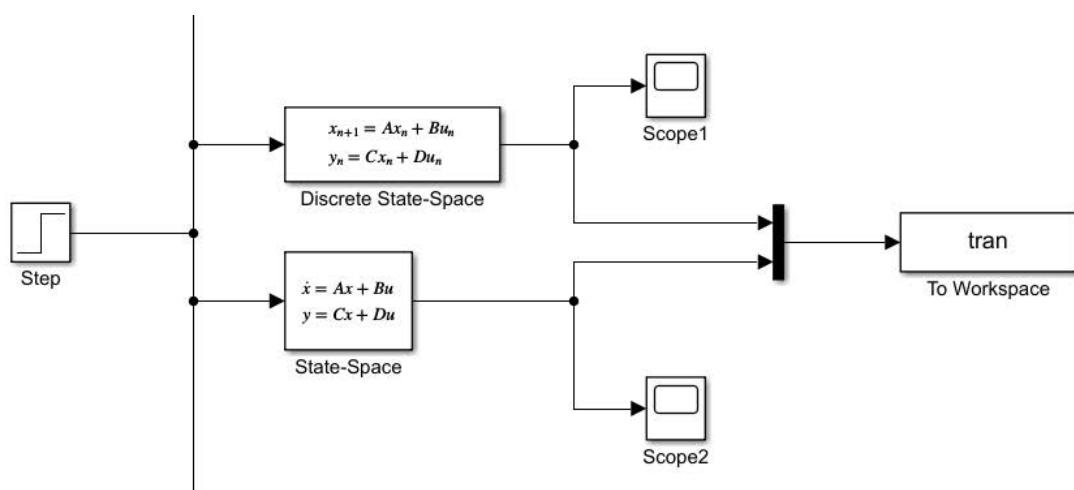
```
D =
```

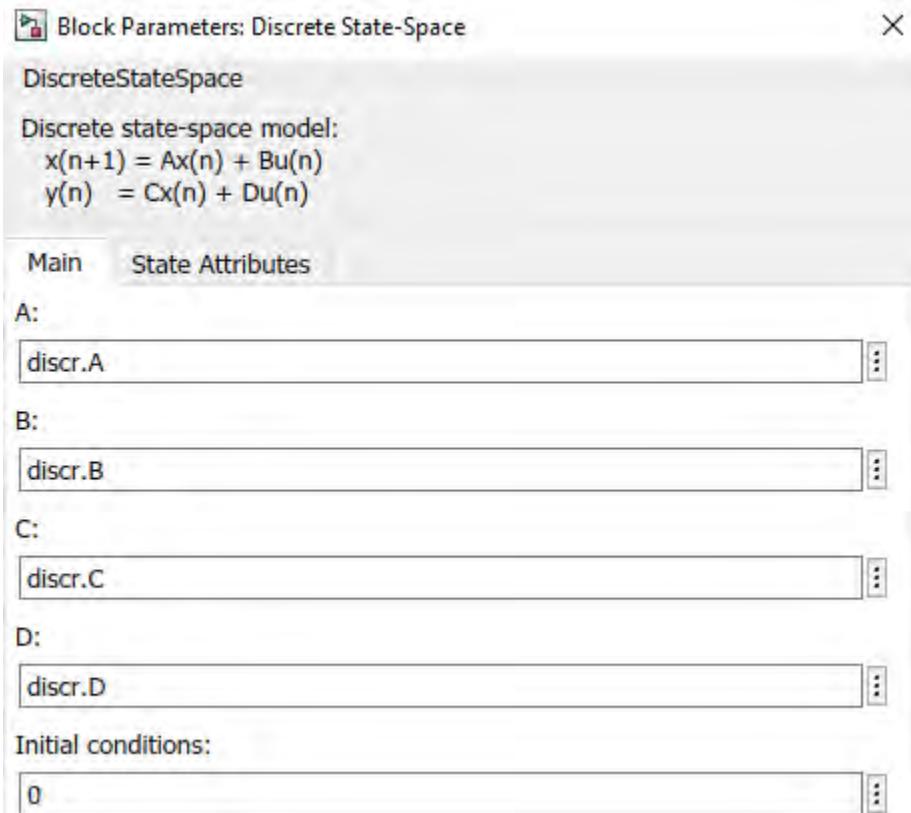
```
    u1
```

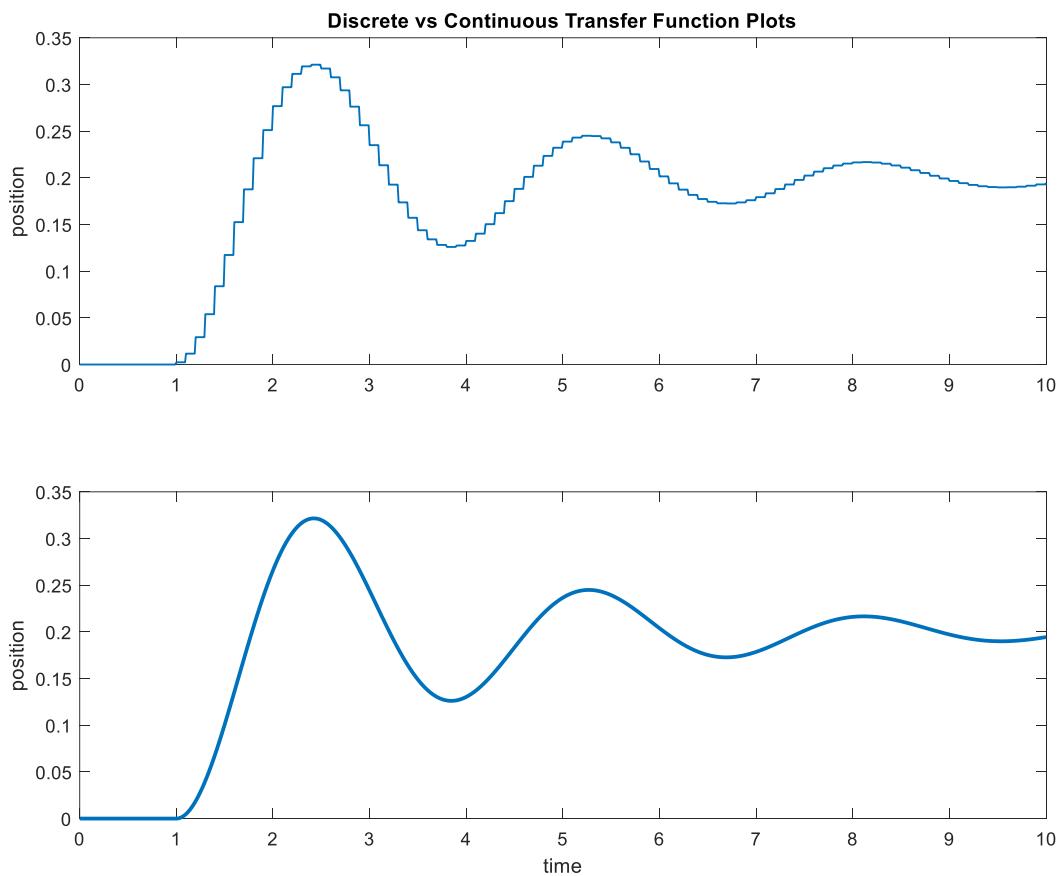
```
y1  0.002387
```

Sample time: 0.1 seconds

Discrete-time state-space model.







`DF=c2d(TF,0.1)`

`DF =`

$$0.004865 \ z + 0.004753$$

-----

$$z^2 - 1.884 \ z + 0.9324$$

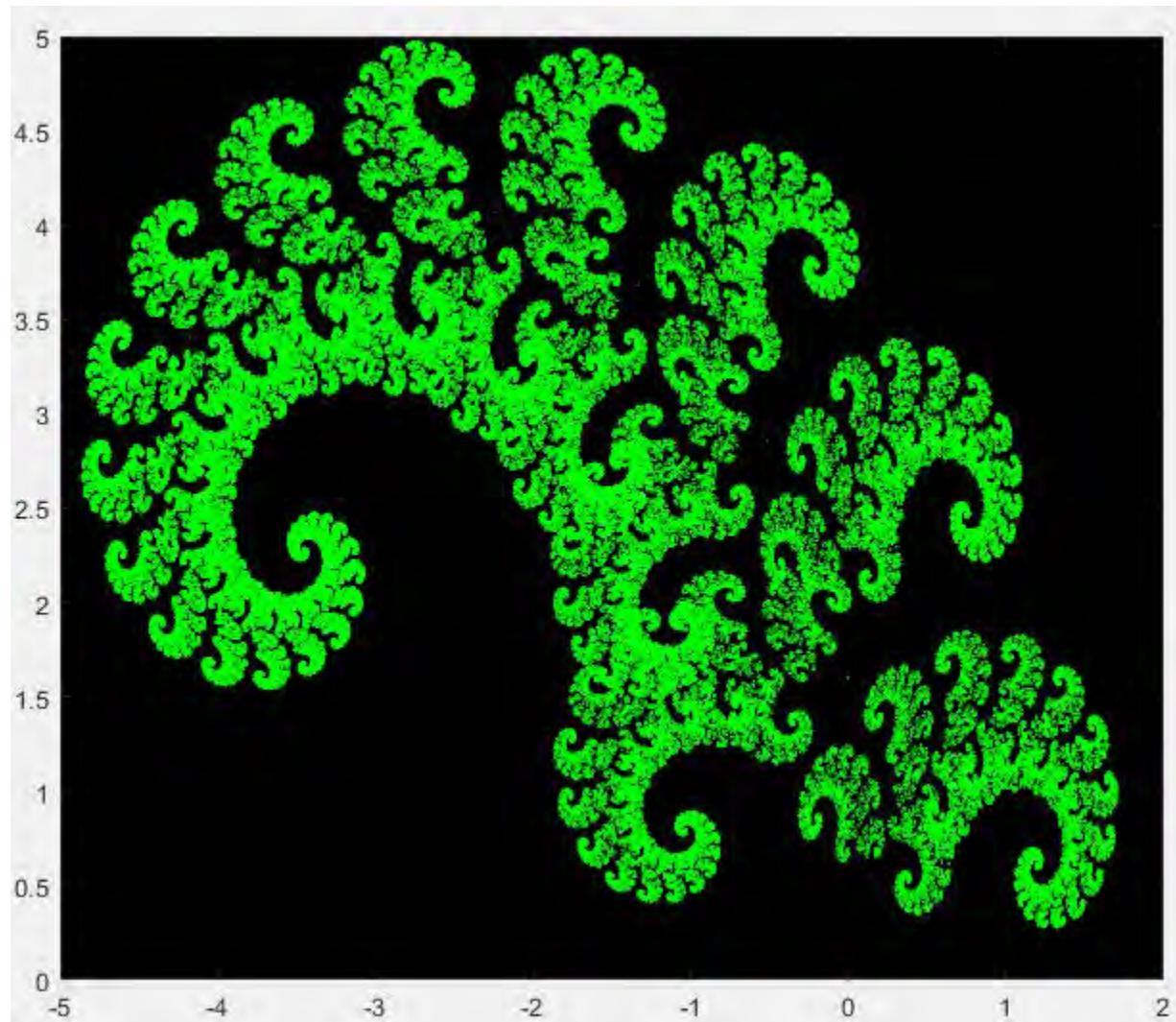
Sample time: 0.1 seconds

Discrete-time transfer function.

## **Problem - 5:**

**MATLAB Code for generating the plot:**

```
function hw1q5
x=[0;0];
A=[0.7873 -0.3230;0.3230 0.7873];
B=[0.0841 -0.3286;0.2930 0.0895];
C=[ -0.2458 0.1523;0.1722 0.3358];
b1=[0;1.6];
b2=[0;0.44];
p=[0.8 0.9 1.0];
i=0;
set(gca,'color',[0 0 0])
plot(x(1),x(2),'g.','markersize',1)
hold on
while i<100001
r=rand;
if r < p(1)
    x=((A*x)+b1);
elseif r < p(2)
    x=((B*x)+b1);
else
    x=((C*x)+b2);
end
J=x(1,1);
K=x(2,1);
set(gca,'color',[0 0 0])
plot(J,K,'g.','markersize',1)
i=i+1;
end
end
```



**EE5327, Fall 2020**  
**Homework 2: Discrete Equations and**  
**Recursive Estimation**  
**Due 9/24/2020**

**Problem 1) 50 points**

Implement the following transfer functions in parallel in Simulink with a fixed step solver and a time step of 0.1 seconds

$$\begin{array}{c} 1 \\ \hline s^2 + 0.5 \quad + 1 \\ + 3 \\ \hline s^2 + 2 \quad + 6 \\ \hline s^2 + 3 \quad + 5 \\ \hline s^2 + 6 \quad + 5 \end{array}$$

Using the standard step input to the transfer functions for this problem, do the following:

- a) Run the simulation for 10 seconds and plot the outputs. (10 points)
- b) Implement the systems in discrete blocks (use c2d) and compare those output with the outputs of the continuous time systems. Use the Tustin transformation and a sample time of 0.1 seconds in the c2d command. (10 points)
- c) Now add a normal random signal with a sample time of 0.1 seconds and noise variance of 0.001 to the step input and rerun the simulation. Compute the difference between the continuous system outputs and the respective discrete system outputs. (10 points)
- d) For the third transfer function above, determine the difference equation for that input/output relationship using the discrete numerator and denominator and implement that equation in a MATLAB function block in Simulink. Compare the output of this block to the discrete transfer function block. They should be practically identical. (10 points)
- e) Use the tf2ss command to convert the second and third transfer functions from transfer function form to state space form and show the resulting A, B, C, and D matrices. What do you notice about the D matrices between the two? (10 points)

**Problem 2) Recursive Estimator 50 points**

Implement the recursive estimator in Simulink:

$$X_{-1} = X + \frac{1}{+1} (Z_{-1} - X)$$

Modify the Simulink model configuration parameters to use a fixed-step solver with a fixed-step size of 0.1 seconds. To a step input signal add Gaussian noise of character zero mean and 0.01 variance and make sure the sample time on the Gaussian noise is also set to 0.1 seconds. This becomes your measurement signal.

- a) Run the simulation for 15 seconds and plot the noisy input as well as the estimate as a function of time. (10 points)

Note that this estimator does a pretty good job of filtering out the noise, but it is estimating what is essentially a constant. Now, remove the step input and replace it with a square wave of magnitude 1 and period of 0.2 Hz (Use the Signal Generator block) which creates a time-varying signal. Now do the following:

- b) Run the simulation again for 15 seconds with this new input signal and plot the noisy input and the estimate as functions of time. (10 points)
- c) Noting the slowness of the estimator response and wanting a better response, put a limiter on the maximum value of k used in the denominator in the above equation to 10. Rerun the 15 second simulation and plot the estimate as a function of time. (5 points)
- d) If some is good, more must be better. Now limit the max value of k to be 4 and repeat the analysis. (5 points)
- e) Finally, limit the max value of k to be 1 and repeat the analysis. (5 points)
- f) Comment on the results, comparing the ability of the estimator to react faster and the noise level passed along. (5 points)
- g) Finally, noting this estimator acts like a low-pass filter (shown below), by trial and error, determine an approximate transfer function by selecting the time constant in a filter for the case where k = 4. Plot the estimate to the output of the filter with the time constant you found, feeding both the noisy measurements. (10 points)

$$\frac{1}{+1}$$

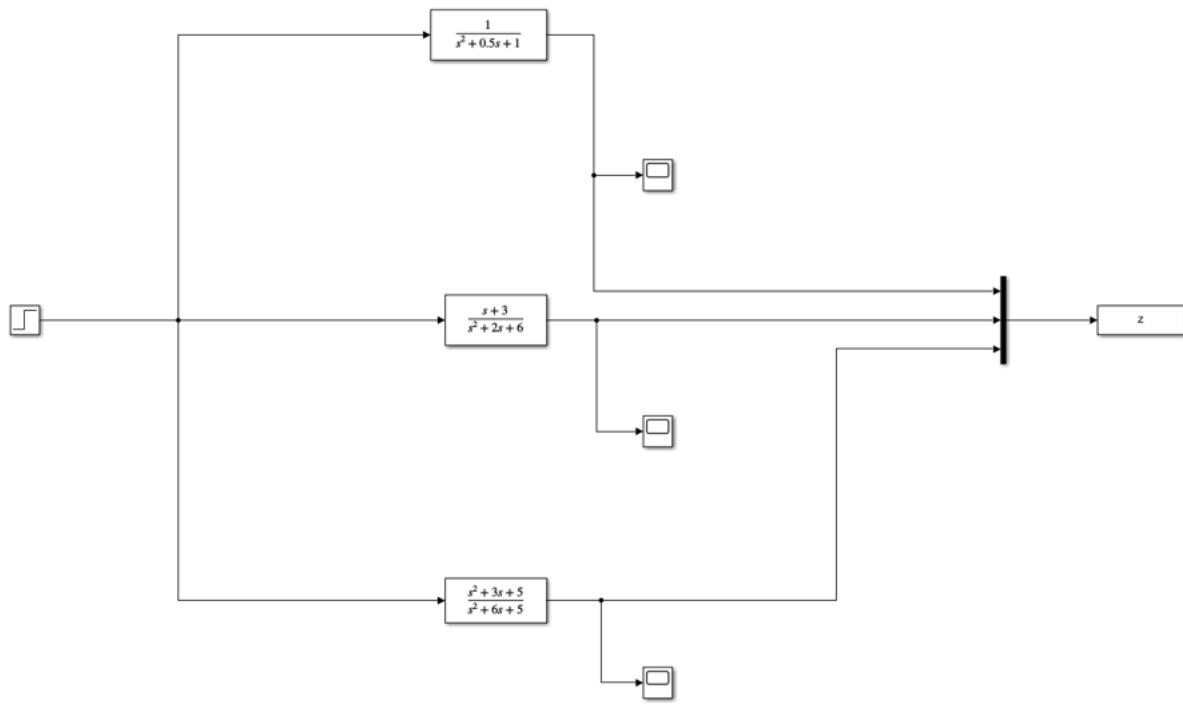
**ATUL SHROTRIYA**

**Homework – 2**

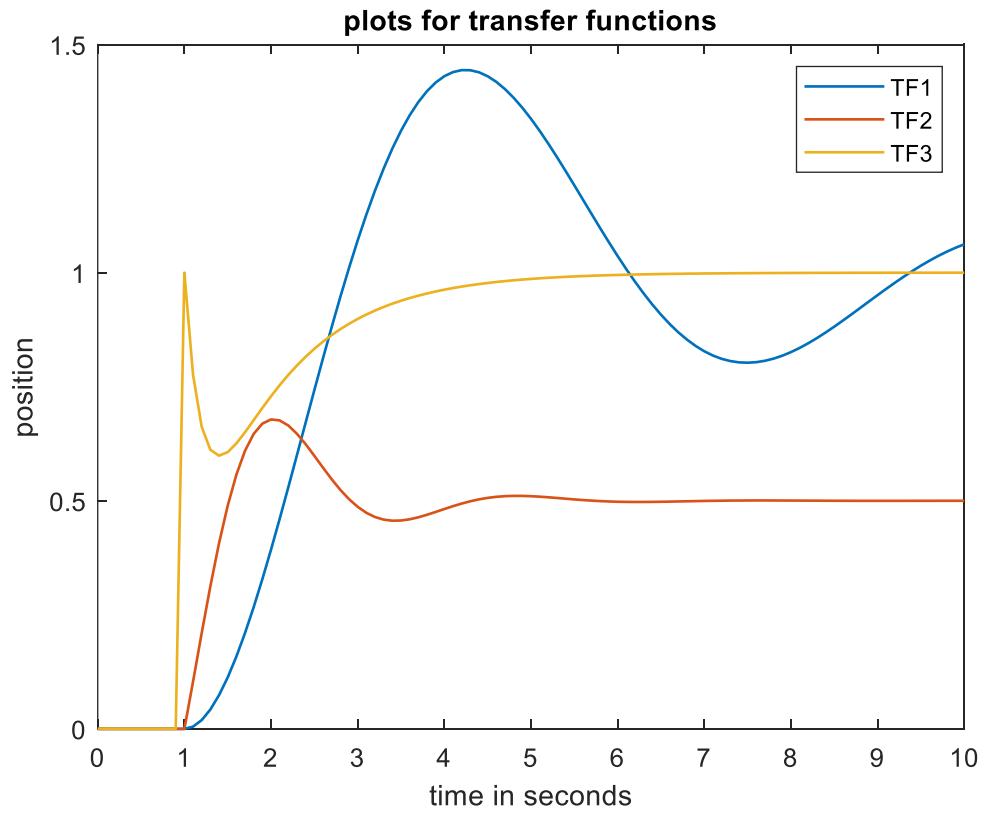
**System Identification and Estimation**

## Problem – 1:

### a. Simulink Model:



Plot:



Plot Command:

```
plot(t,z(:,1),t,z(:,2),t,z(:,3))
```

NOTE: TF in legend refers to Transfer Function and DTF to Discrete Transfer Function. The legend would have extended into the plots so I kept the names short.

**b. Commands to implement the system in discrete blocks**

```
t=0:0.1:10;  
L=1;  
M=[1 0.5 1];  
N=[1 3];  
O=[1 2 6];  
P=[1 3 5];  
Q=[1 6 5];  
T1=tf(L,M)  
T1d=c2d(T1,0.1,'Tustin')  
T2=tf(N,O)  
T2d=c2d(T2,0.1,'Tustin')  
T3=tf(P,Q)  
T3d=c2d(T3,0.1,'Tustin')  
[A1d,B1d,C1d,D1d]=tf2ss(T1d.Numerator{1,1},T1d.Denominator{1,1})
```

```
T1 =  
1  
-----  
s^2 + 0.5 s + 1  
Continuous-time transfer function.  
T1d =  
0.002433 z^2 + 0.004866 z + 0.002433  
-----  
z^2 - 1.942 z + 0.9513
```

```
Sample time: 0.1 seconds  
Discrete-time transfer function.  
T2 =  
s + 3  
-----  
s^2 + 2 s + 6  
Continuous-time transfer function.
```

```
T2d =
0.05157 z^2 + 0.01345 z - 0.03812
```

-----

$$z^2 - 1.767 z + 0.8206$$

Sample time: 0.1 seconds

Discrete-time transfer function.

T3 =

$$s^2 + 3 s + 5$$

-----

$$s^2 + 6 s + 5$$

Continuous-time transfer function.

T3d =

$$0.8857 z^2 - 1.505 z + 0.6571$$

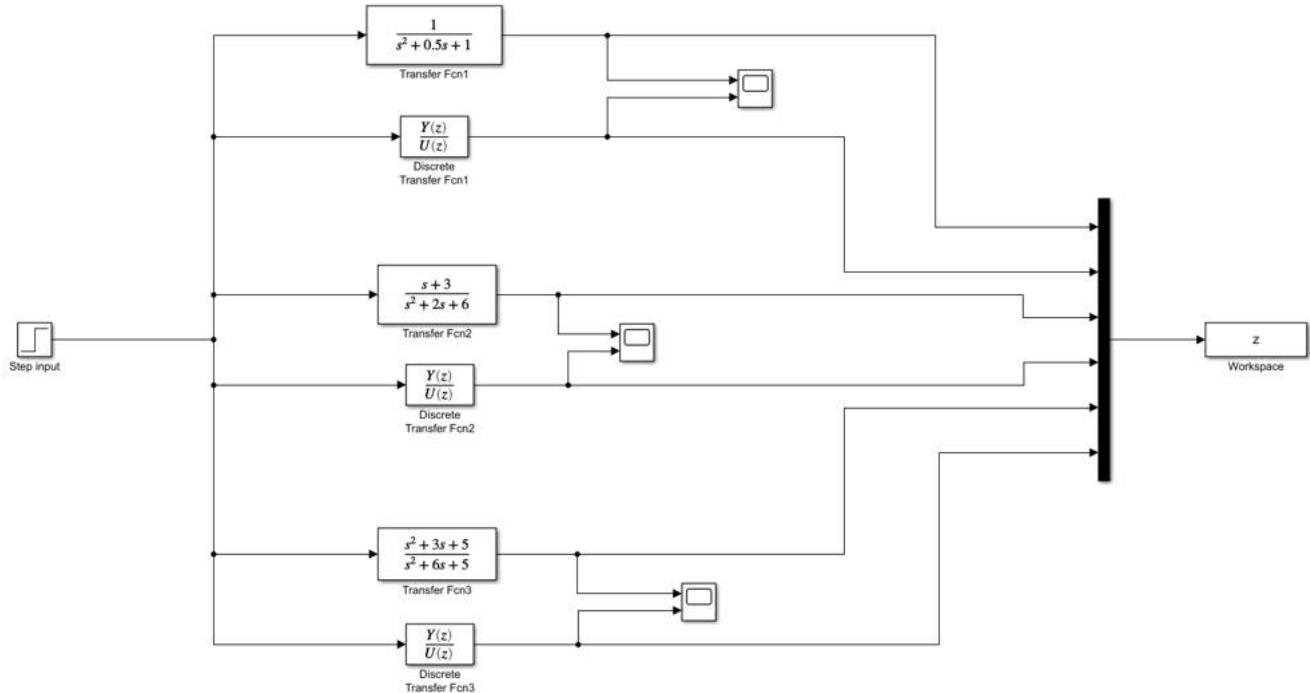
-----

$$z^2 - 1.505 z + 0.5429$$

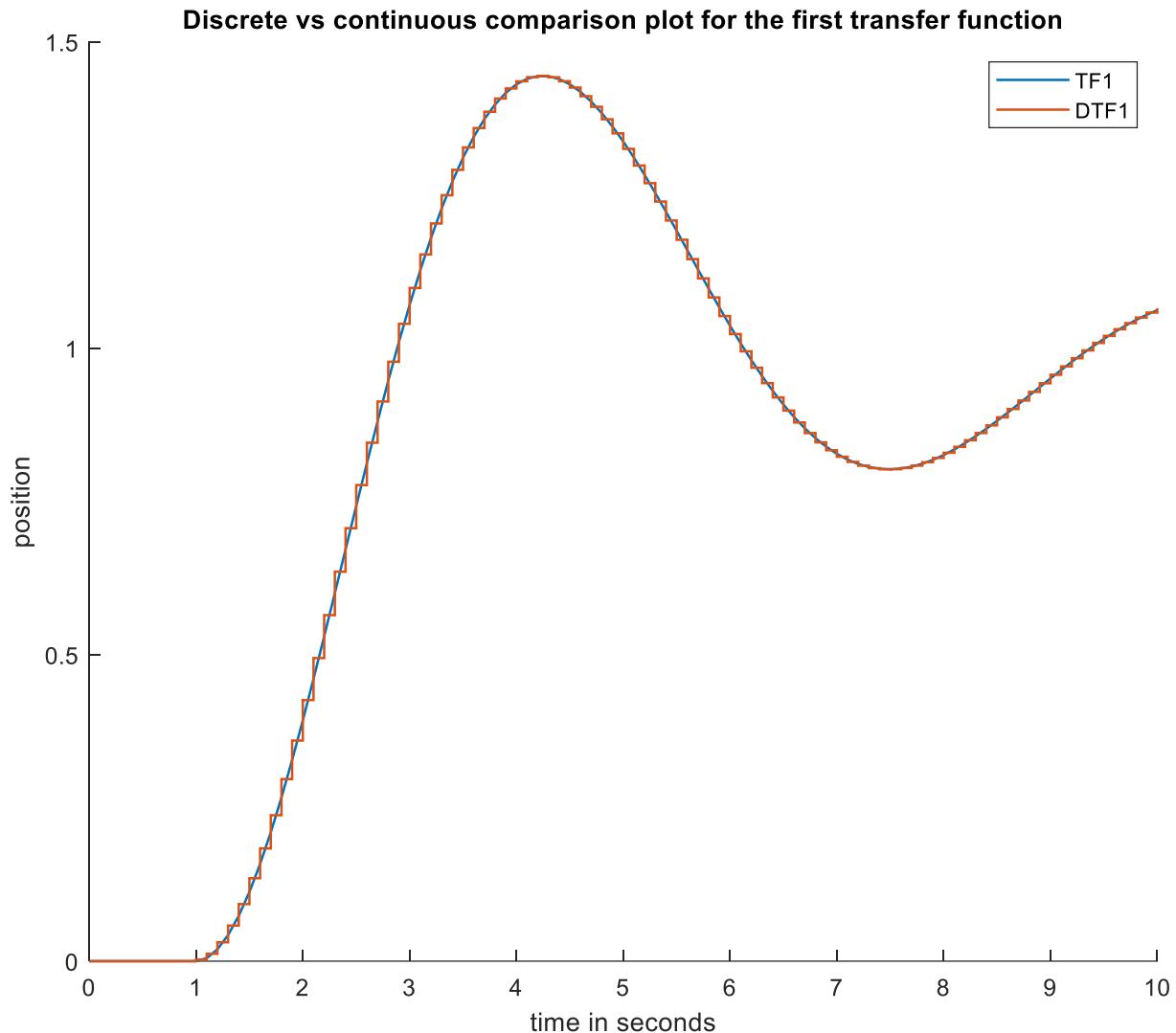
Sample time: 0.1 seconds

Discrete-time transfer function.

## Simulink Model



## Comparison Plots

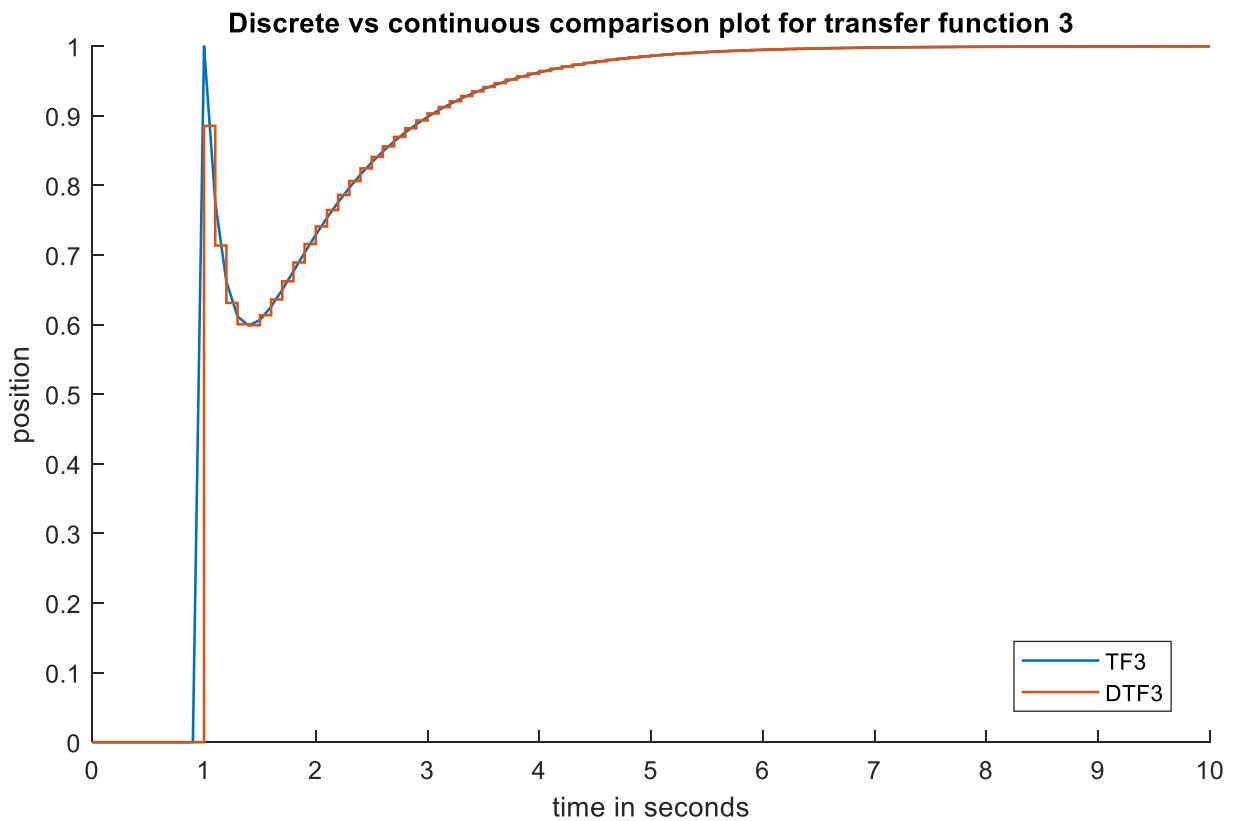
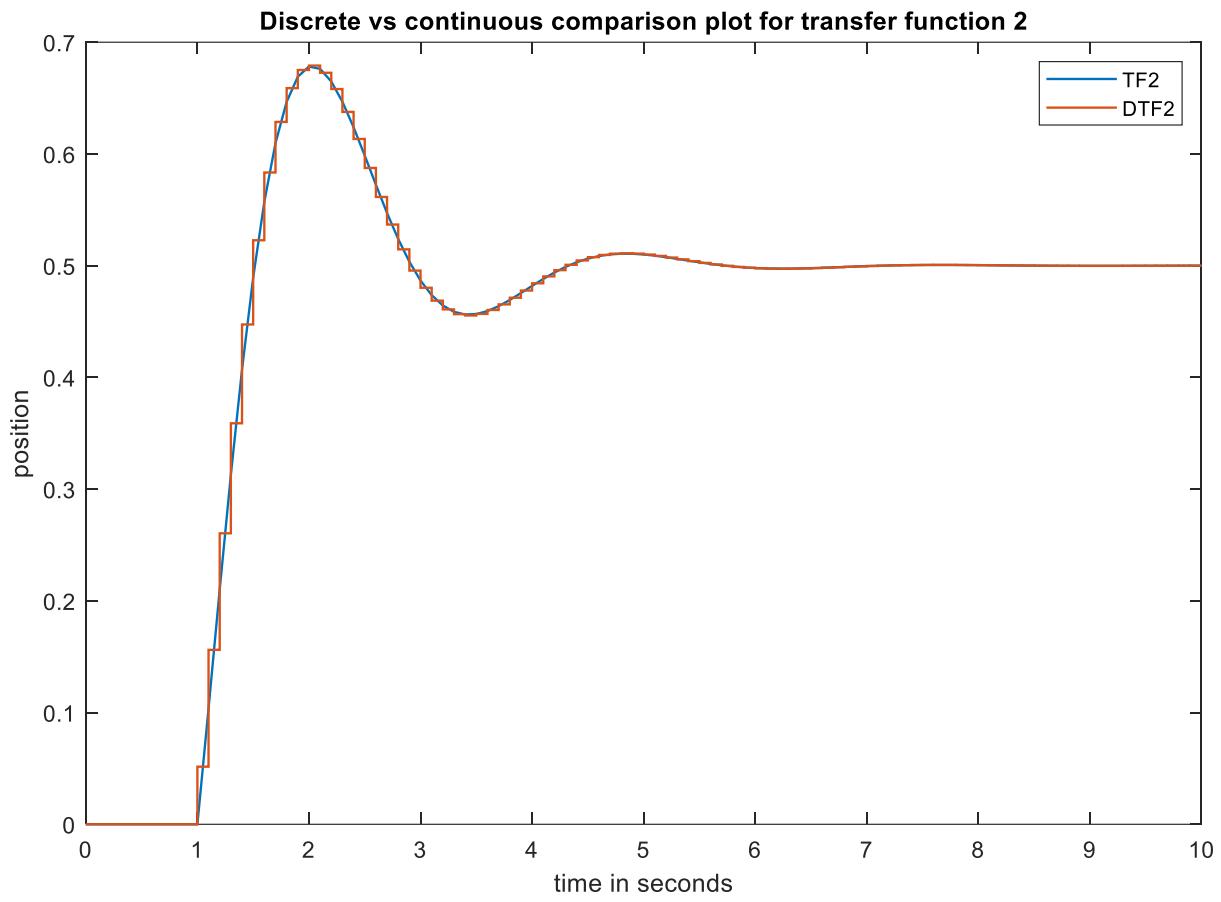


## Plot Commands

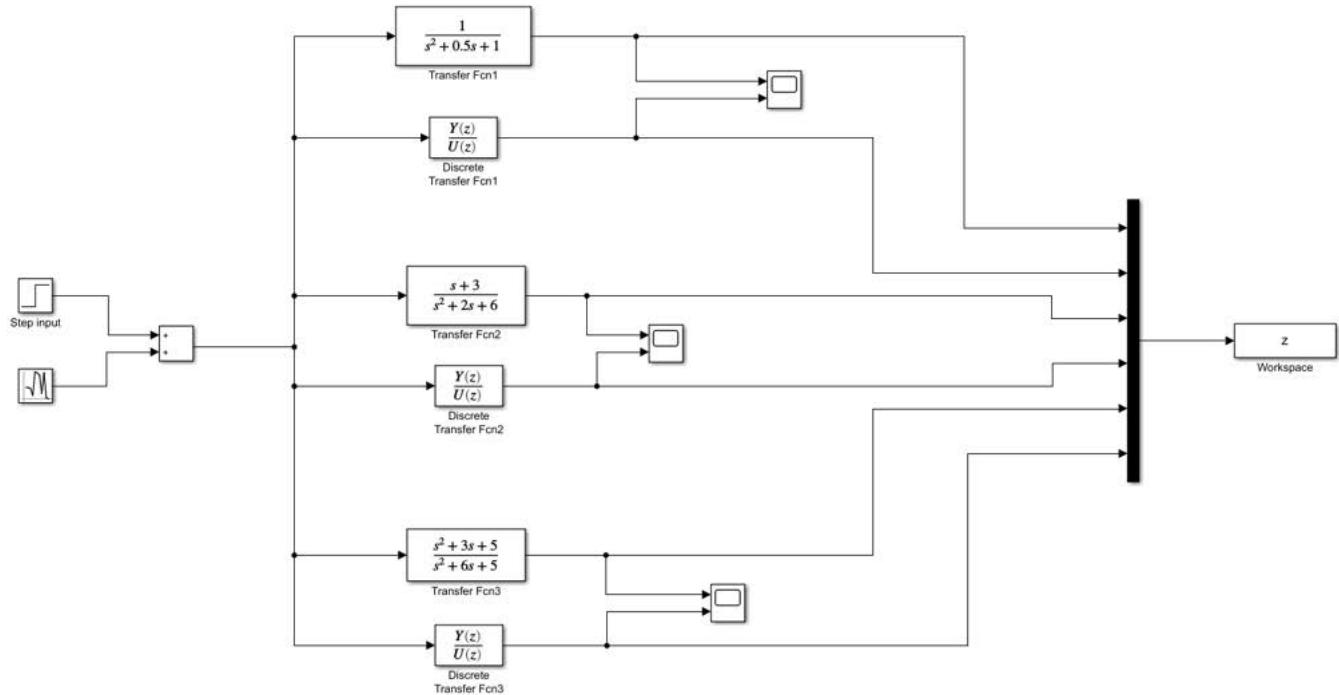
```
hold on  
figure(1)  
plot(t,z(:,1))  
stairs(t,z(:,2))
```

The same command was used for the plots given below with the change in z columns i.e,

```
plot(t,z(:,3))  
stairs(t,z(:,4))  
plot(t,z(:,5))  
stairs(t,z(:,6))
```



As we can see, the discrete and continuous plots are quite similar with discrete ones being slightly choppier.



C.

Block Parameters: Discrete Transfer Fcn X

Discrete Transfer Fcn

Implement a z-transform transfer function. Specify the numerator and denominator coefficients in descending powers of z. The order of the denominator must be greater than or equal to the order of the numerator.

Main	Data Types	State Attributes
<b>Data</b>		
Source	Value	
Numerator:	Dialog	T1d.Numerator{1,1}
Denominator:	Dialog	T1d.Denominator{1,1}
Initial states:	Dialog	0
External reset: None		
Input processing: Elements as channels (sample based)		
<input type="checkbox"/> Optimize by skipping divide by leading denominator coefficient (a0)		
Sample time (-1 for inherited):		
0.1		

OK Cancel Help Apply

**Commands for difference:**

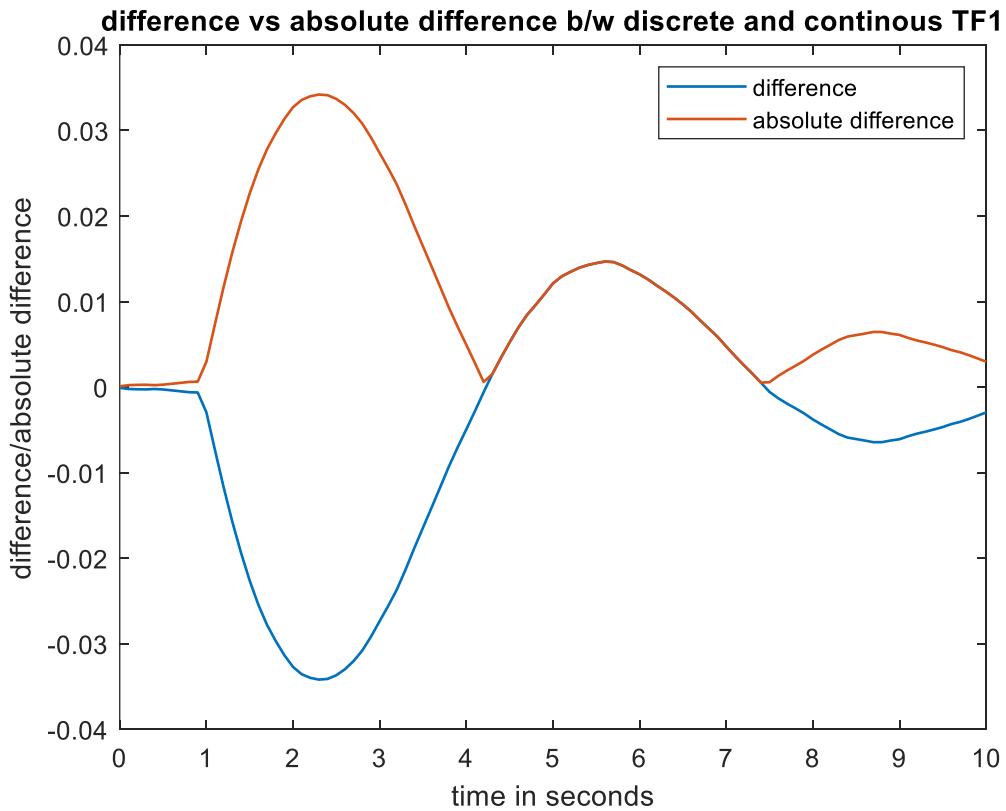
```
for i=1:1:101  
i1(i,1)=z(i,1)-z(i,2);  
i2(i,1)=z(i,3)-z(i,4);  
i3(i,1)=z(i,5)-z(i,6);  
end
```

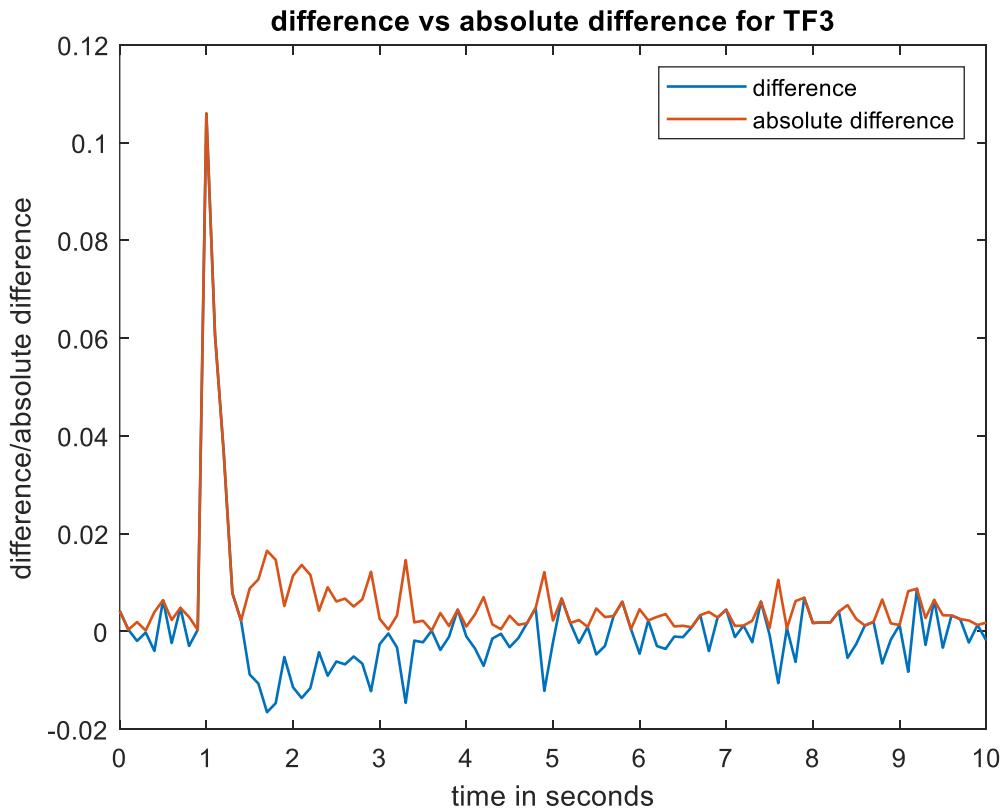
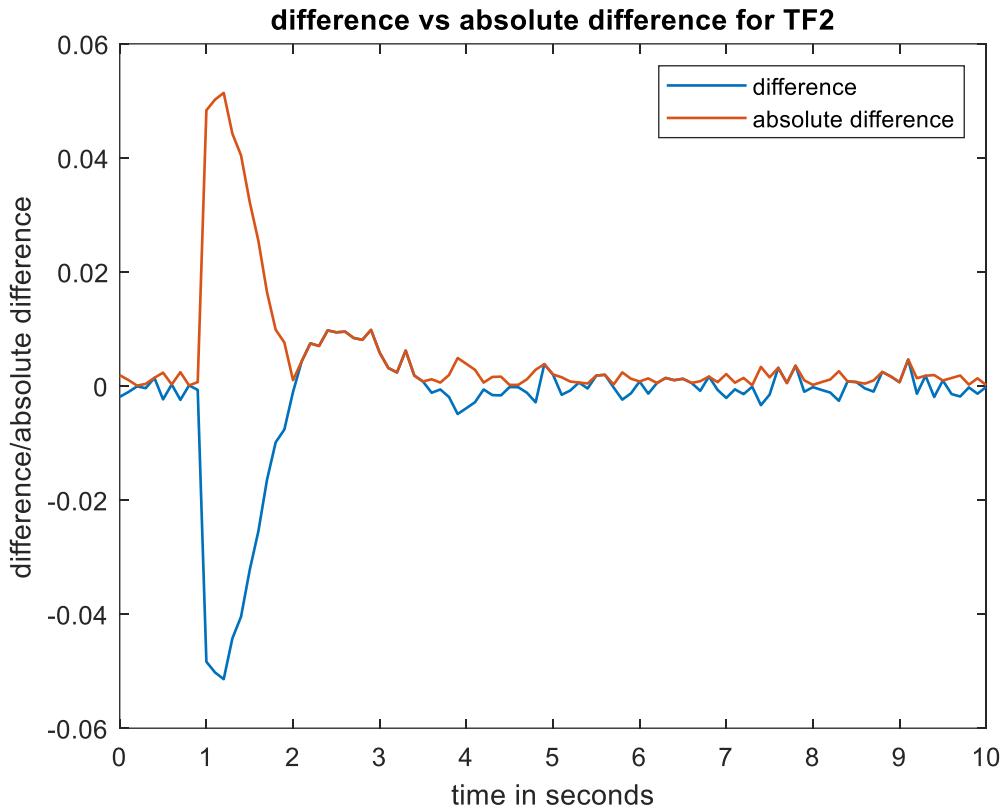
**Commands for absolute difference:**

```
for i=1:1:101  
h1(i,1)=abs(z(i,1)-z(i,2));  
h2(i,1)=abs(z(i,3)-z(i,4));  
h3(i,1)=abs(z(i,5)-z(i,6));  
end
```

**Commands for comparison plot between difference and absolute difference:**

```
plot(t,i1(:,1),t,h1(:,1))  
plot(t,i2(:,1),t,h2(:,1))  
plot(t,i3(:,1),t,h3(:,1))
```

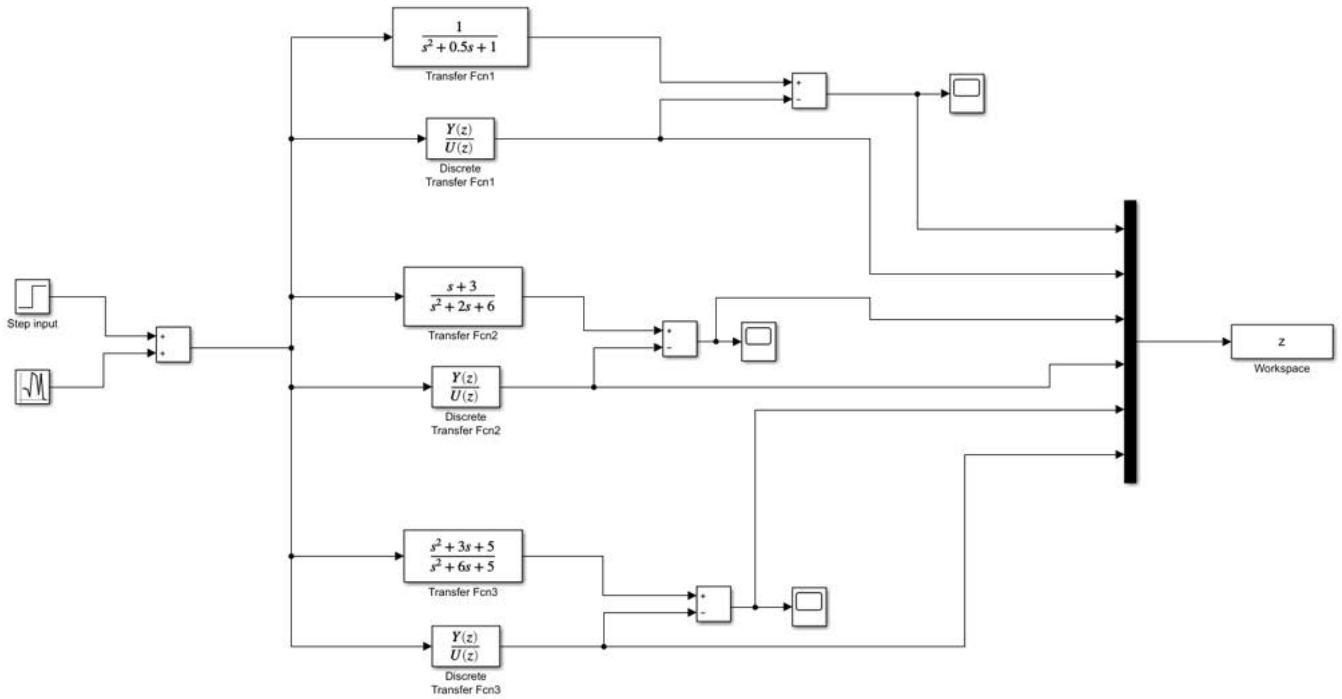




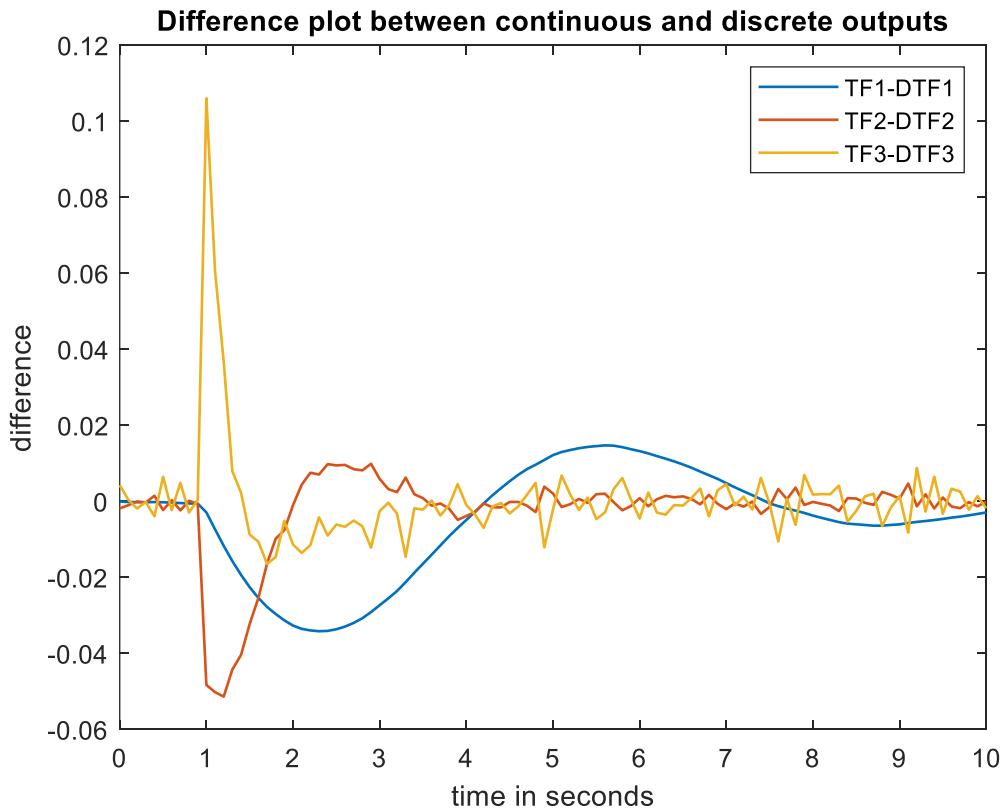
We can see how the values peak initially when the input is given and slowly settle down towards zero as the system tries to reach steady state.

Same thing has also been implemented through the Simulink model below but I didn't know the absolute difference block so it's just differences and their comparison.

### Simulink Model:



**Plot Command:** `plot(t,z(:,1),t,z(:,3),t,z(:,5))`



It is observed from the above plot that, both the first and second transfer function go towards negative in the beginning of the plot. With the decrease in relative degree the peak increases. However, it completely flips for the third transfer function when the relative degree is 0. This is primarily because here the output is directly related to the input. Step is immediate for continuous model but in small parts for discrete model.

#### d. Difference Equation:

$$(k) = . . * (k-1) - . . * (k-2) + . . * (k-3) + . . * (k-4) - . . * (k-5)$$

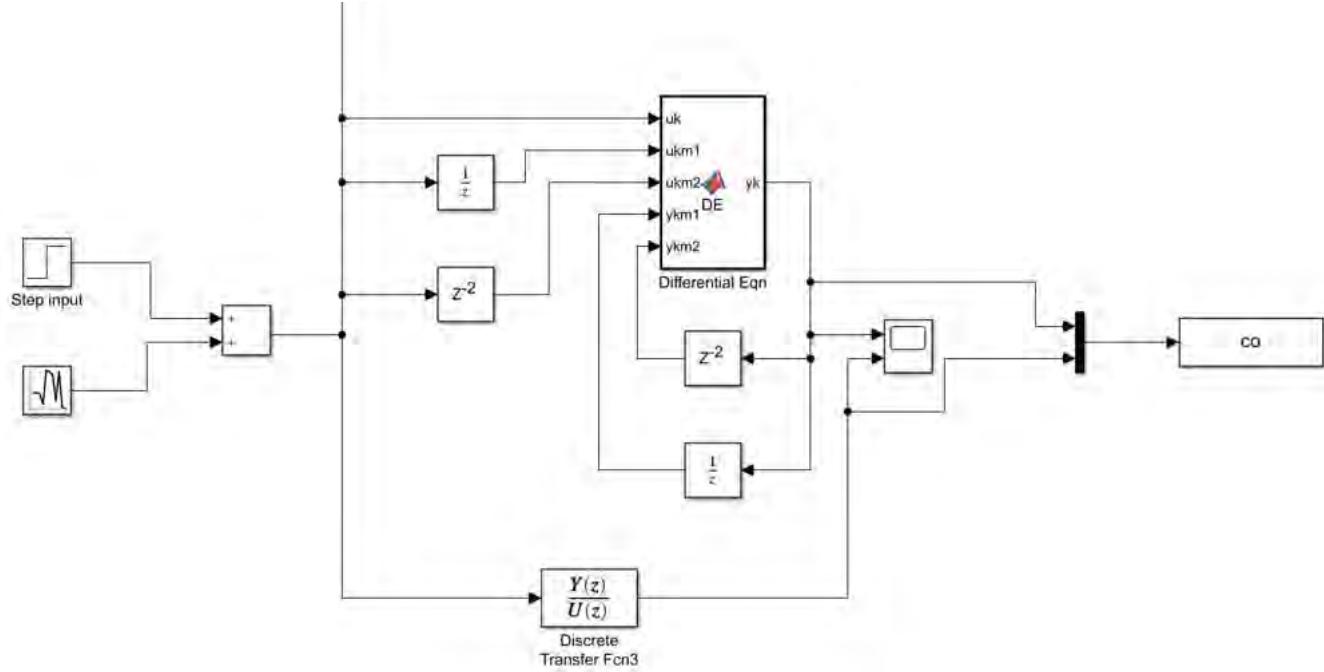
Here k represents the current time step.

#### Implementation in MATLAB function block:

```
function yk = DE(uk, ukm1, ukm2, ykm1, ykm2)

yk = 0.8857*uk-1.505*ukm1+0.6571*ukm2+1.505*ykm1-0.5429*ykm2;
```

#### Simulink Model With Noise:



#### Plot Command:

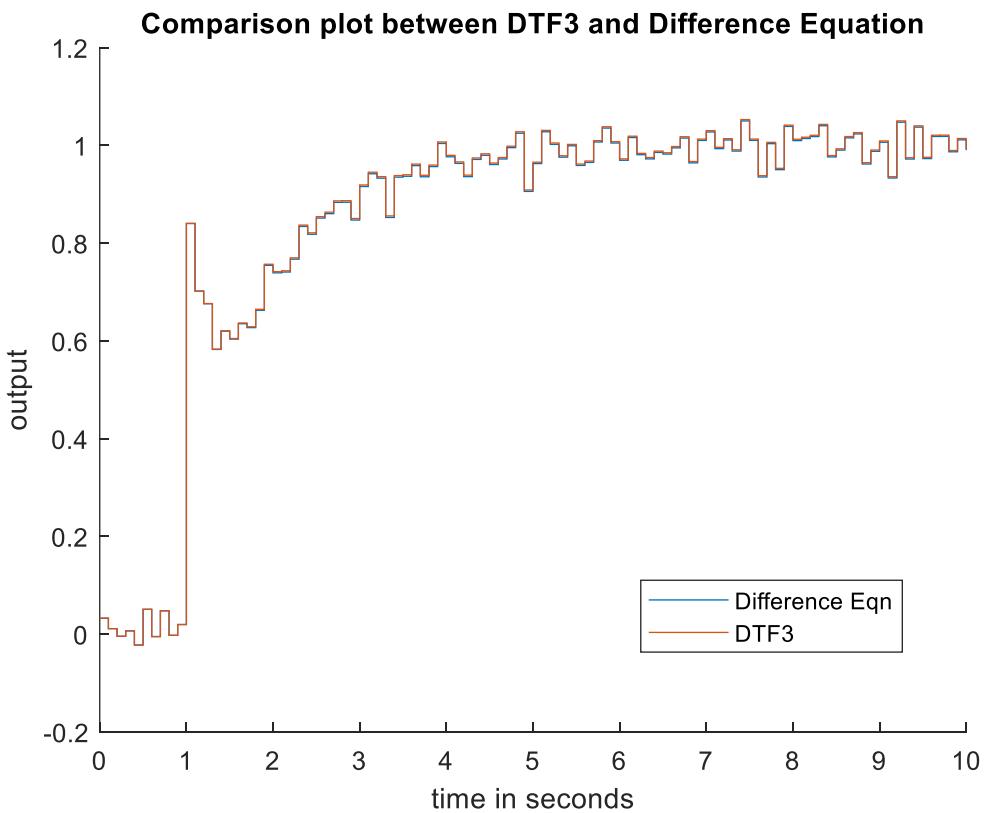
```
hold on
figure(1)
stairs(t,co(:,1))
stairs(t,co(:,2))
```

Okay so there was a slight difference on the plots so I retrieved the exact values to define the function. The new function is given below. The plot command remains the same. Had to change one line on the second plot to dashes and dots otherwise only one color was visible. Generated plots are shown on the next page.

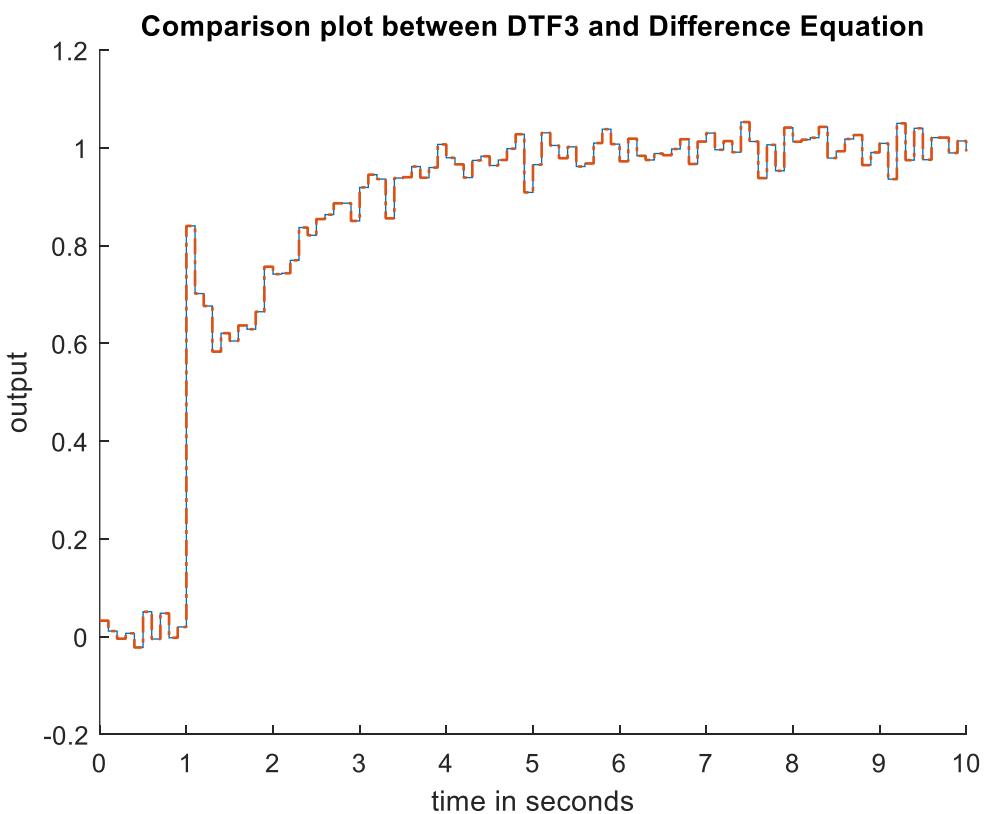
#### New Function EXACT VALUES:

```
function yk = DE(uk, ukm1, ukm2, ykm1, ykm2)
yk = 0.885714285714286*uk - 1.504761904761905*ukm1 + 0.657142857142857*ukm2 +
1.504761904761905*ykm1 - 0.542857142857143*ykm2;
```

### **Output Comparison Plot:**



### **Output Comparison Plot EXACT VALUES:**



e. Commands for converting transfer functions to state space:

```
>> [A2,B2,C2,D2]=tf2ss([1,3],[1,2,6])
```

A2 =

$$\begin{matrix} -2 & -6 \\ 1 & 0 \end{matrix}$$

B2 =

$$\begin{matrix} 1 \\ 0 \end{matrix}$$

C2 =

$$\begin{matrix} 1 & 3 \end{matrix}$$

D2 =

$$0$$

```
>> [A3,B3,C3,D3]=tf2ss([1,3,5],[1,6,5])
```

A3 =

$$\begin{matrix} -6 & -5 \\ 1 & 0 \end{matrix}$$

B3 =

$$\begin{matrix} 1 \\ 0 \end{matrix}$$

C3 =

$$\begin{matrix} -3 & 0 \end{matrix}$$

D3 =

$$1$$

Here, D2 is the D matrix of the second transfer function whereas D3 is the D matrix of the third transfer function. It is clearly observed that the second transfer function has a D matrix of 0 whereas it is 1 for the third transfer function.

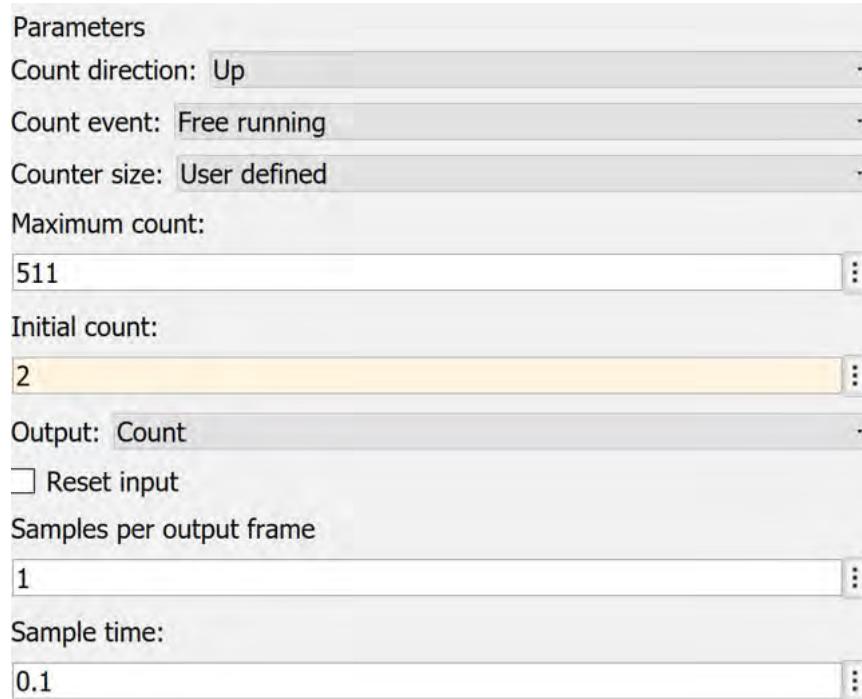
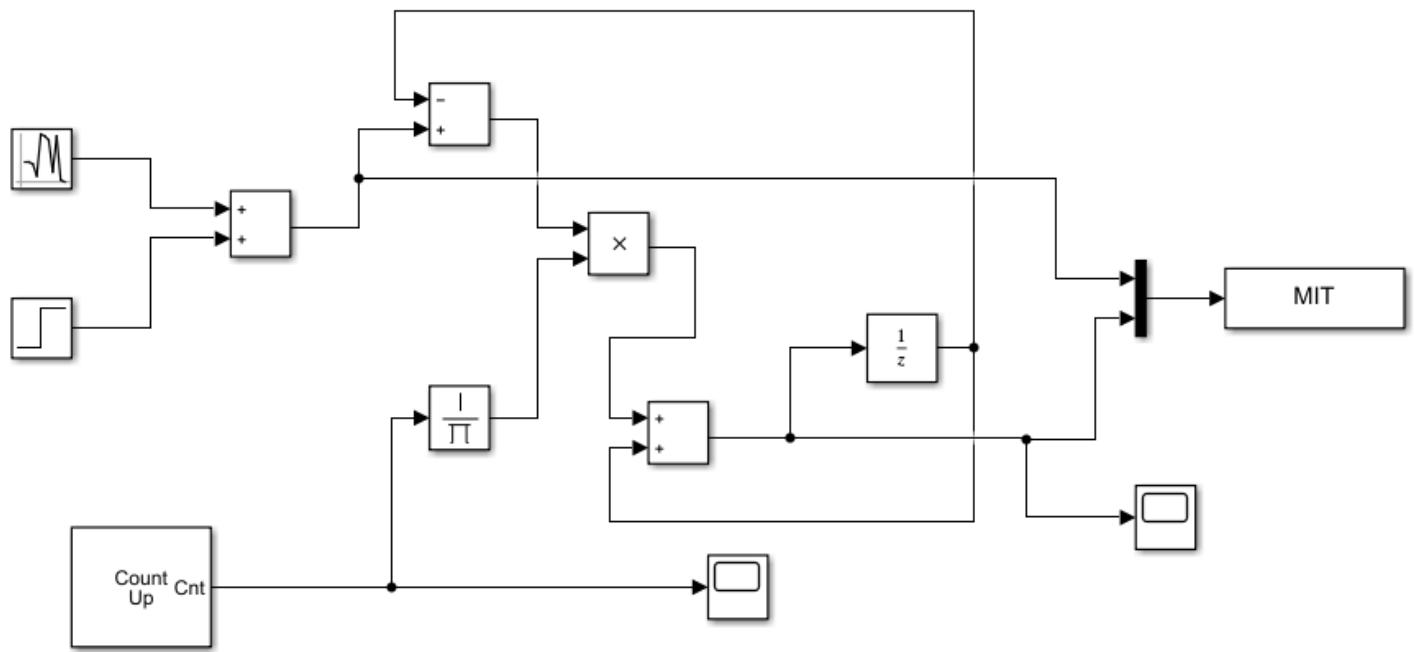
This can also be deduced by looking at the transfer functions themselves. The second transfer function has a denominator degree 2 and numerator degree 1. Whereas, both numerator and denominator degrees are 2 for the third transfer function.

D is one because output is directly affected by input for the third equation. It is hence not zero on the output equation. (1 due to same coefficient). This is not the case for second transfer function and thus D is zero for its state space representation.

$$= + , h , h \quad h$$

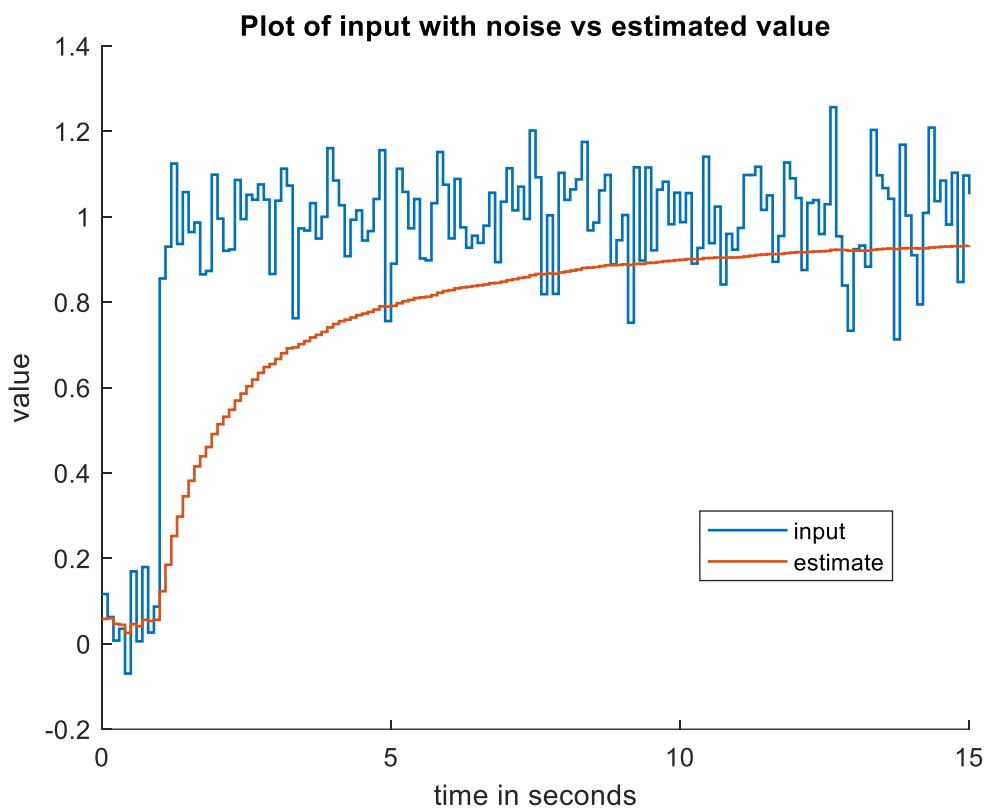
## Problem - 2:

a. Simulink Model (since period is in time units, I have done everything twice)



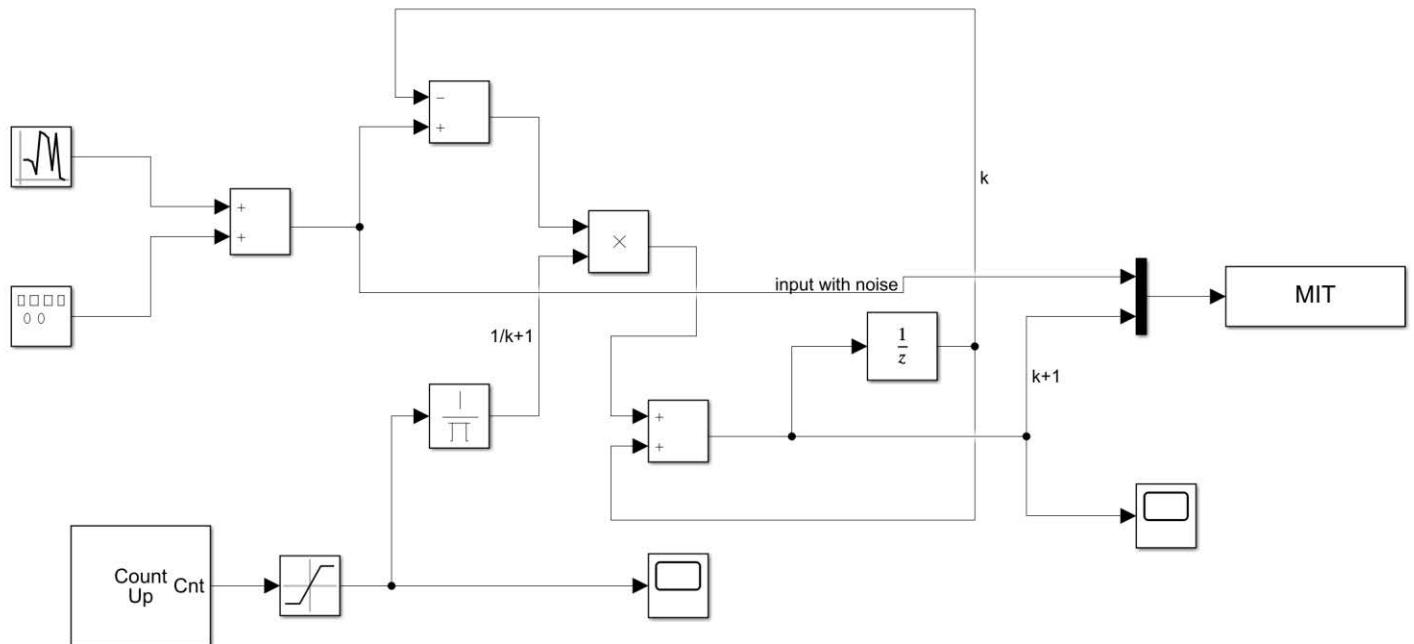
Note: The counting starts from 2 to ensure that it is  $k+1$ . For limiting, I have used the  $k+1$  value i.e., 11, 5 and 2 respectively.

## Comparison Plot

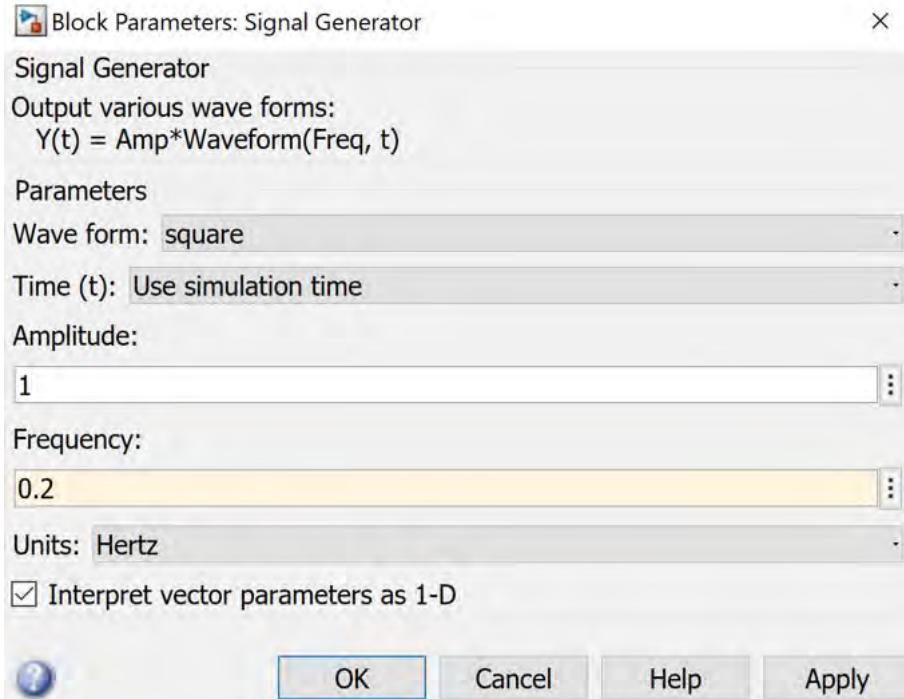


**Plot commands:** hold on; stairs(tout,MIT(:,1)); stairs(tout,MIT(:,2))

### b. Simulink Model



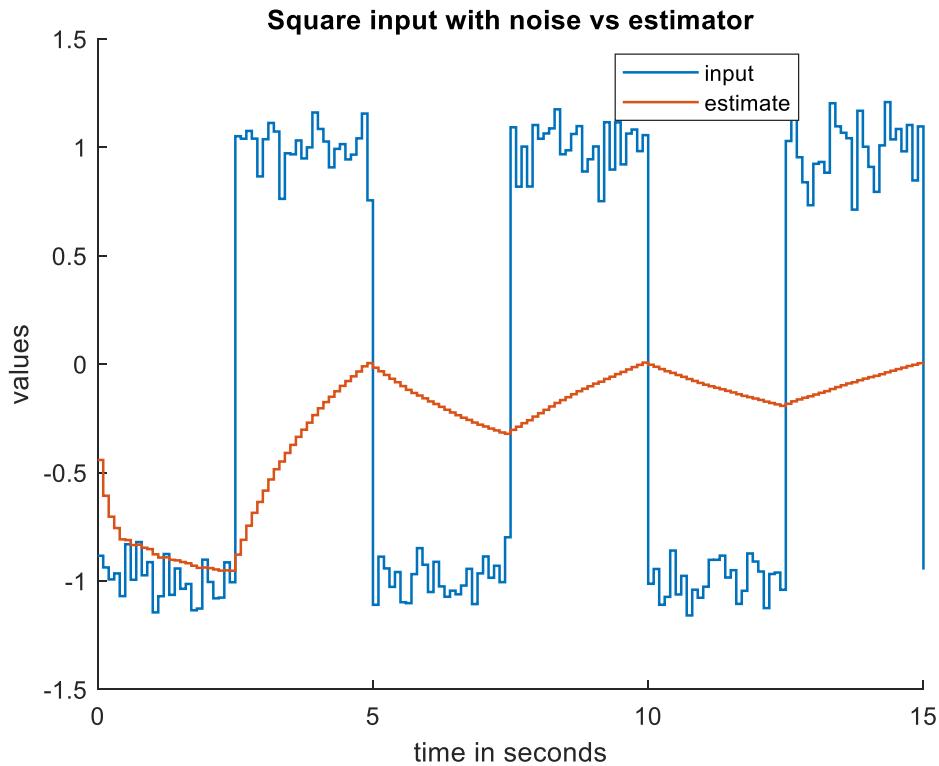
## Square Wave Generator Parameters



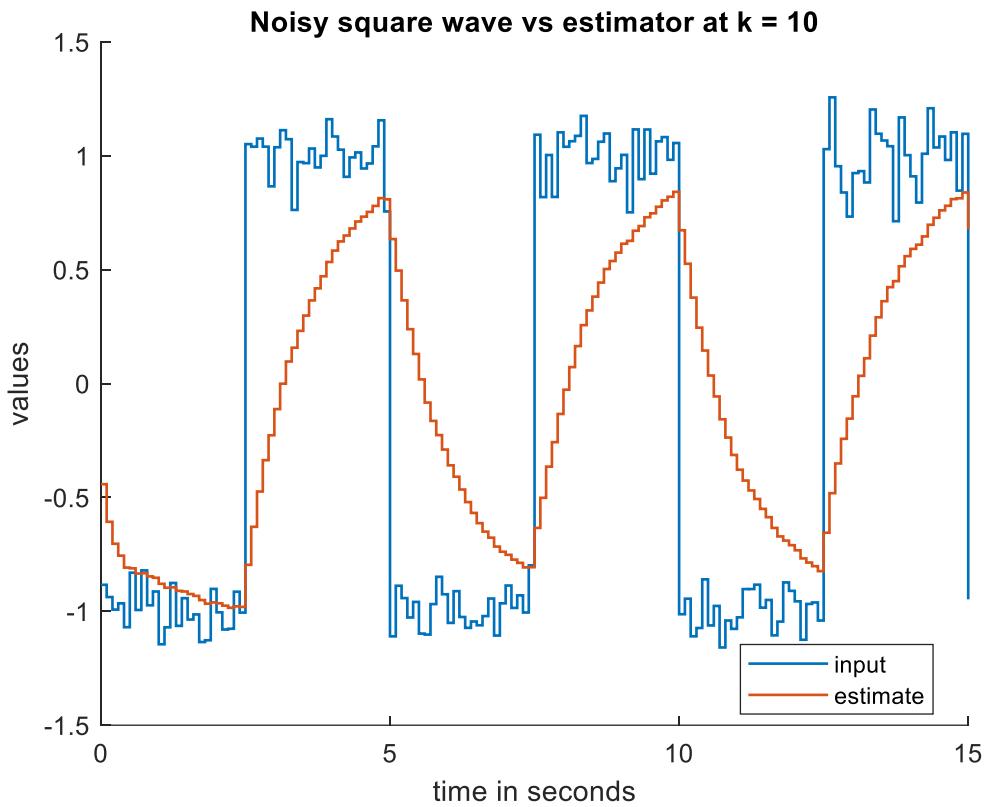
## Plot Commands

```
>> hold on  
stairs(tout,MIT(:,1))  
stairs(tout,MIT(:,2))
```

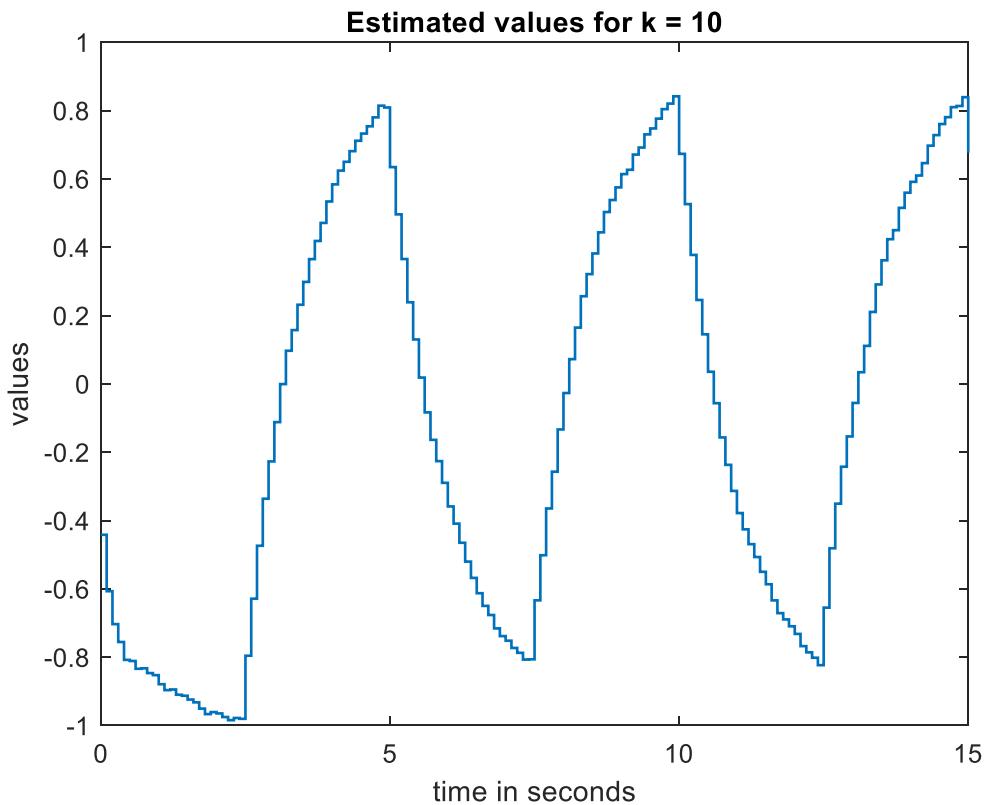
## Comparison Plot



c. For k limited to 10,

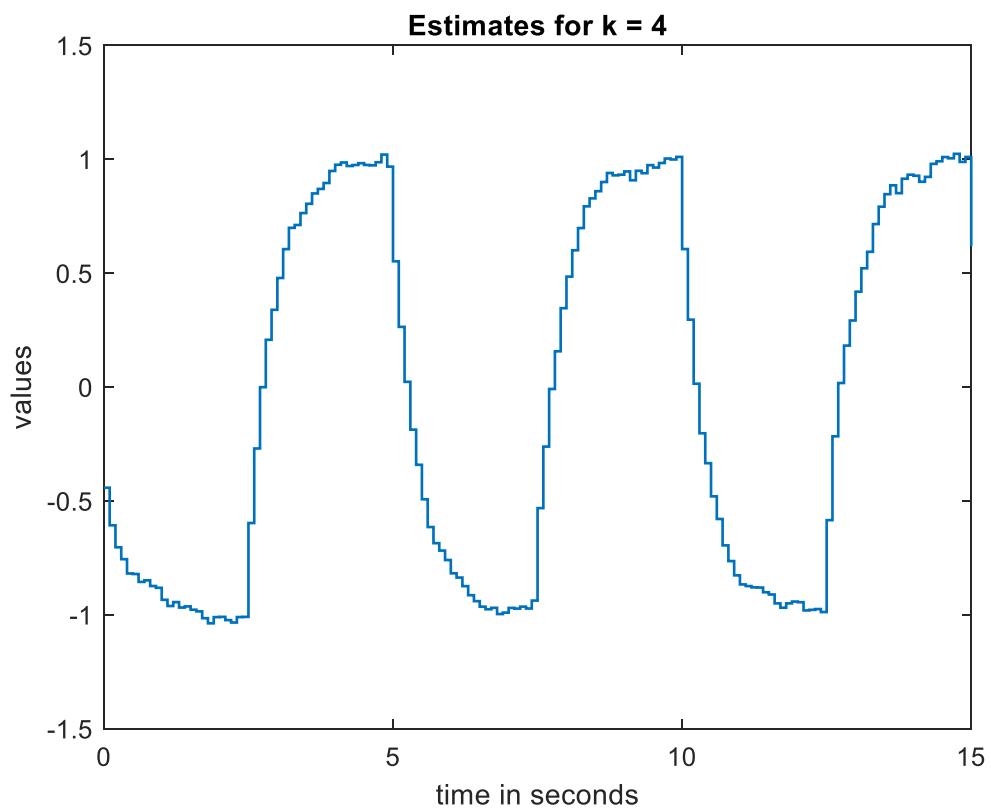
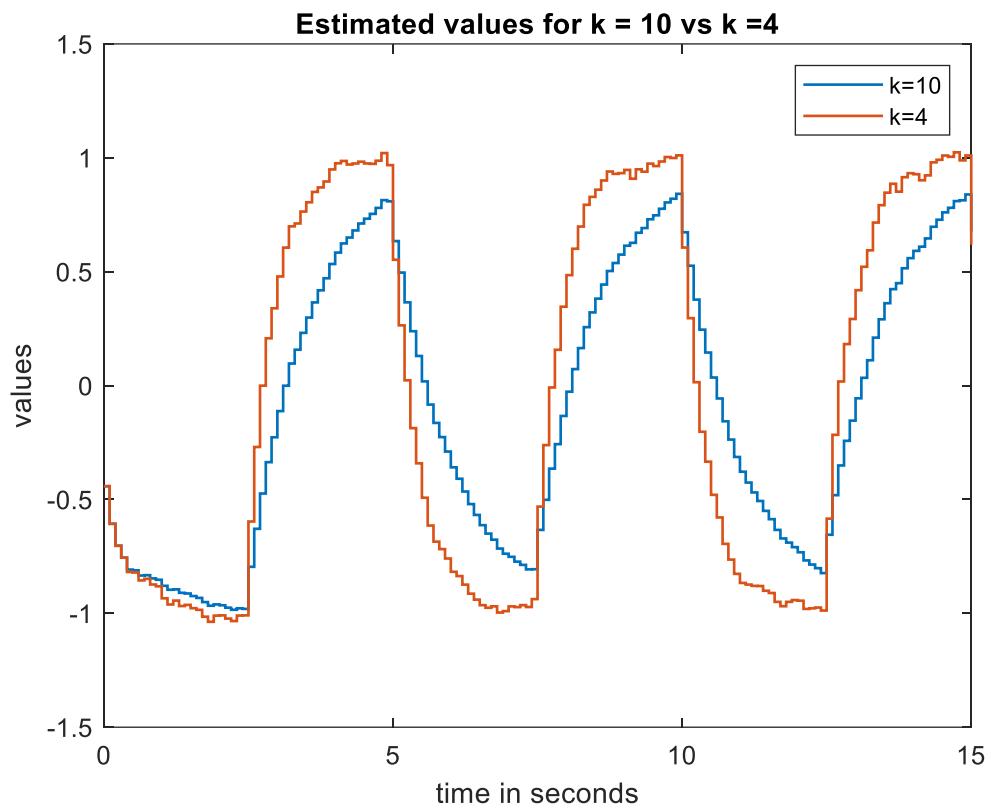


Only the estimated values

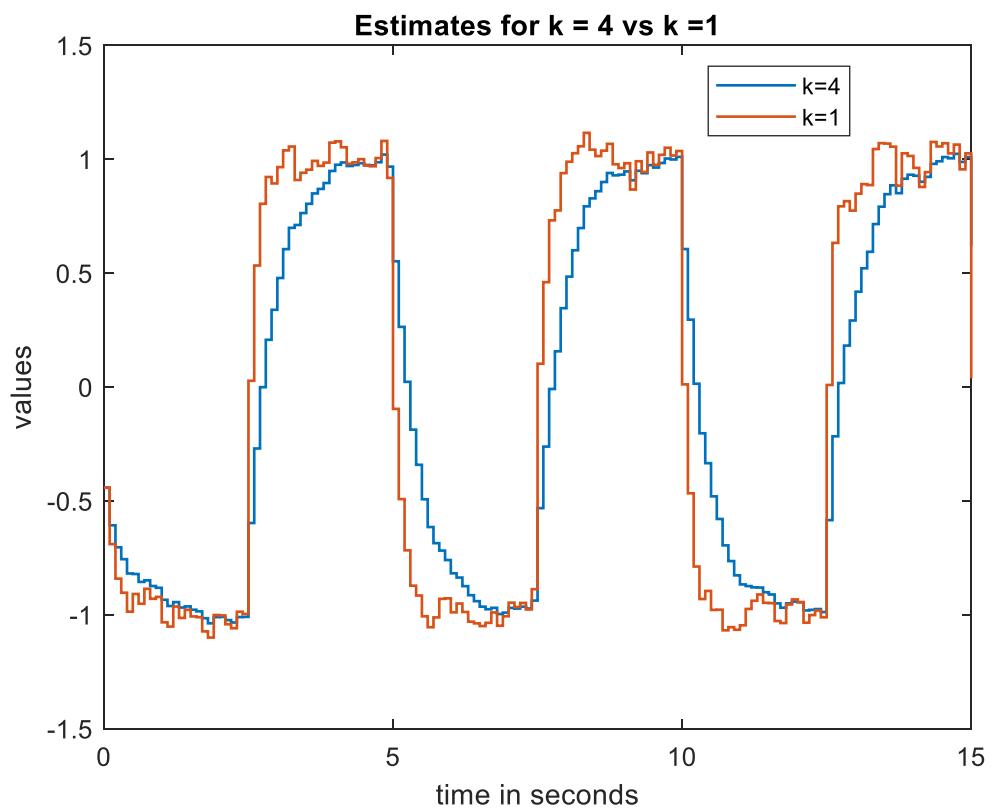
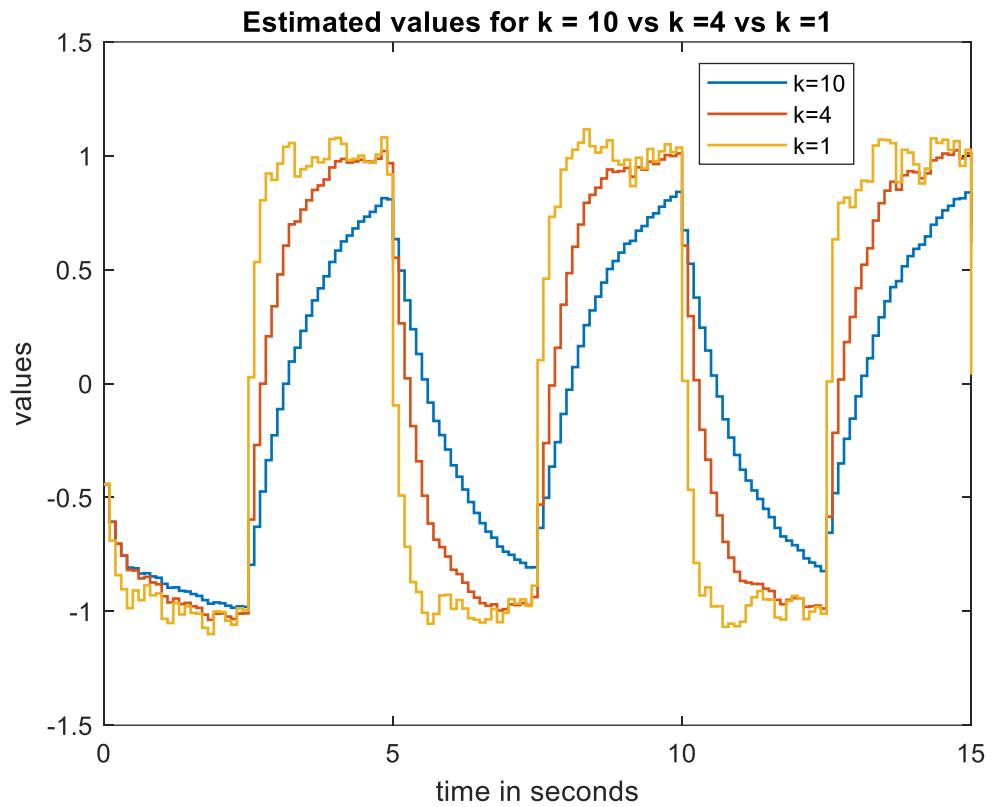


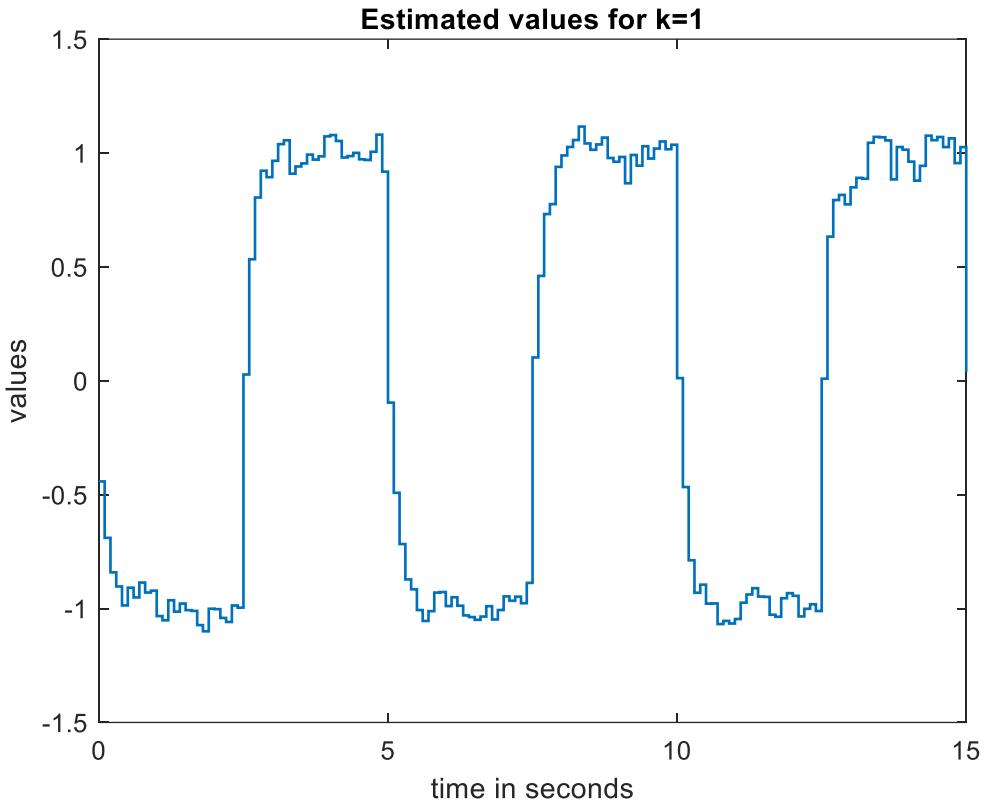
**Plot Command:** >> stairs(tout/MIT(:,2))

d. Plots with  $k = 4$ .



e. Plots with  $k = 1$ .





**Plot Commands: (for part d)**

```
hold on
stairs(tout/MIT(:,2))
figure(2)
stairs(tout/MIT(:,2))
```

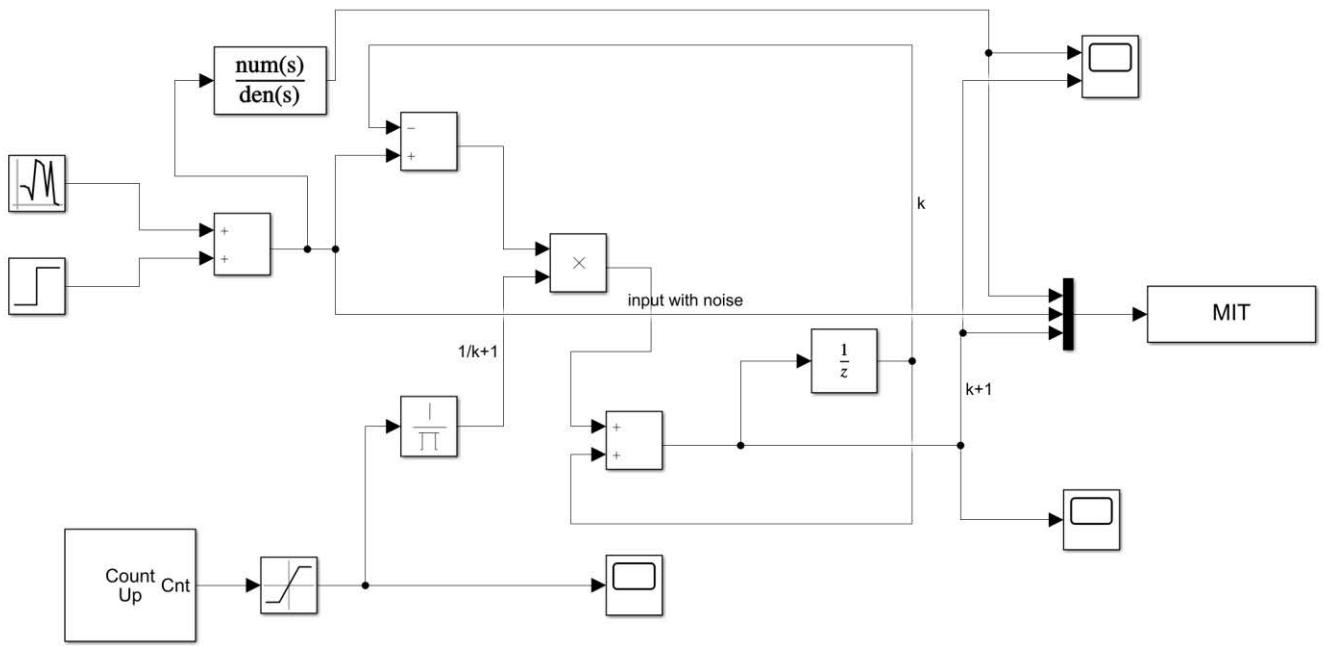
**Plot Commands: (for part e)**

```
hold on
figure(1)
stairs(tout/MIT(:,2))
figure(2)
stairs(tout/MIT(:,2))
figure(3)
stairs(tout/MIT(:,3))
```

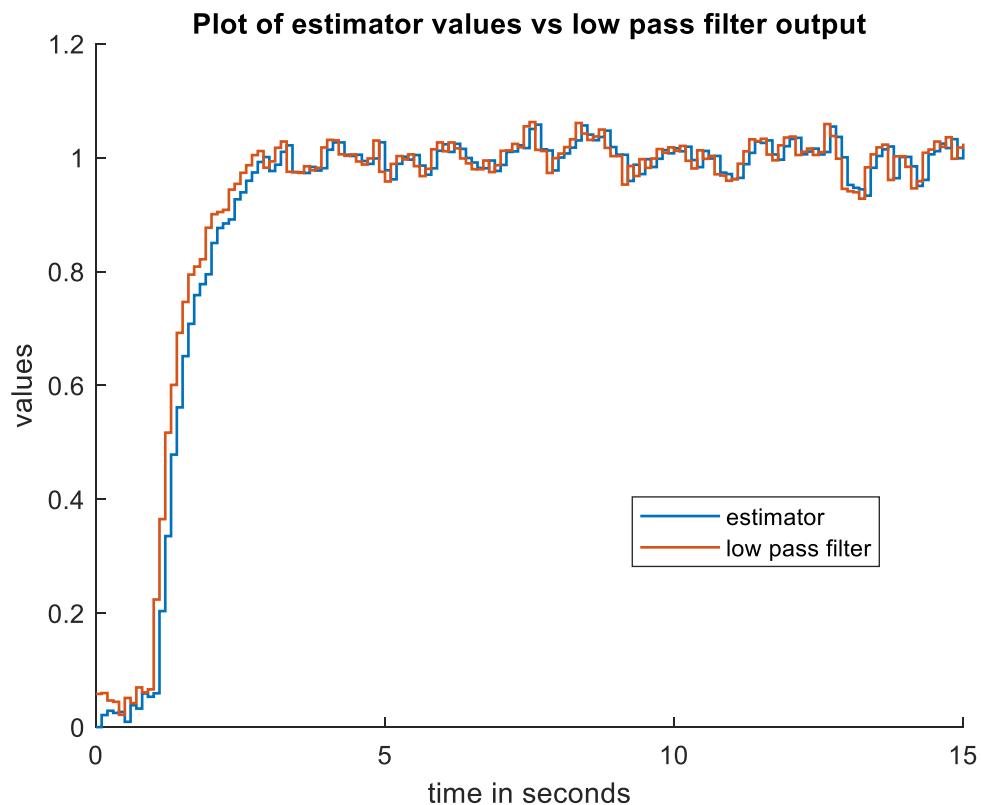
- f. As taught by Dr. Niestroy in class, a higher  $k$  results in lower noise as new values with noise have lower effect on the next estimated value. This is why  $k = 10$  responds much more slowly but doesn't let through any noise. On the other hand,  $k = 4$  responds much faster but allows some noise to pass through. The last case with  $k=1$  is only filtering half the noise but it has the fastest response time. The low filtering capability for the case with  $k=1$  can be readily observed by putting  $k = 1$  in the equation as shown below:

$$\hat{+} = .^* \hat{+} .^* \hat{+}$$

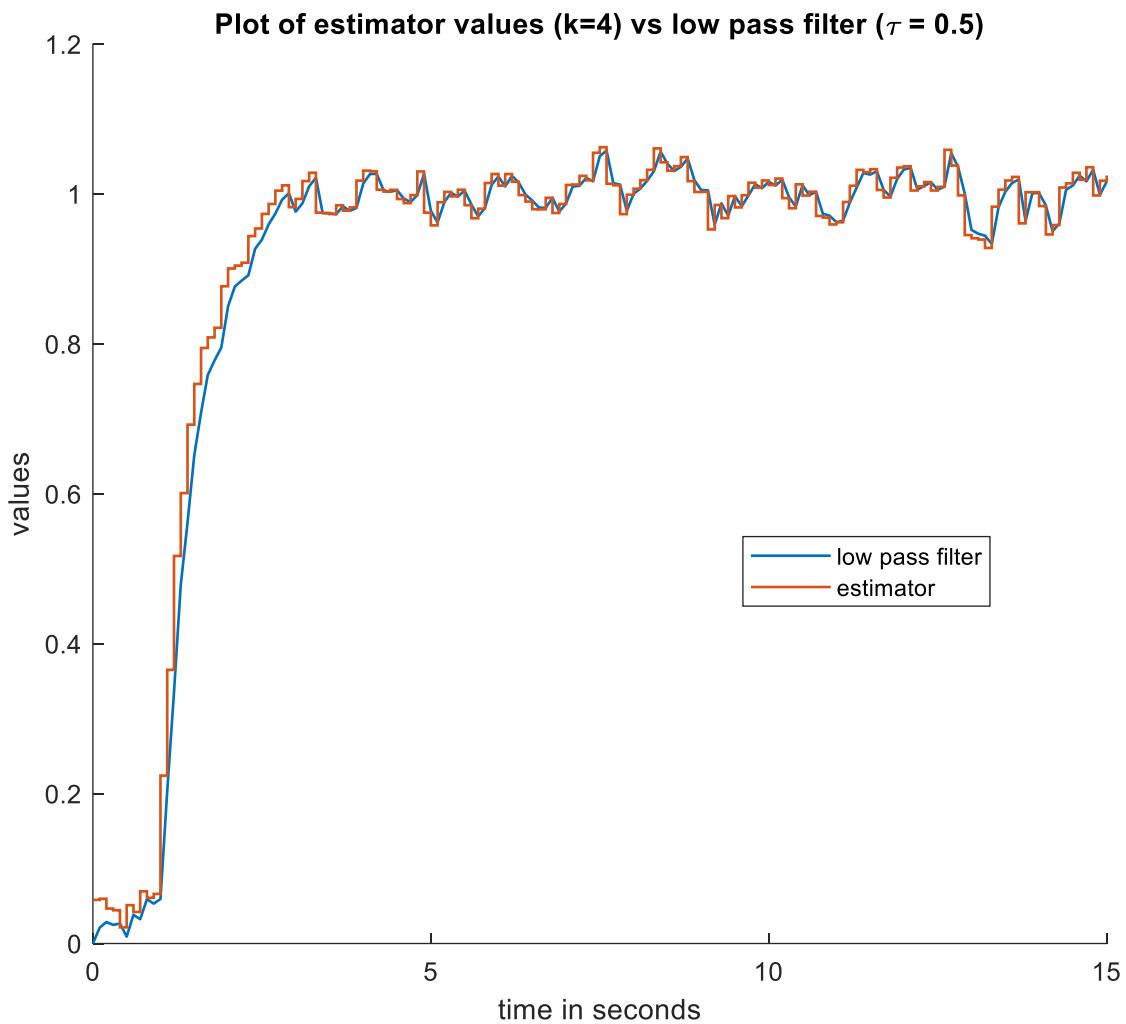
This has been repeated in the appendix to verify when square waves do not have an average value of 0.5.



g.



Since transfer function is a continuous block, it's better to plot it normally than in steps for comparison.

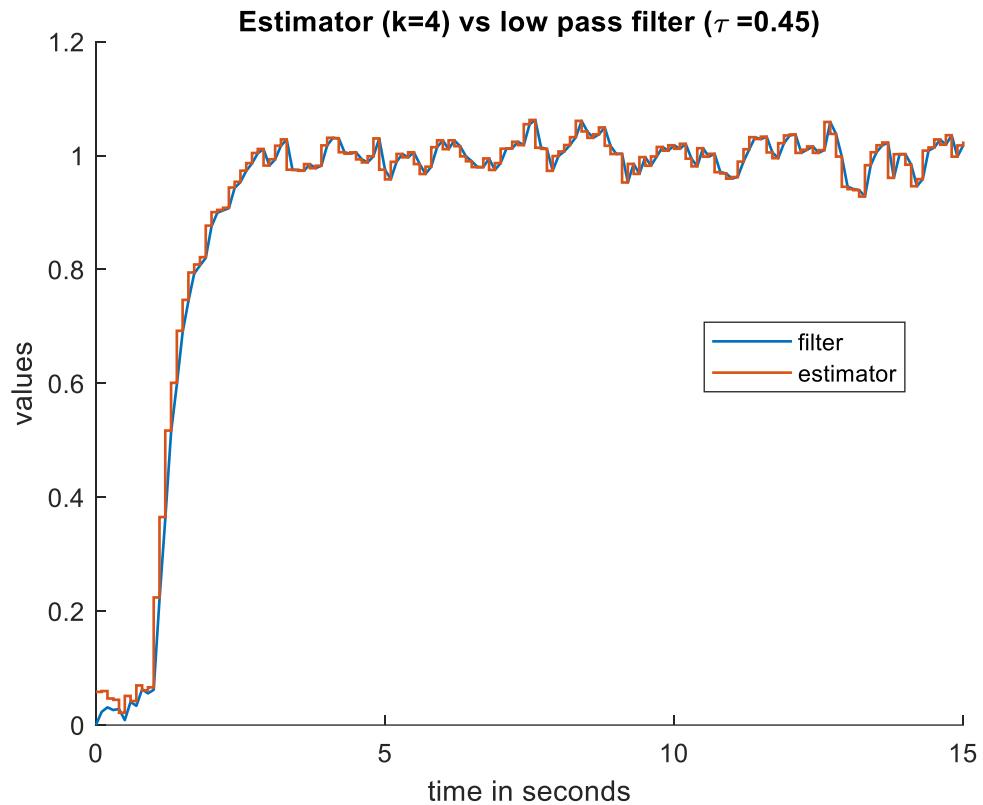


For both the square wave input and the step input (with noise),  $\tau = 0.43$ ,  $\tau = 0.44$  and  $\tau = 0.45$  are all good values for the low pass filter to act as the recursive estimator. I will choose  $\tau = 0.45$  because it passes through the steps most accurately.

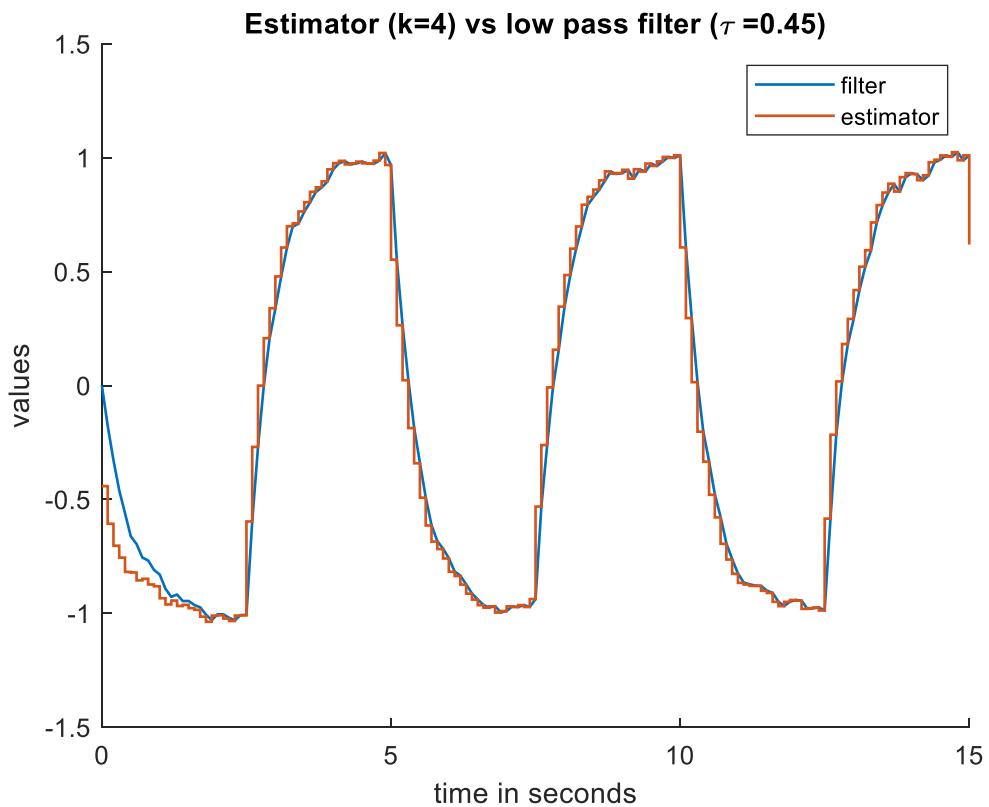
The only thing that changes in the above Simulink diagram is the step input which is changed back to the square wave signal generator. I didn't include that because it would have been redundant.

**The final comparison plots are shown on the next page.** After that page, I have added all the hit and trials I did for your reference in the Appendix.

**Response comparison for step input with noise to estimator with k =4 and low pass filter for tau = 0.45**

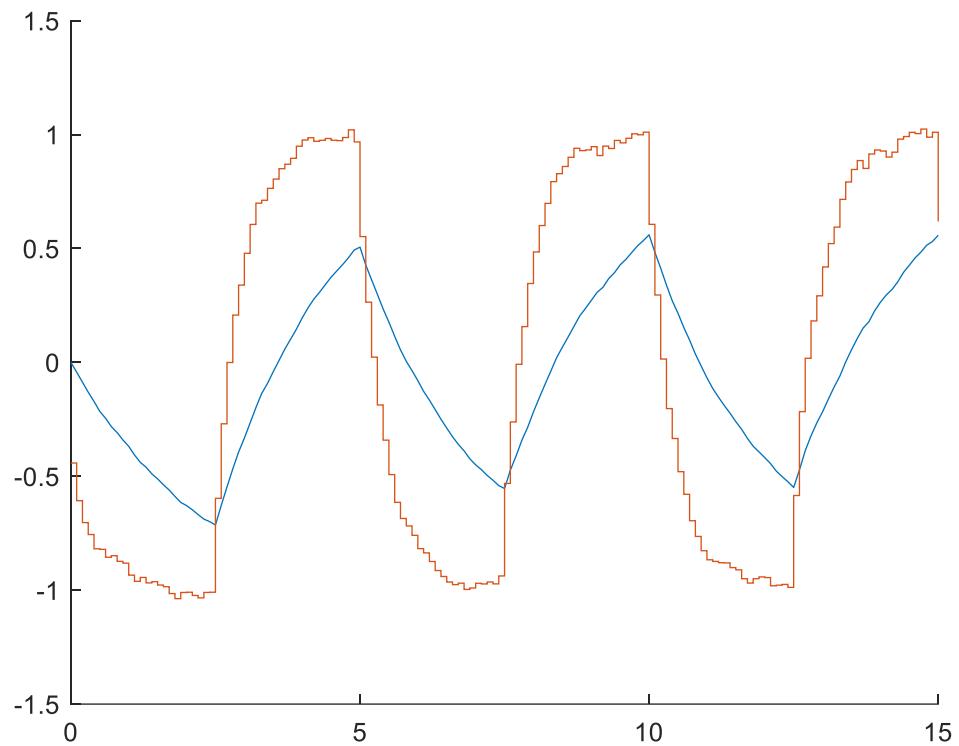


**Response comparison for square wave input of magnitude 1 and frequency 0.2 Hz with noise given to estimator with k =4 and low pass filter for tau = 0.45**

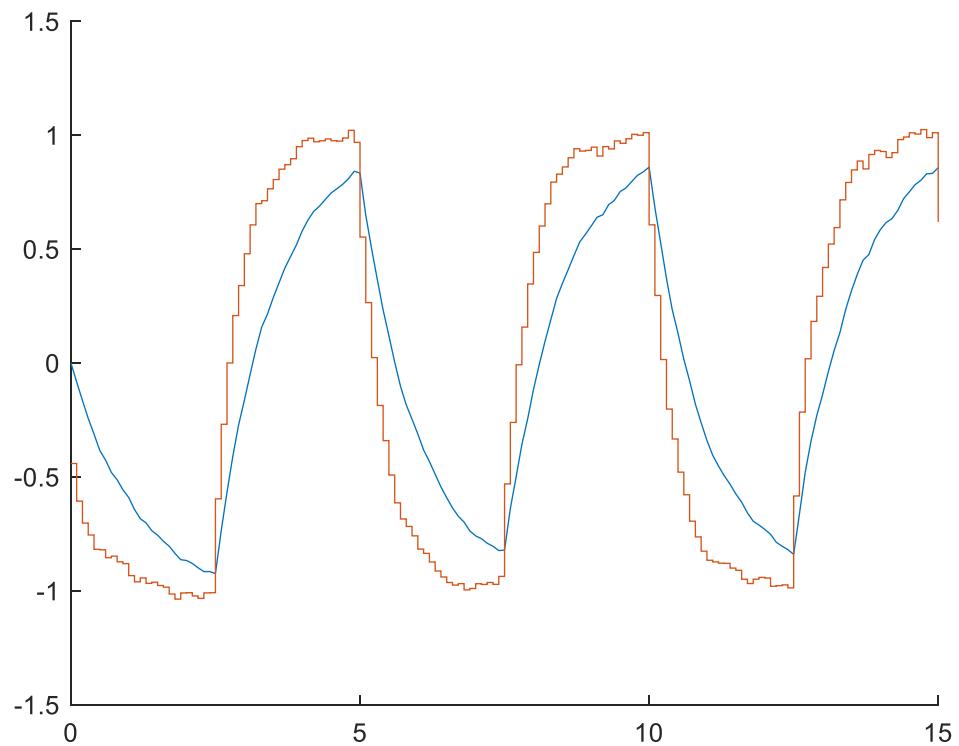


## Appendix:

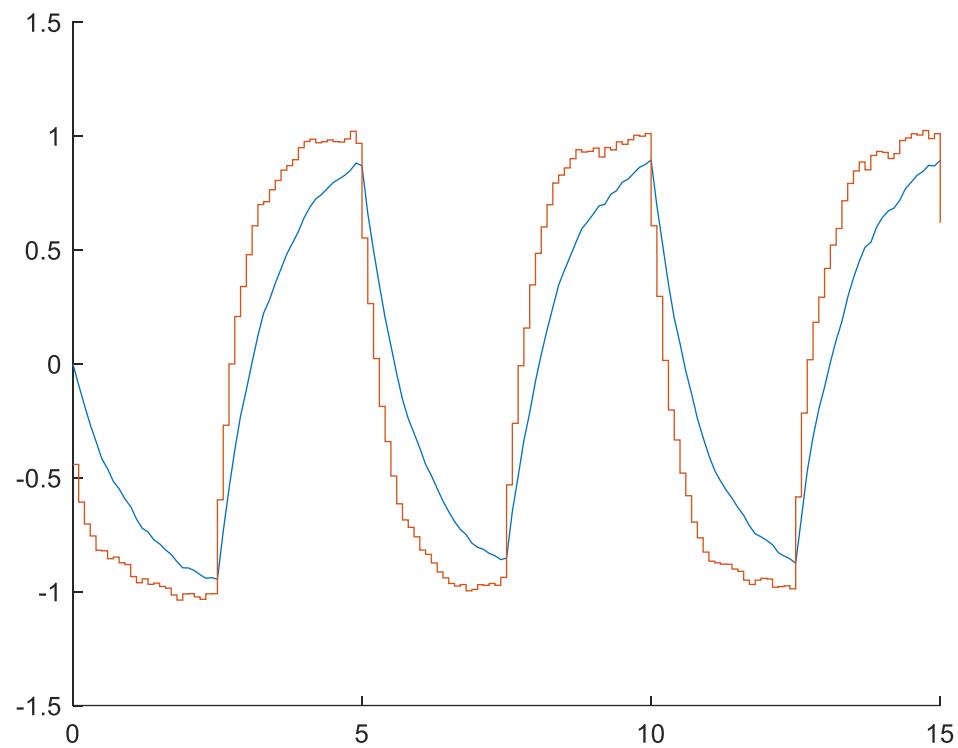
Tau = 2



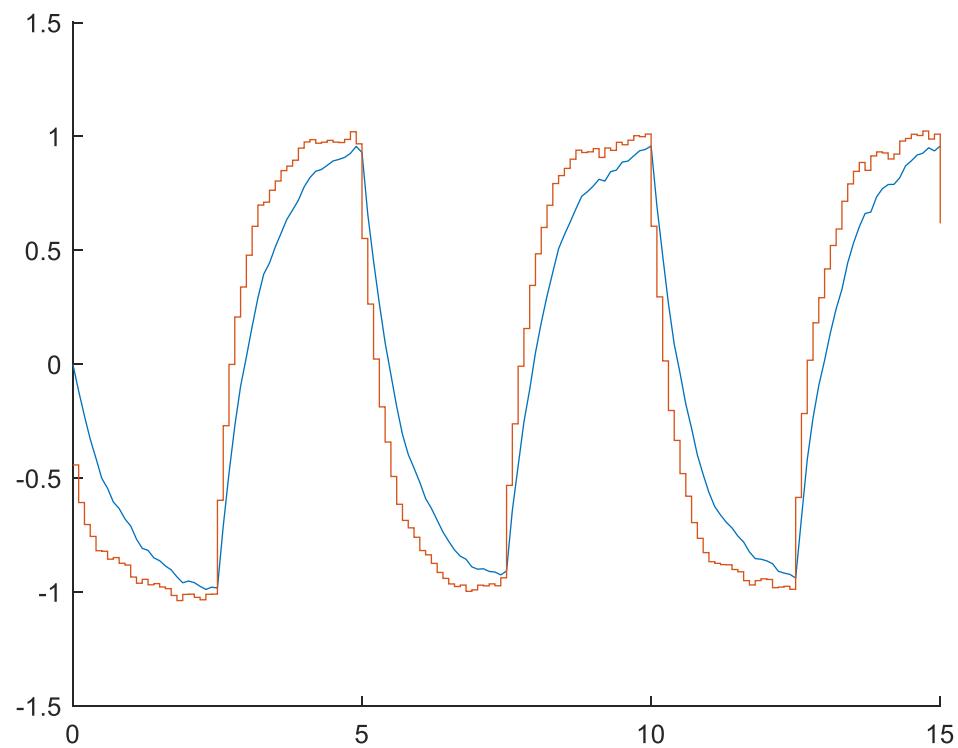
Tau = 1



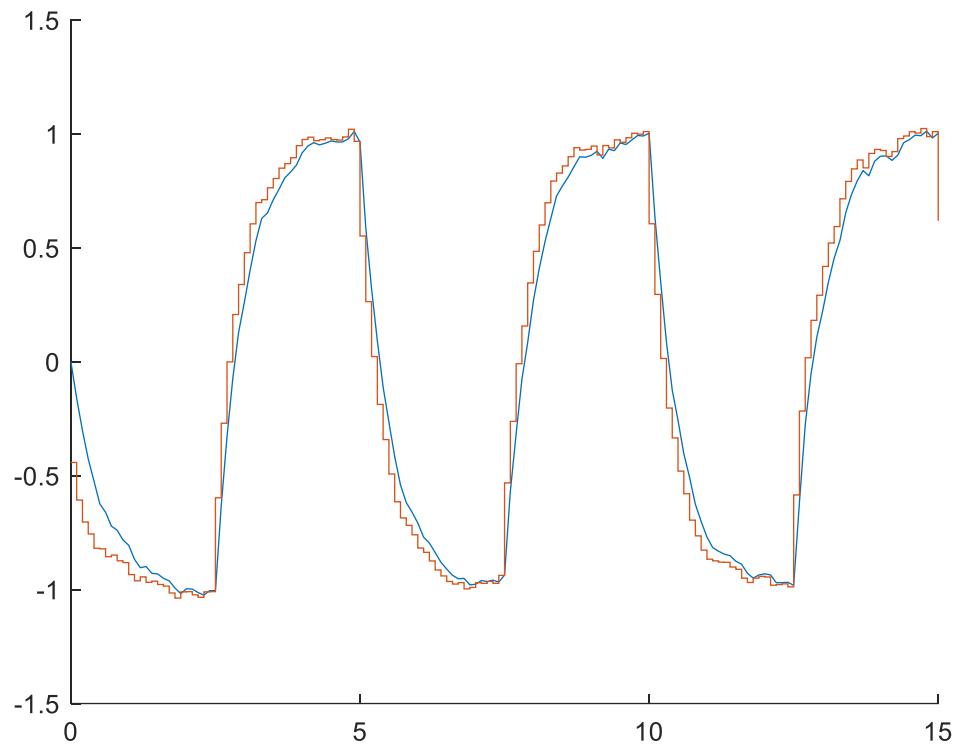
Tau = 0.9



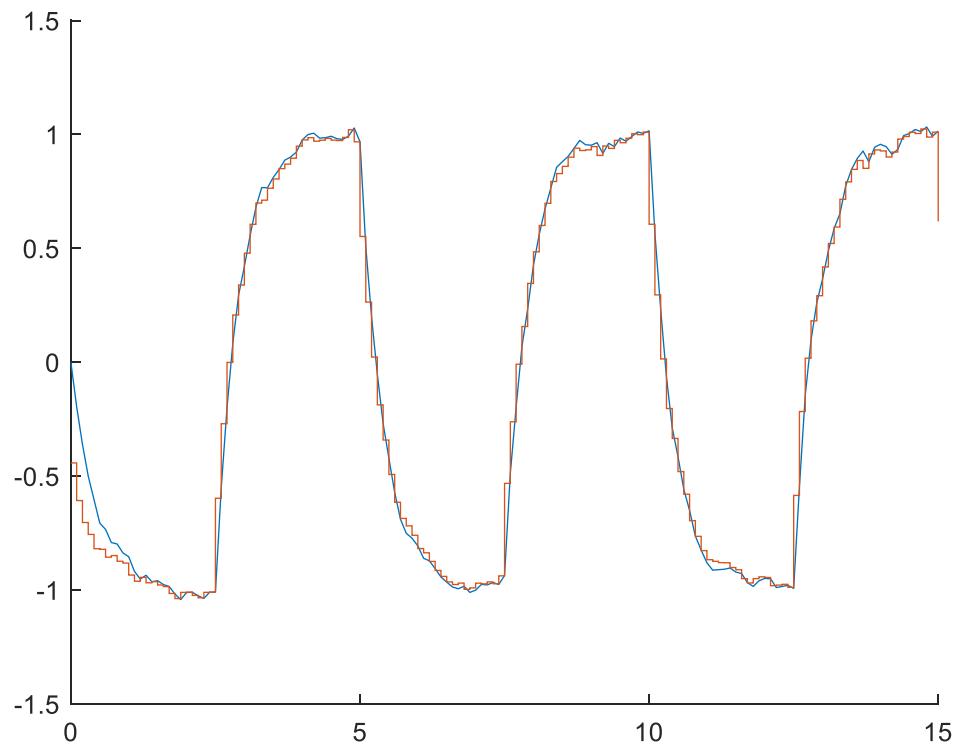
Tau = 0.7



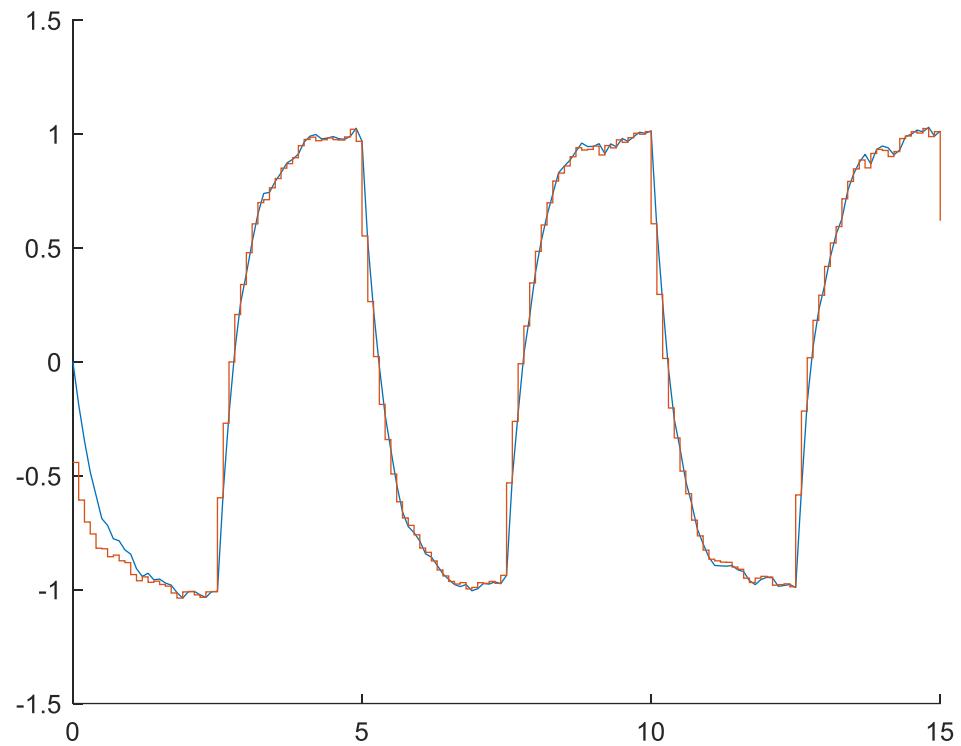
Tau = 0.5



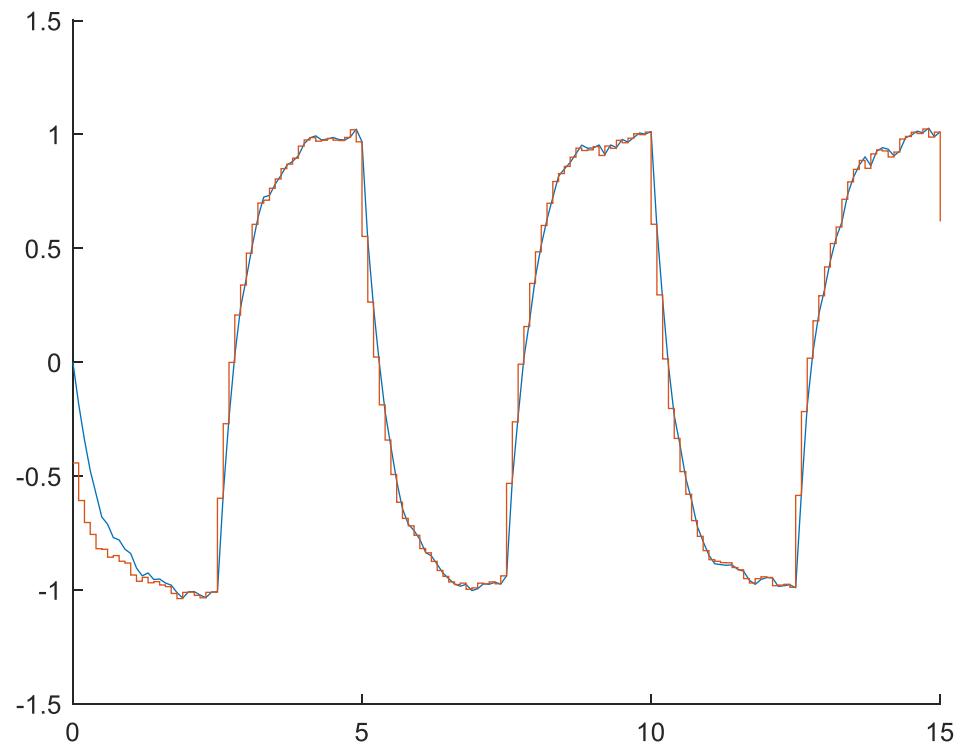
Tau = 0.4



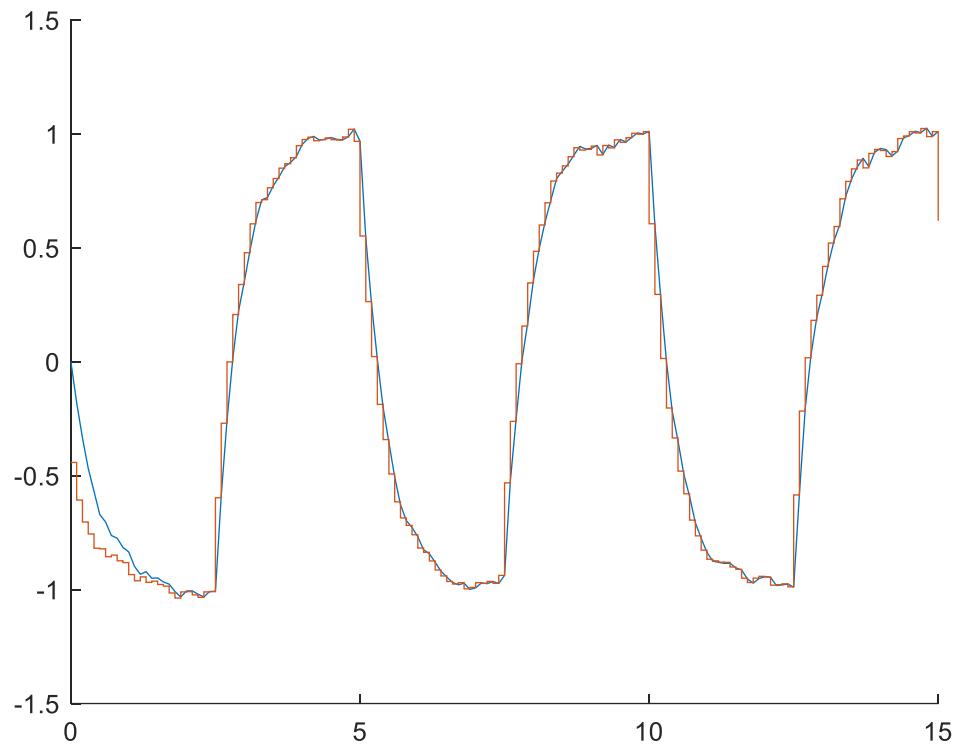
Tau = 0.42



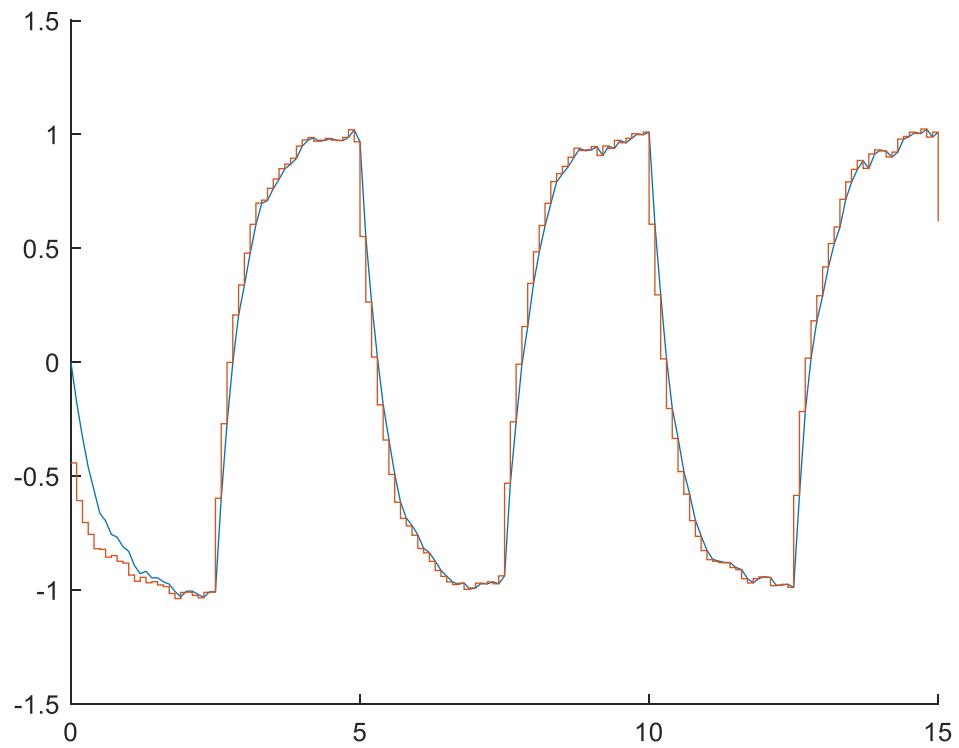
Tau = 0.43



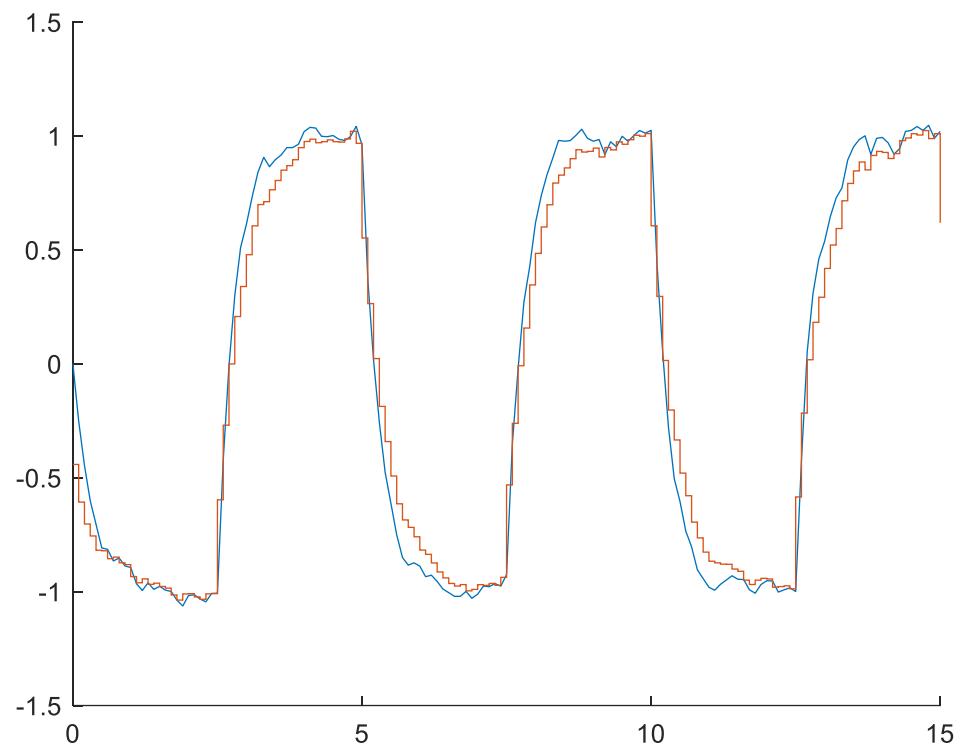
Tau = 0.44



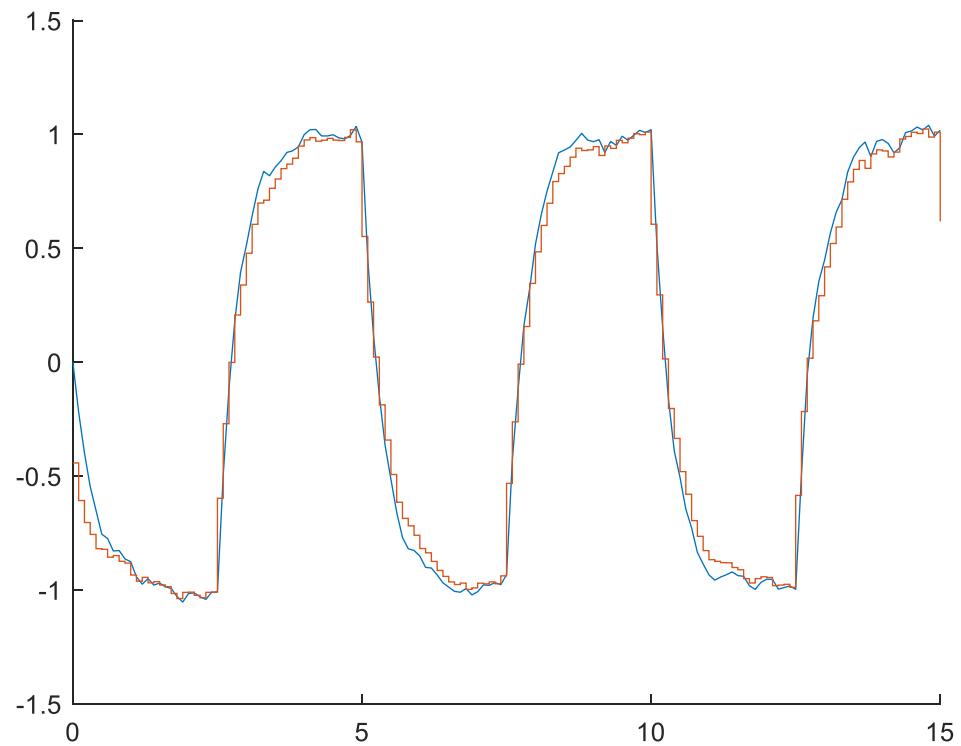
Tau = 0.45



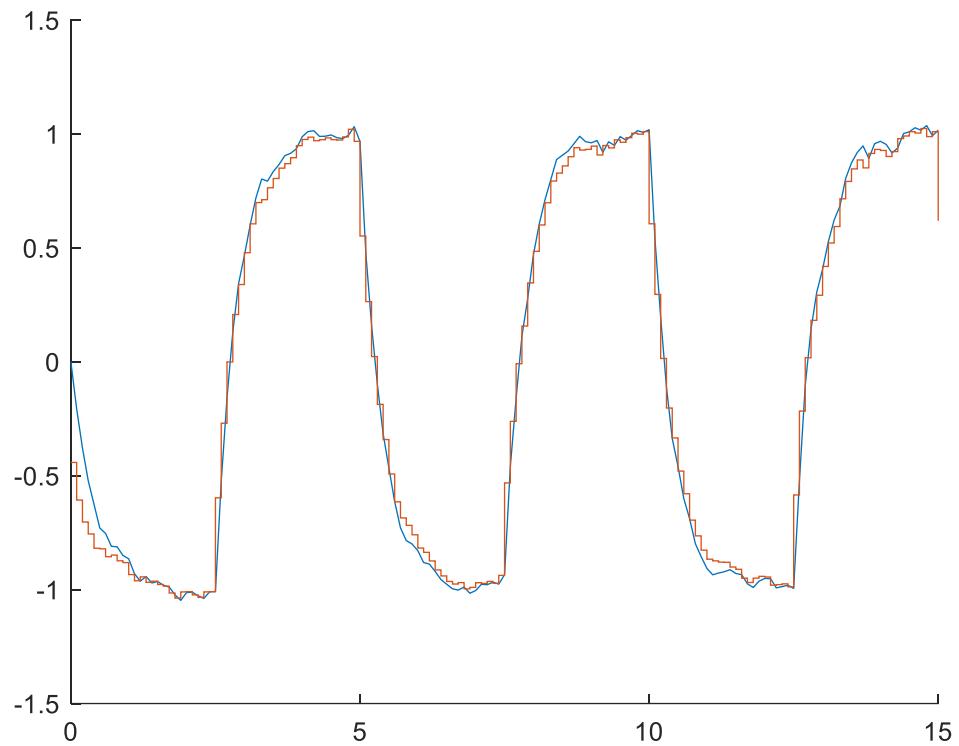
Tau = 0.3



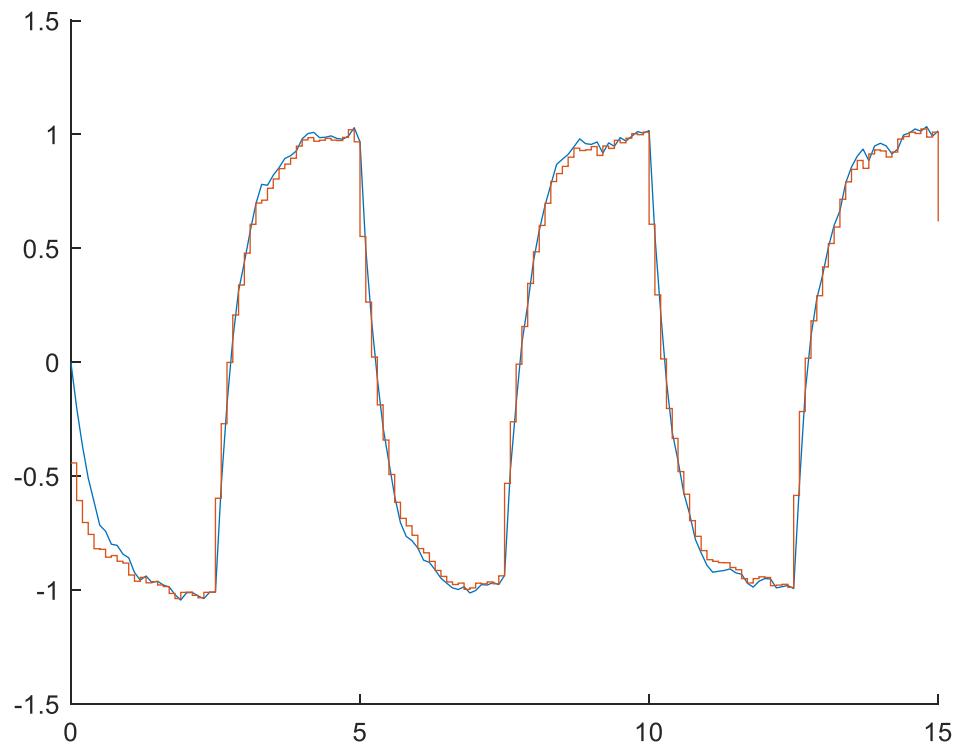
Tau = 0.35



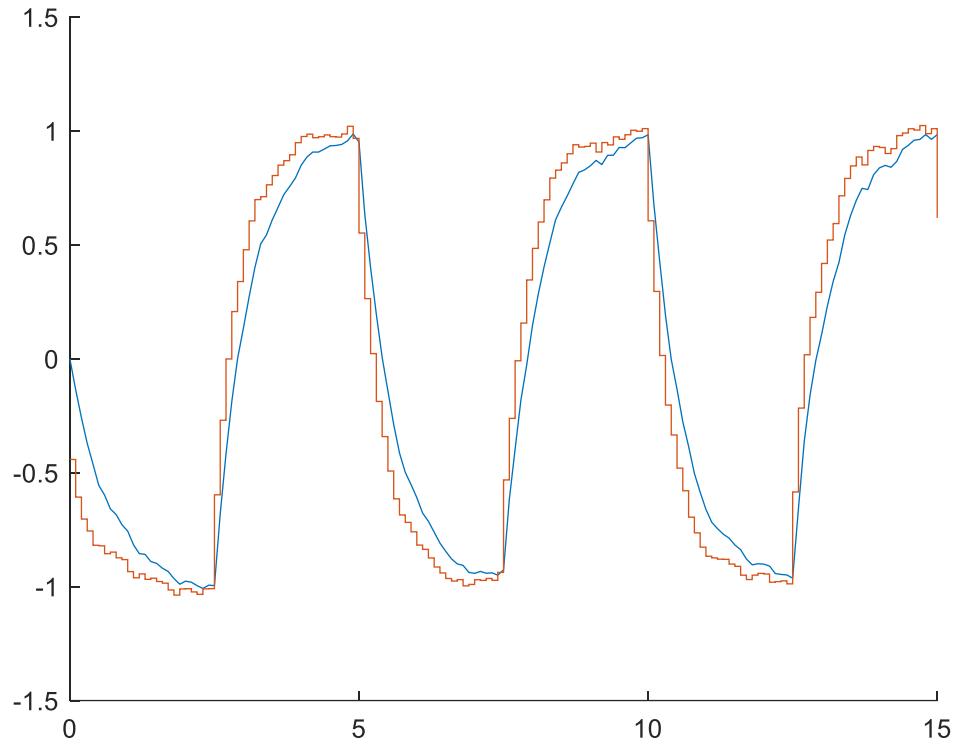
Tau = 0.375



Tau = 0.39

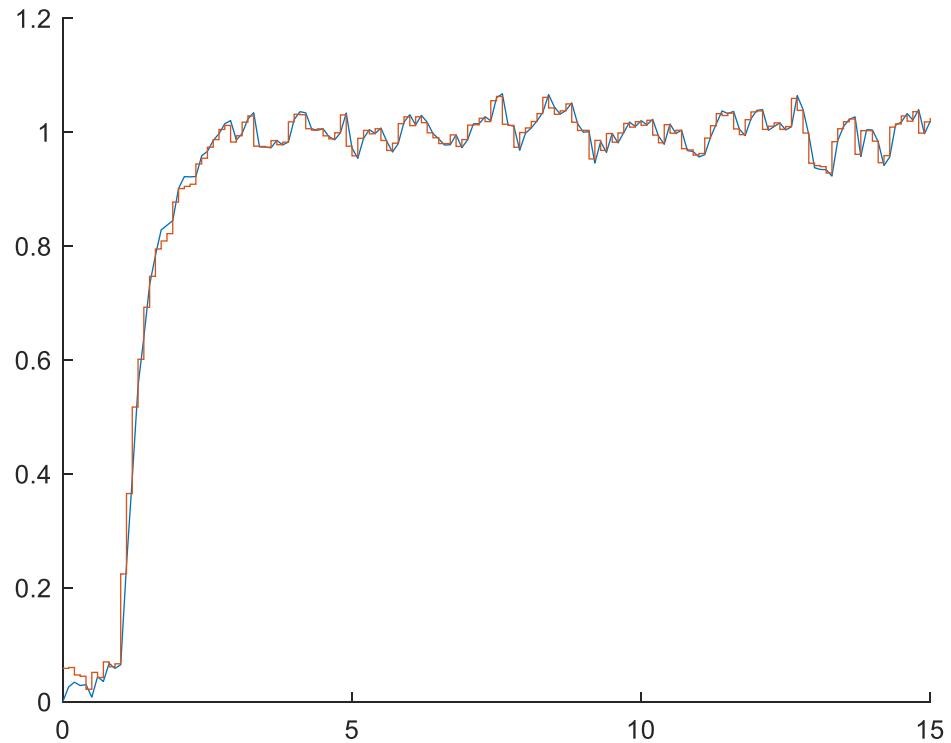


Tau = 0.6

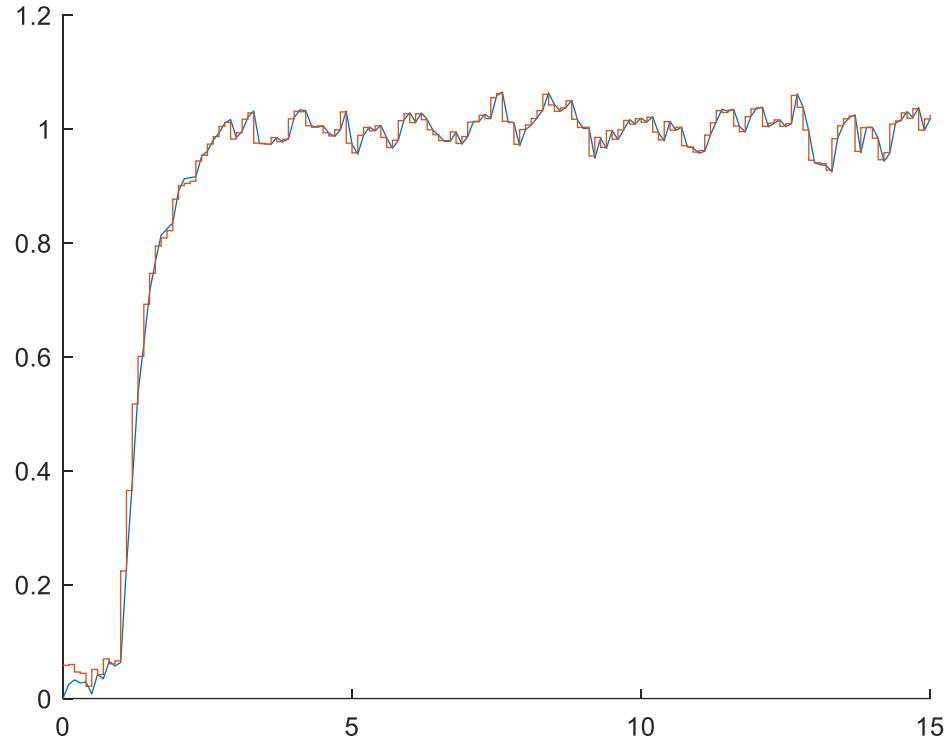


**Iterations for the step input with noise**

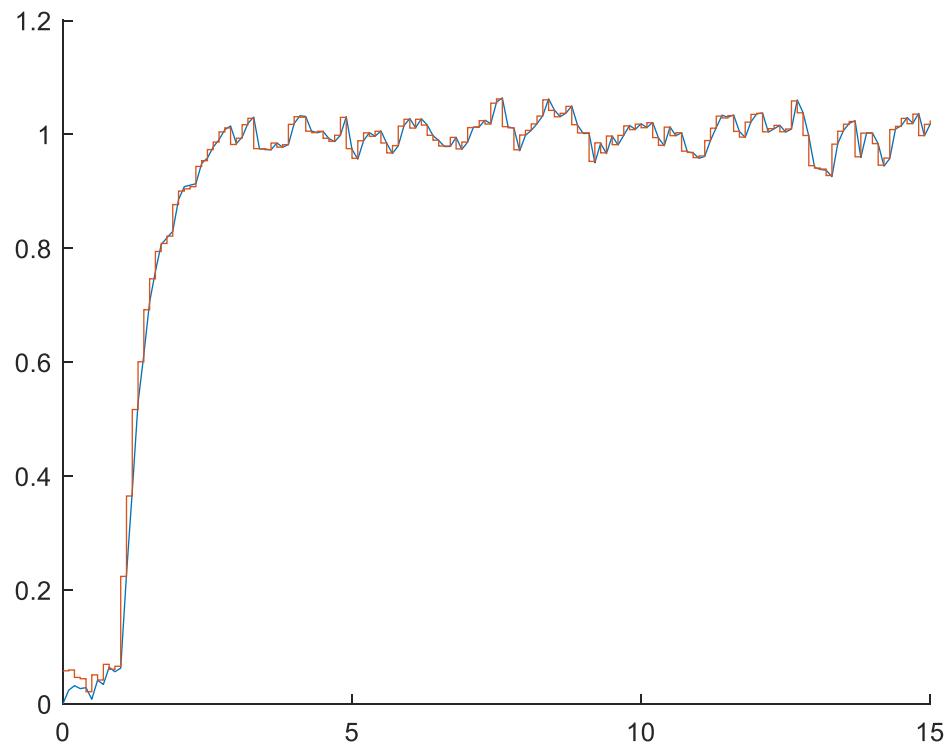
Tau = 0.4



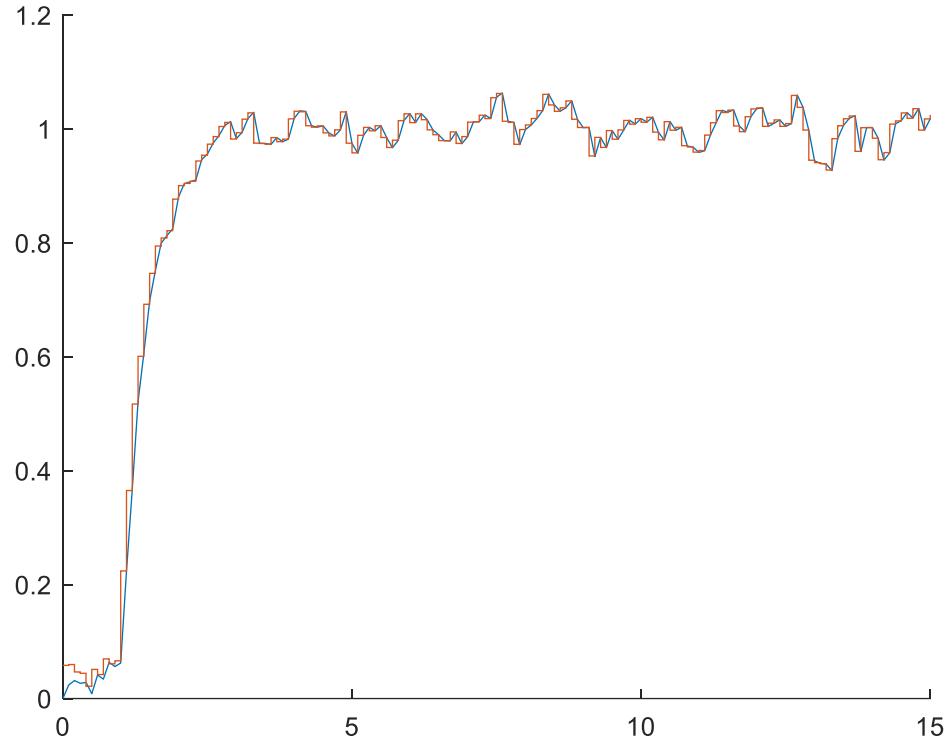
Tau = 0.42



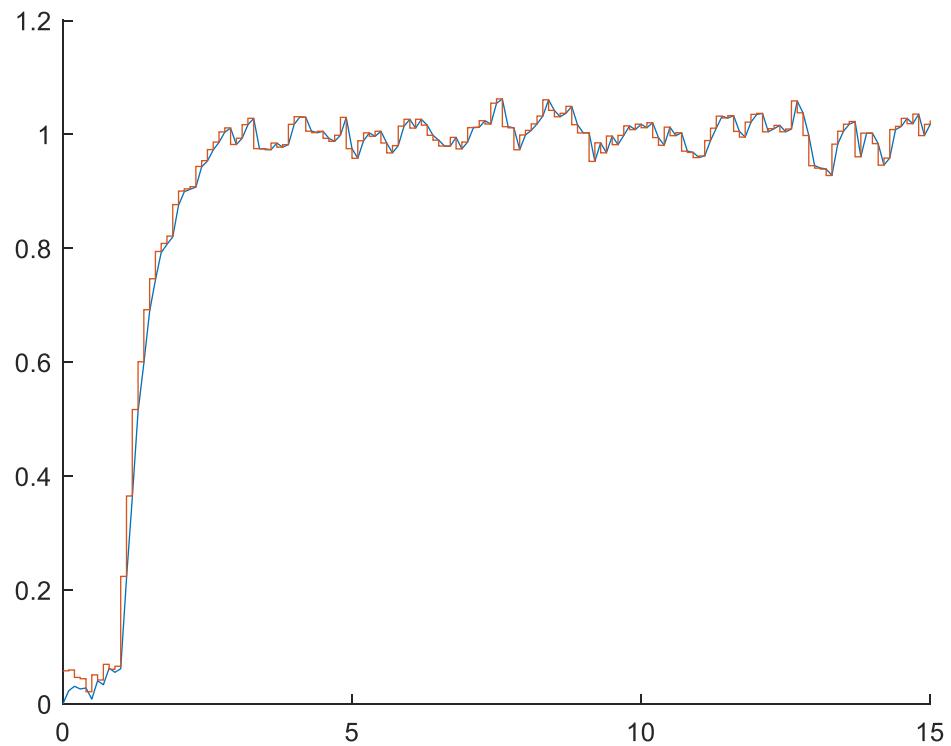
Tau = 0.43



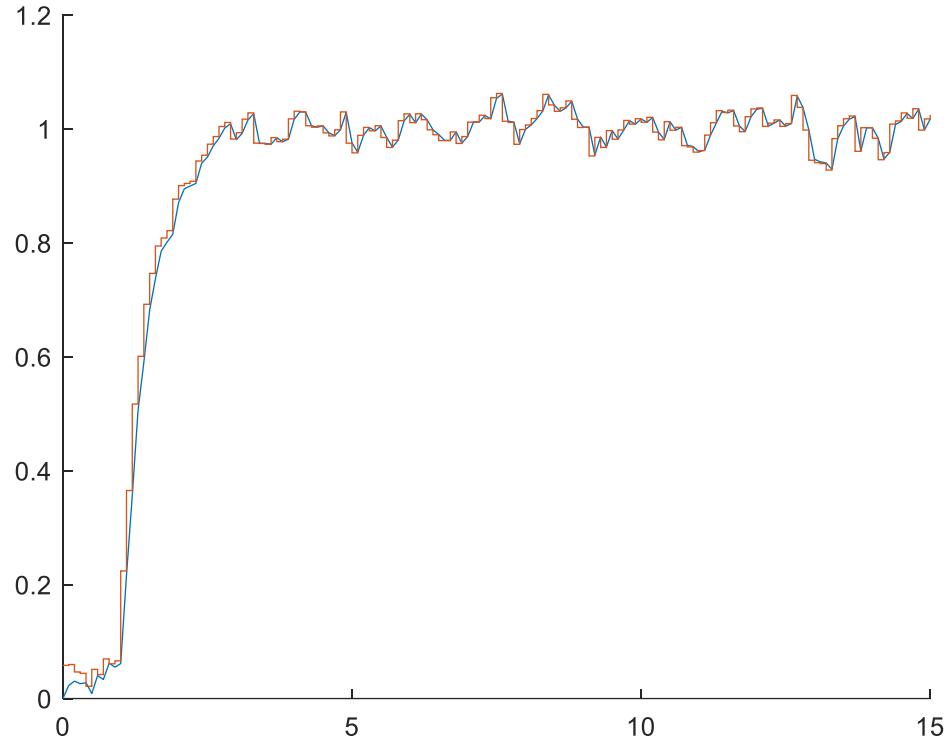
Tau = 0.44



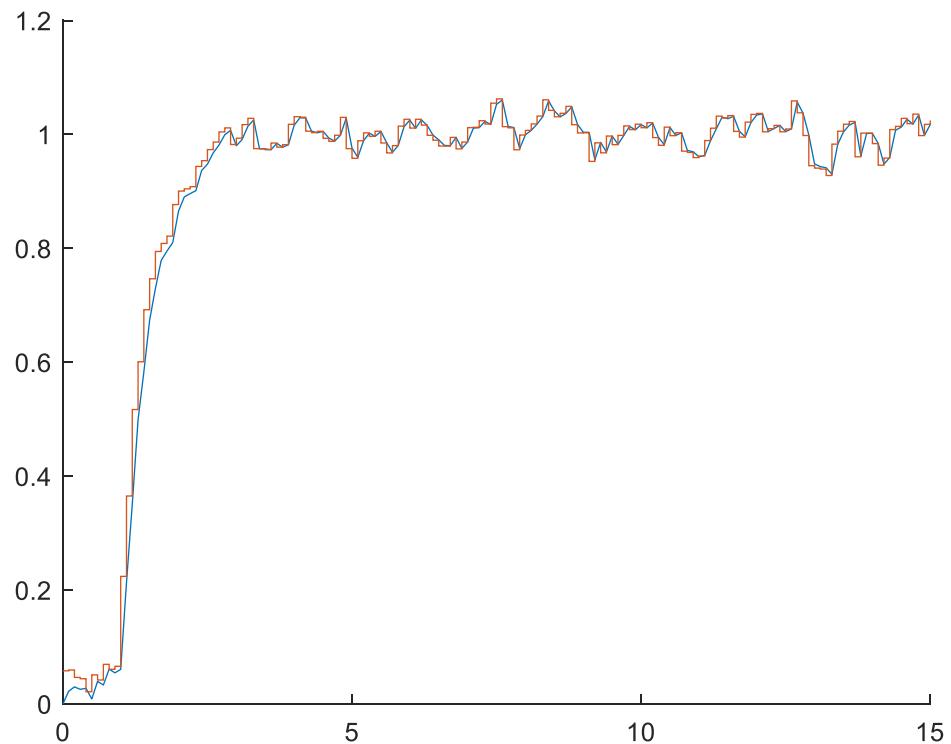
Tau = 0.45



Tau = 0.46



Tau = 0.47



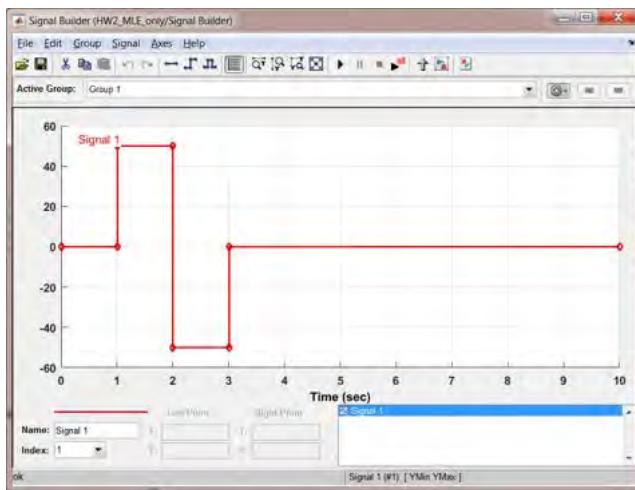
**EE5327, Fall 2020**  
**Homework 3: Maximum Likelihood Estimation**  
**And Discrete-Time Kalman Filter**  
**Due 10/8/2020**

**Problem 1) 40 points**

The below system represents the longitudinal dynamics of an aircraft trimmed as some flight condition. The control input is the elevator position in degrees and the states and outputs are as follows:

$$x = \begin{bmatrix} : pitch\ angle, rad \\ q : pitch\ rate, rad/sec \\ u : x - axis\ velocity, ft/sec \\ w : z - axis\ velocity, ft/sec \end{bmatrix} \quad y = \begin{bmatrix} : aircraft\ total\ velocity, ft/sec \\ : pitch\ angle, deg \\ q : pitch\ rate, deg/sec \\ : angle\ of\ attack, deg \end{bmatrix}$$

- a) Implement this aircraft in Simulink (using the State-Space block?) and use the below figure to create the control input to the system using the Signal Builder block from the Sources library. Input the same signal to both inputs. Plot the state (X) and output (Y) response to the below control input. (15 points)
- b) Now, create the measurement (Z) by adding normal random noise to each of the outputs. Use the variance values of [0.01, 0.01, 0.1, 0.08] (assume these are also the diagonal values in the R matrix) for the noise, respectively, and add the noise to the output. Change the sample time on the random noise to 0.01 seconds. Generate the measurement time histories. (10 points)
- c) Next, implement the **maximum likelihood estimator** and create the time history of the estimated states ( $\hat{x}$ ). (10 points)
- d) Finally, generate figures that overlay the outputs (Y) and measurements (Z) on the same figure and the true states (X) and estimated states ( $\hat{x}$ ) on a different figure. (5 points)



```
a = ...
[ -0.0000  1.0000 -0.0000 -0.0000
 -0.0001 -0.6707  0.0001 -0.0108
 -31.9238 -60.0651 -0.0065  0.0508
 -2.1418  888.0565 -0.0432 -0.7067];
```

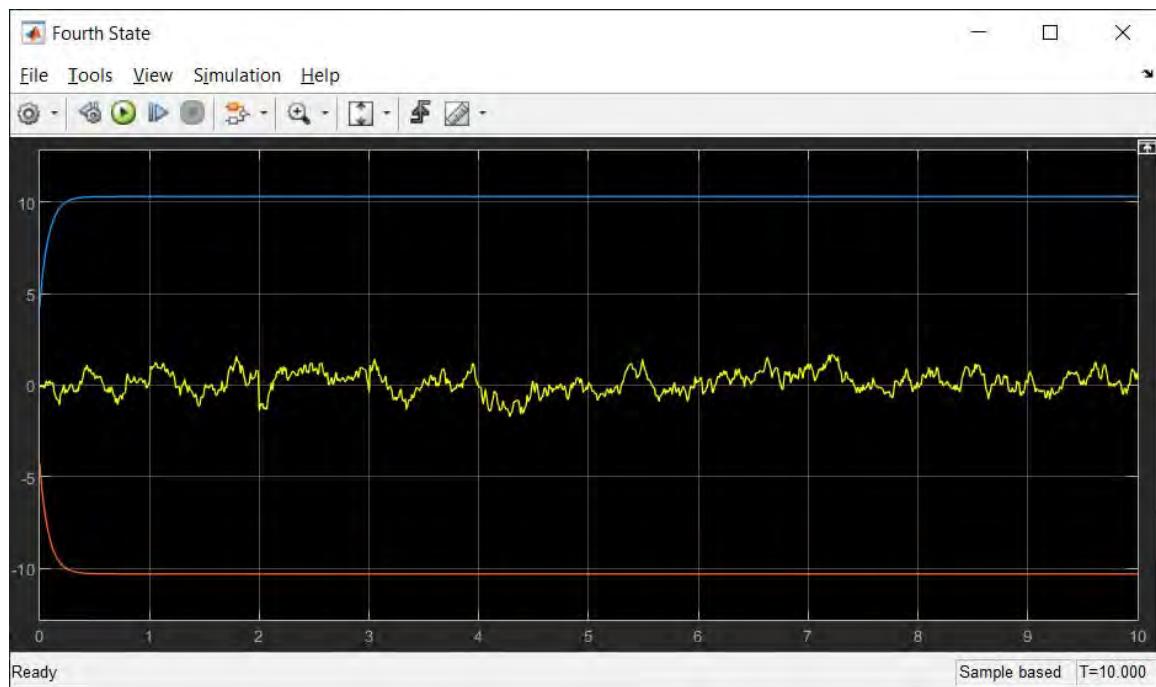
```
b = ...
[ -0.0000 -0.0000
 -0.0003 -0.0003
  0.0002  0.0002
 -0.9184 -0.9184];
```

```
c = ...
[0.1600  0.0028  0.9978  0.0668
 57.2958 -0.2865  0.0000  0.0000
 0.0000  57.4879 -0.0000  0.0031
 0.0000 -0.2849 -0.0043  0.0641];
```

**Problem 2)** Discrete-Time Kalman Filter 50 points

To the diagram from problem 1, add a discrete-time Kalman filter and compare the results as follows:

- Implement the discrete-time Kalman filter (DKF) in the Simulink diagram you created for Problem 1. Feel free to use MATLAB or Embedded MATLAB blocks to make creation easier. Take the initial value of the covariance matrix P to be identity. Use Q as identity as well but keep the R values from Problem 1 and the same input. Plot the estimates from the DKF as compared to the estimates from the maximum likelihood estimator. Comment on the similarities or differences in the estimates from the two methods (20 points)
- Now, let's look closer at the covariance matrix and the corresponding errors between the true states and the estimated states. For the Q and R values listed above, plot the time history of each state error on a scope/graph along with the + and – square root value of the respective covariance matrix. An example is shown below in which the yellow line is the state error, the blue line is the  $+\sqrt{P(4,4)}$  and the red line is the  $-\sqrt{P(4,4)}$ . (20 points)
- Now experiment by running four additional cases of varying Q and R values. Run the cases where  $Q = 0.01*\text{eye}(4)$  and  $Q = 100*\text{eye}(4)$  and note the changes in the graphs that the effects of process noise have on the estimates. Make  $Q = \text{eye}(4)$  again and then run cases where  $R = 100$  times the original values and where  $R = 0.01$  times the original values to see the effects of measurement noise on the estimates. (10 points)
- Based upon these variations in the Q and R matrices, which of them is closest to being ‘tuned’, meaning that the error in the states lies within +/- the square root of the associated covariance matrix elements about 80-90% of the time, but not completely within those bounds. (10 points)



**Atul Shrotriya**  
**UTA ID - 1001812437**

**System Identification and Estimation**  
**Homework 3**

## Problem - 1:

### a. Initializing matrices

```
A=[0 1 0 0;-0.0001 -0.6707 0.0001 -0.0108; -31.9238 -60.0651 -0.0065 0.0508; -2.1418  
888.0565 -0.0432 -0.7067]
```

A =

```
0 1.0000 0 0  
-0.0001 -0.6707 0.0001 -0.0108  
-31.9238 -60.0651 -0.0065 0.0508  
-2.1418 888.0565 -0.0432 -0.7067
```

```
>> B=[0 0;-0.0003 -0.0003;0.0002 0.0002;-0.9184 -0.9184]
```

B =

```
0 0  
-0.0003 -0.0003  
0.0002 0.0002  
-0.9184 -0.9184
```

```
>> C=[0.1600 0.0028 0.9978 0.0668; 57.2958 -0.2865 0 0; 0 57.4879 0 0.0031;0 -0.2849 -  
0.0043 0.0641]
```

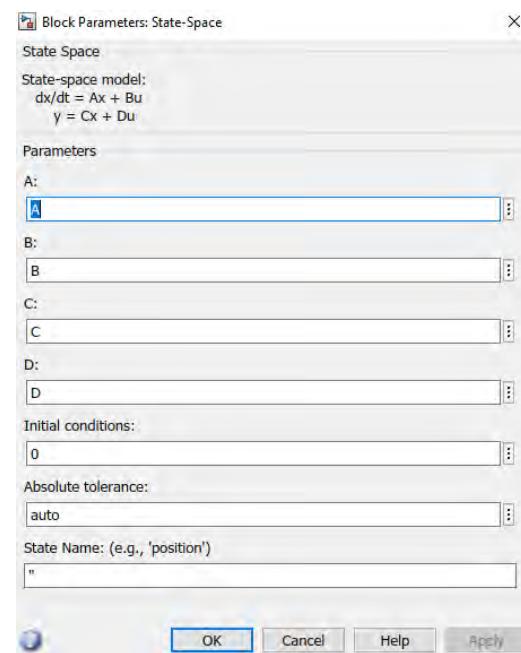
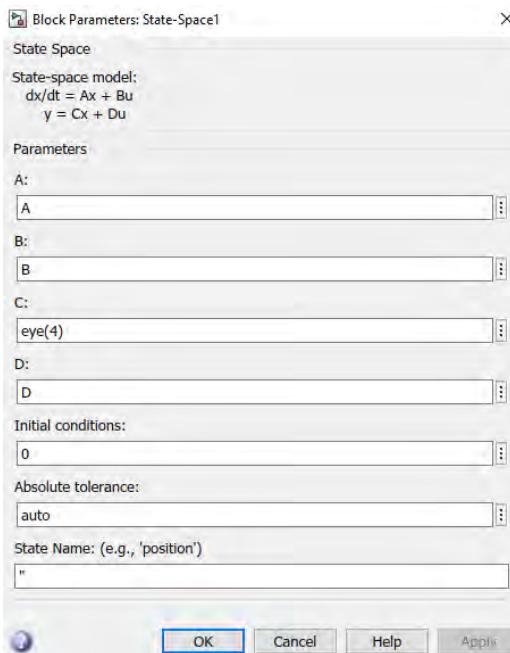
C =

```
0.1600 0.0028 0.9978 0.0668  
57.2958 -0.2865 0 0  
0 57.4879 0 0.0031  
0 -0.2849 -0.0043 0.0641
```

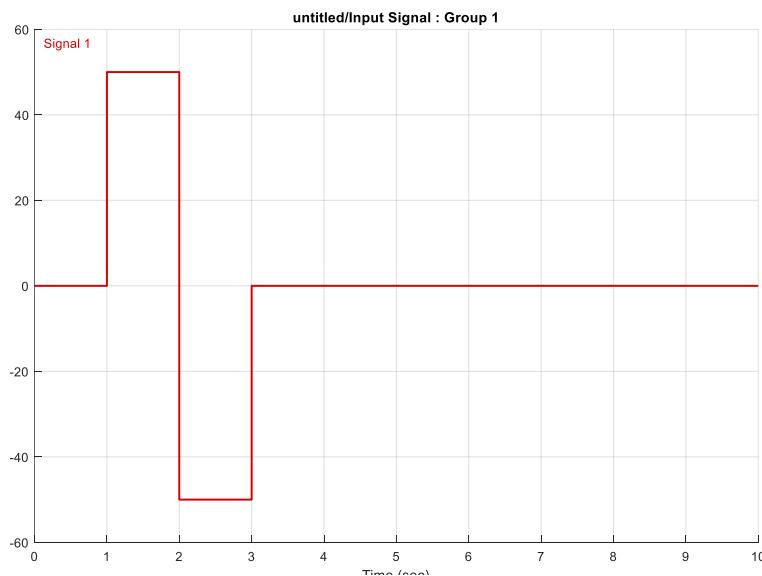
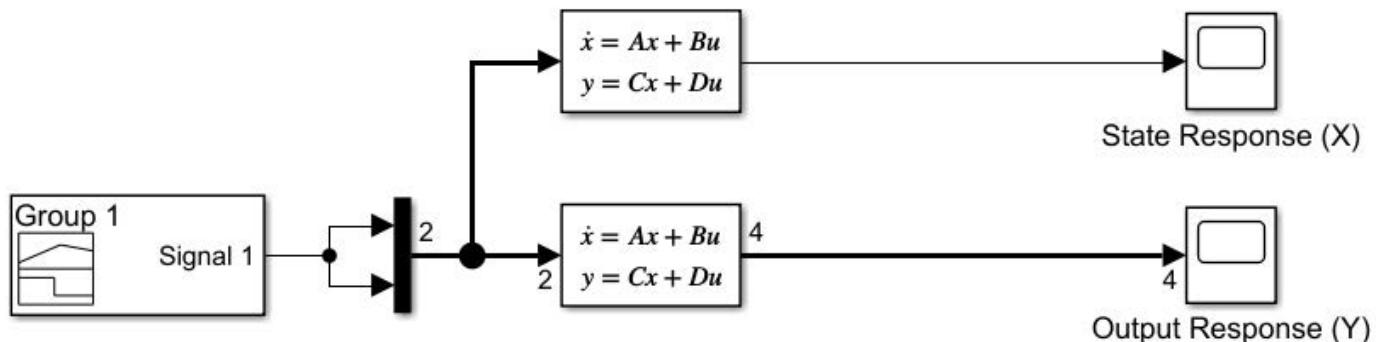
```
>> D=zeros(4,2)
```

D =

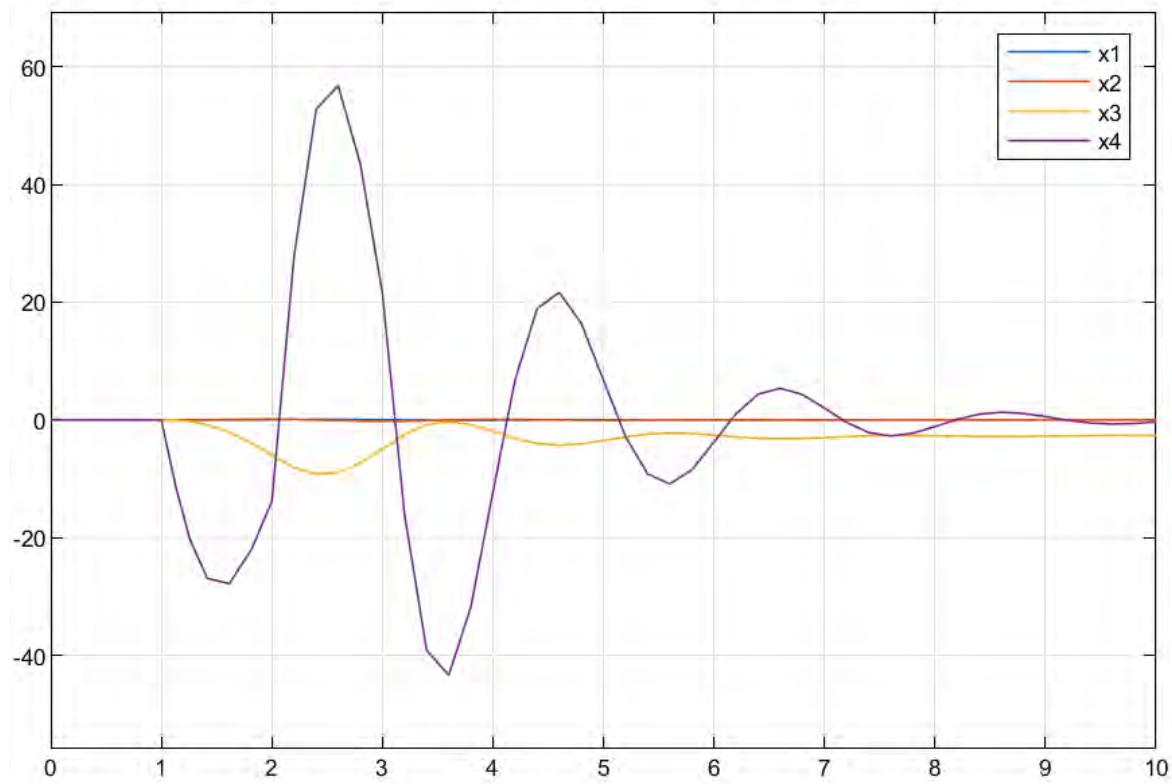
```
0 0  
0 0  
0 0  
0 0
```



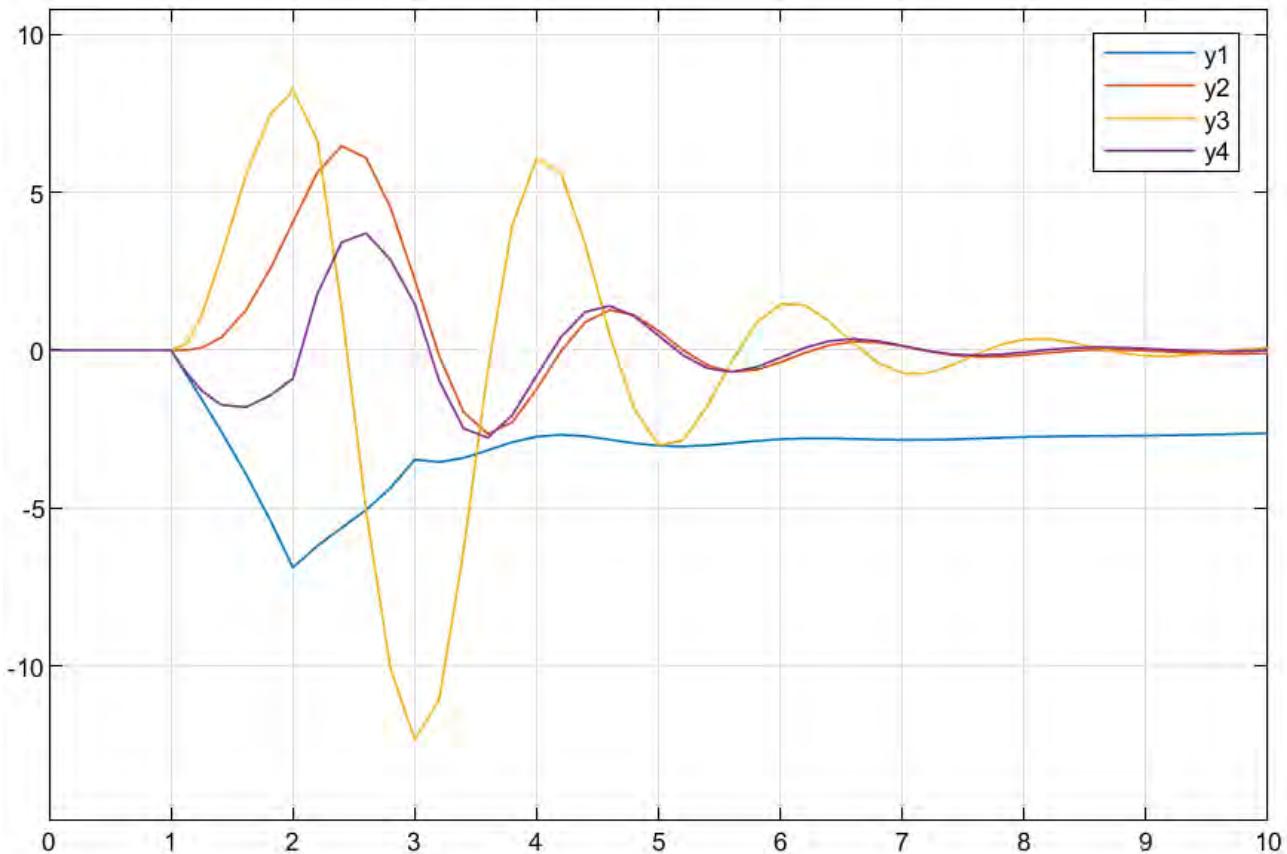
### Simulink Diagram:



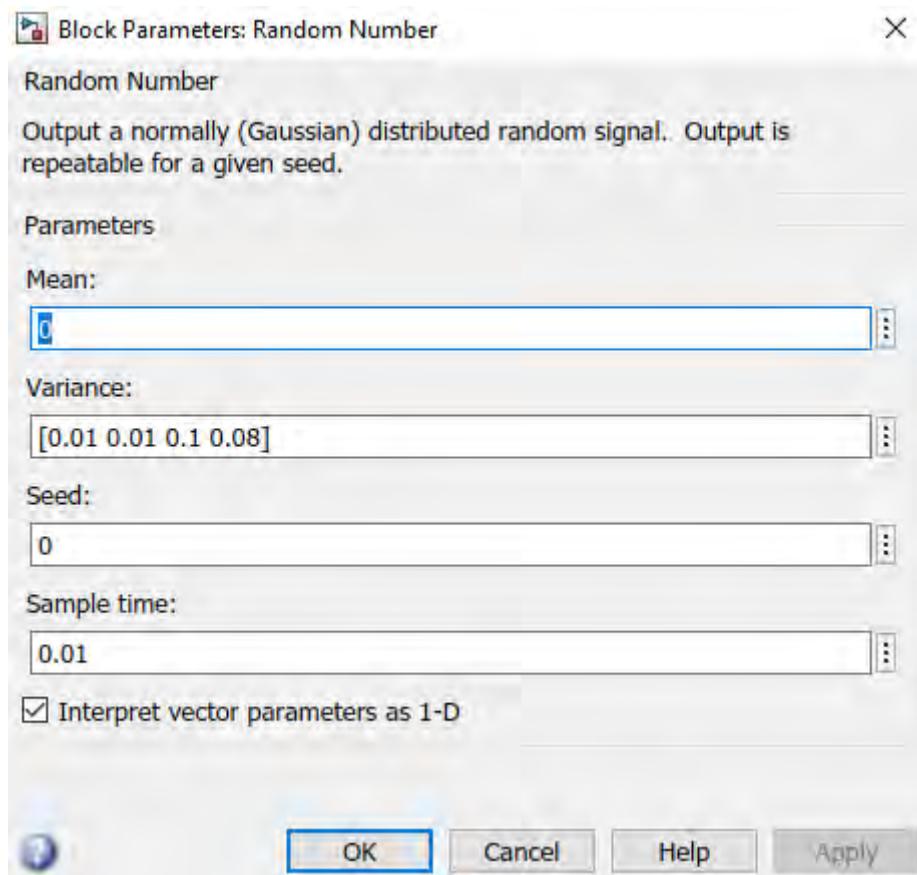
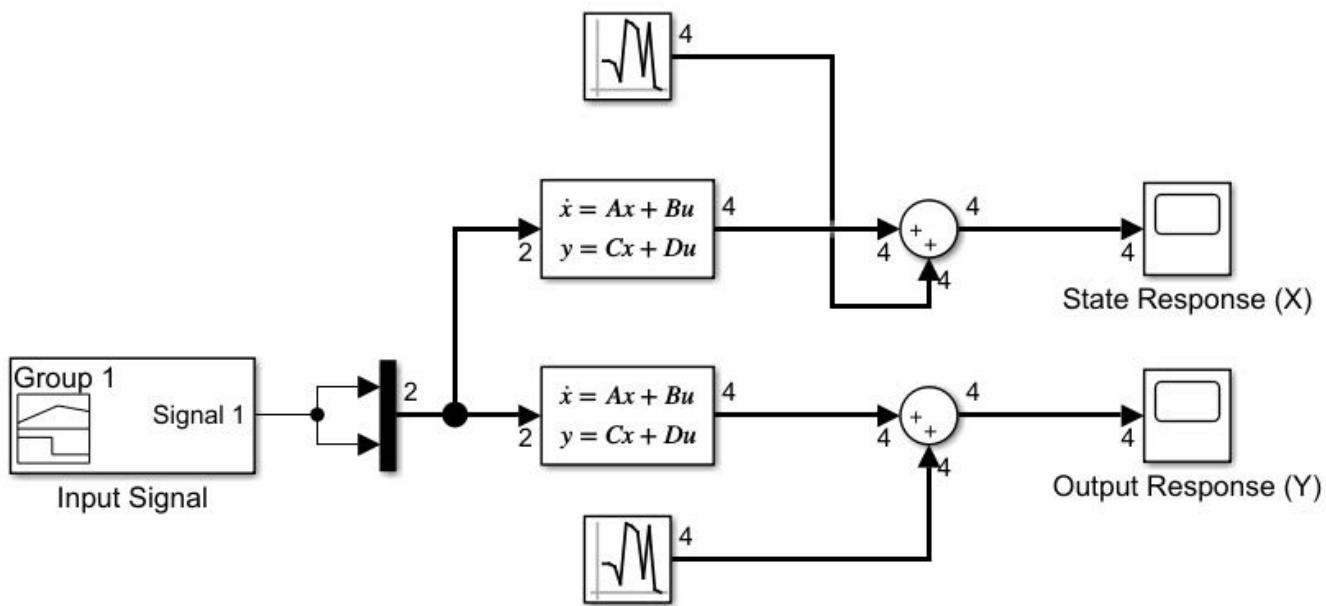
### State Response (x)



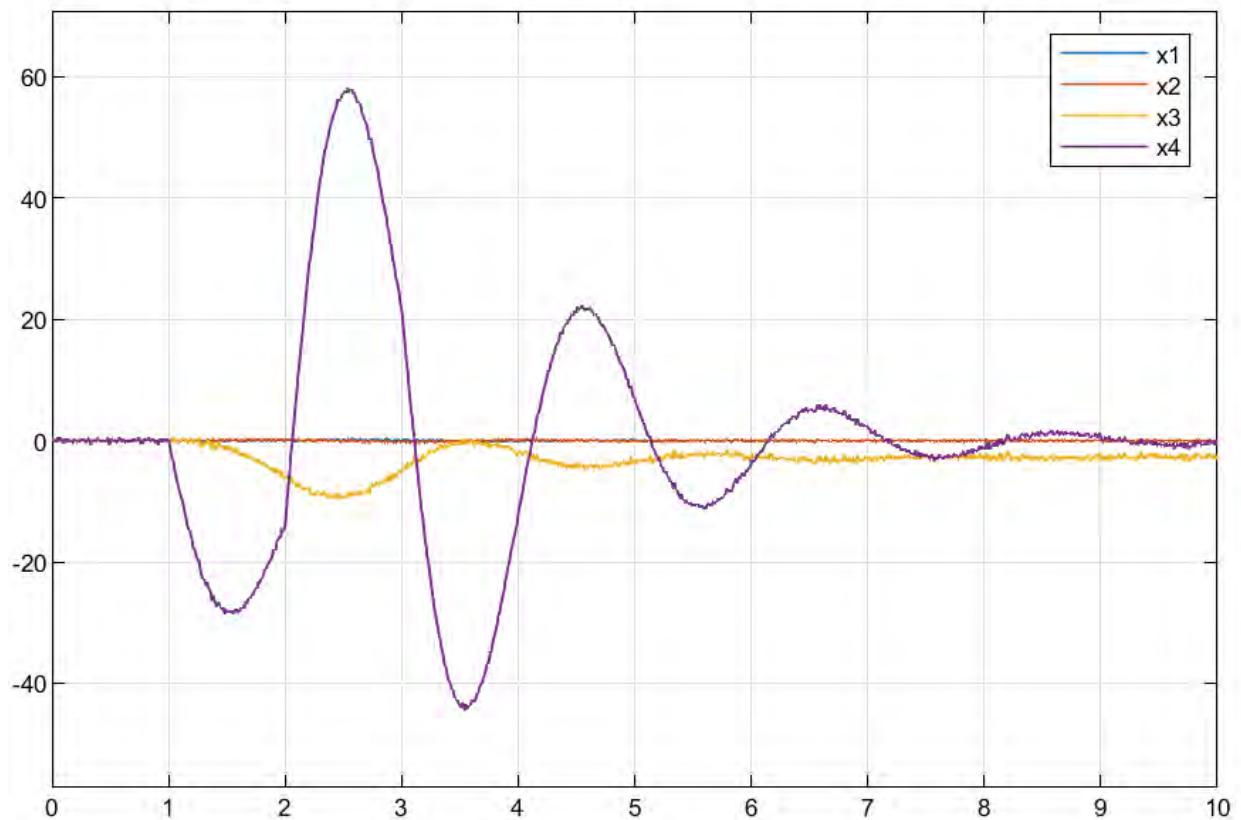
### Output Response (Y)



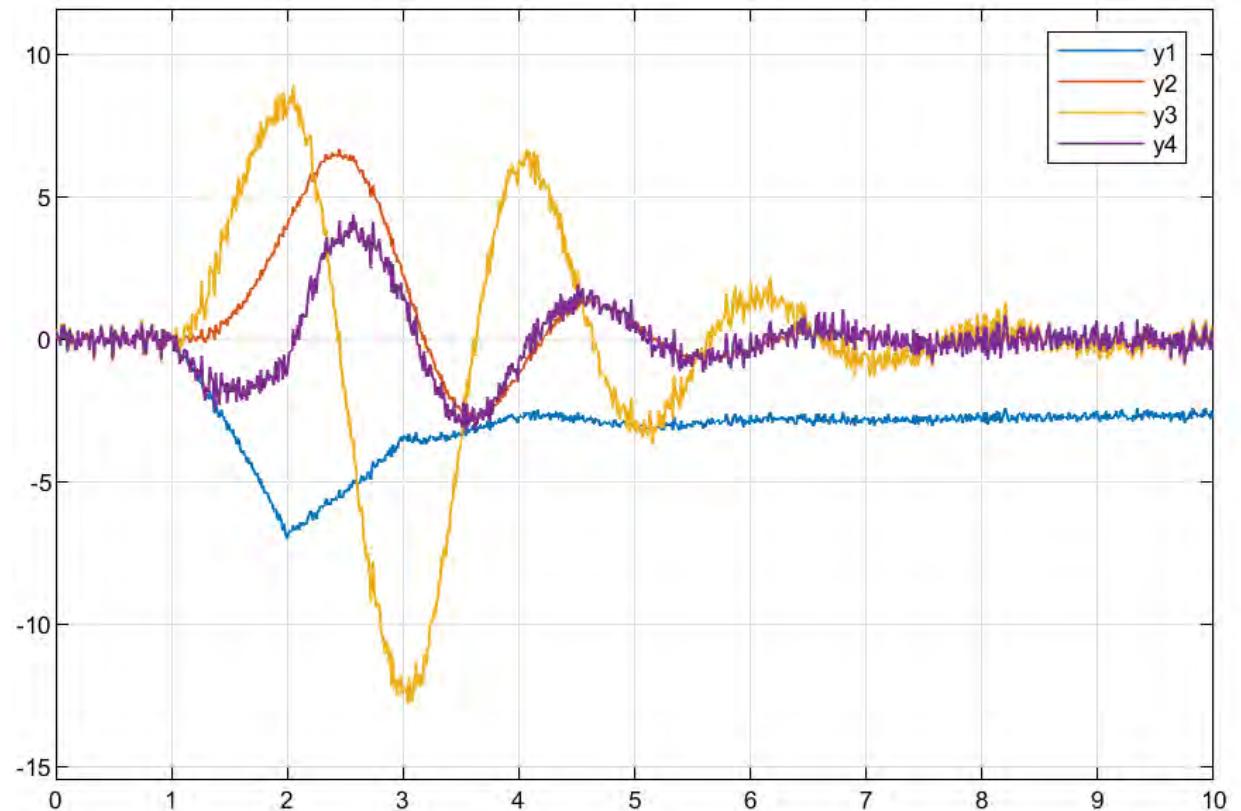
b. Simulink diagram



### State Response (X)



### Output Response (Y)



### c. Commands

```
>> CT=transpose(C)
CT =
    0.1600    57.2958      0      0
    0.0028   -0.2865   57.4879   -0.2849
    0.9978      0      0   -0.0043
    0.0668      0   0.0031    0.0641

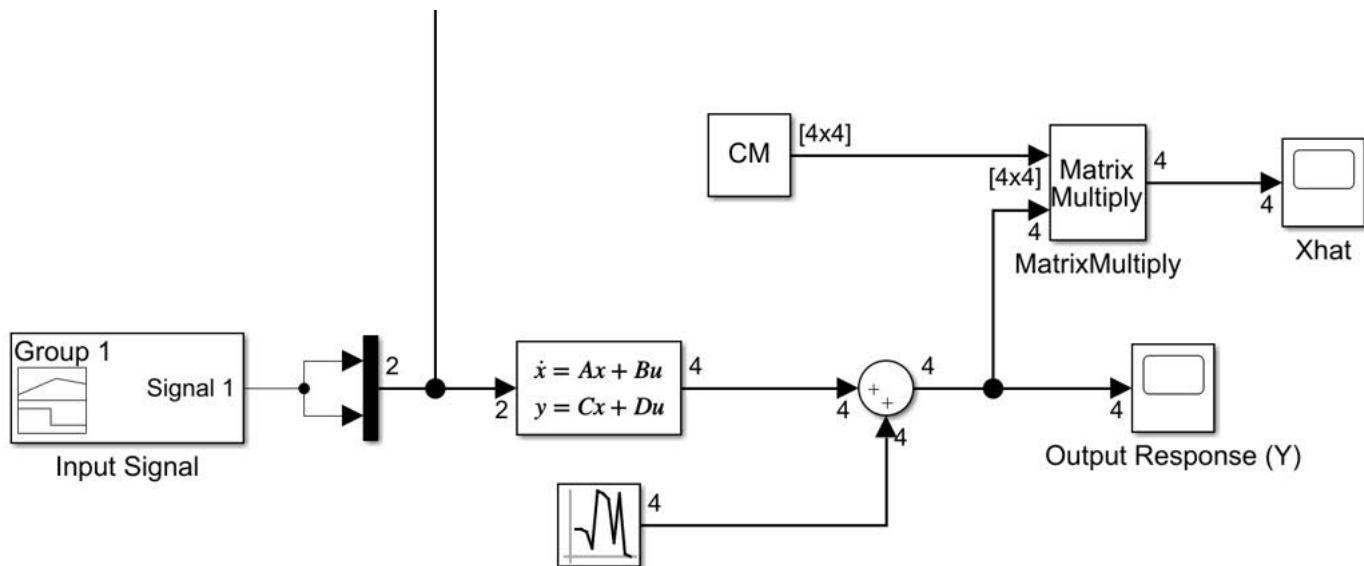
>> R=diag([0.01 0.01 0.1 0.08])
R =
    0.0100      0      0      0
        0    0.0100      0      0
        0      0   0.1000      0
        0      0      0   0.0800

>> C1=inv(CT*inv(R)*C)
C1 =
    0.0000    0.0000   -0.0000   -0.0000
    0.0000    0.0000    0.0001   -0.0009
   -0.0000    0.0001    0.0964   -1.2906
   -0.0000   -0.0009   -1.2906  19.2881

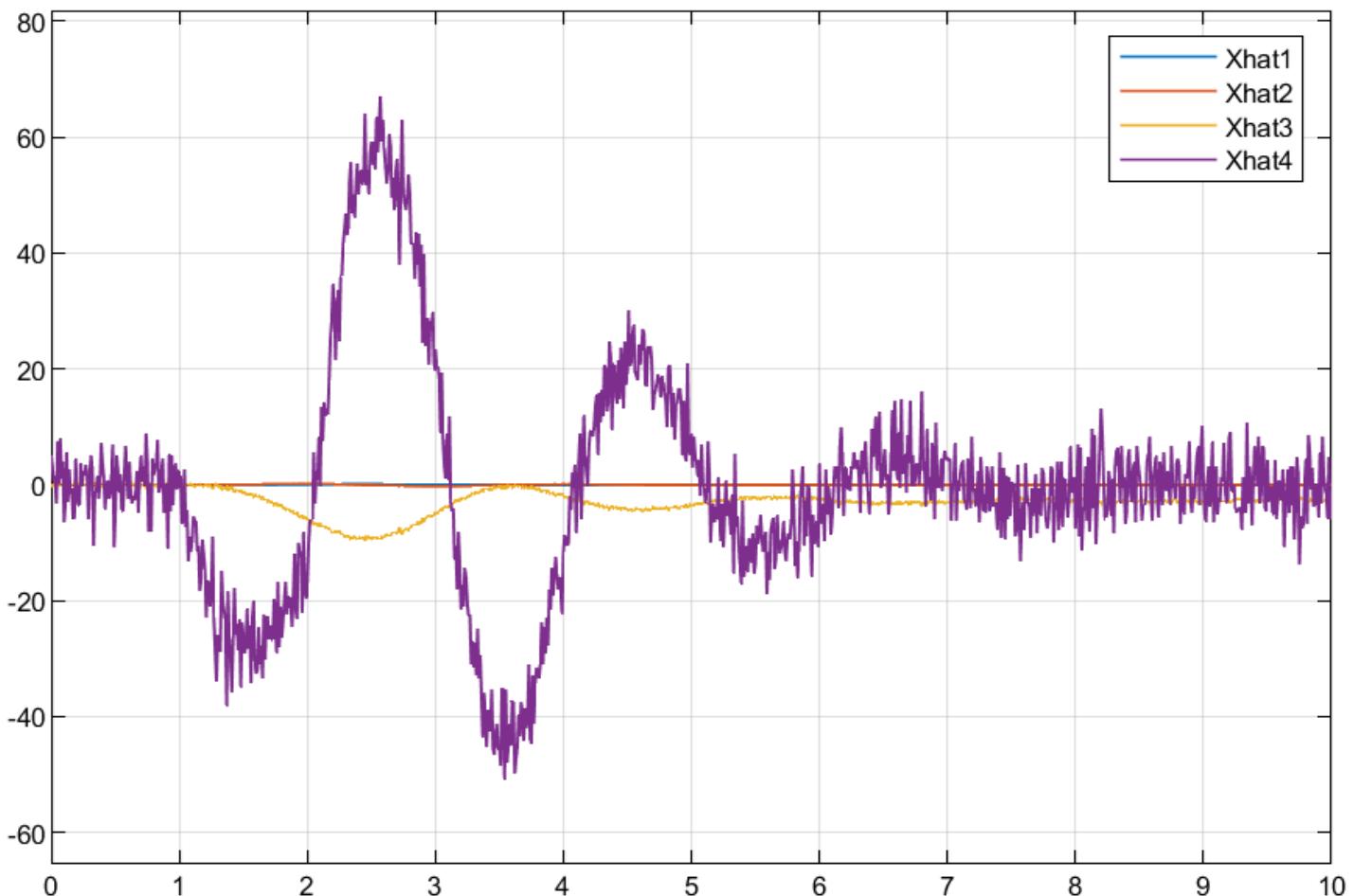
>> C2=CT*inv(R)
C2 =
    1.0e+03 *
    0.0160    5.7296      0      0
    0.0003   -0.0286    0.5749   -0.0036
    0.0998      0      0   -0.0001
    0.0067      0   0.0000    0.0008

>> CM=C1*C2
CM =
   -0.0000    0.0175    0.0001   -0.0000
   -0.0000    0.0000    0.0174   -0.0008
    0.9977   -0.0028   -0.0052   -1.0395
    0.0669   -0.0002    0.0769  15.5272
```

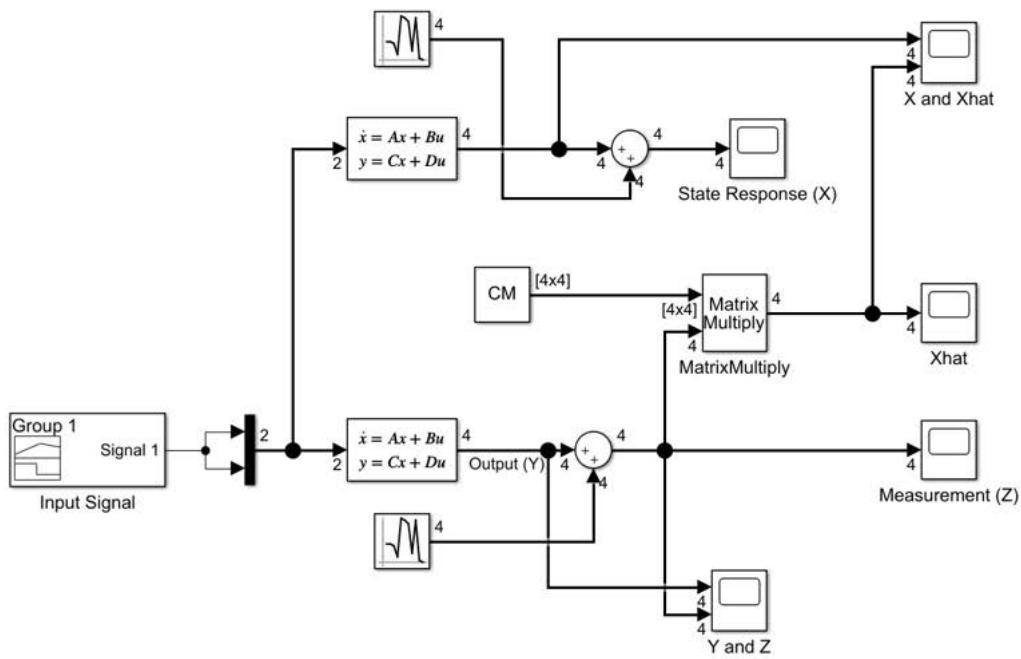
## Simulink Diagram



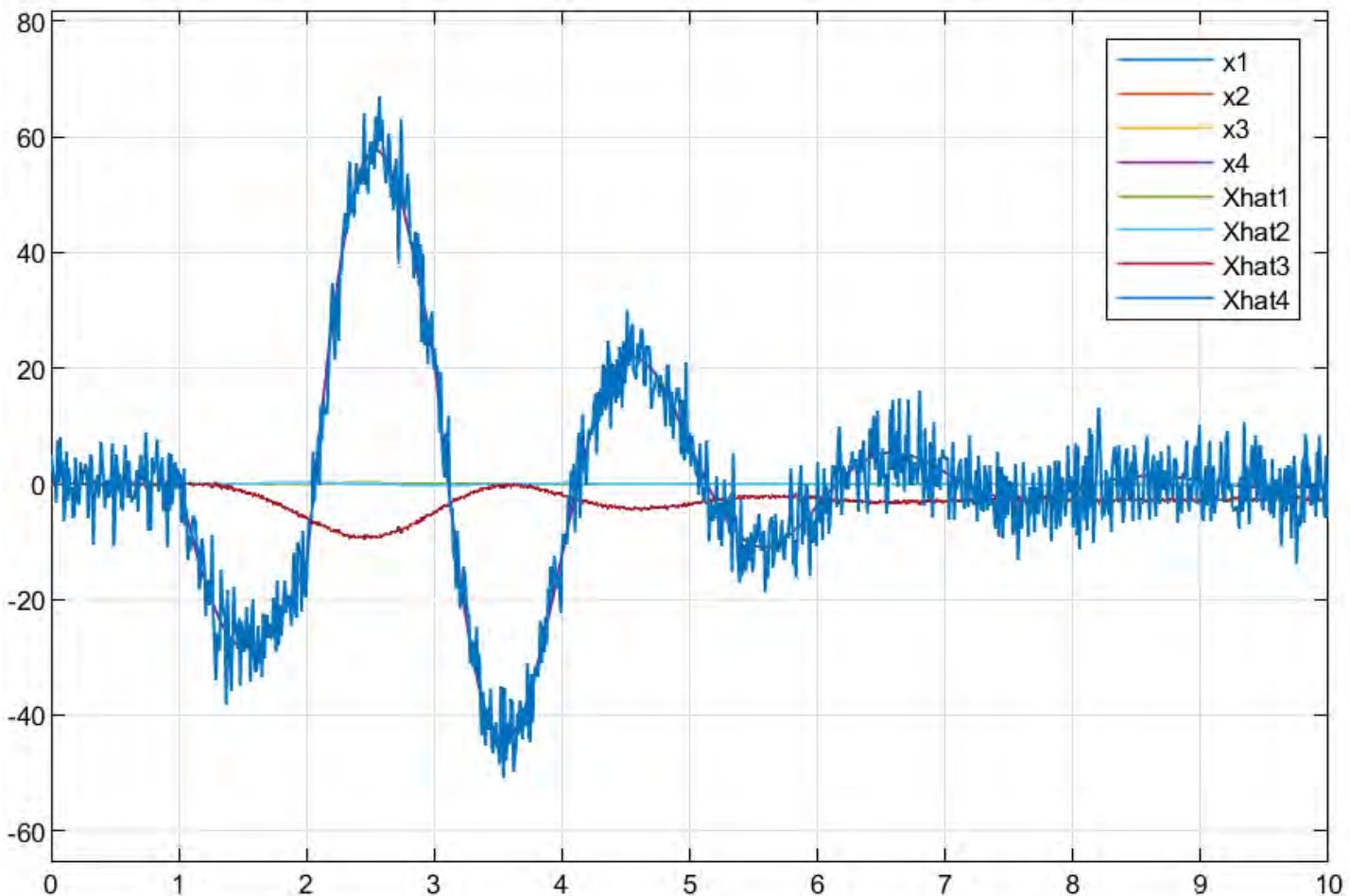
Estimated States (Xhat):

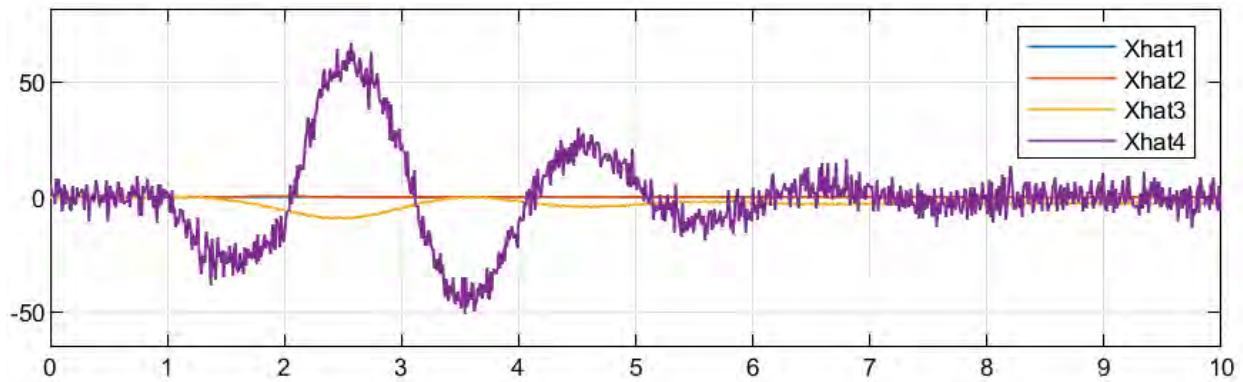
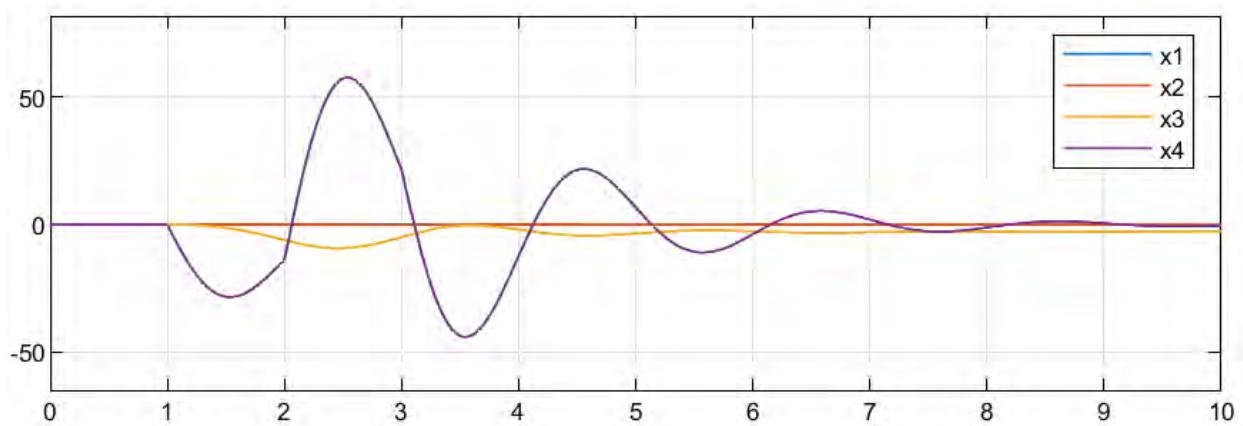


#### d. Simulink Diagram

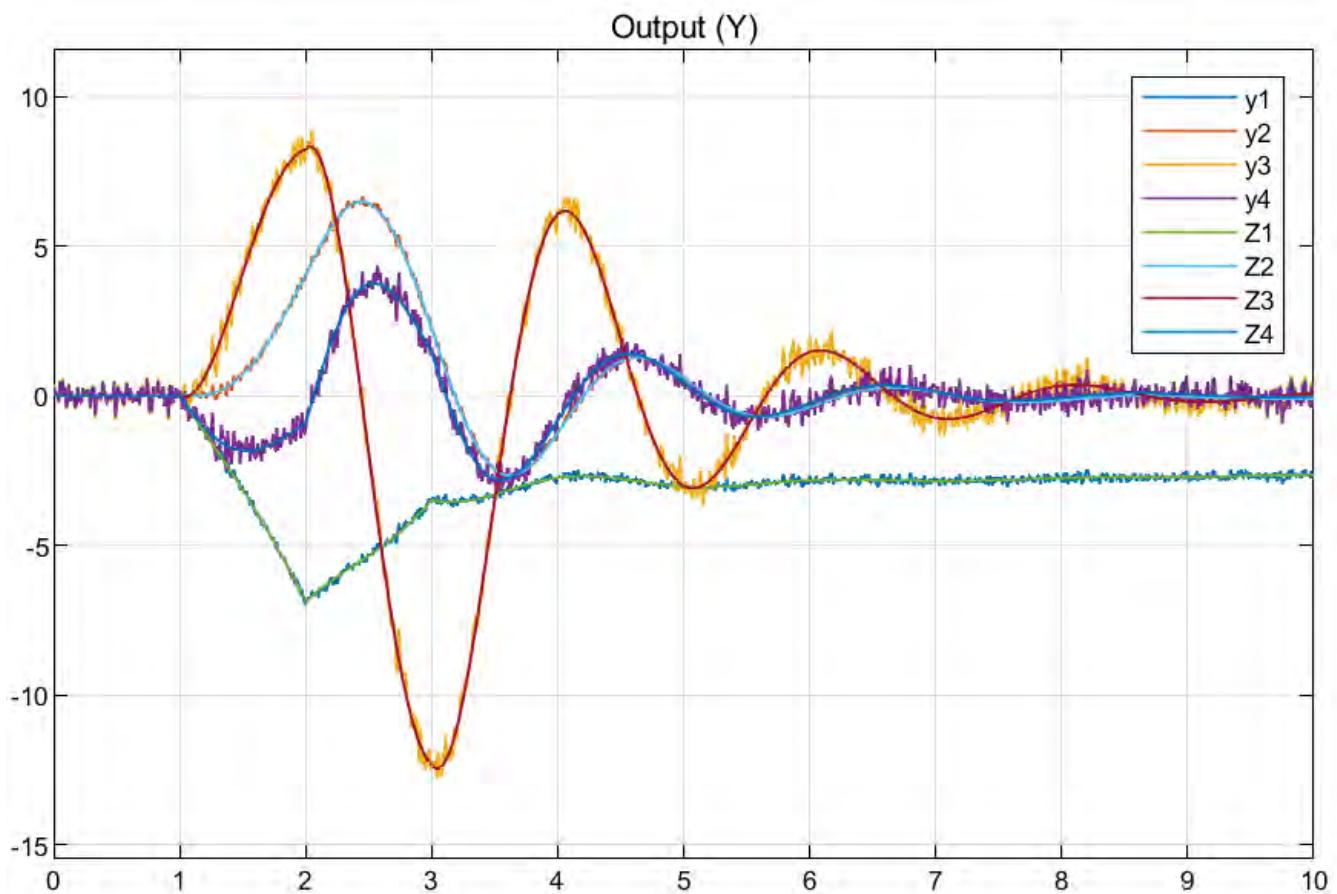


X and Xhat



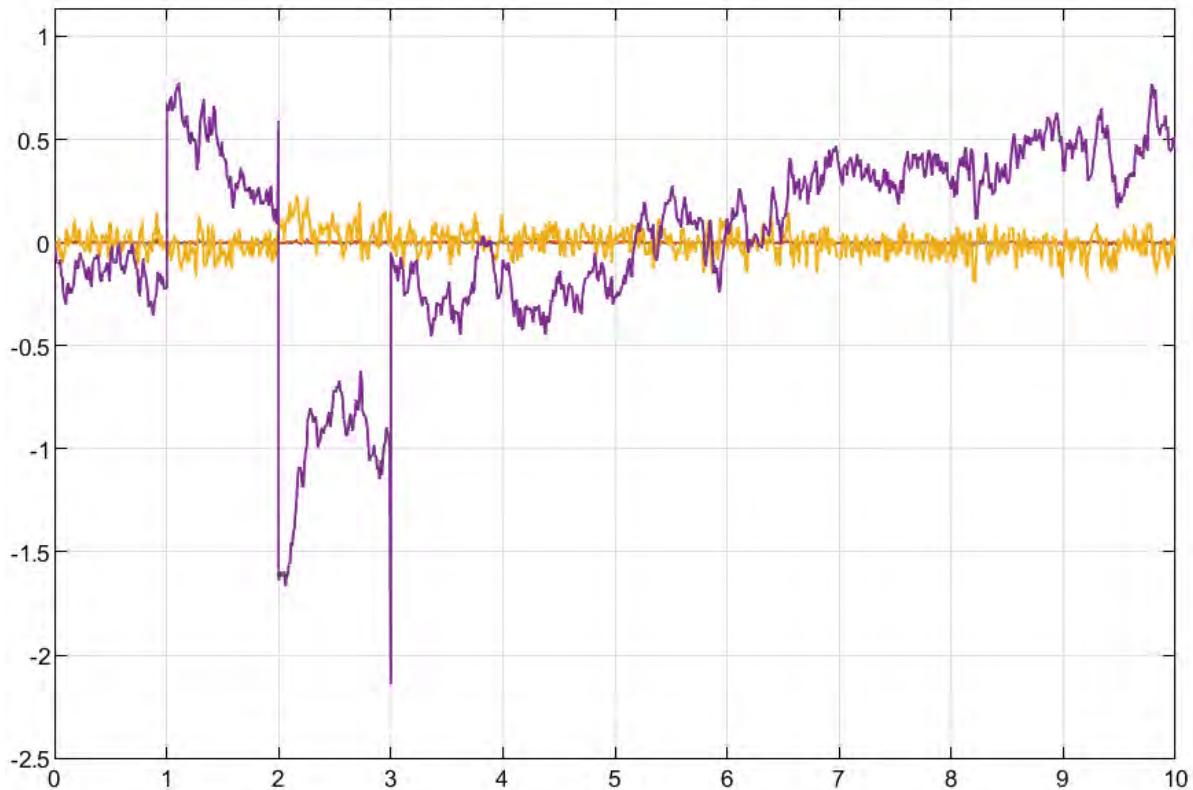


## Y and Z

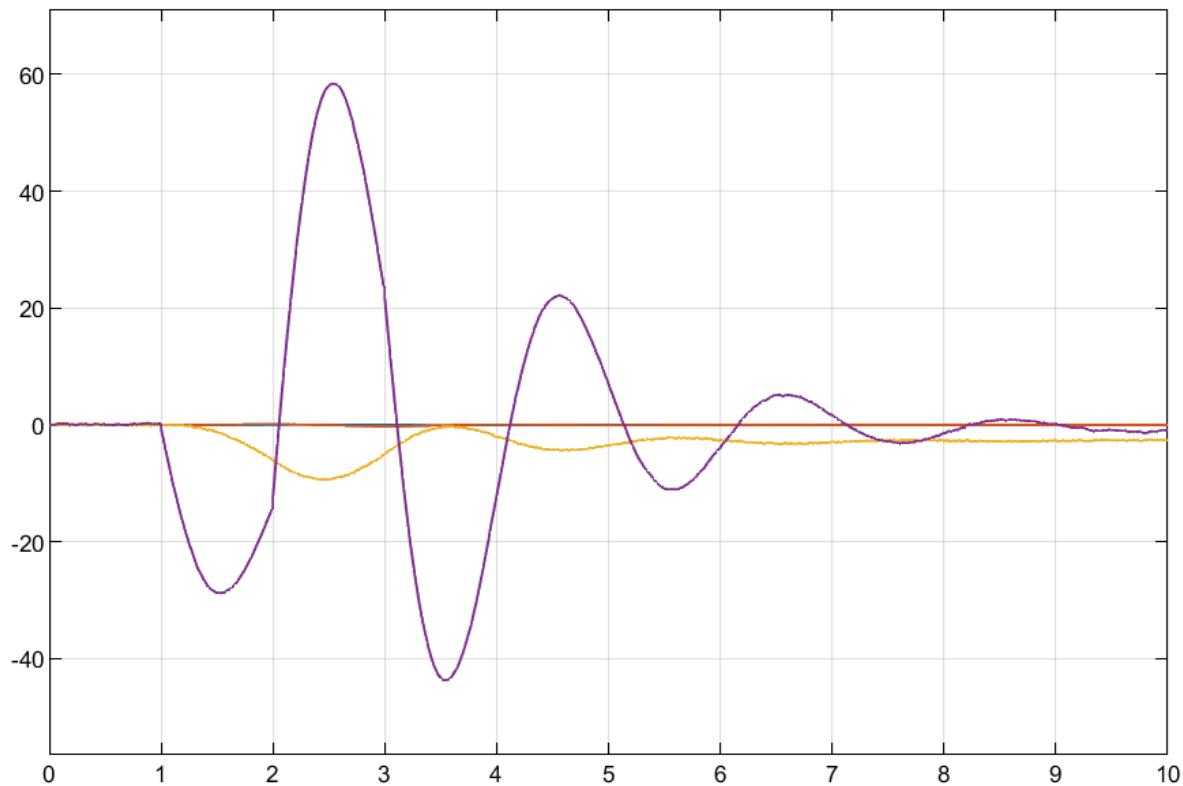


## Problem - 2:

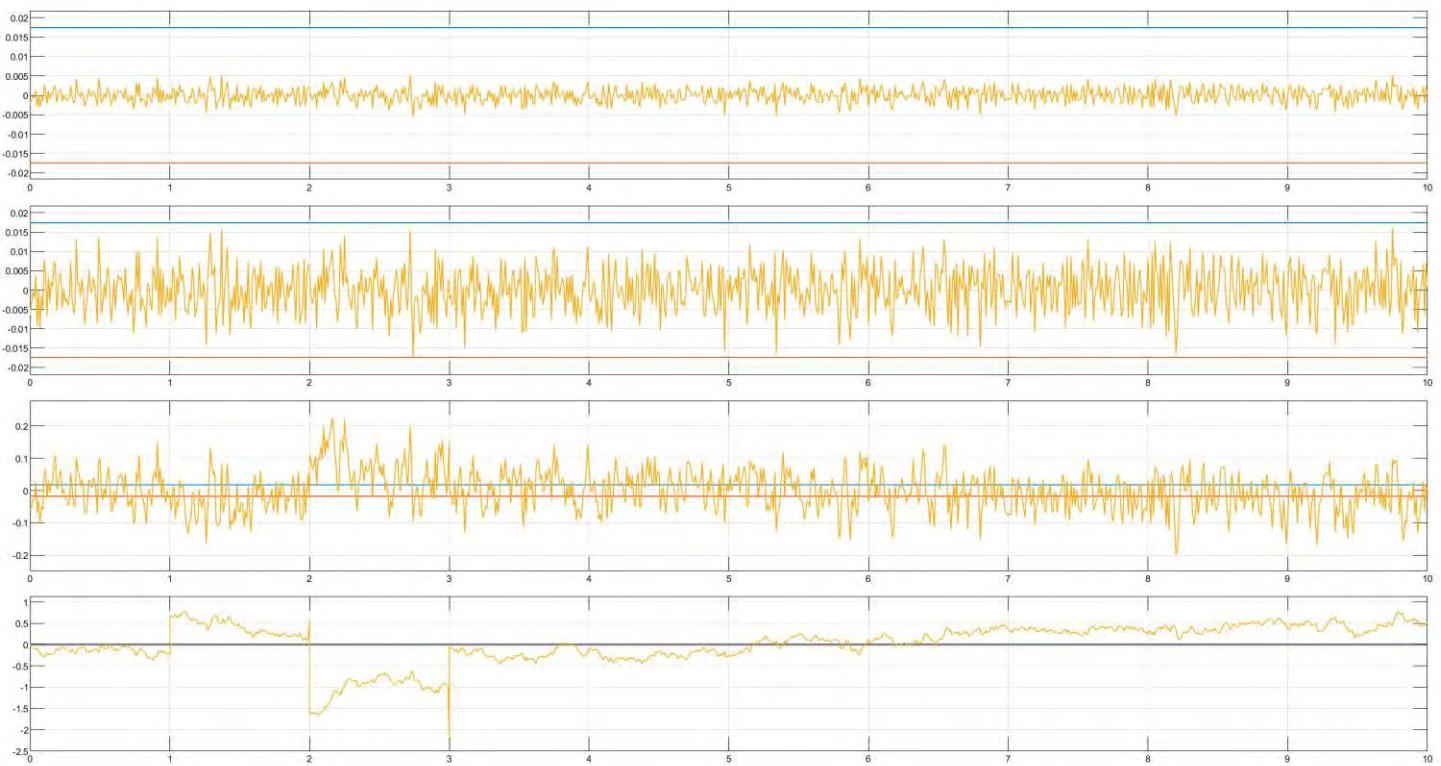
### a. Error Plot



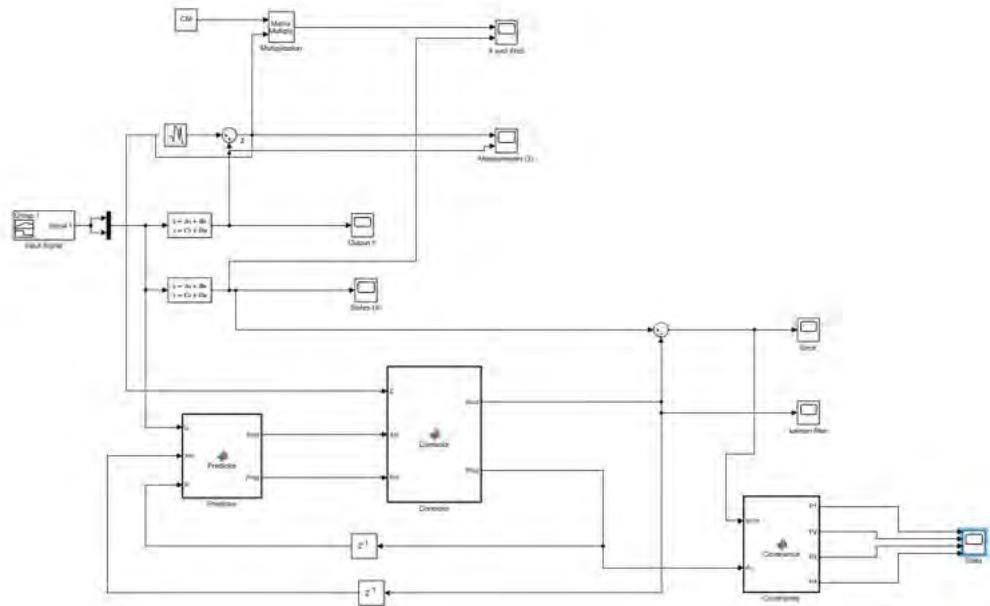
### Kalman Filter



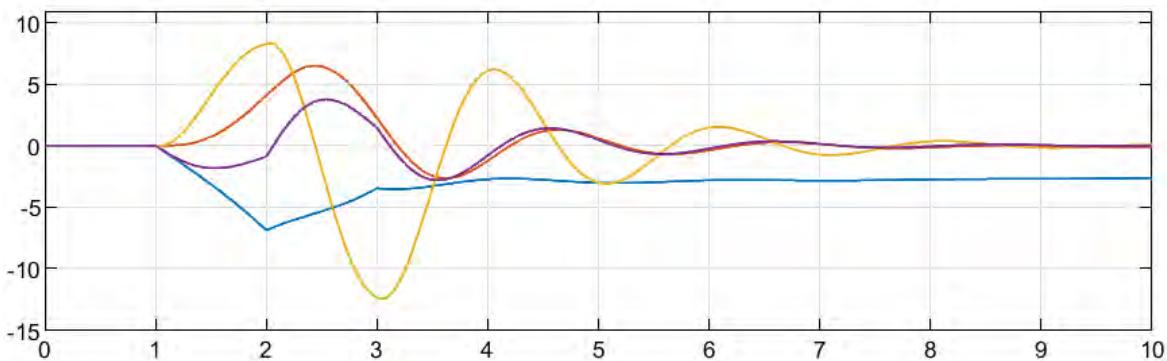
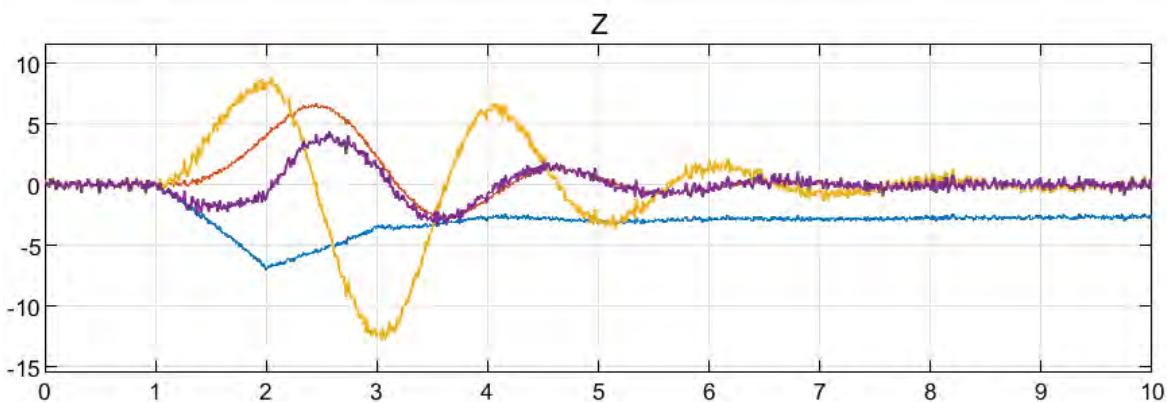
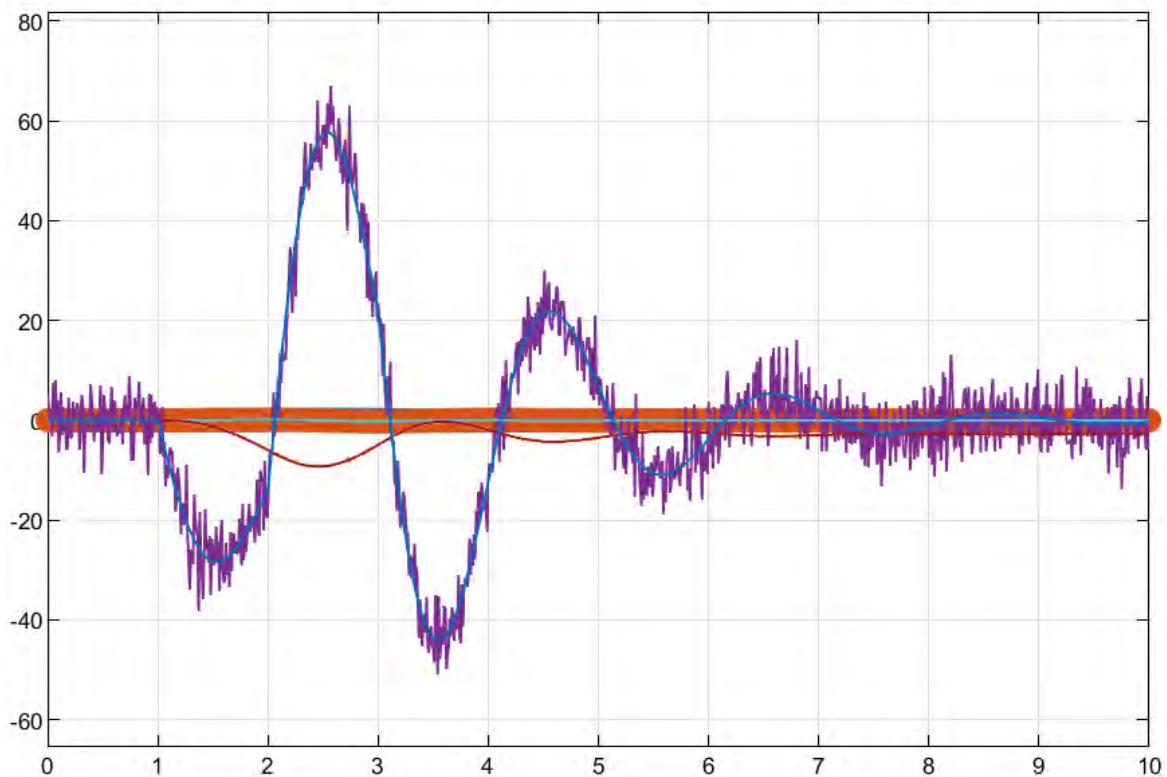
## States



## Simulink Block

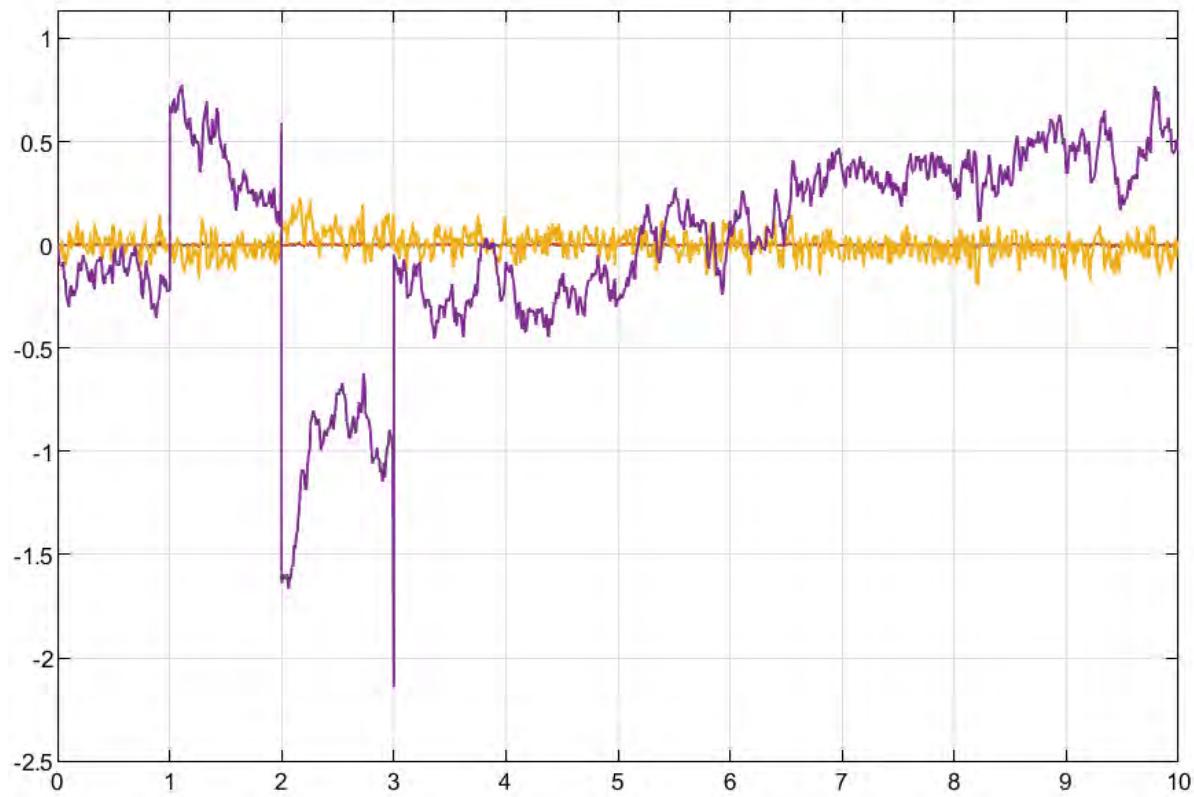


### Maximum Likelihood estimator

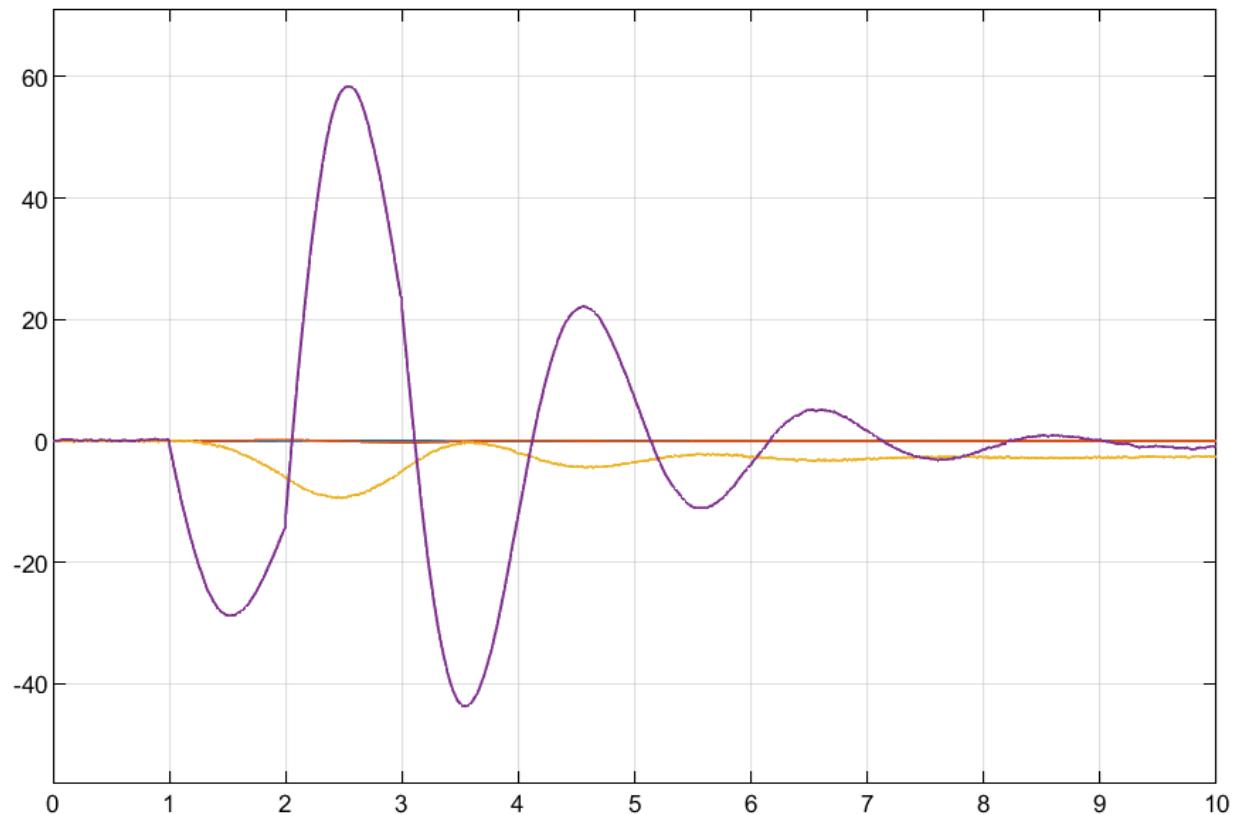


**Conclusion:** Kalman filter provides a much better output as compared to the Maximum likelihood estimator.

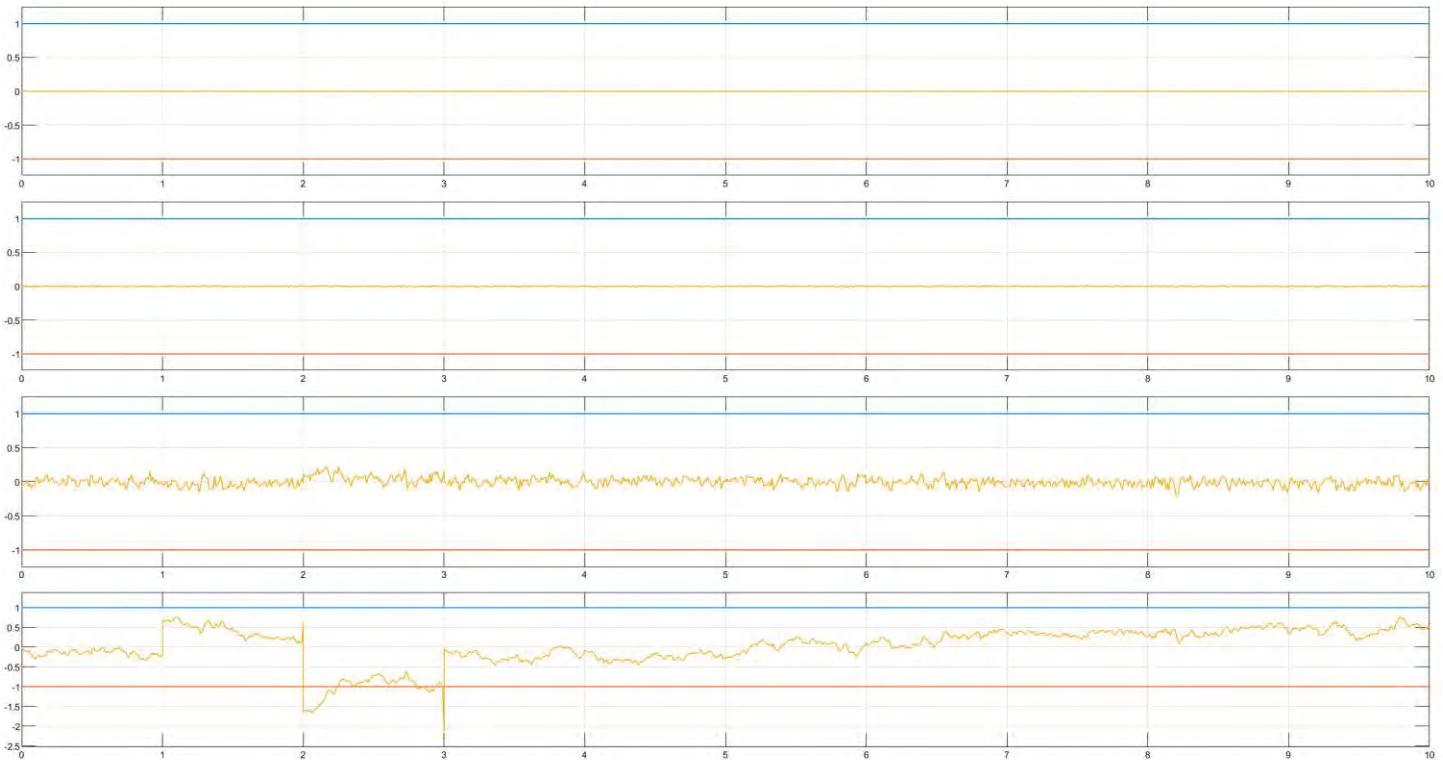
### b. Error



### Kalman Filter

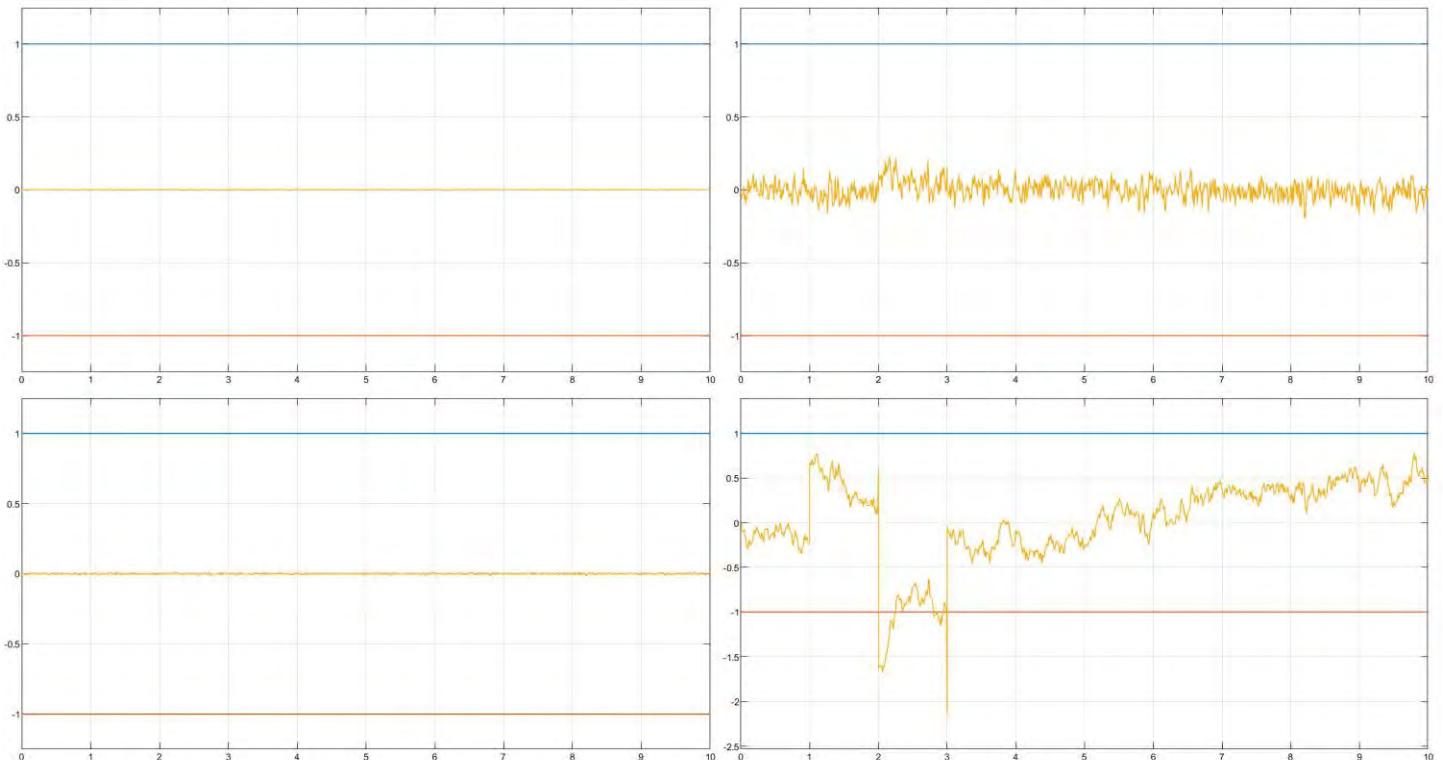


## States

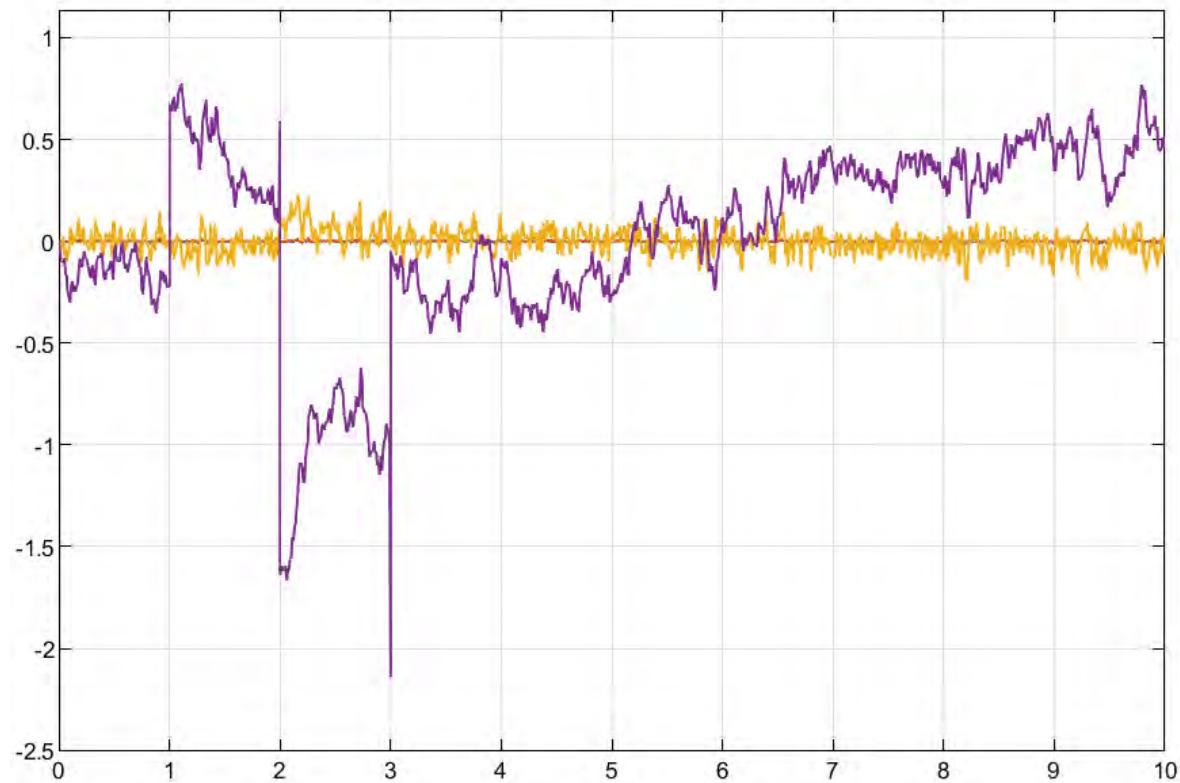


### c. First Case

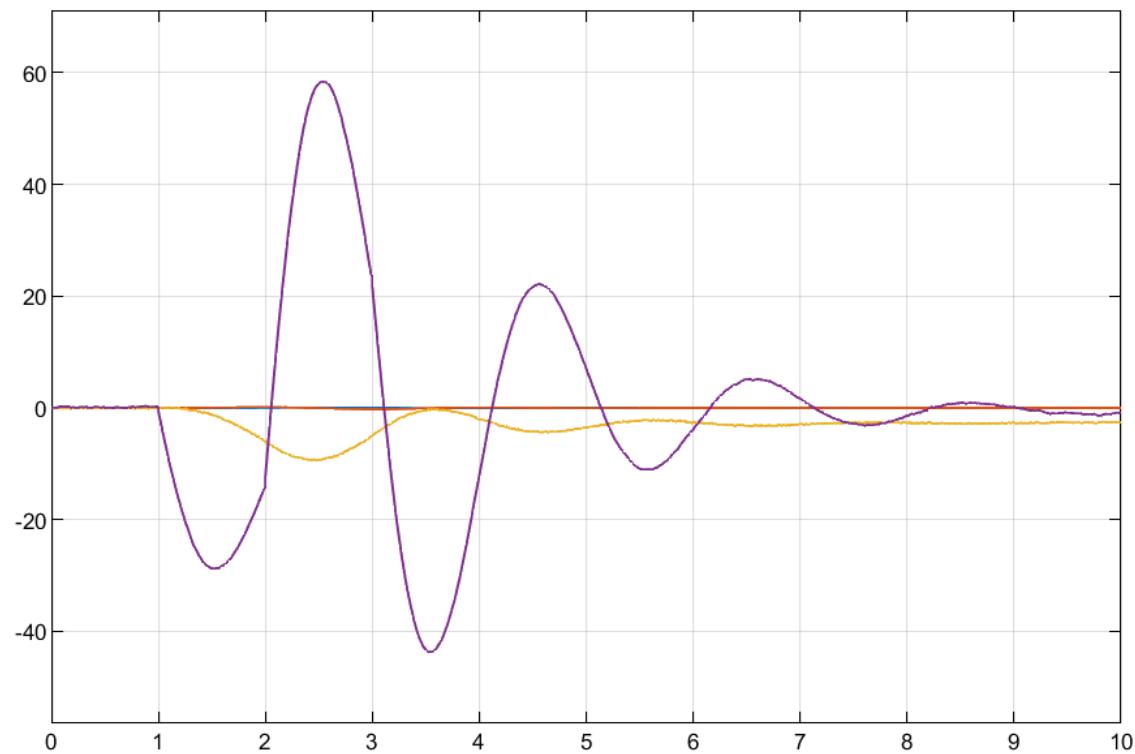
## States



## Error

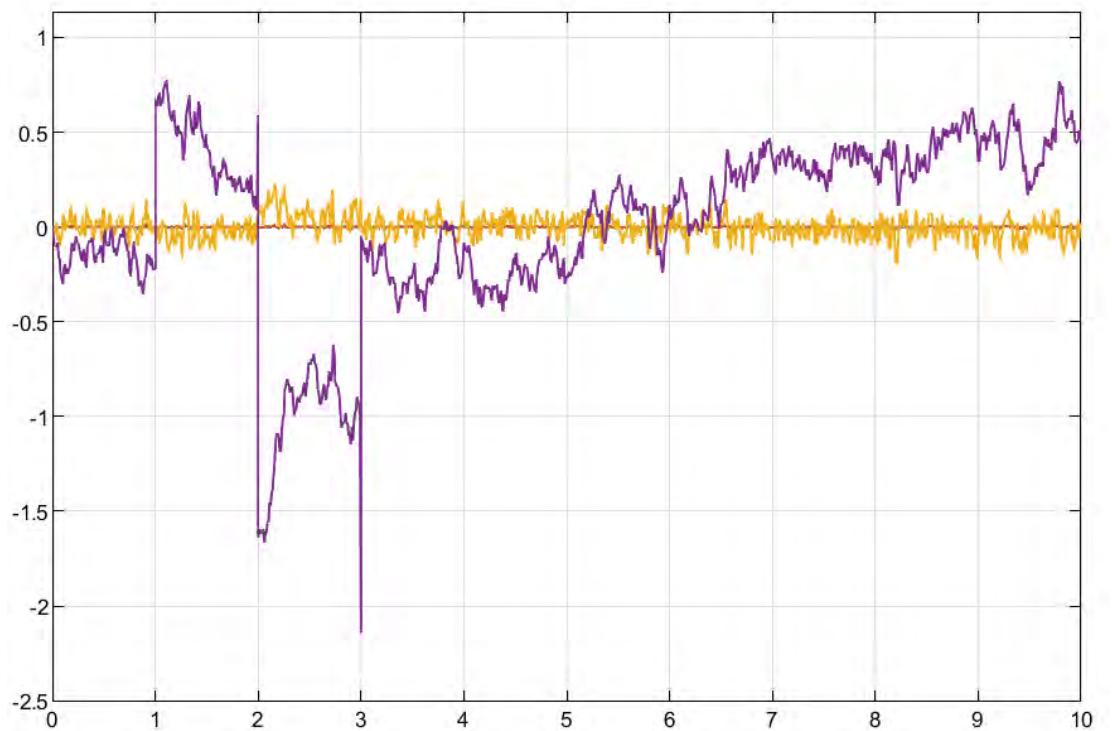


## Kalman Filter

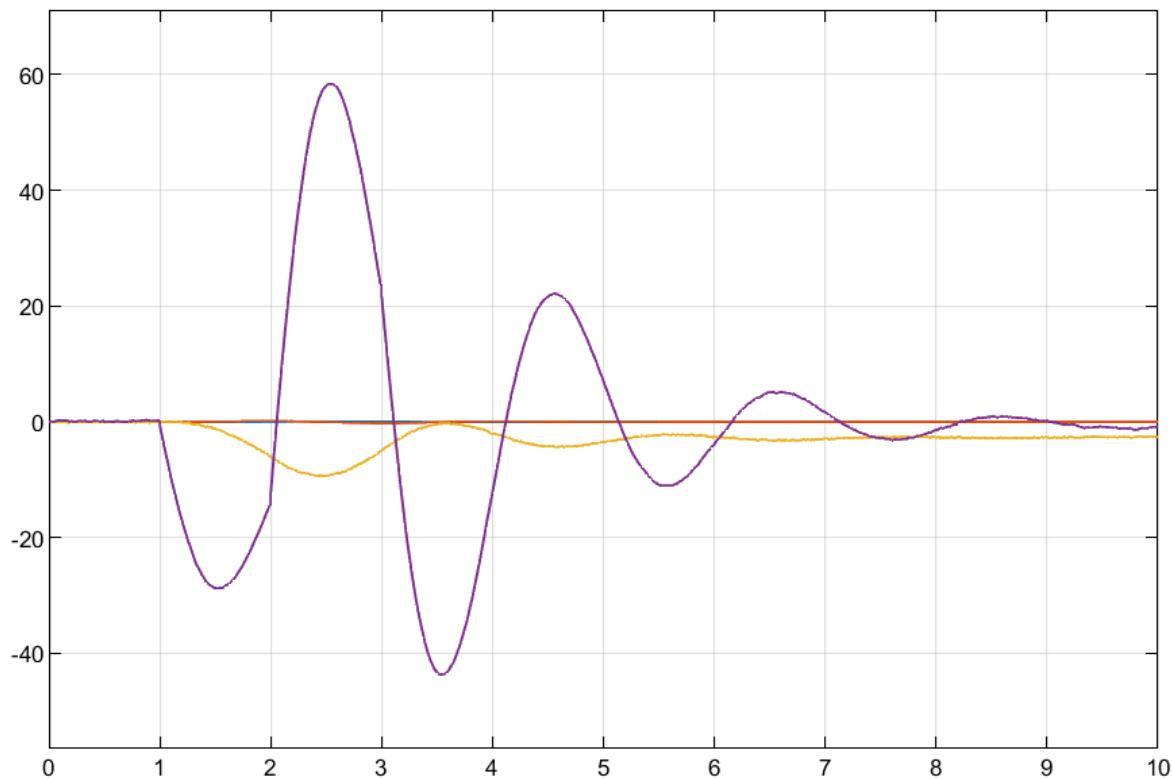


## Second Case

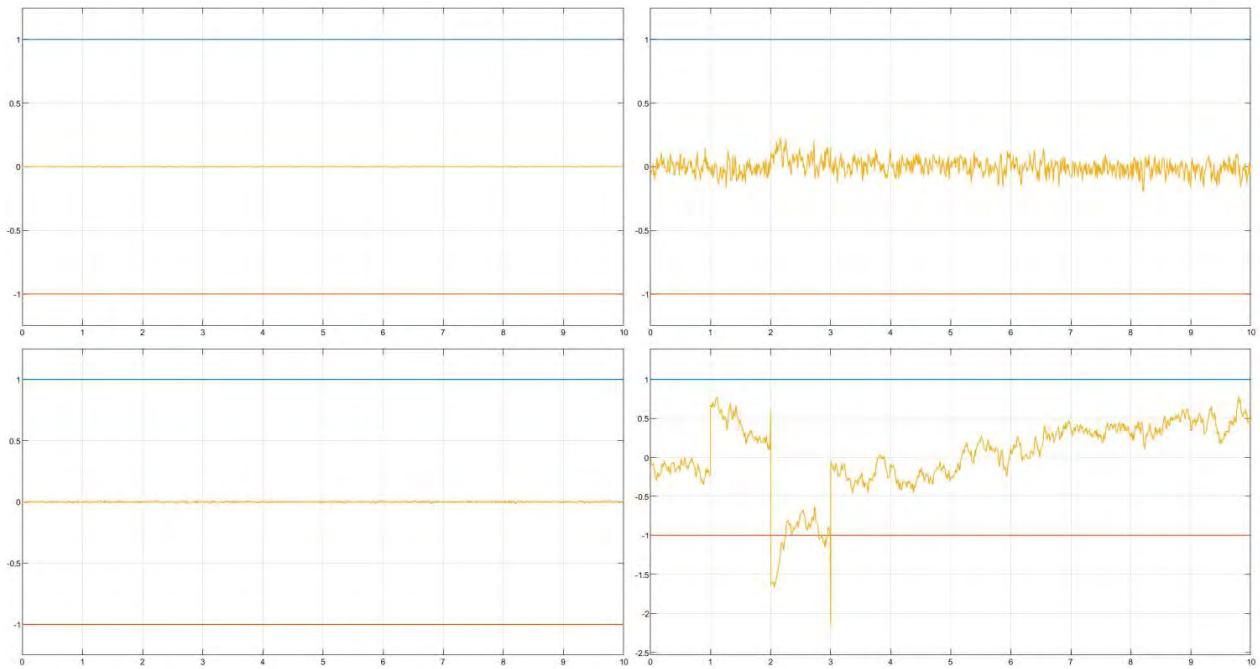
### Error



### Kalman Filter

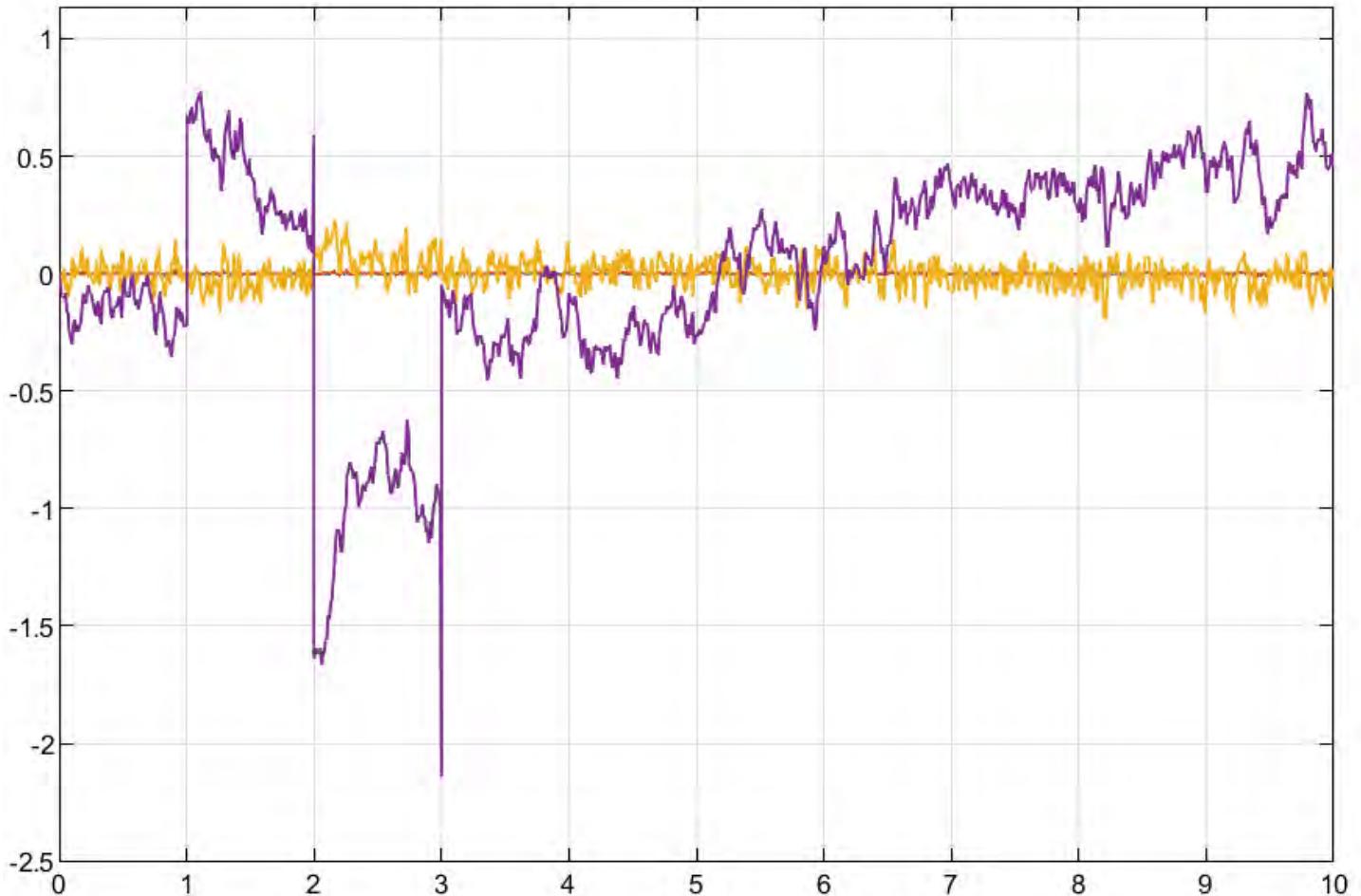


### State

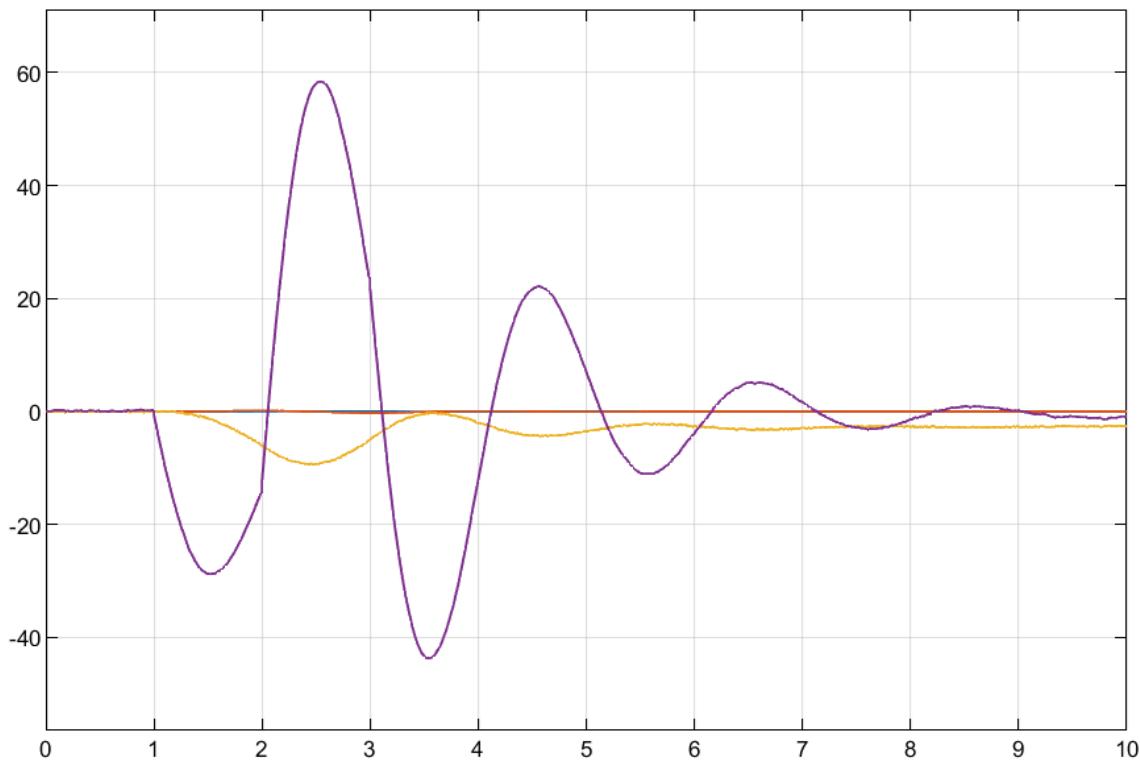


### Third Case

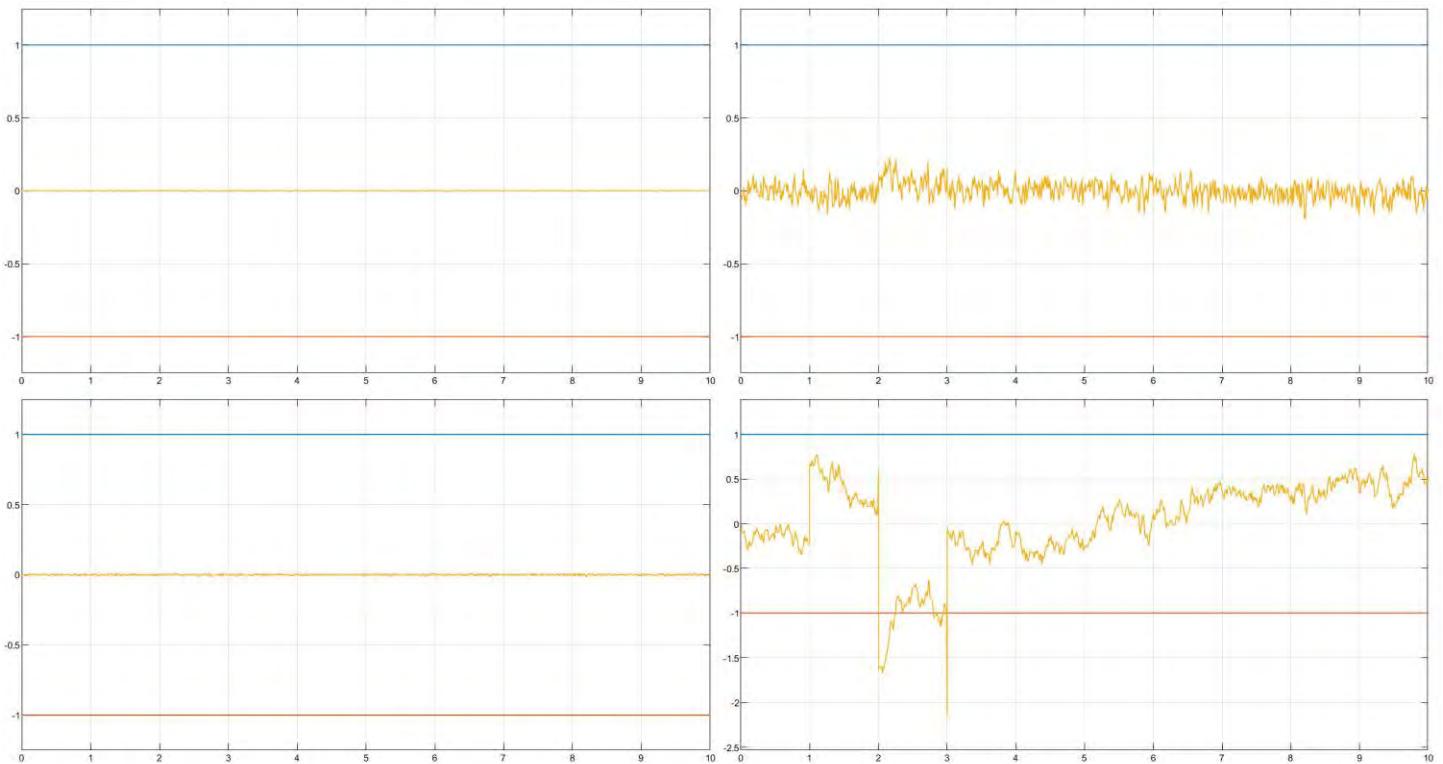
#### Error



## Kalman Filter

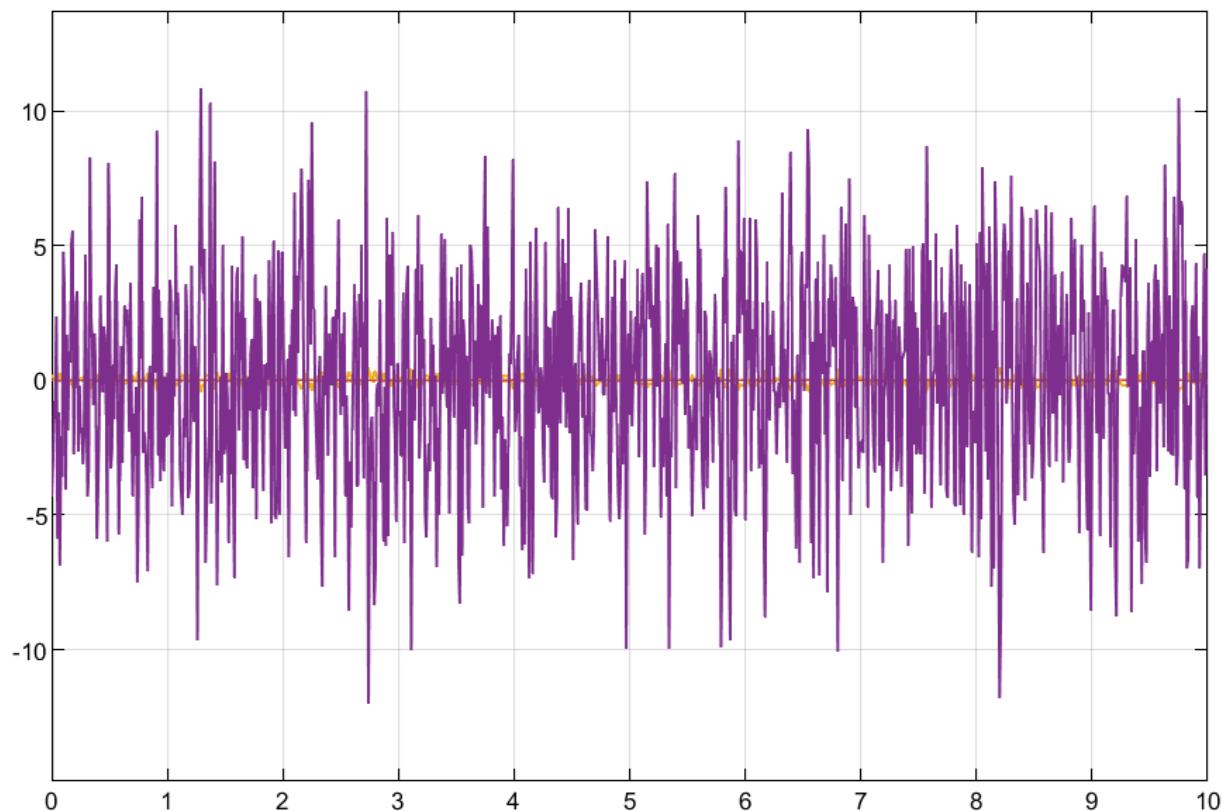


States

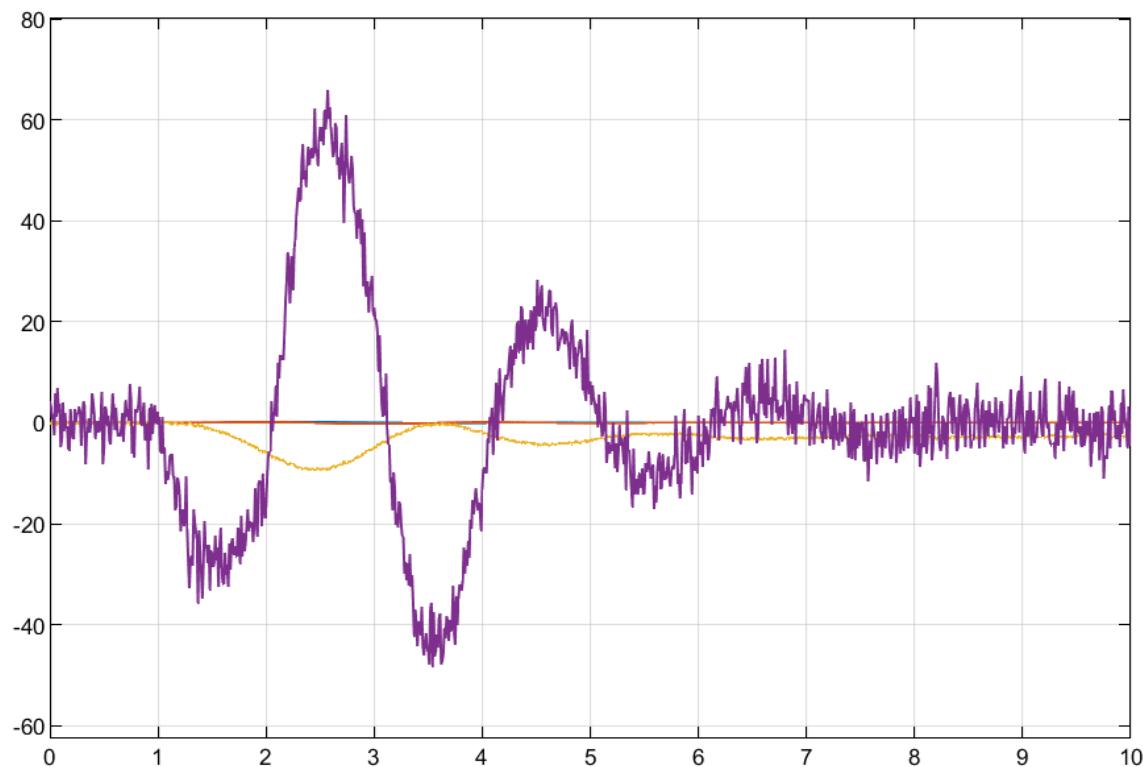


#### Fourth Case

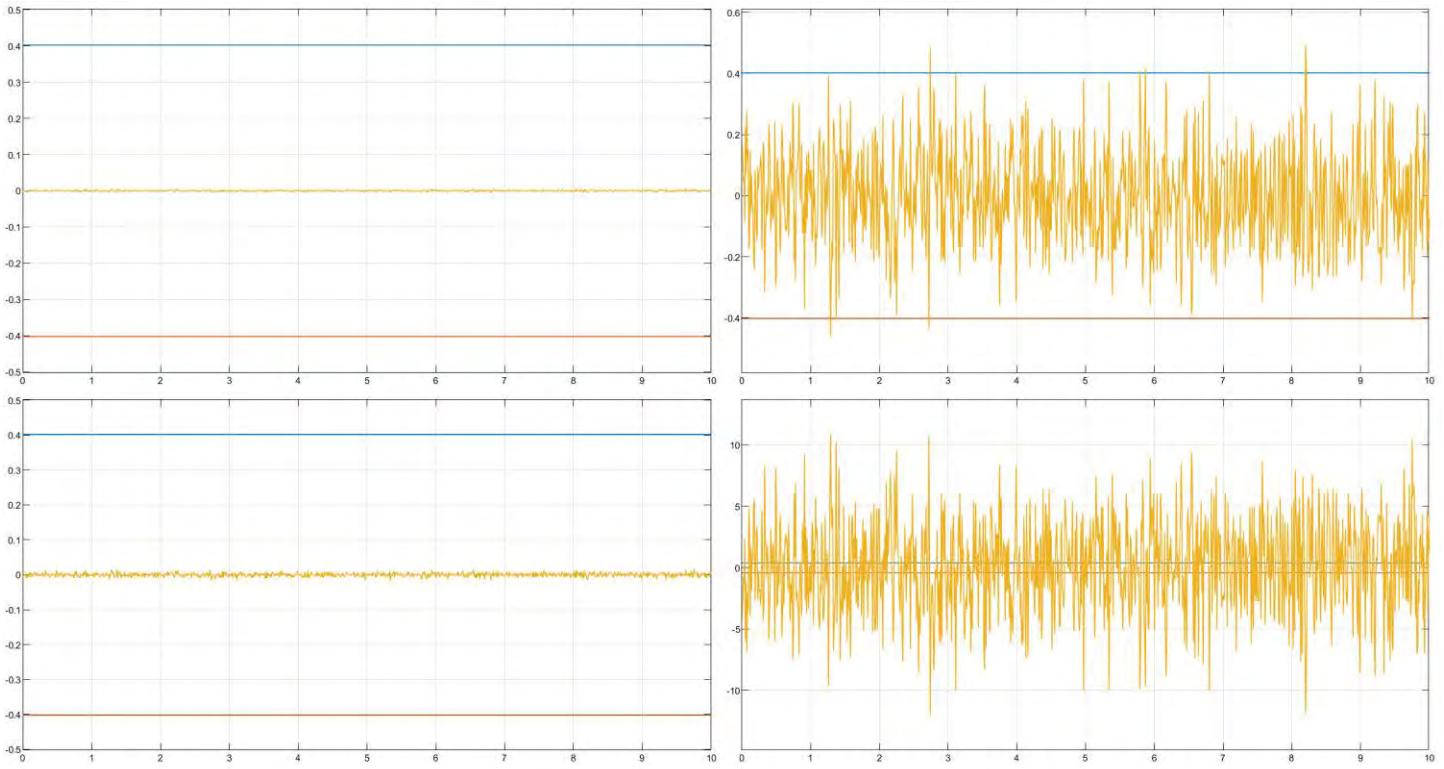
##### Error



##### Kalman Filter



## States



d. From the above plots, it is observed that changing the values of Q does not affect the output. Thus, we can infer that covariance is independent of Q. On the other hand, R is inversely proportional to the noise. Therefore, we can see from the Fourth Case that very small value of R allows a lot of noise into the Corrector block.

**EE5327, Fall 2020**  
**Extended Homework/Project 1**  
**Due 10/22/2020**

**Please turn in this paper with your exam:**

**Pledge of honor: “On my honor I have neither given nor received aid on this project except from the TA or class instructor.”**

**Signature:** \_\_\_\_\_

**Extended Kalman Filter** 100 points (there are two pages to this homework)

Implement the following equations in Simulink:

$$\begin{aligned}\dot{x}_1 &= \mu x_1 + u \\ \dot{x}_2 &= \lambda(x_2 - x_1^2)\end{aligned}$$

With  $\mu = -0.1$  and  $\lambda = -1$ . Make the input,  $u$ , be a Signal Generator block set to a square wave with amplitude 3 and a 1 Hz frequency.

Use a 20 second simulation with a fixed step solver with a time step of 0.01 seconds and initial conditions of the states to be  $[5, 0]$ . Add a process noise before the integrators in the true physics equations (above) with zero mean and variance of 1 with different seeds for each. Implement measurement noise added to the true states (the output of the integrators which include the process noise effects) to create the measurement vector. Set the measurement noise variance to be  $[0.1, 0.1]$  (zero mean) for each of the two states. Make sure to give each random number generator a different seed.

Modify the MATLAB blocks you used in Homework 3 to implement the extended Kalman filter, noting that the dynamics are now nonlinear (so you'll have to compute the Jacobian and implement it in the predictor block) but assume the measurement equation is linear and all states are directly observable (i.e.  $H = \text{eye}(2)$ ). Set the initial value of  $P$  to be  $\text{eye}(2)*5$  and the initial values of the estimated states to be  $[5, 0]$ , the same as the true states. The variances of the process and measurement noises ( $Q$  and  $R$ ) will be up to you to determine, given the requirements that the error between the true states (with the process noise integrated into them) and the estimated states should be less than 0.5.

Adjust the  $Q$  and  $R$  matrix values to try to get the error to be within the square root of the covariance about 80% of the time or higher and the error magnitude less than 0.5 for each state.

After performing and documenting the above tasks, now set the filter to have an initial state of  $[0, 0]$  and rerunning. Document the results and comment on whether the incorrect guess at the filter states made a difference in this case.

Sometimes we don't get the physics model correct but the filter still can do a pretty good job of estimating the states. Change your predictor model (but not the true physics) to have  $\mu = -0.01$  and rerun with the initial state  $[5, 0]$ . Modify the Q and R values if you must to meet the requirements of both state errors being less than 0.5 in magnitude for more than 80% of the time. While I think this should work, if you can't achieve the goal, document your attempts and note which got you the closest.

**EE5327, Fall 2020**  
**Extended Homework/Project 1**  
**Due 10/22/2020**

**Please turn in this paper with your exam:**

Pledge of honor: “On my honor I have neither given nor received aid on this project except from the TA or class instructor.”

**Signature:** Atul Shrotriya

**Extended Kalman Filter** 100 points (there are two pages to this homework)

Implement the following equations in Simulink:

Word ate the equations while converting from pdf format.

With  $\alpha = -0.1$  and  $\beta = -1$ . Make the input,  $u$ , be a Signal Generator block set to a square wave with amplitude 3 and a 1 Hz frequency.

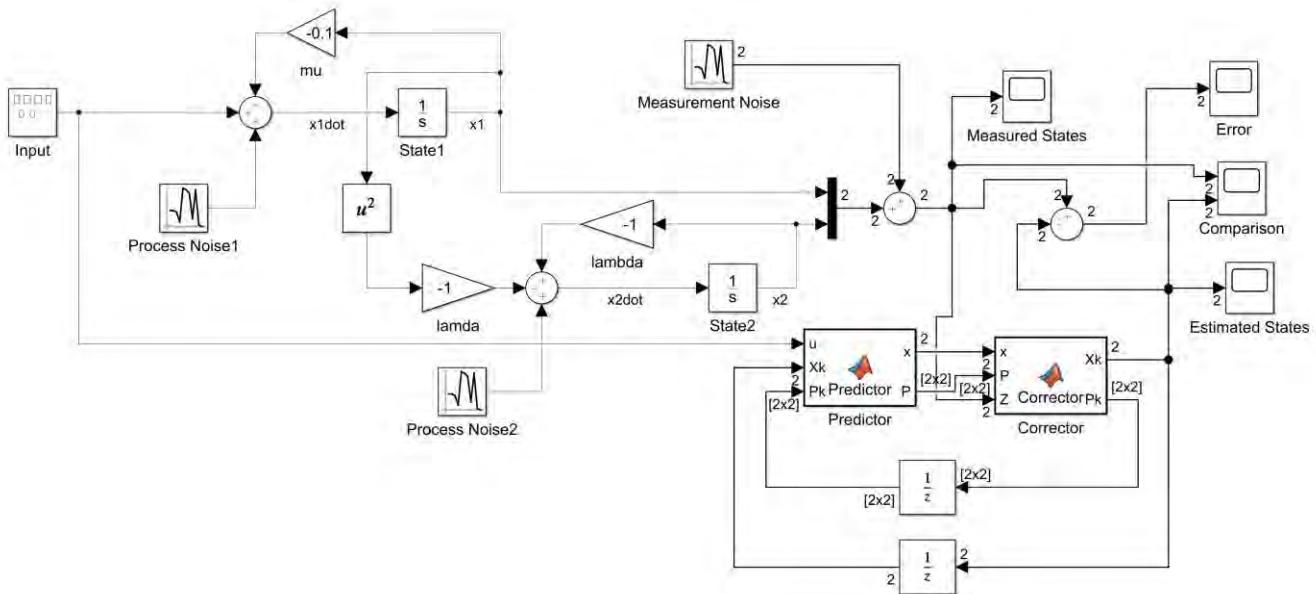
Use a 20 second simulation with a fixed step solver with a time step of 0.01 seconds and initial conditions of the states to be  $[5, 0]$ . Add a process noise before the integrators in the true physics equations (above) with zero mean and variance of 1 with different seeds for each. Implement measurement noise added to the true states (the output of the integrators which include the process noise effects) to create the measurement vector. Set the measurement noise variance to be  $[0.1, 0.1]$  (zero mean) for each of the two states. Make sure to give each random number generator a different seed.

Modify the MATLAB blocks you used in Homework 3 to implement the extended Kalman filter, noting that the dynamics are now nonlinear (so you'll have to compute the Jacobian and implement it in the predictor block) but assume the measurement equation is linear and all states are directly observable (i.e.  $H = eye(2)$ ). Set the initial value of  $P$  to be  $eye(2)*5$  and the initial values of the estimated states to be  $[5, 0]$ , the same as the true states. The variances of the process and measurement noises ( $Q$  and  $R$ ) will be up to you to determine, given the requirements that the error between the true states (with the process noise integrated into them) and the estimated states should be less than 0.5. Adjust the  $Q$  and  $R$  matrix values to try to get the error to be within the square root of the covariance about 80% of the time or higher and the error magnitude less than 0.5 for each state.

After performing and documenting the above tasks, now set the filter to have an initial state of  $[0, 0]$  and rerunning. Document the results and comment on whether the incorrect guess at the filter states made a difference in this case.

Sometimes we don't get the physics model correct but the filter still can do a pretty good job of estimating the states. Change your predictor model (but not the true physics) to have  $\alpha = -0.01$  and rerun with the initial state  $[5, 0]$ . Modify the  $Q$  and  $R$  values if you must to meet the requirements of both state errors being less than 0.5 in magnitude for more than 80% of the time. While I think this should work, if you can't achieve the goal, document your attempts and note which got you the closest.

## Simulink Diagram



### Predictor Code

```

function [x,P] = Predictor(u,Xk,Pk)
mu=-0.1; lam=-1;
A = [mu 0;-2*lam*Xk(1) lam];
B= [1;0];
Ad=expm(A*0.01);
Bd=inv(A)*(Ad-eye(2))*B;
Q=eye(2);
F=[-A Q;zeros(2,2) transpose(A)];
G=expm(F*0.01);
Qd=transpose(G(3:4,3:4))*G(1:2,3:4);
x=Ad*Xk + Bd*u;
P=(Ad*Pk*transpose(Ad)+Qd);
end

```

### Corrector Code

```

function [Xk,Pk] = Corrector(x,P,Z)
H=eye(2); R=0.1*eye(2); Rd=R/0.01;
K=P*transpose(H)*inv(H*P*transpose(H)+Rd);
Pk=(eye(2)-K*H)*P*transpose(eye(2)-K*H)+K*Rd*transpose(K);
Xk=x+K*(Z-H*x);
end

```

## Noise Parameters

 Block Parameters: Process Noise1	 Block Parameters: Process Noise2	 Block Parameters: Measurement Noise
Random Number	Random Number	Random Number
Output a normally (Gaussian) distributed noise signal. The signal is repeatable for a given seed.	Output a normally (Gaussian) distributed noise signal. The signal is repeatable for a given seed.	Output a normally (Gaussian) distributed noise signal. The signal is repeatable for a given seed.
Parameters	Parameters	Parameters
Mean:	Mean:	Mean:
0	0	0
Variance:	Variance:	Variance:
1	1	[0.1 0.1]
Seed:	Seed:	Seed:
9	3	[6 7]
Sample time:	Sample time:	Sample time:
0.01	0.01	0.01
<input checked="" type="checkbox"/> Interpret vector parameters as 1-D	<input checked="" type="checkbox"/> Interpret vector parameters as 1-D	<input checked="" type="checkbox"/> Interpret vector parameters as 1-D

## Input Parameters

 Block Parameters: Input ×

Signal Generator  
Output various wave forms:  
 $Y(t) = \text{Amp} * \text{Waveform}(\text{Freq}, t)$

Parameters

Wave form: square

Time (t): Use simulation time

Amplitude:

3

Frequency:

1

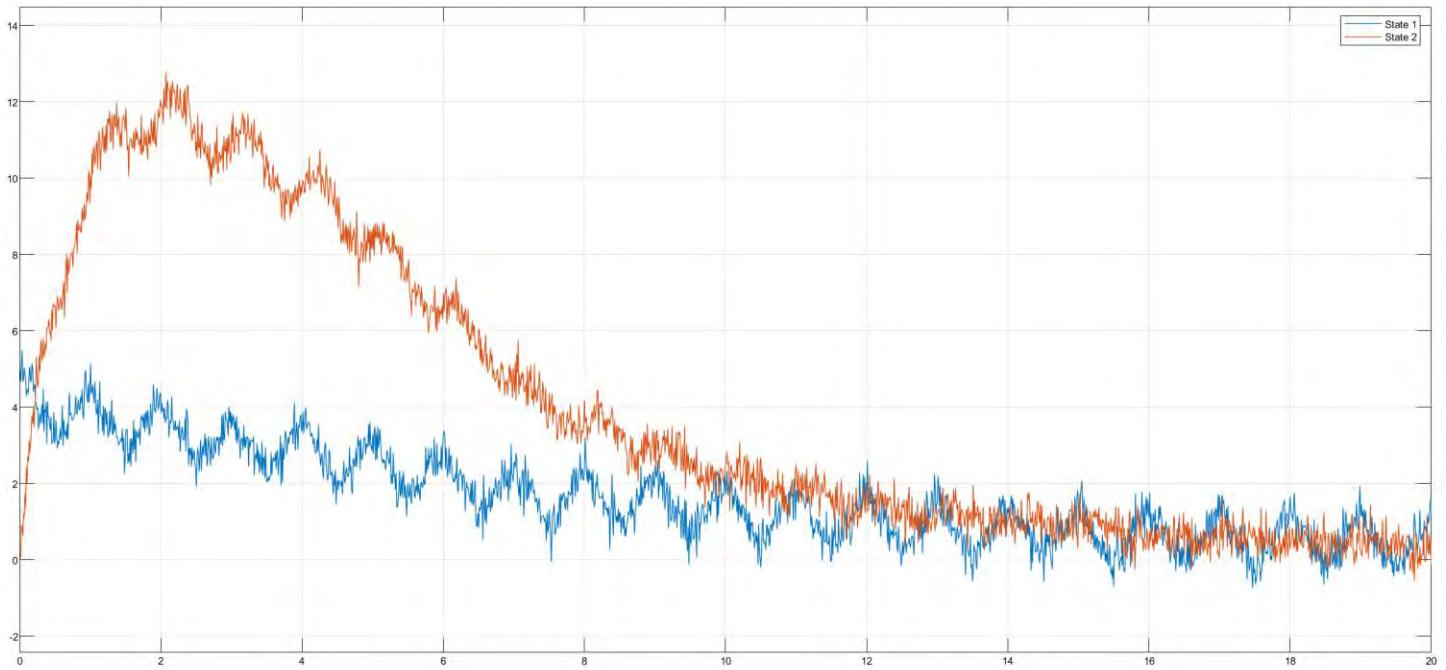
Units: Hertz

Interpret vector parameters as 1-D

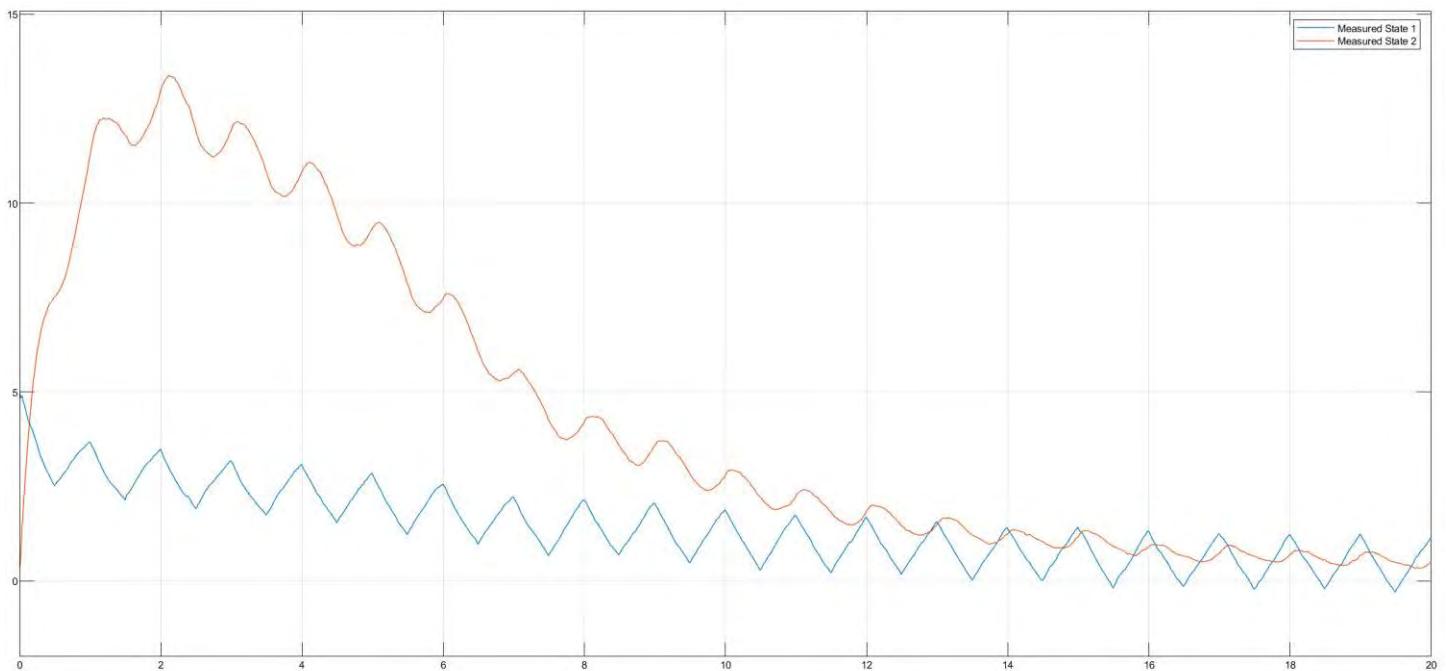
## Initial State Conditions

 Block Parameters: State1	 Block Parameters: State2	 Block Parameters: Unit Delay	 Block Parameters: Unit Delay
Integrator Continuous-time integration of the input signal.	Integrator Continuous-time integration of the input signal.	UnitDelay Sample and hold with one sample per unit delay.	UnitDelay Sample and hold with one sample per unit delay.
Parameters	Parameters	Main    State Attributes	Main    State Attributes
External reset: none	External reset: none	Initial condition: eye(2)*5	Initial condition: [5 0]
Initial condition source: internal	Initial condition source: internal	Input processing: Elements	Input processing: Elements
Initial condition:	Initial condition:	Sample time (-1 for inherited)	Sample time (-1 for inherited)
5	0	-1	-1

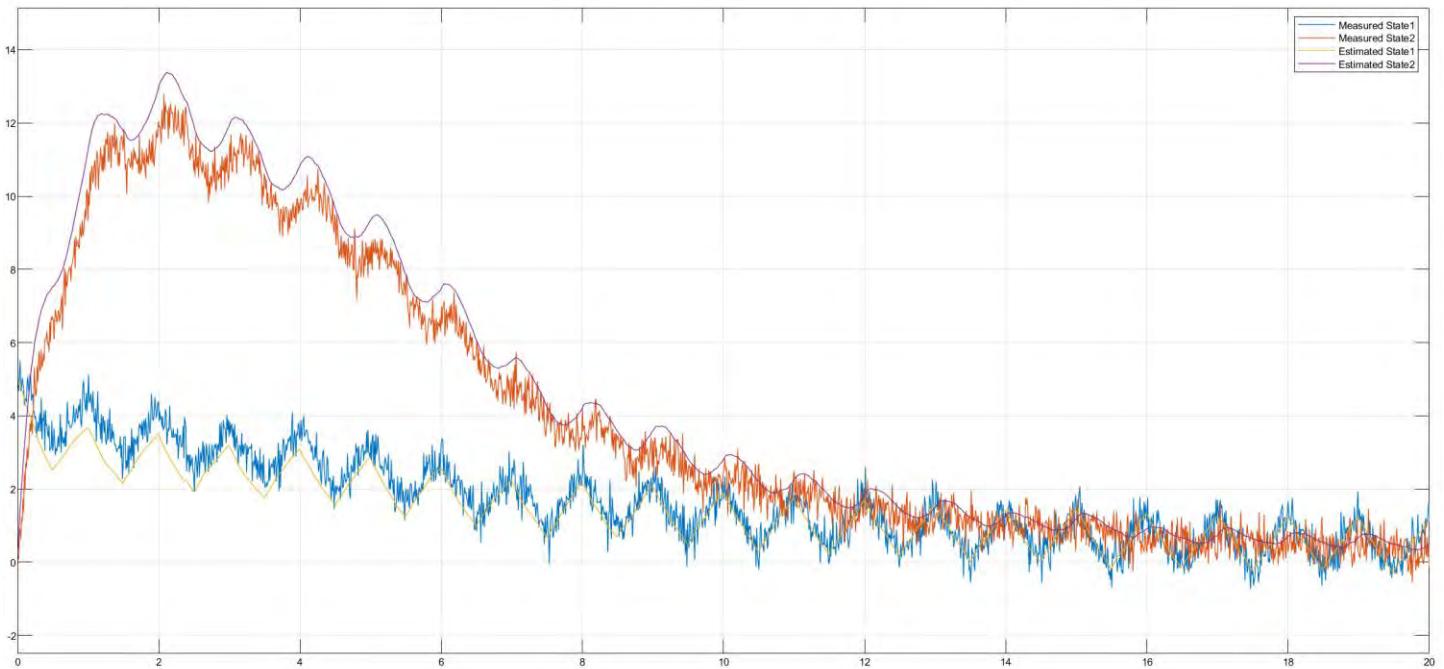
## Measured States



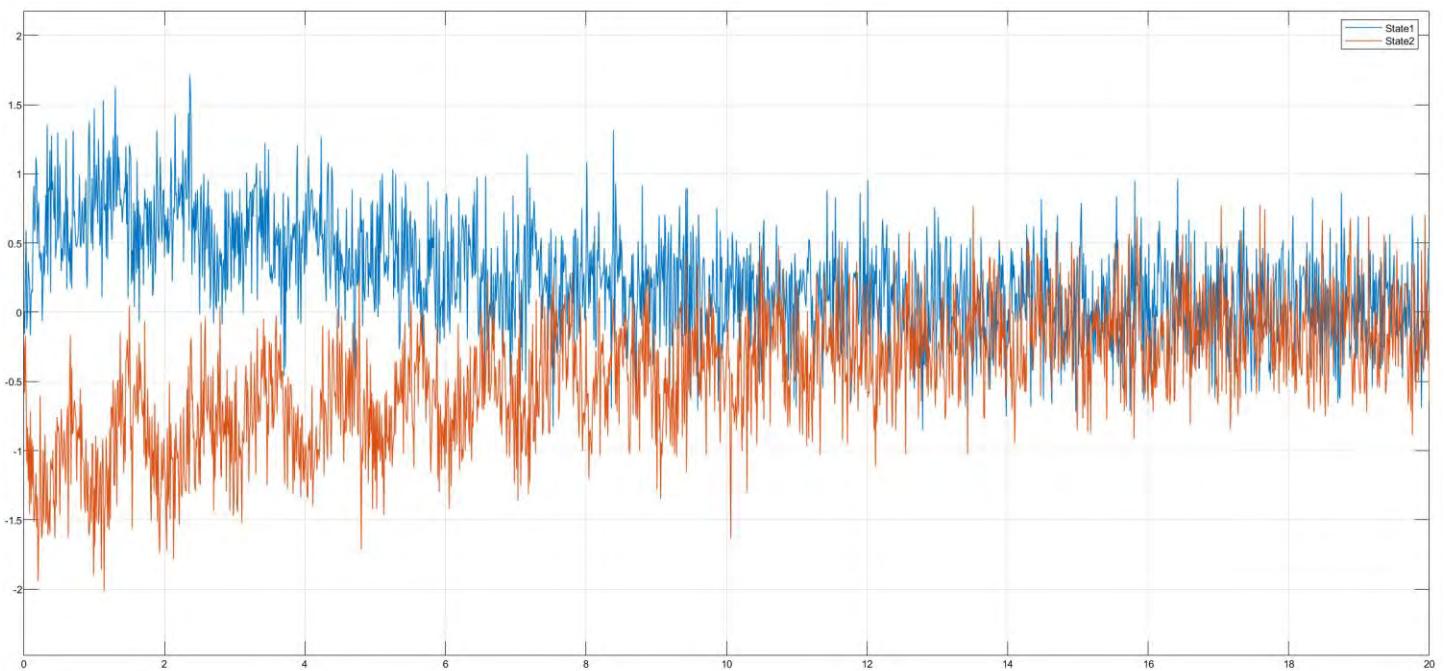
## Filtered States



### Comparison between measured and estimated states

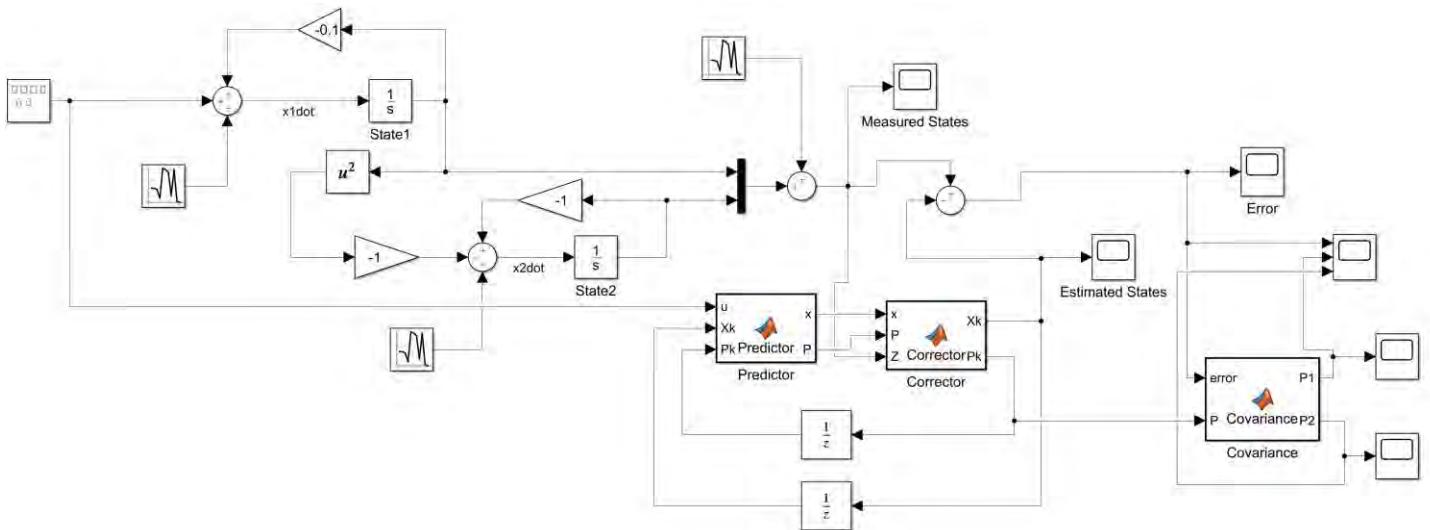


### Error Plot



As observed from the comparison and error plots, the Kalman Filter is slightly inaccurate in the initial stages but it converges quickly within 10 seconds and brings the error values within the 0.5 range.

## Simulink Diagram



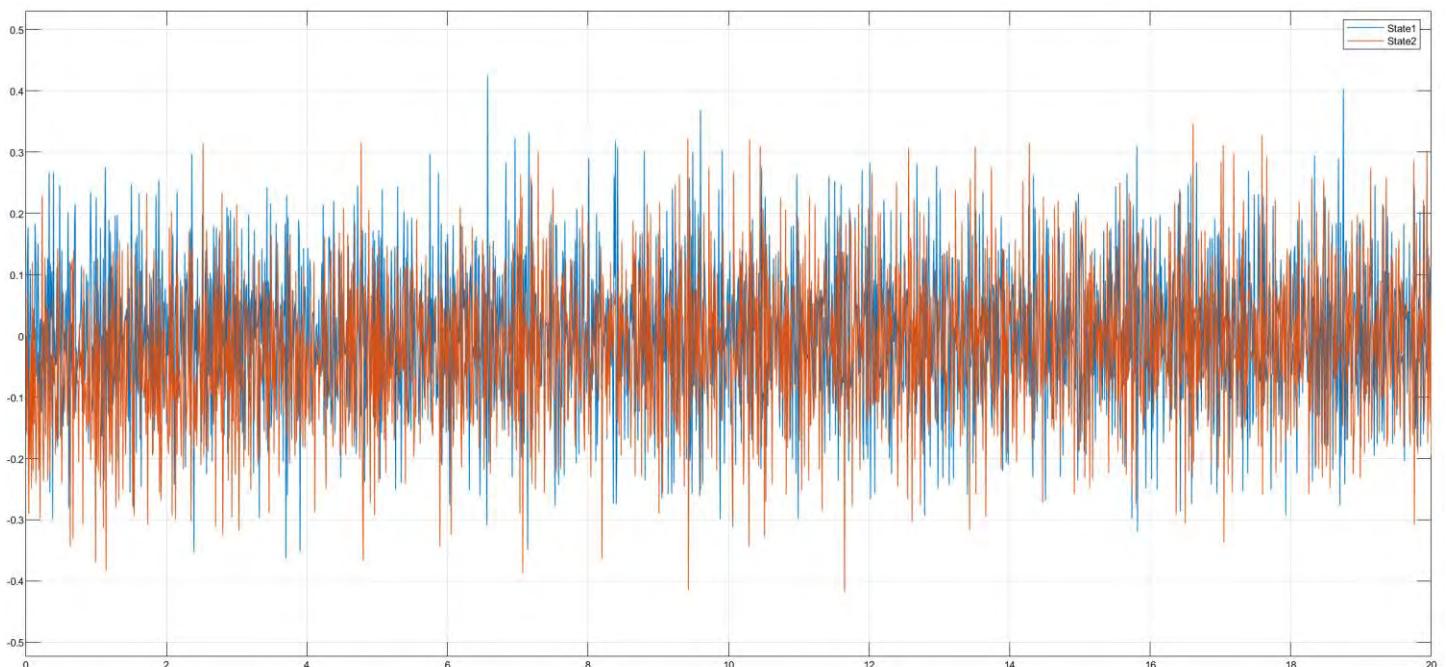
## Covariance Function Code

```
function [P1,P2] = Covariance(error,P)
P1=[+sqrt(P(1,1)) -sqrt(P(1,1)) error(1)];
P2=[+sqrt(P(2,2)) -sqrt(P(2,2)) error(2)];
end
```

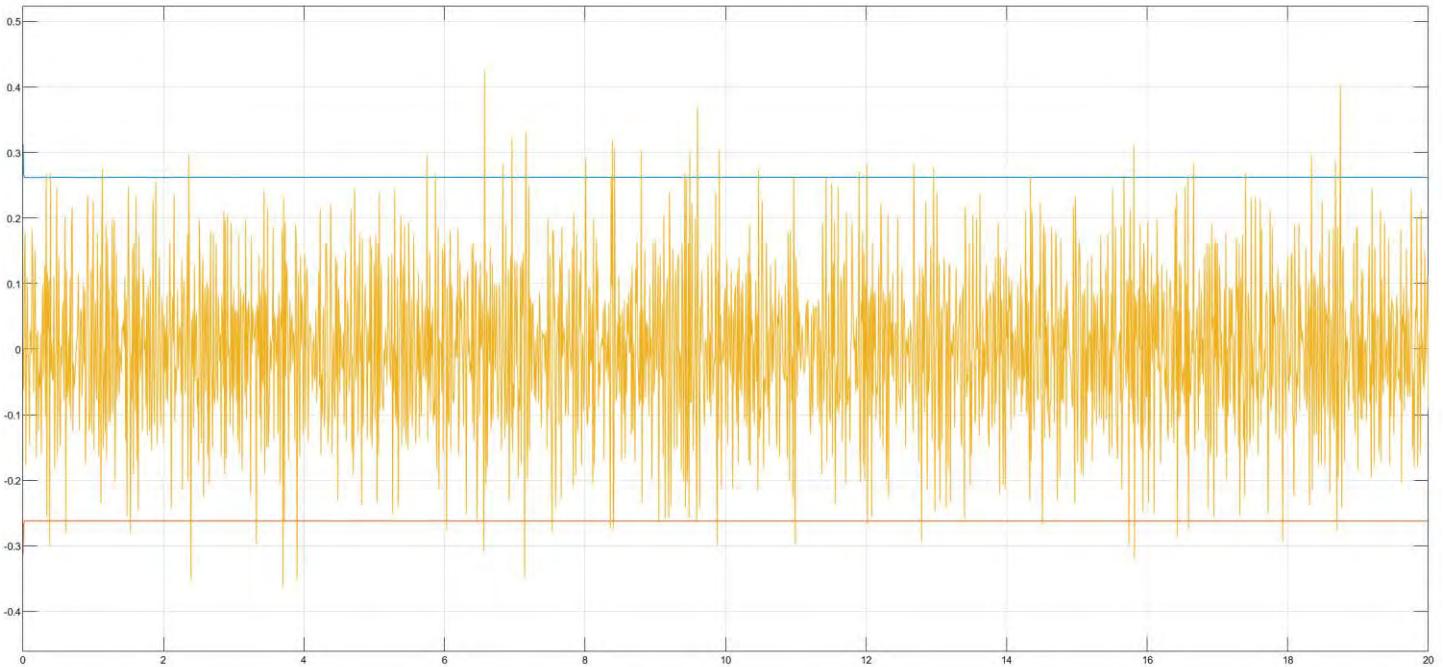
The following Q and R matrices have been used to get error within 0.5 and the error within the covariance about 80% of the time

```
Q=15*eye(2);
R=0.001*eye(2);
```

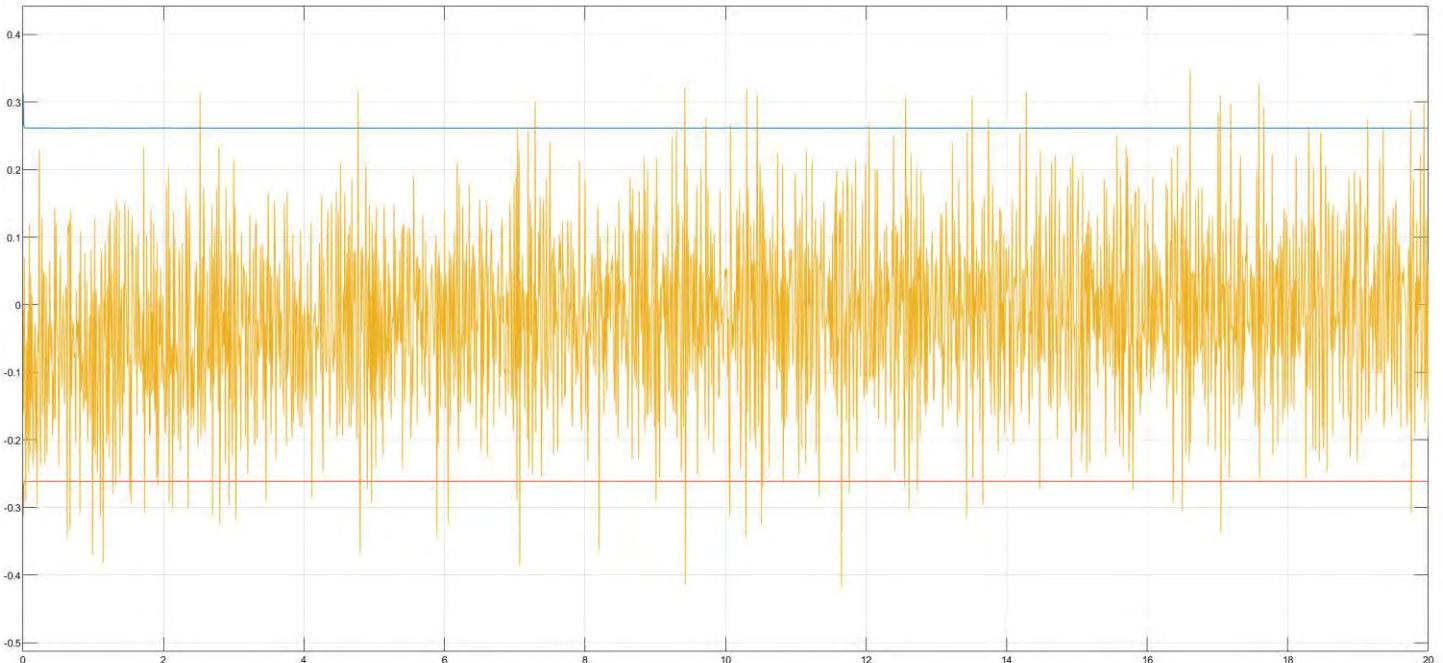
## Error Plot (within 0.5)



### Covariance Plot for State 1

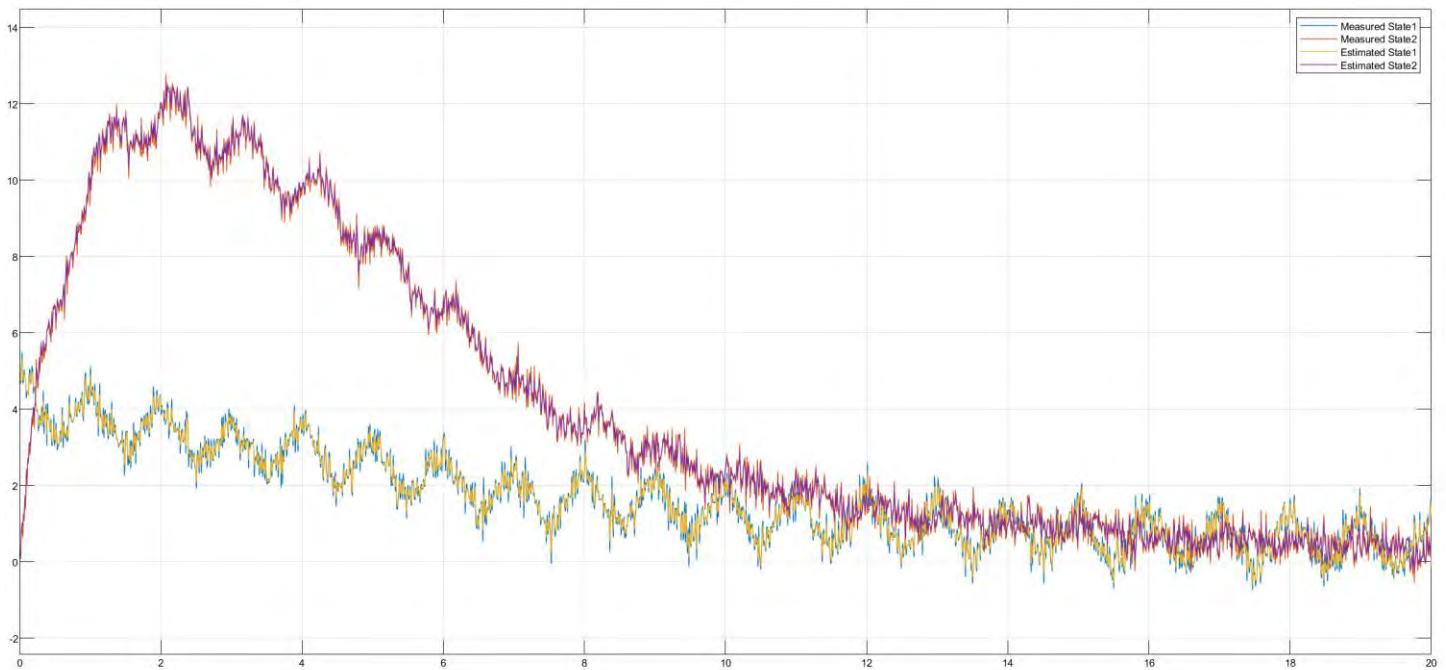


### Covariance Plot for State 2



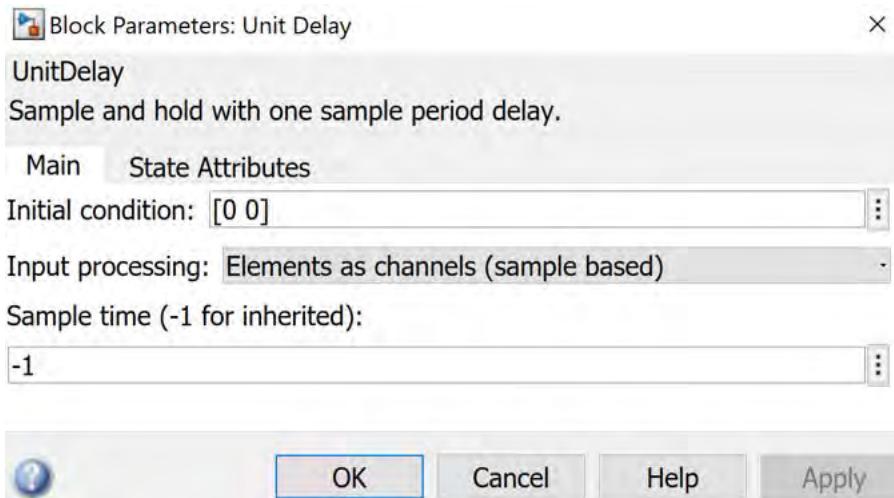
**Note:** The magnitudes cross the covariance lines multiple times. However, the question mentioned that they have to be kept within the limits about 80% of the time. So, the average time of those peaks is around 20% of the total time. Otherwise I could have tried to manipulate the Q and R values further to get them completely within the limits.

### Comparison Plots with new Q and R matrices

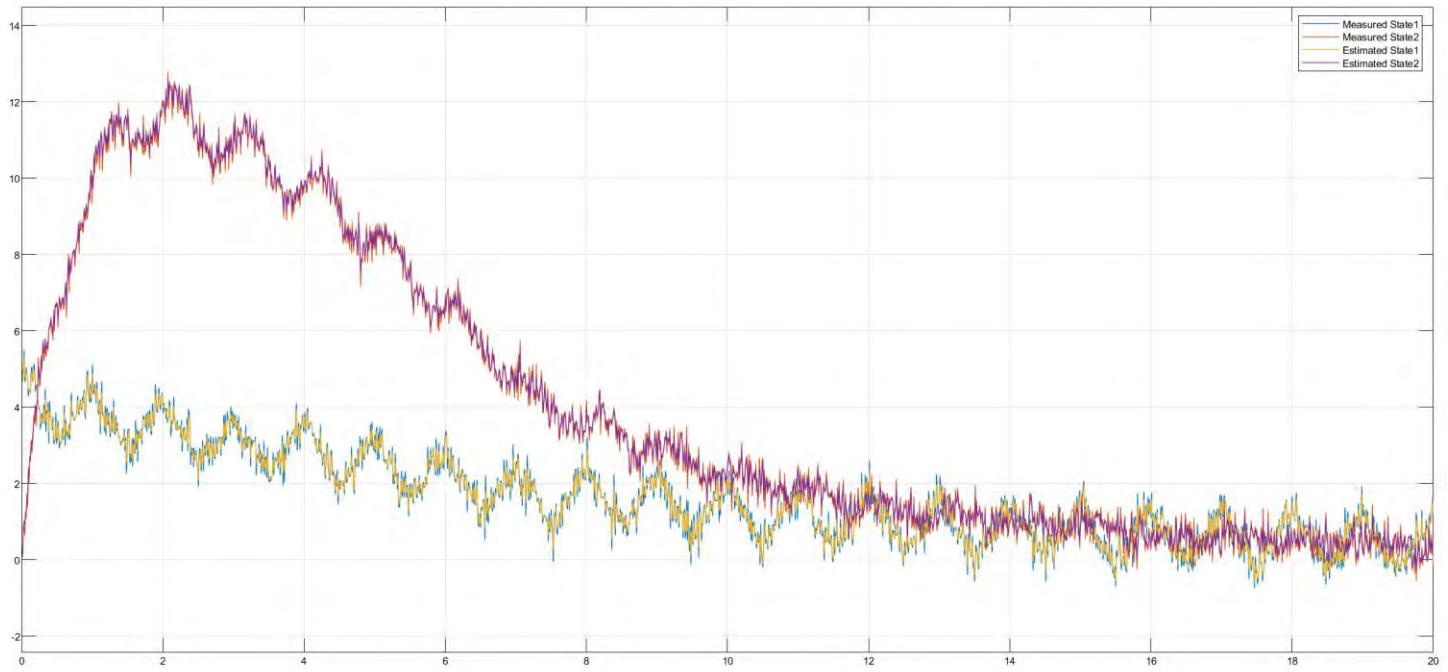


Now everything in the Simulink diagram is kept same, only the initial states of the filter are changed to  $[0,0]$  for observing the effects on estimated values

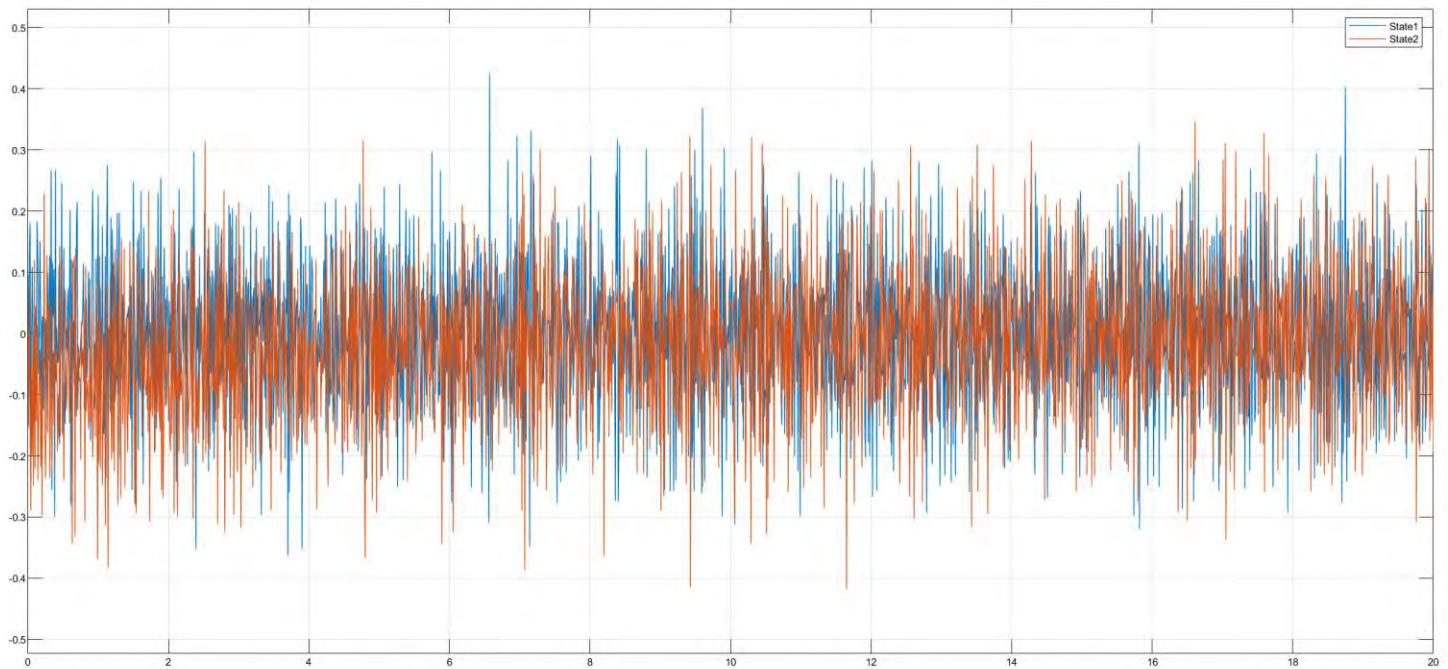
### Defining the initial states in Filter



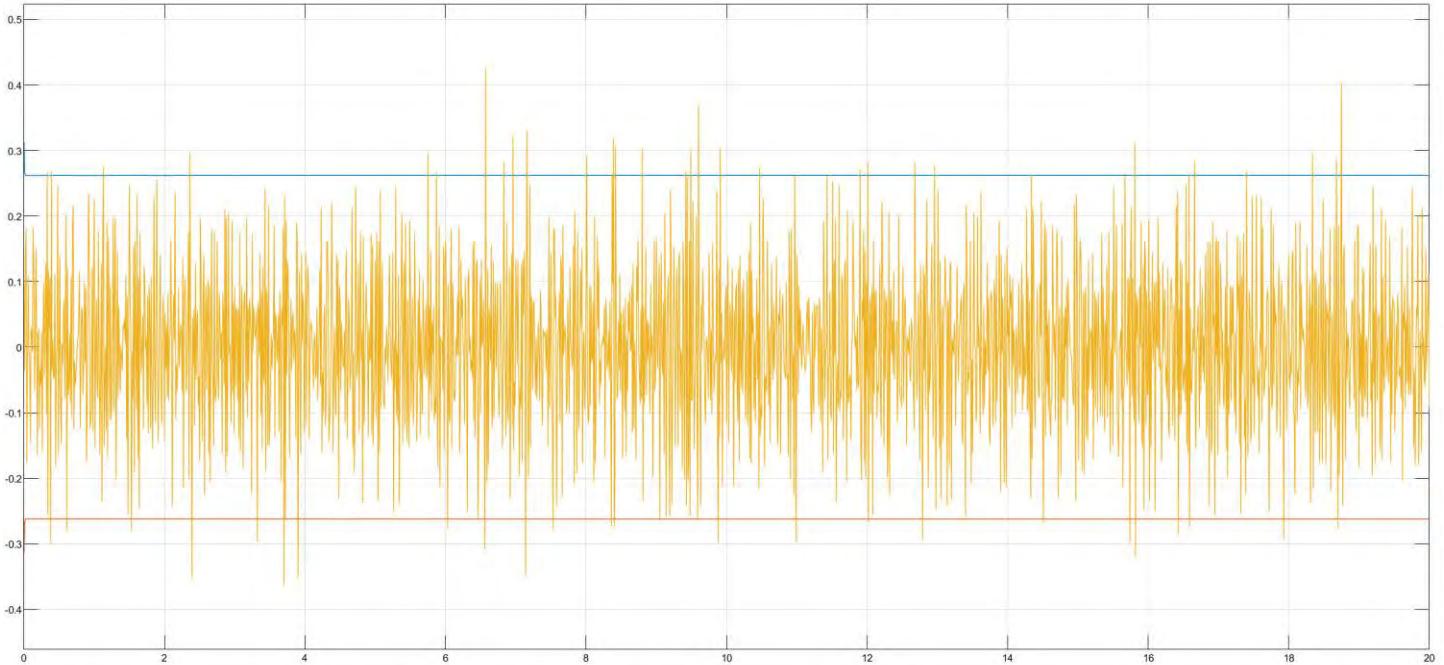
## Comparison Plots for Measured and Estimated States



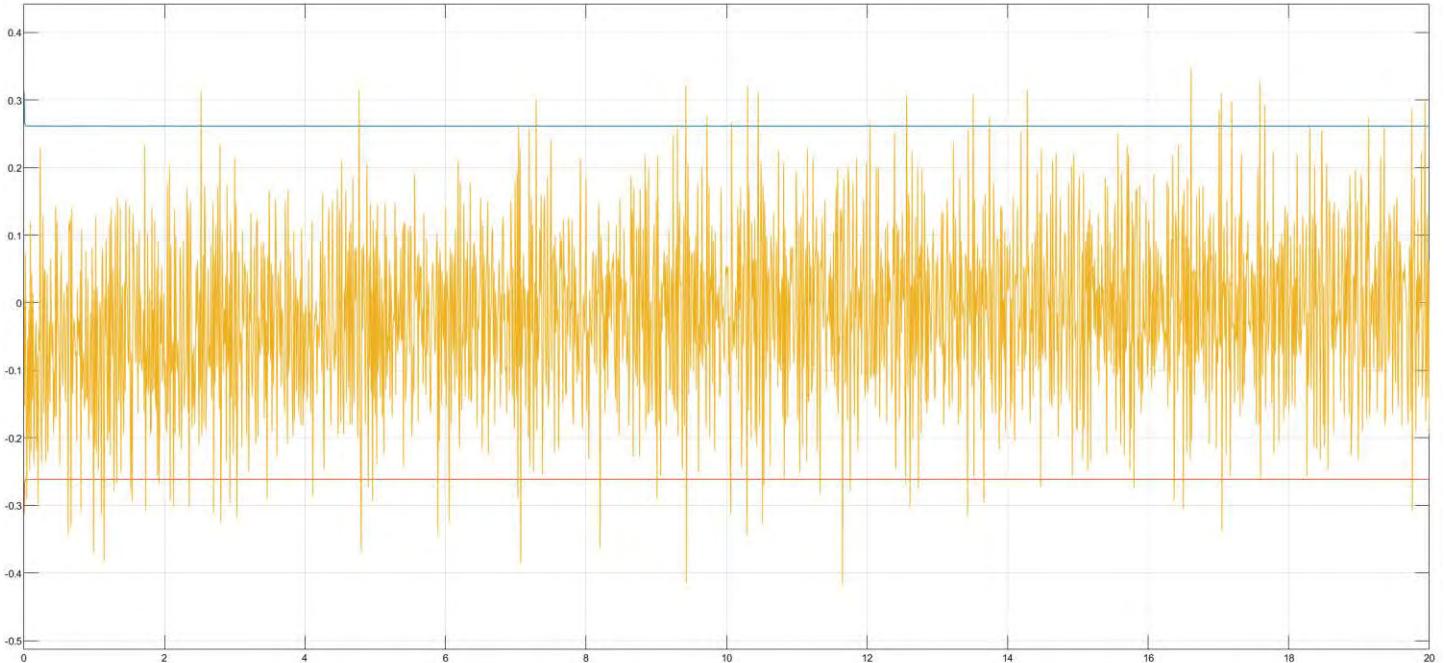
## Error Plot



### Covariance Plot for State 1



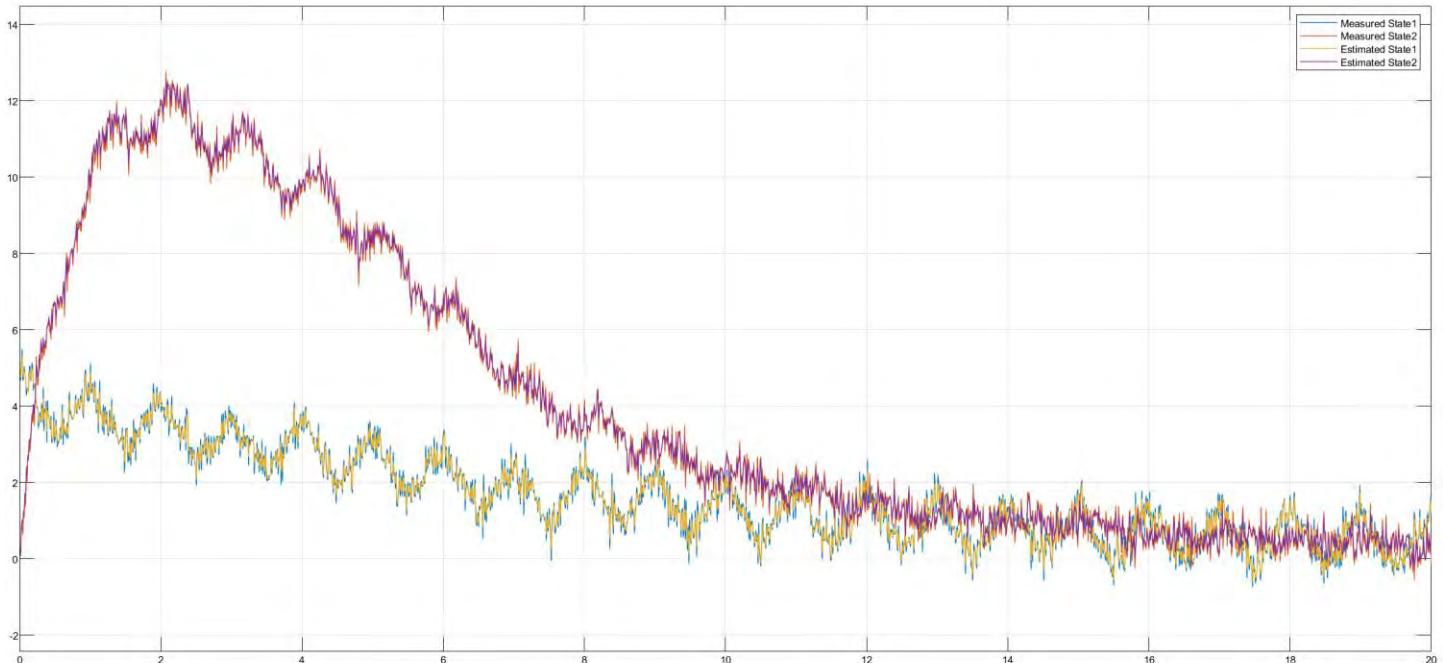
### Covariance Plot for State 2



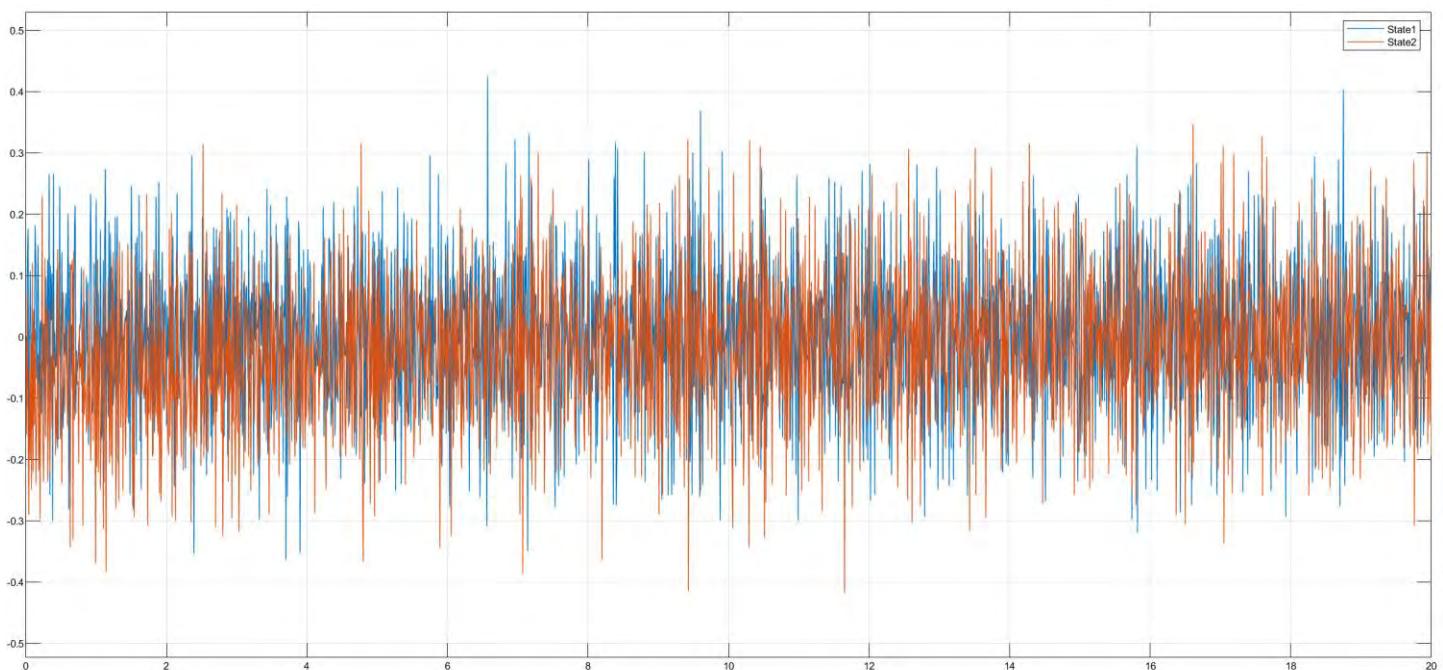
For this particular case/system, the change in initial value of first state from 5 to 0 has no noticeable effect on the output of the filter. For many systems, a significant change in the initial values of the filter may throw it off into another direction as explained by Dr. Niestroy in class.

Now, only the value of mu is changed to -0.01 and the initial states are reverted back to [5,0] and the simulation is rerun.

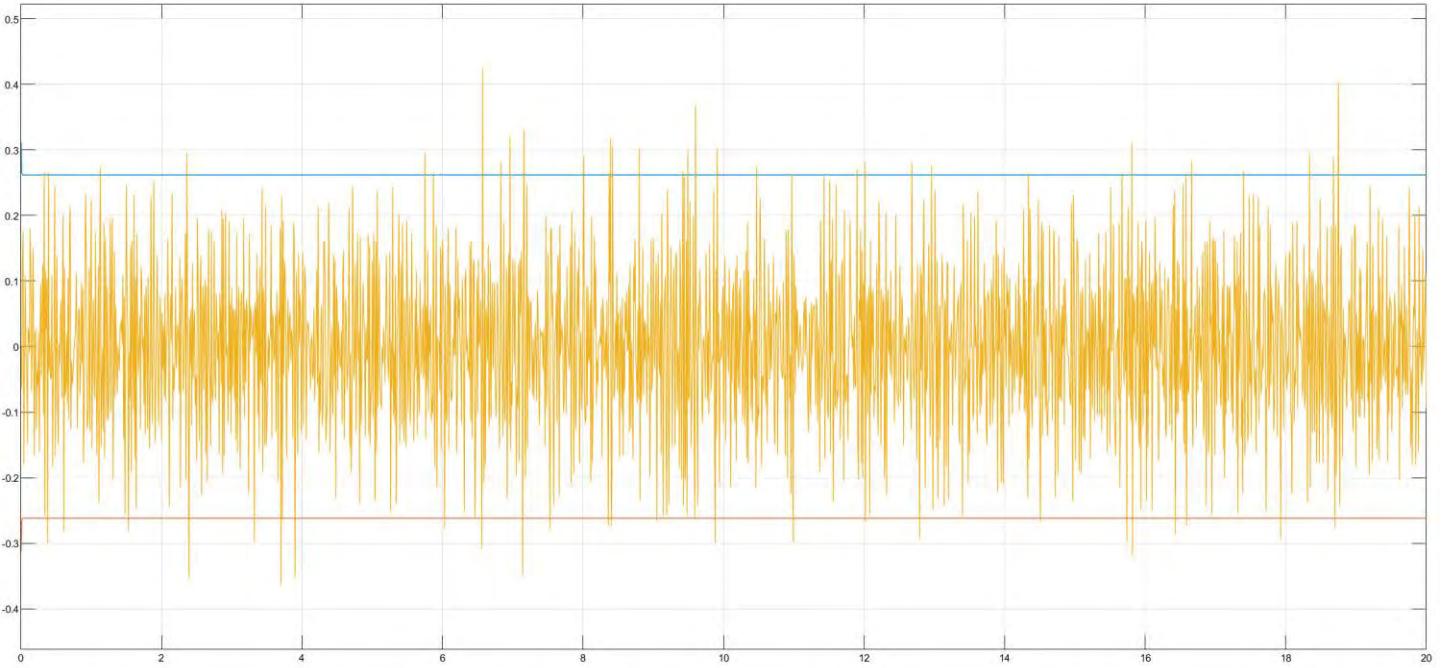
Comparison Plots for measured and estimated values



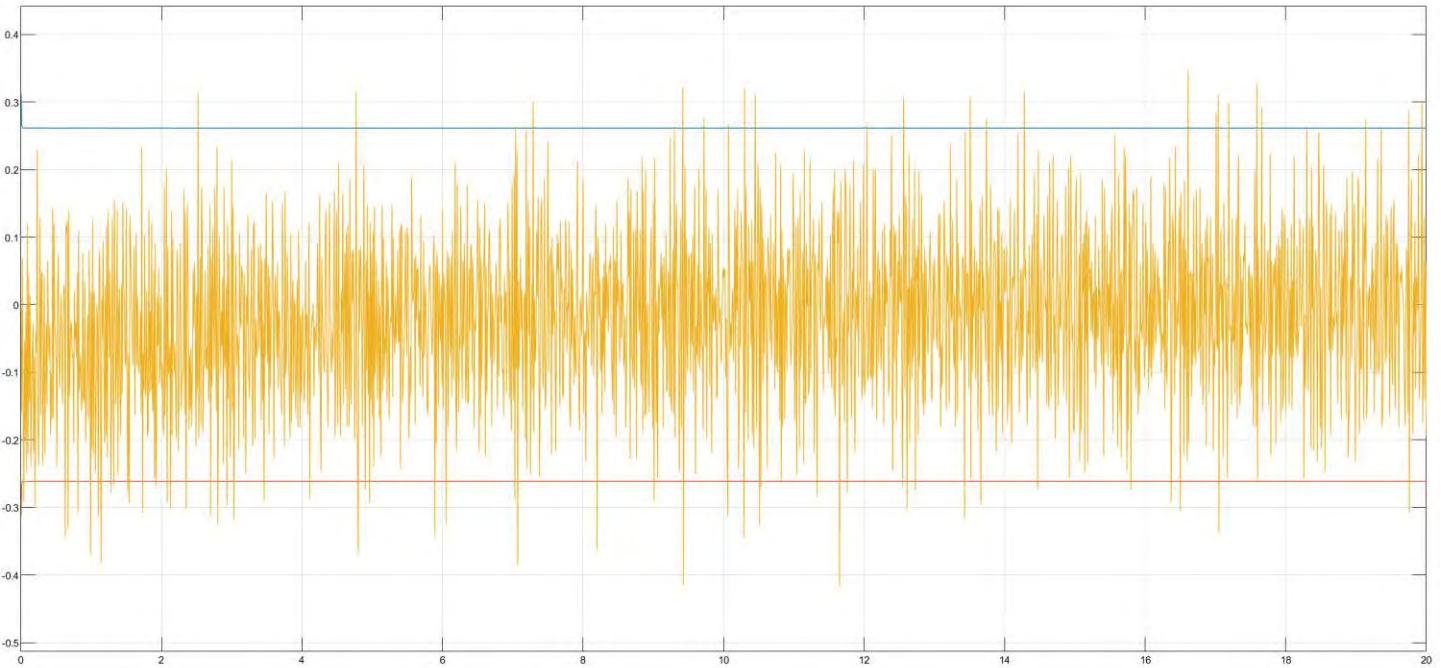
Error Plot



**Covariance Plot for State 1**



**Covariance Plot for State 2**



As observed from the above plots, there is no noticeable change in the output of the filter when the value of  $\mu$  is changed from  $-0.1$  to  $-0.01$  in the predictor block and the initial states are reversed to the original values of  $[5, 0]$ .

This means that for this particular case, the filter does a good job even when the physics specified in the predictor block is slightly incorrect.

**EE5327, Fall 2020**  
**Homework 4: System Identification Tool**  
**Due 11/05/2020**

Generate time input and output time history data to be used in the Sys ID tool to create approximate models using just that data. Use a unit step input occurring at 0.01 seconds, with a simulation stop time of 20 seconds and a fixed step solver with step size of 0.01 seconds. Generate two variables, input and output needed by the tool.

$$y(t) = \frac{-s^2 - 0.9s + 1}{s^3 + 1.5s^2 + 1.5s + 1} u(t)$$

**Problem 1) Polynomial functions ARX/ARMAX (30 points)**

Use the System Identification App/toolbox to create input/output models of the data generated for the system described above. Here, though, you only need the input (the step function) and the output of the second order system.

- a) Use the system identification app to create an ARX model of the system dynamics with the Focus set to Simulation. When you import the data, make sure to specify the begin time (0) and the sample time (0.01 sec). Then, choose to estimate a polynomial model, ARX structure. You get to pick the order.
- b) Extract the A(z) and B(z) polynomials from the model and compare those polynomials with the result you find by using c2d to convert the original s-plane function. They should be close for at least the A(z) expression. The B(z) might be different, but let's see what the simulation looks like in part c).
- c) Note that, in Simulink, you can put a polynomial model into an Idmodel block, found under the System Identification Toolbox pallet in Simulink. Export your model from the Sys ID tool for part c) to MATLAB and use the Idmodel block in Simulink to implement the approximated model. Compare the time histories.
- d) Now add the zero mean, 0.001 variance noise to the output and use that data to repeat step a) and step c). What does noise do to the ability to produce an accurate model of the system? Feel free to play with the tool settings to make the model as good as you can get it. You can also try the ARMAX setting with various order models.

**Problem 2) Transfer Function Approximation (30 points)**

For the same system as above, use the System Identification app to identify a transfer function.

- a) Use the system identification app to create third order transfer function denominator, second order numerator using the data without noise.
- b) Compare the poles and zeros of the identified transfer function with the true values.
- c) Export the model and use the Idmodel block to import the identified transfer function into Simulink. Compare the time histories of true vs identified.

- d) Now repeat steps a), b), and c) when using the output which has the random noise added. What does noise do to the ability to produce an accurate model of the system in this case?

**Problem 3) State Space ID (30 points)**

Repeat the process as in Problem 1, but this time identify a state space model.

- a) Use the system identification app to create a state space model using the data without noise. You might have to turn off the ‘Allow unstable models’ checkbox under the Estimation Options section of the GUI and set the Focus to Simulation.
- b) Compare the poles and zeros of the identified model with the true values.
- c) Export the model and use the Idmodel block to import the identified state space system into Simulink. Compare the time histories of true vs identified.
- d) Now repeat steps a), b), and c) when using the output which has the random noise added. Feel free to experiment with the different tool settings to try to make the model as good as you can get it.
- e) What does noise do to the problem, i.e. does it help or hurt?

**Problem 4) Discussion (10 points)**

Comment on which of the three methods you tried above produced the best model fit for the noise-free case and for the noisy data case. Which would you probably use if you had to develop a model from data that has noise to it? Note this is probably problem-dependent so it would be best to try different methods to see what works best for your situation.

**ATUL SHROTRIYA**

**UTA ID - 1001812437**

**System Identification Toolbox**

**Homework - 4**

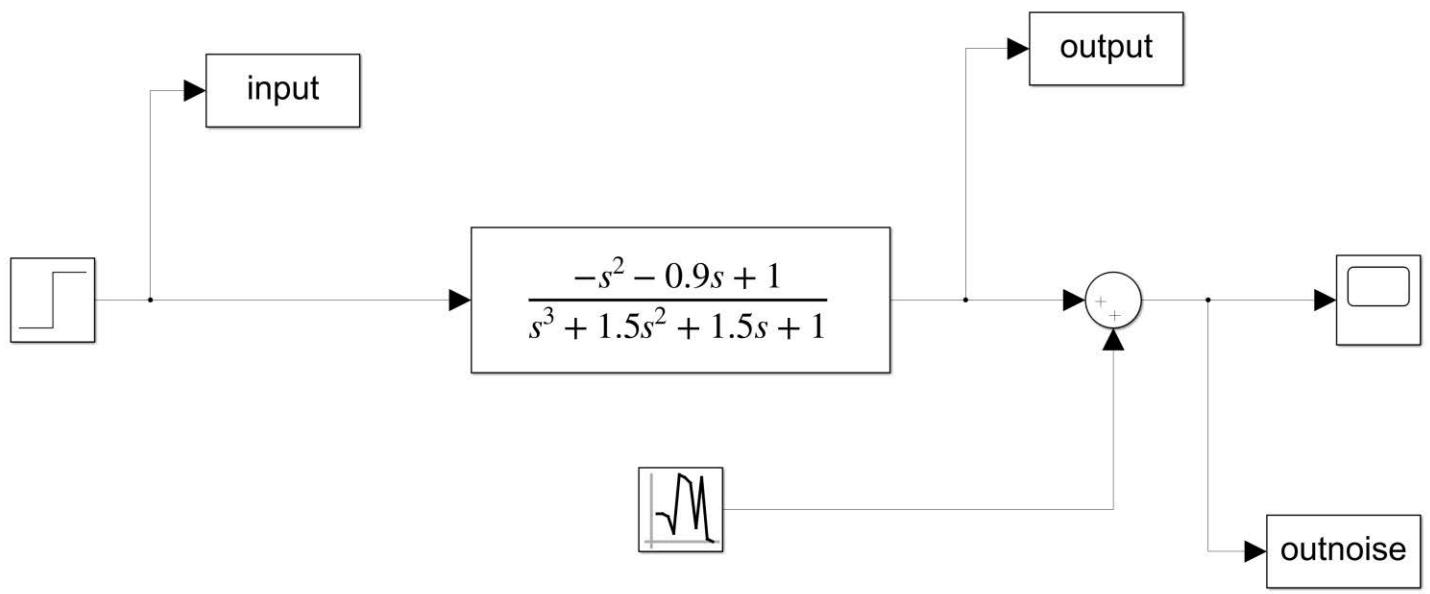
**Submitted - November 5, 2020**

**Course:**

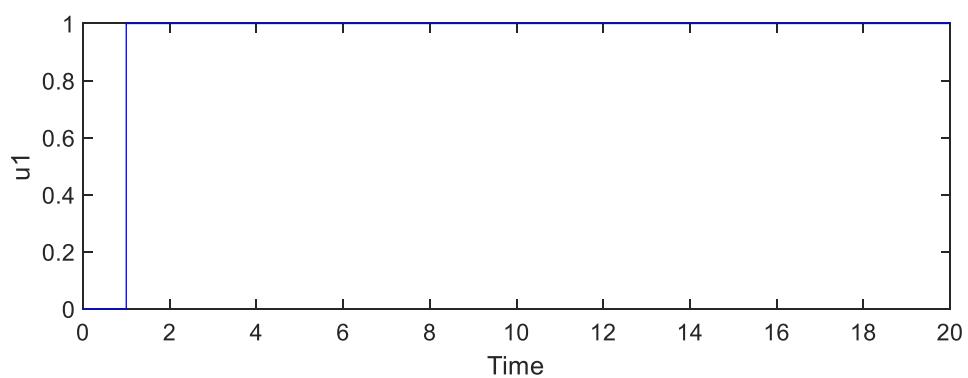
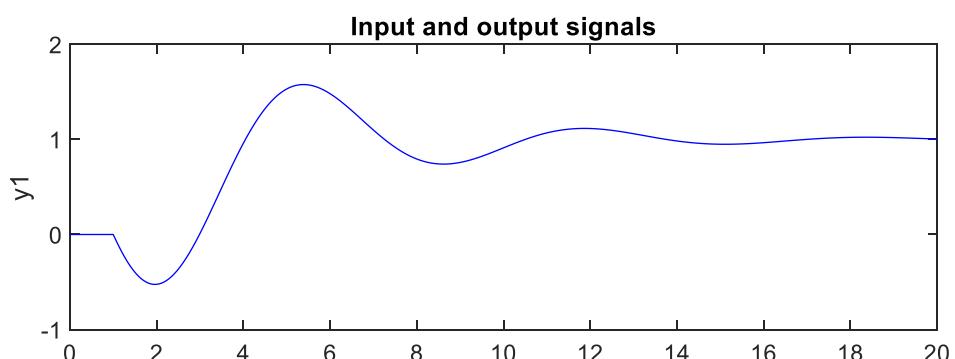
**System Identification and Estimation**

## Problem - 1

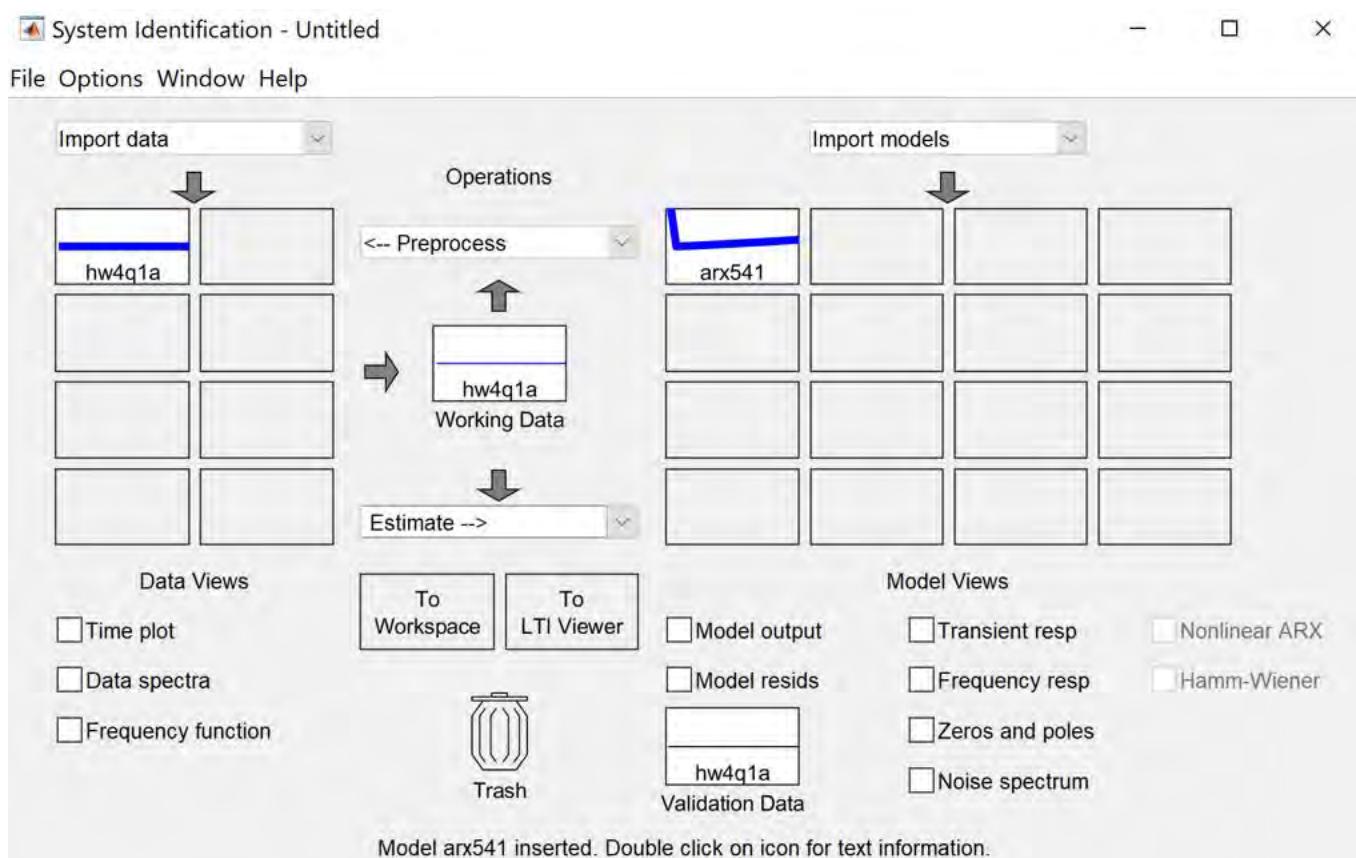
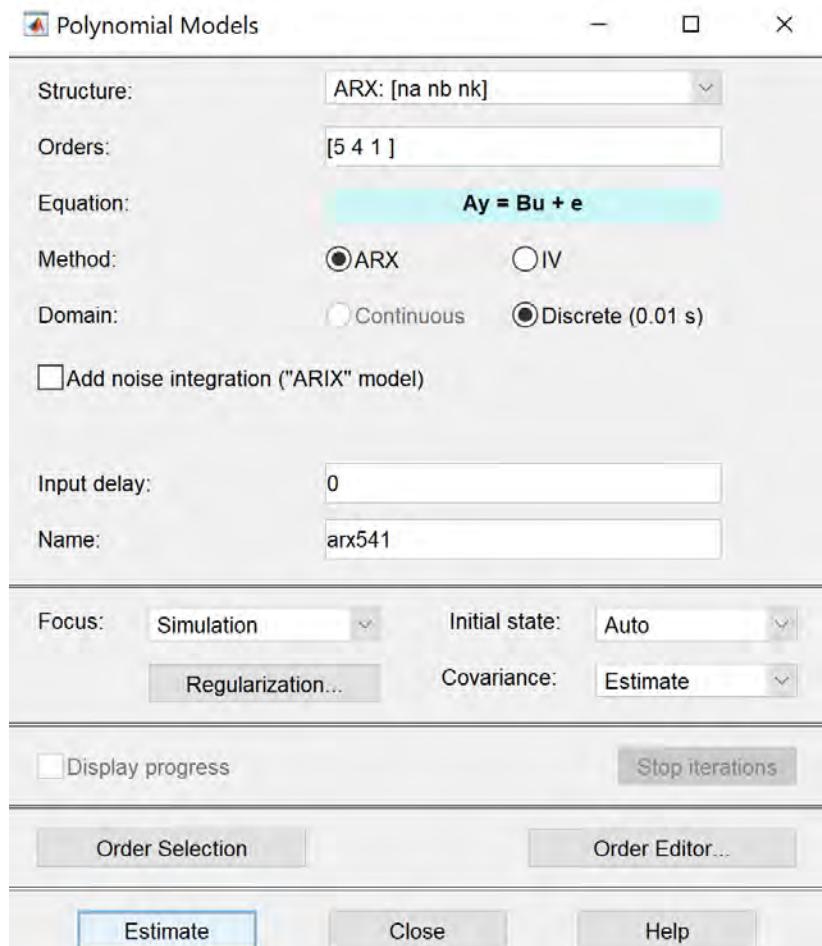
a.



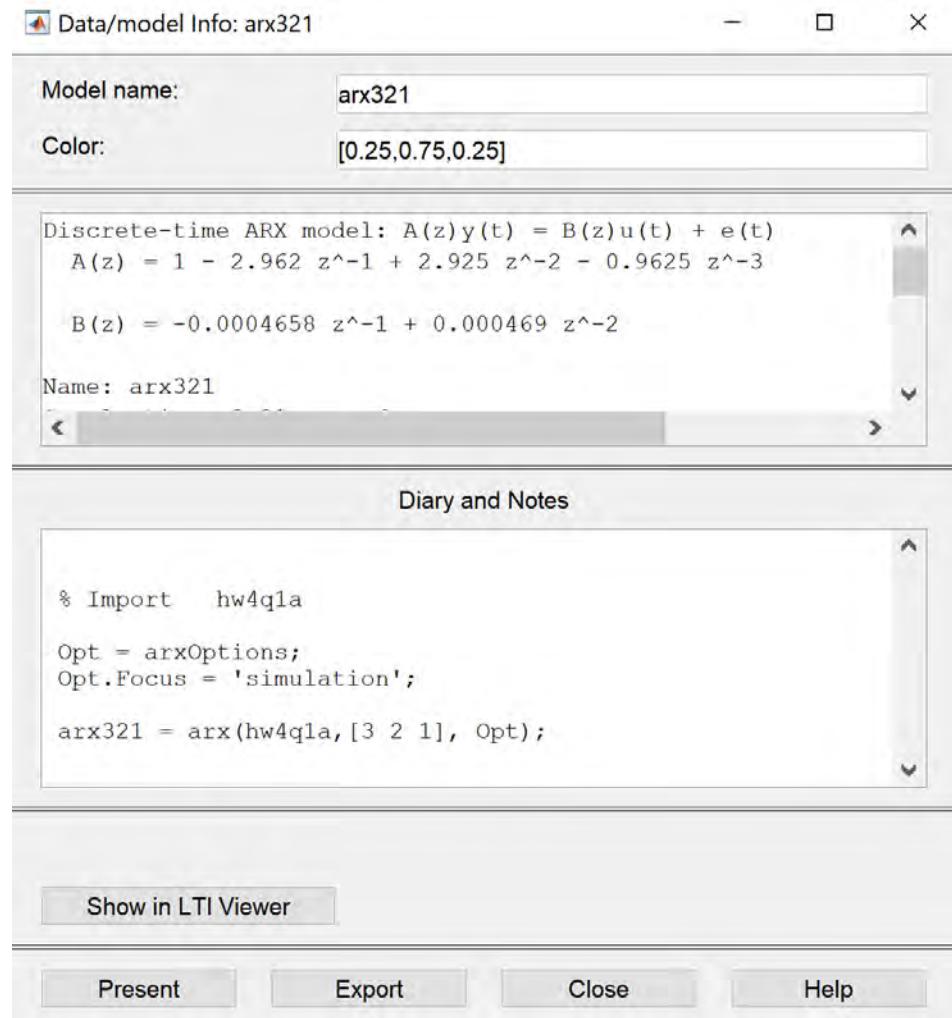
Simulink Diagram



Plots for Input and Output signals



## Importing data and estimating the polynomial function



b.

### MATLAB Commands for c2d

```
>> F=tf([-1 -0.9 1],[1 1.5 1.5 1])
```

```
F =
```

$$\frac{-s^2 - 0.9 s + 1}{s^3 + 1.5 s^2 + 1.5 s + 1}$$

Continuous-time transfer function.

```
>> Fd=c2d(F,0.01)
```

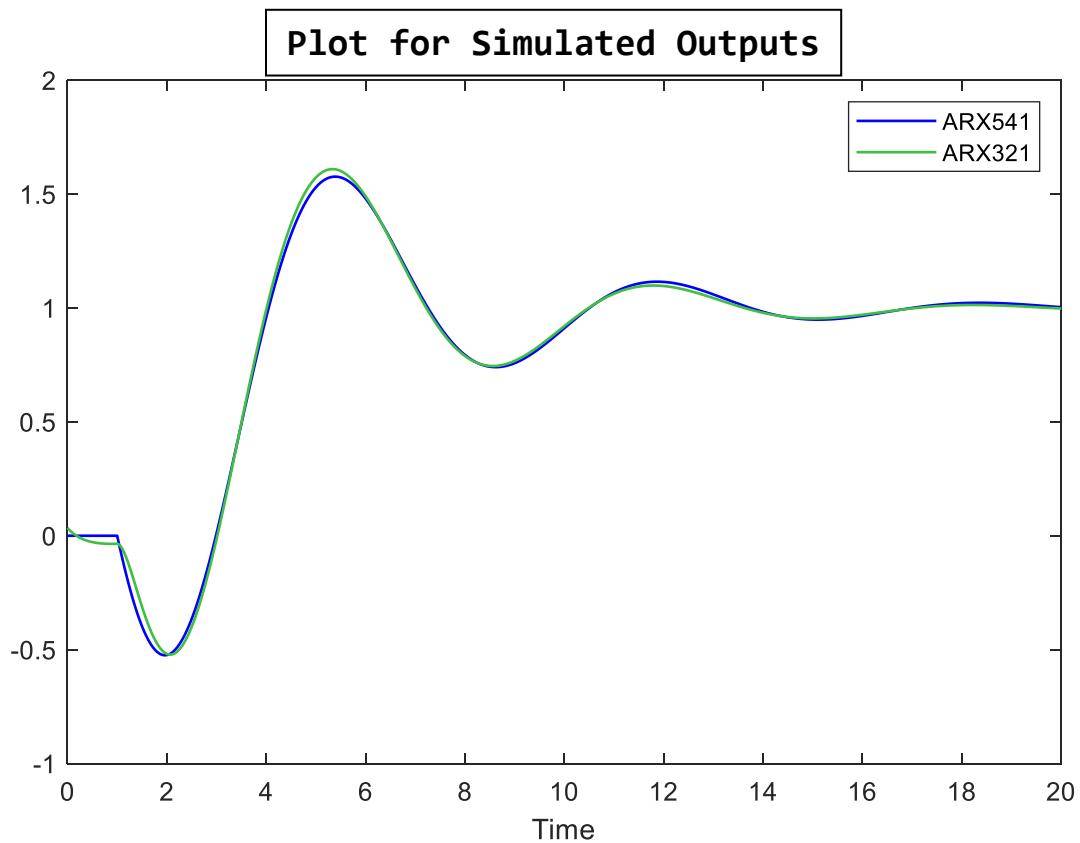
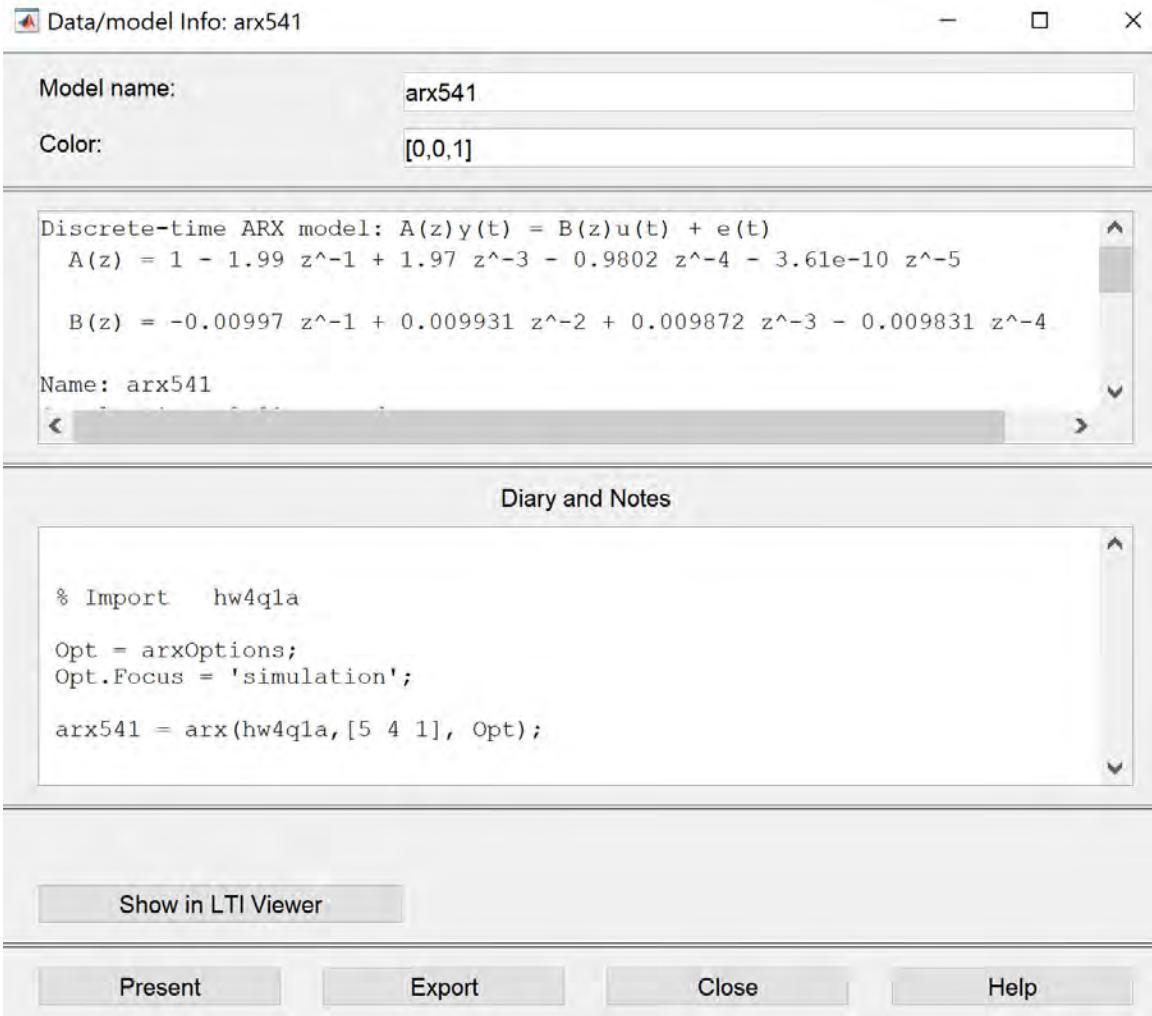
```
Fd =
```

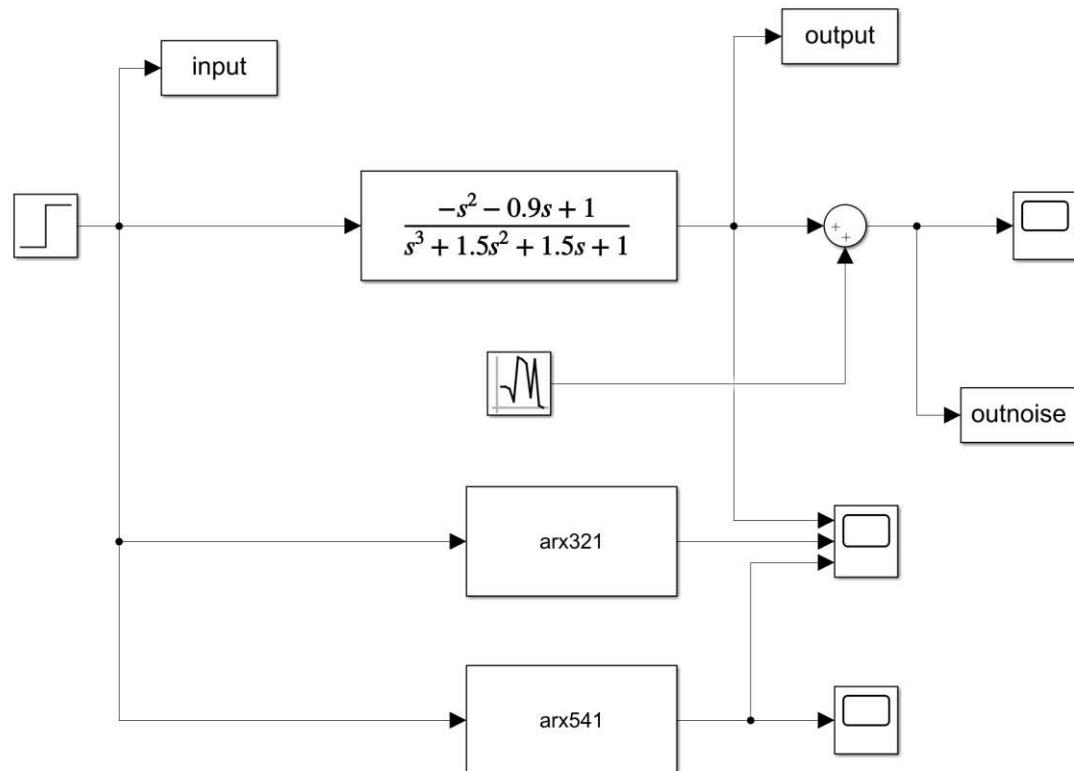
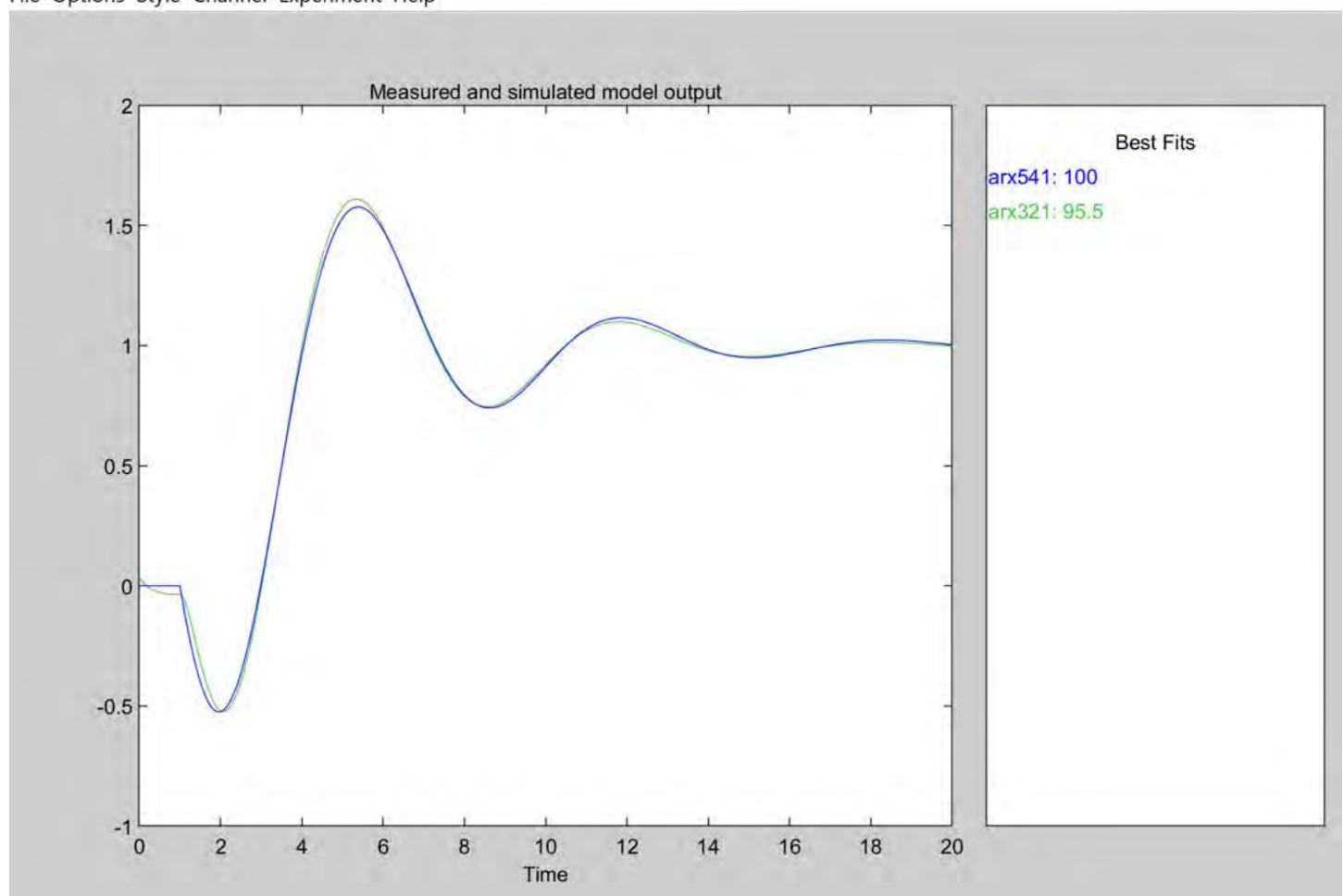
$$\frac{-0.00997 z^2 + 0.01985 z - 0.00988}{z^3 - 2.985 z^2 + 2.97 z - 0.9851}$$

Sample time: 0.01 seconds

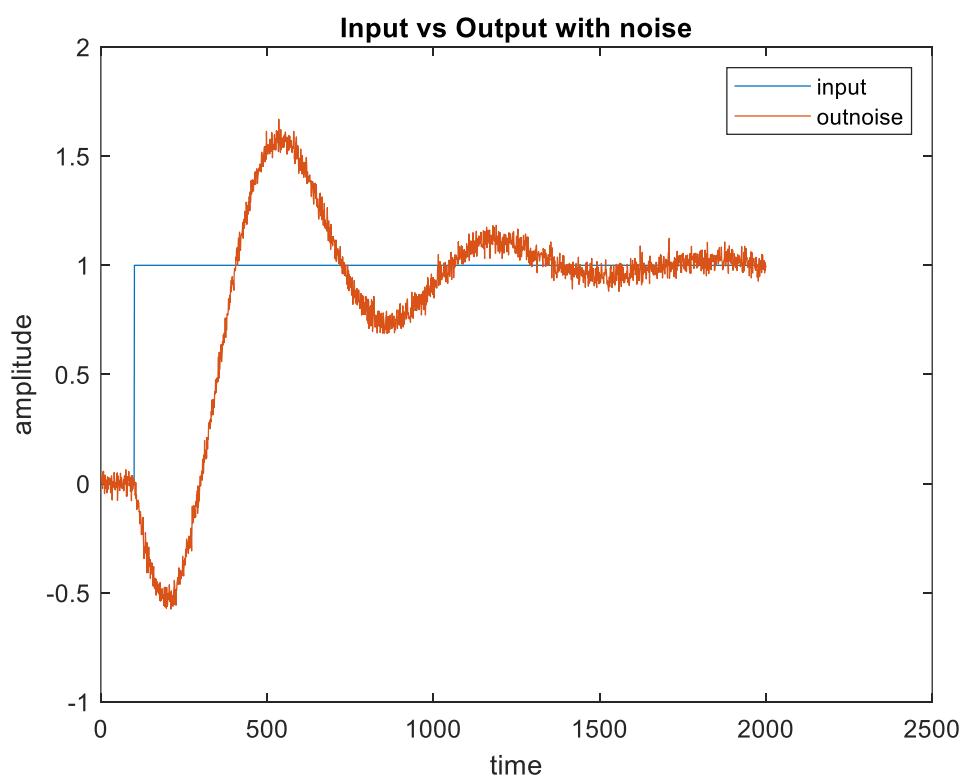
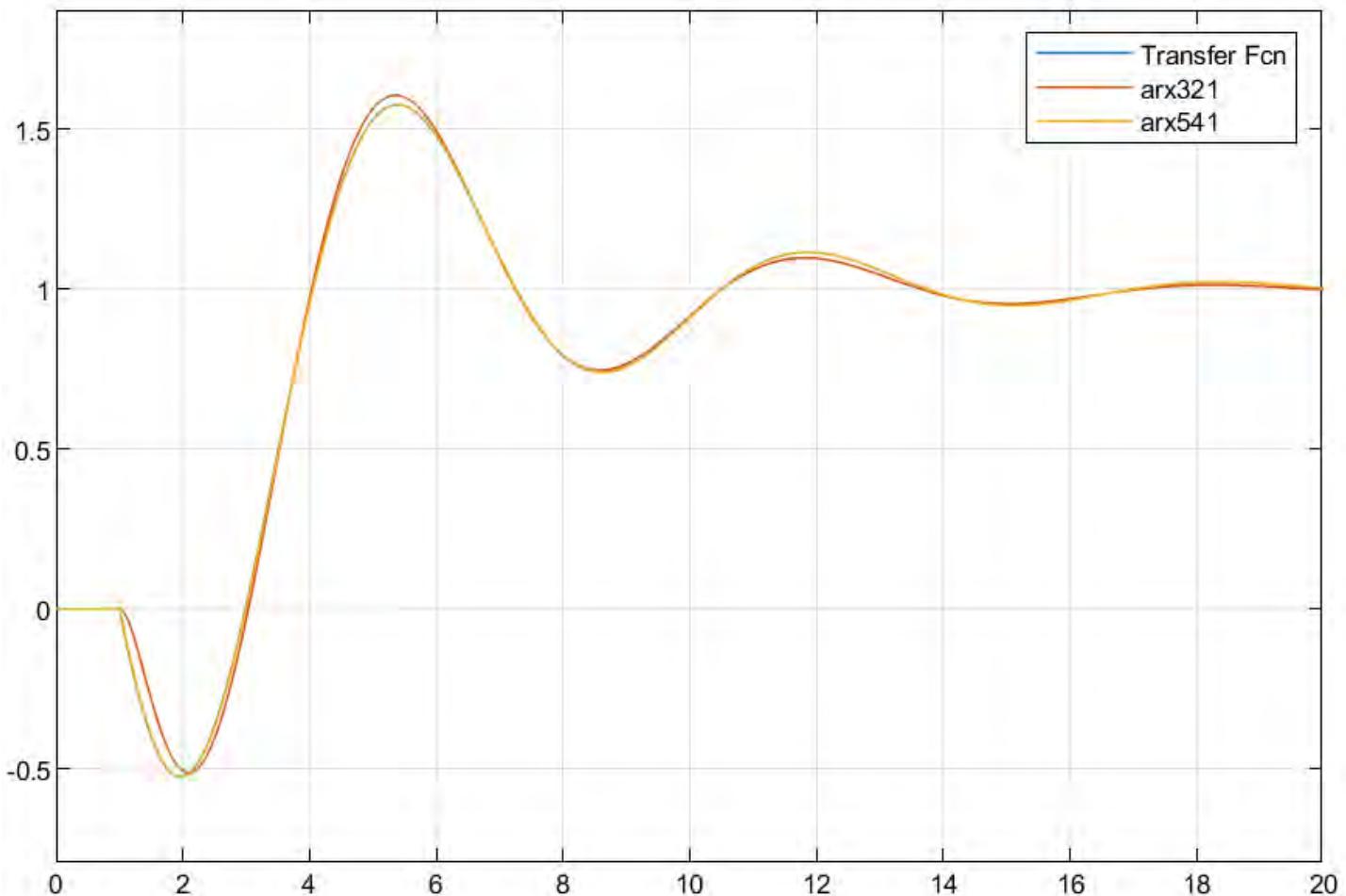
Discrete-time transfer function.

**Conclusion** - The  $A(z)$  is similar to the denominator of the discrete transfer function when it is estimated using the same orders for numerator and denominator. Higher order estimations cannot be compared easily.





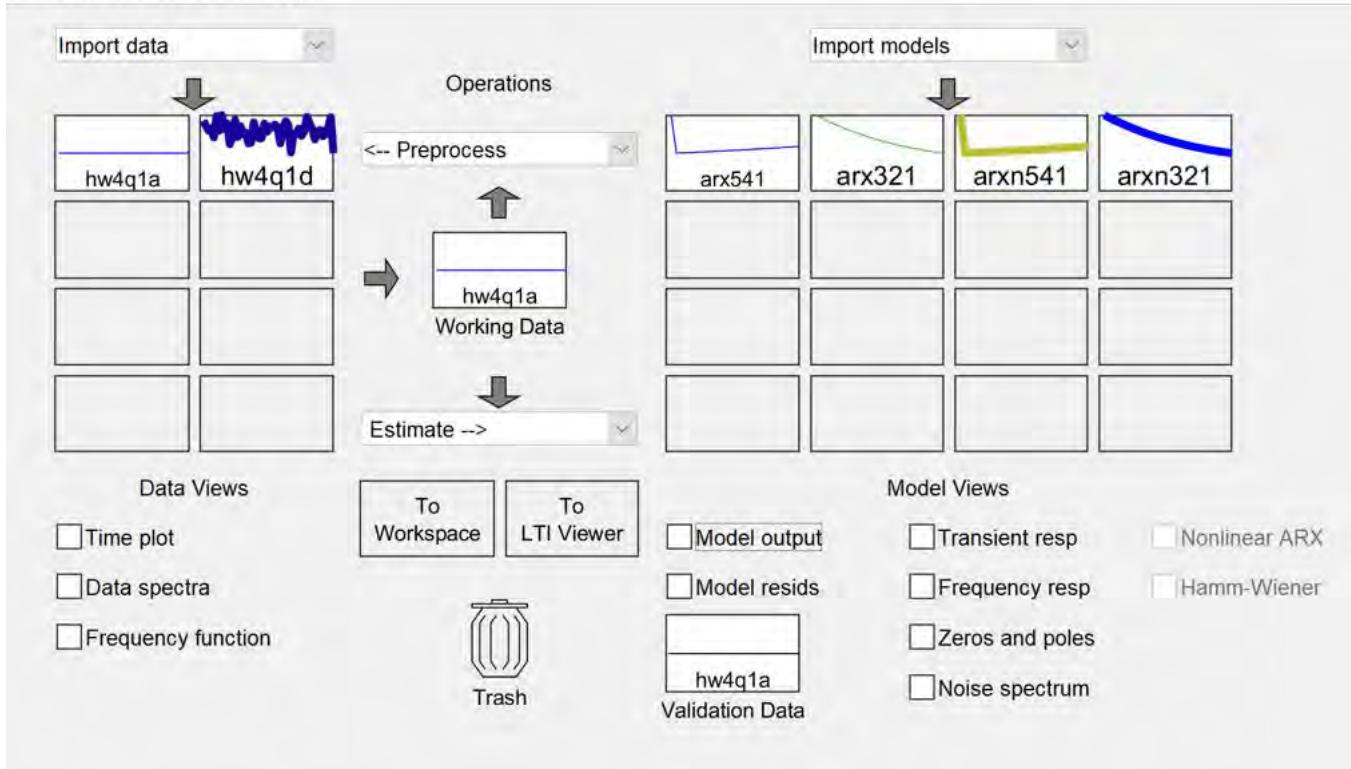
Comparison plots for estimated and actual outputs



d.

# System Identification - Untitled

File Options Window Help



### Polynomial Models

Structure: ARX: [na nb nk]      Orders: [ 3 2 1 ]

Equation:  $Ay = Bu + e$

Method:  ARX       IV

Domain:  Continuous       Discrete (0.01 s)

Add noise integration ("ARIX" model)

Input delay: 0

Name: arx\_n\_321

Focus: Simulation      Initial state: Auto

Regularization...      Covariance: Estimate

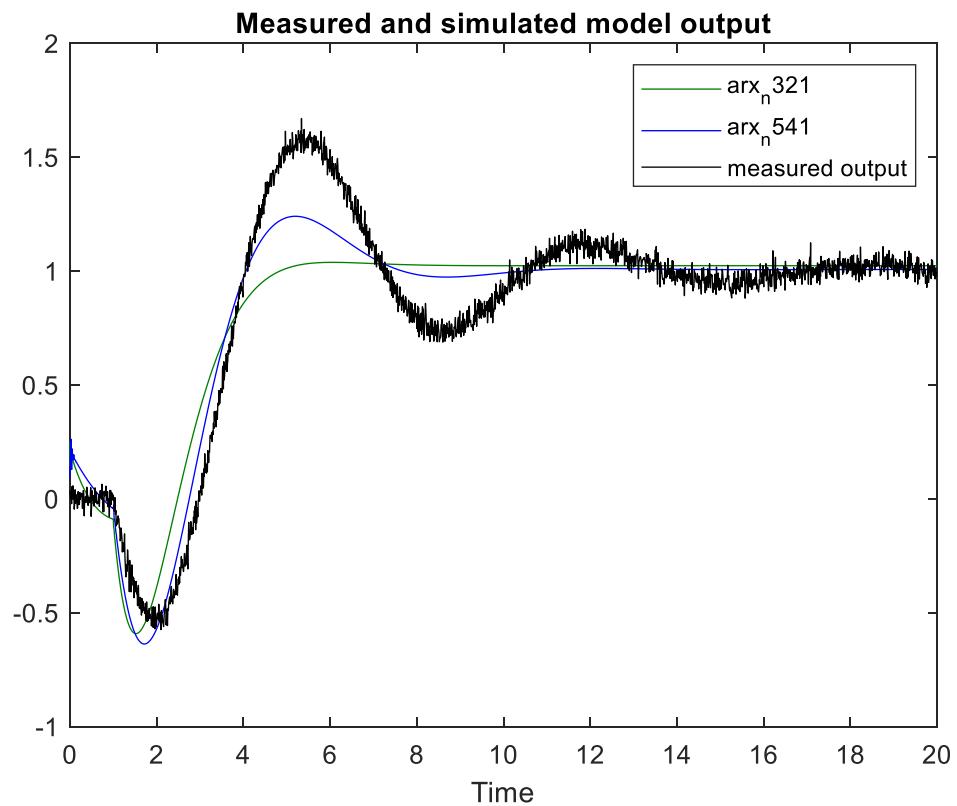
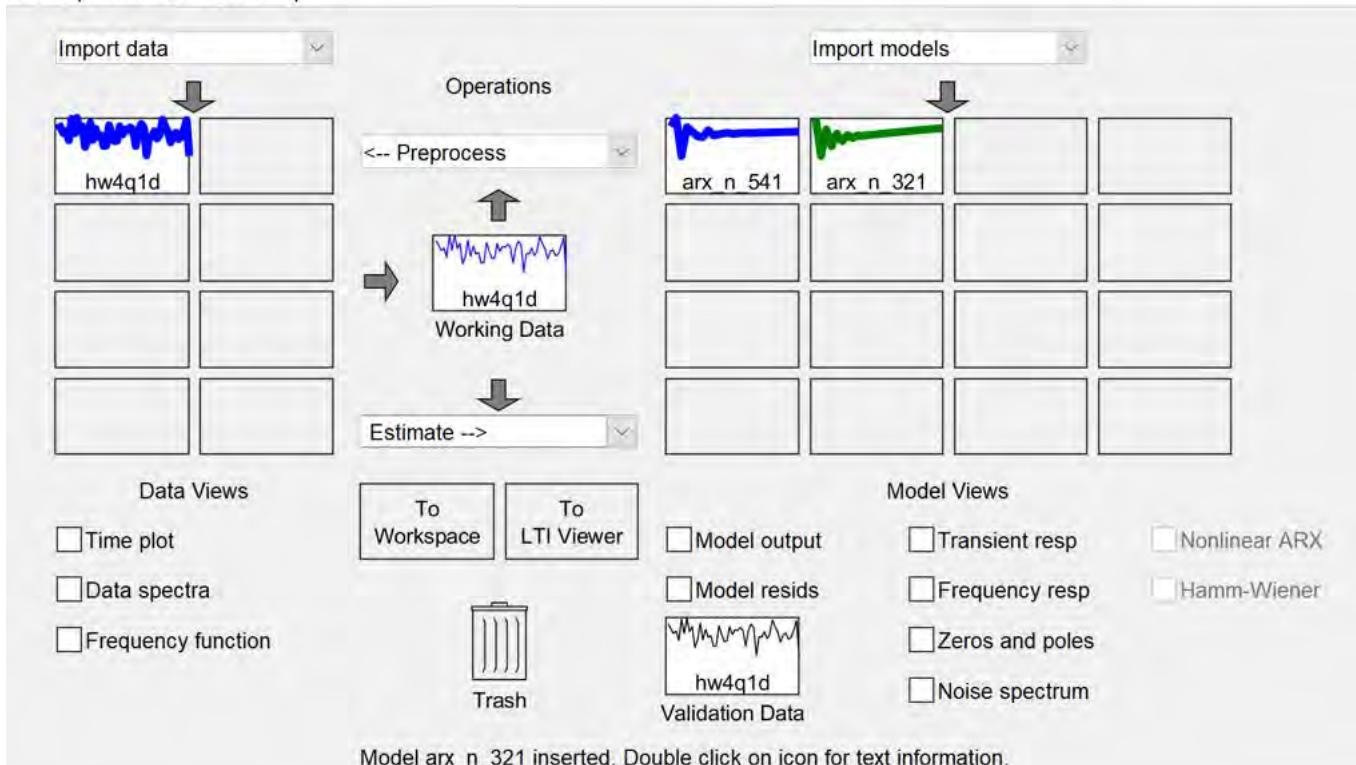
Display progress      Stop iterations

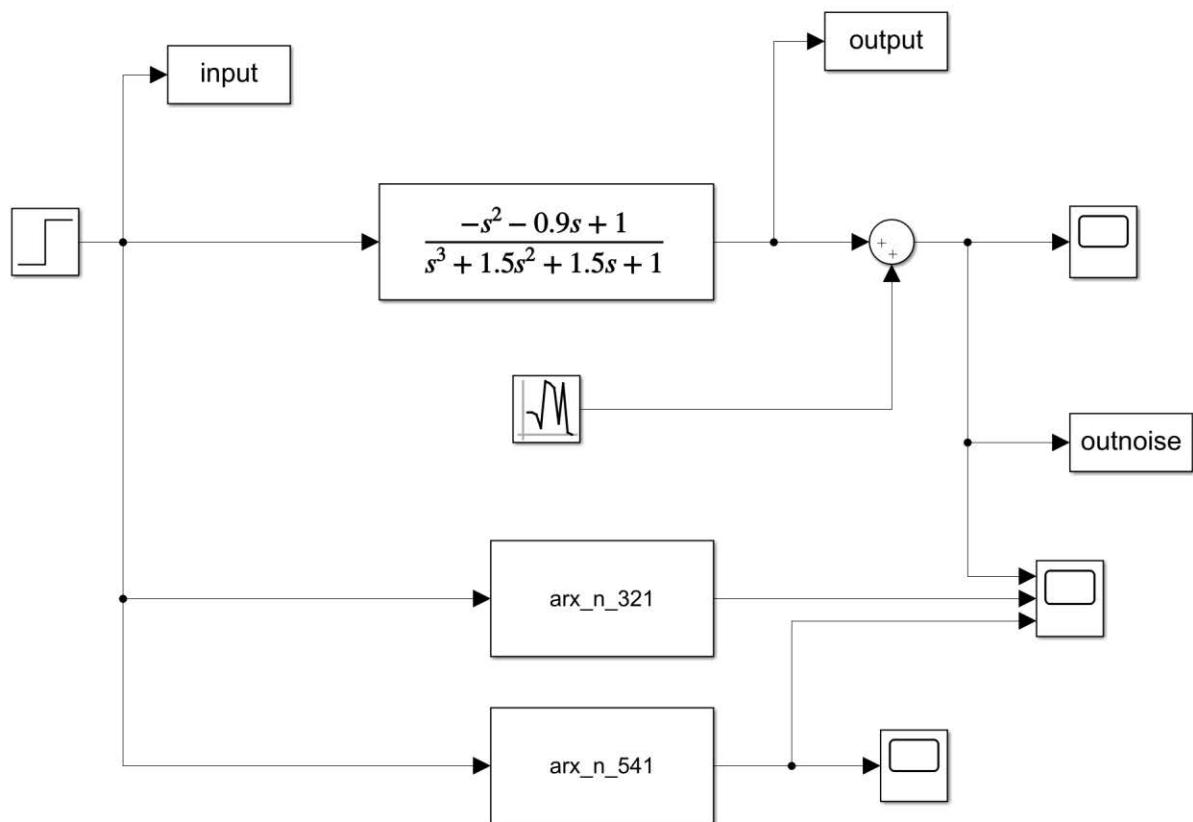
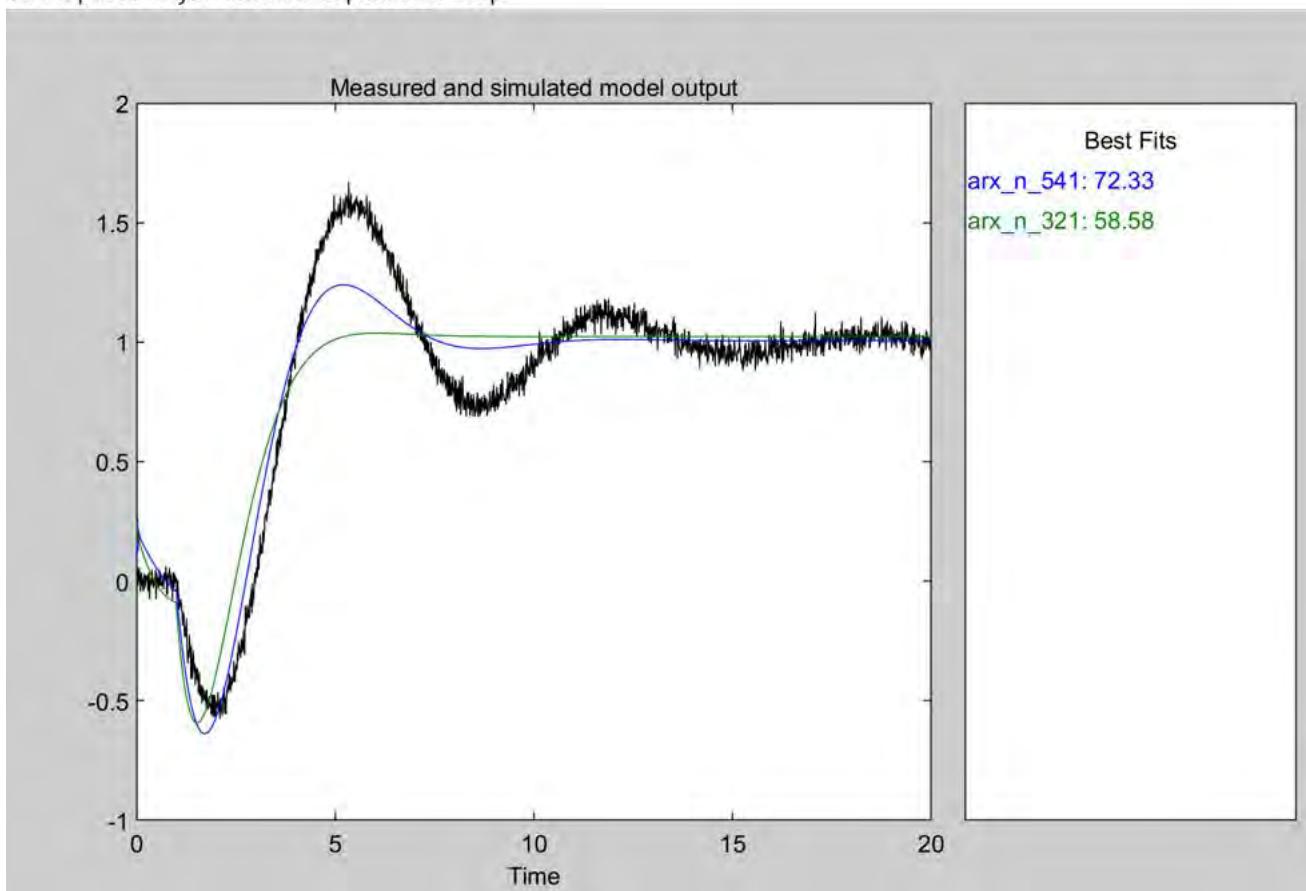
Order Selection      Order Editor...

**Buttons:**

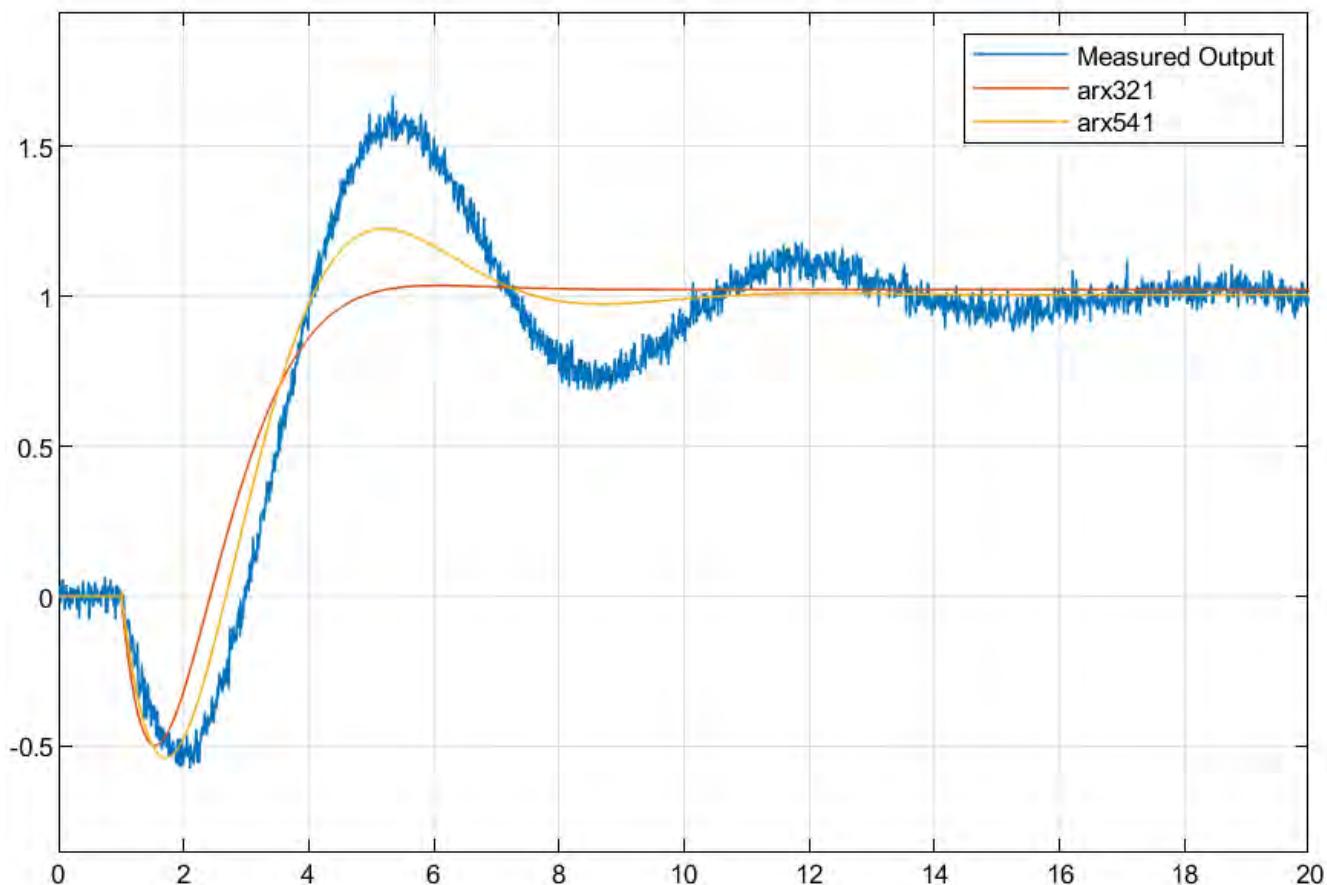
- Estimate (highlighted)
- Close
- Help

**Importing data with noise and estimating polynomial functions**



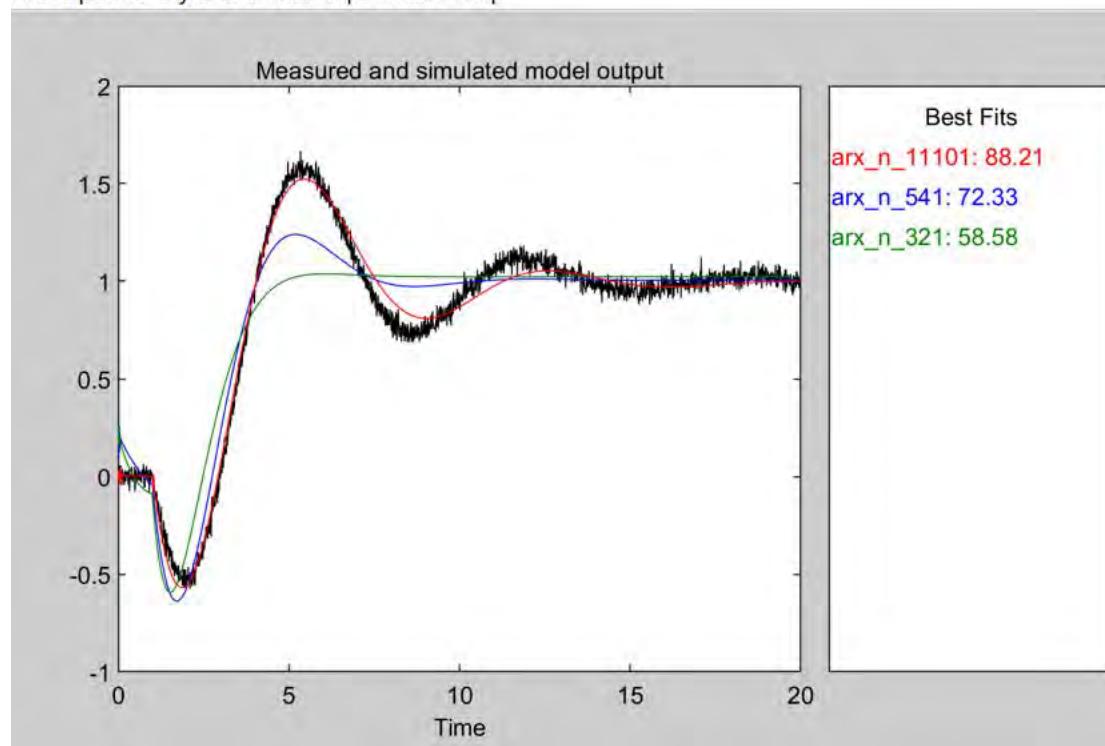


**Conclusion** - Noise significantly reduces the fitting capability of the ARX model

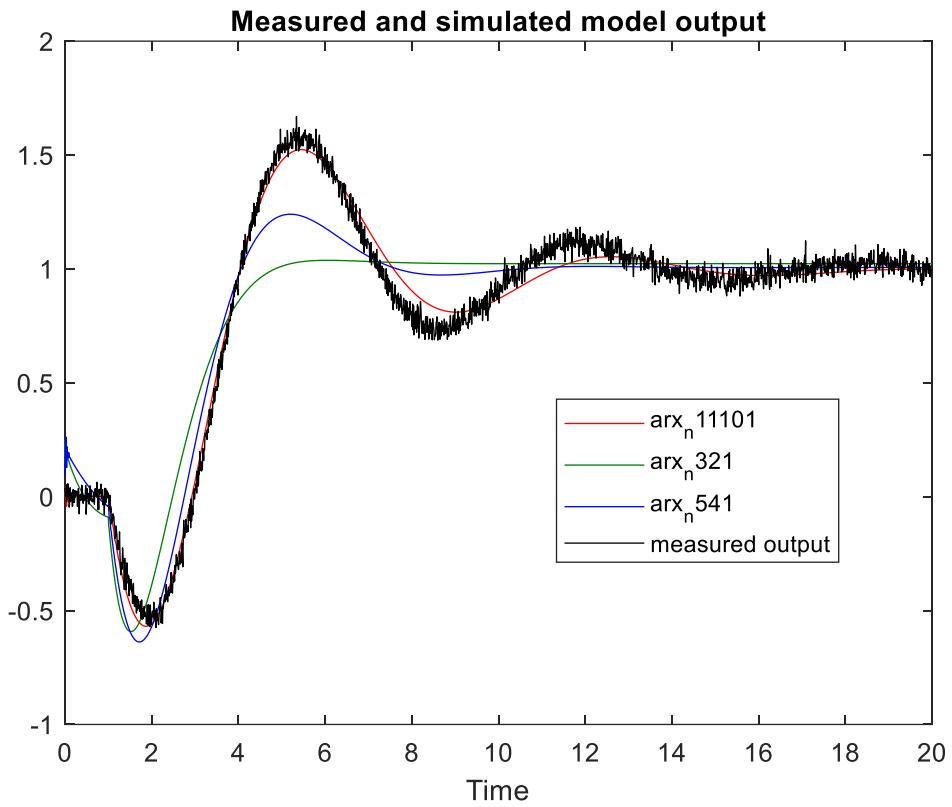


Model Output: y1

File Options Style Channel Experiment Help



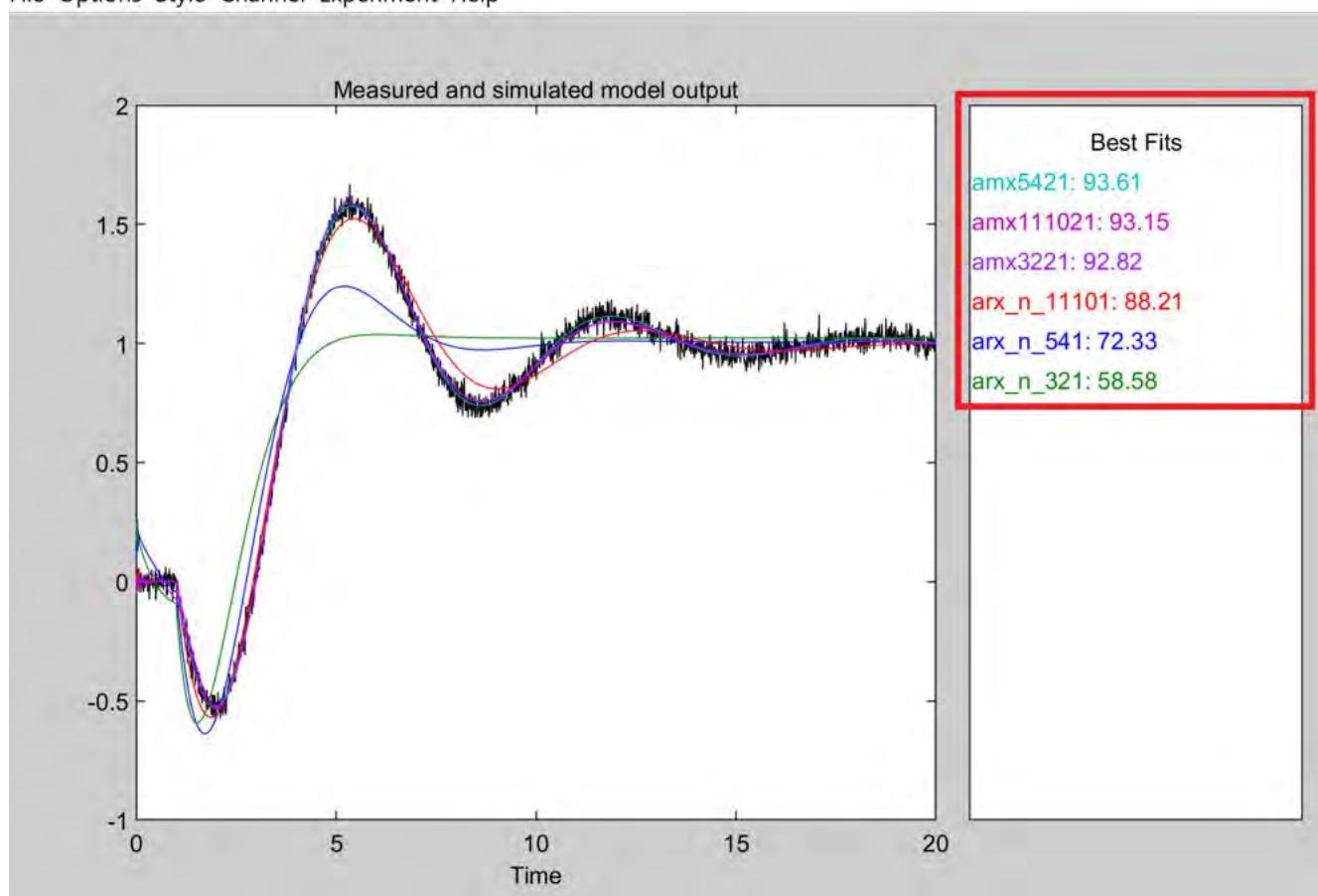
It is observed that using higher orders for estimation in the ARX model allow for better fits in case of data with noise.

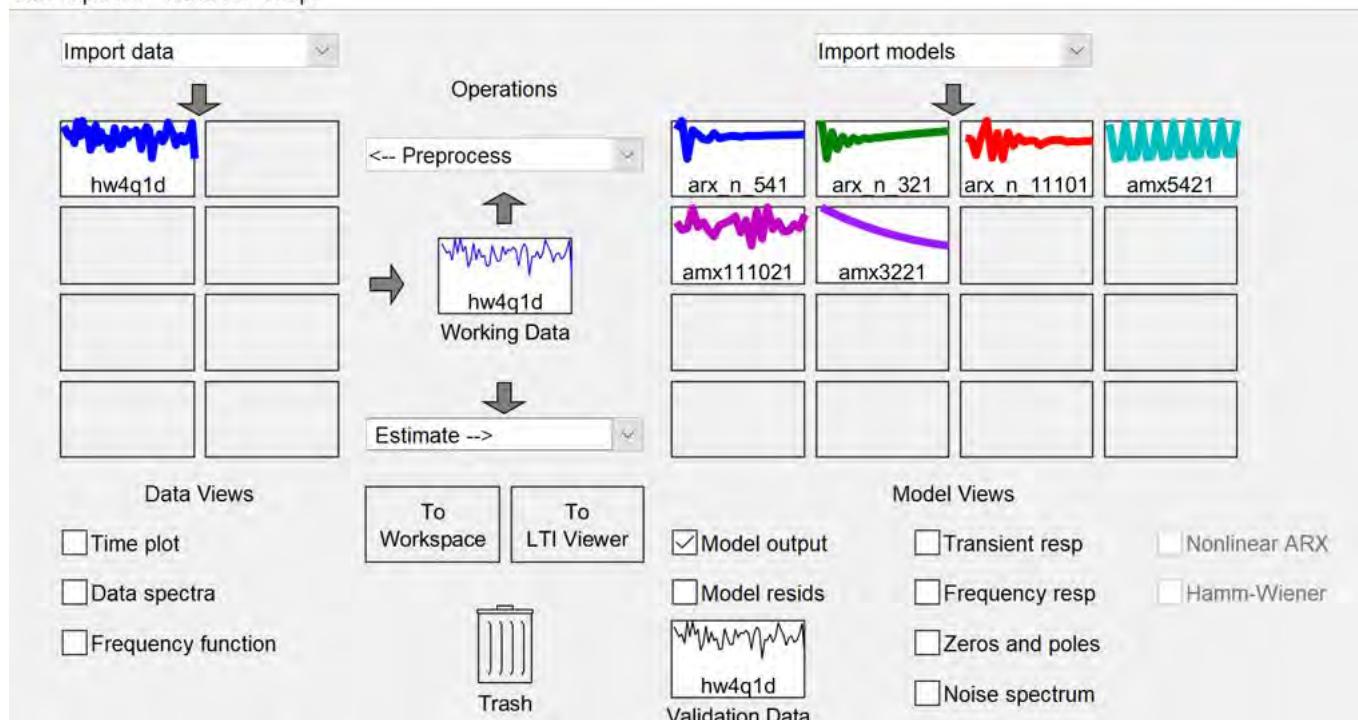


Model Output: y1

- □ ×

File Options Style Channel Experiment Help

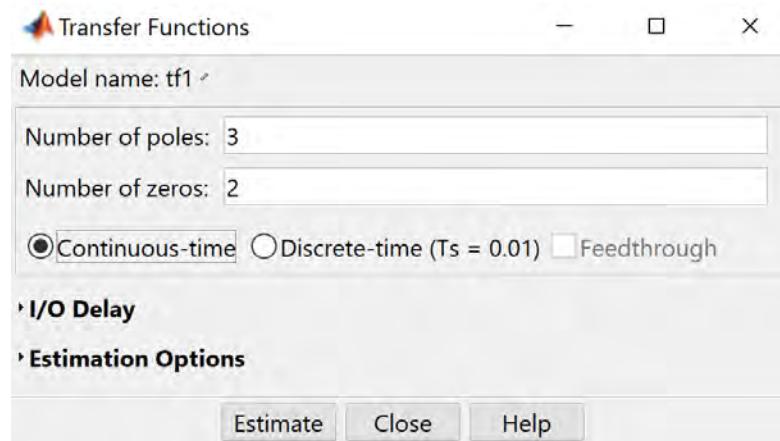




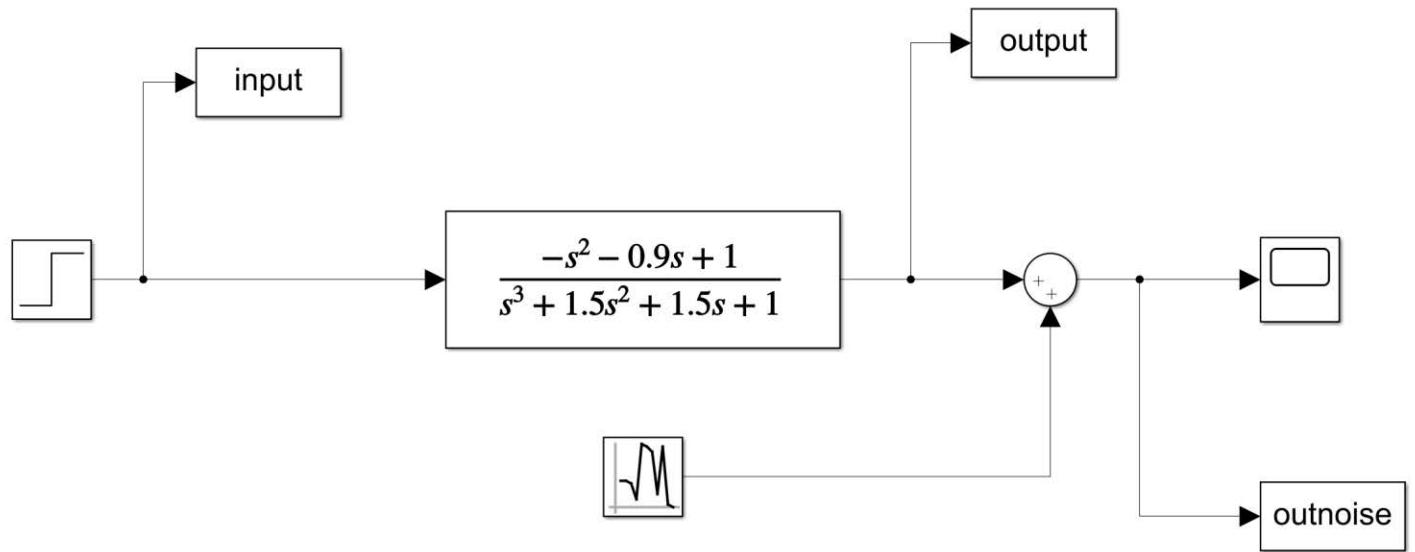
**Conclusion:** ARMAX model is better in prediction than AMX model. The ARMAX model has the best fit at amx5421, it probably starts chasing noise for higher orders of estimation which results in a lesser fit.

## Problem - 2

a.



Setting the initial values for estimation of transfer function after importing data



Simulink Block Diagram

## Plant Identification P...

Transfer Function Identification  
 Estimation data: Time domain data hw4q2  
 Data has 1 outputs, 1 inputs and 2001 samples.  
 Number of poles: 3, Number of zeros: 2  
 Initialization Method: "iv"

### Estimation Progress

Initializing model parameters...  
 Initializing using 'iv' method...  
 done.

Initialization complete.

Algorithm: Nonlinear least squares with automatically chosen line search method

Iteration	Cost	step	Norm of optimality	First-order Expected	Improvement (%) Achieved	Bisections
0	4.10284e-28	-	6.02e+16	2.05e+29	-	-
1	7.4195e-29	1.04e-12	2.55e+16	2.05e+29	01.9	0
2	7.36044e-29	1.98e-14	1.78e+16	1.48e+29	0.8	1
3	7.23355e-29	8.58e-15	1.58e+16	6.71e+28	1.72	2
4	7.09966e-29	2.55e-14	1.05e+16	4.98e+28	1.85	1
5	7.01872e-29	1.56e-14	1.01e+16	1.16e+28	1.14	0
6	6.99928e-29	1.31e-15	1.37e+16	2.51e+28	0.277	4
7	6.94623e-29	7.74e-15	9.45e+15	4.21e+28	0.472	1
8	6.86584e-29	1.55e-14	1.63e+16	2.01e+28	1.44	0
9	6.74044e-29	3.15e-16	1.65e+16	2.92e+28	1.03	5
10	6.64264e-29	2.65e-18	1.63e+16	3.1e+28	1.48	1
11	6.62577e-29	4.43e-16	1.64e+16	5.76e+28	0.253	4
12	6.62577e-29	4.41e-19	1.64e+16	5.42e+28	3.72e-05	11
13	6.62577e-29	3.31e-19	1.64e+16	5.42e+28	1.67e-05	12
14	6.62577e-29	1.44e-19	1.64e+16	5.42e+28	9.16e-06	13
15	6.62577e-29	1.44e-19	1.64e+16	5.42e+28	1.14e-06	14
16	6.62577e-29	1.5e-20	1.64e+16	5.42e+28	1.45e-07	15
17	6.62577e-29	4.51e-21	1.64e+16	5.42e+28	5.35e-08	21
18	6.62577e-29	2.23e-21	1.64e+16	5.42e+28	1.82e-08	22
19	6.62577e-29	1.13e-21	1.64e+16	5.42e+28	9.08e-09	23
20	6.62577e-29	5.63e-22	1.64e+16	5.42e+28	4.54e-09	24

Estimating parameter covariance...  
 done.

### Result

Termination condition: Near (local) minimum, (norm(g) < tol)..  
 Number of iterations: 20, Number of function evaluations: 221

Status: Estimated using TTEST  
 Fit to estimation data: 100%, FPE: 6.69548e-29

## Estimation for Continuous transfer function

## Plant Identification P...

Transfer Function Identification  
 Estimation data: Time domain data hw4q2  
 Data has 1 outputs, 1 inputs and 2001 samples.  
 Number of poles: 3, Number of zeros: 2

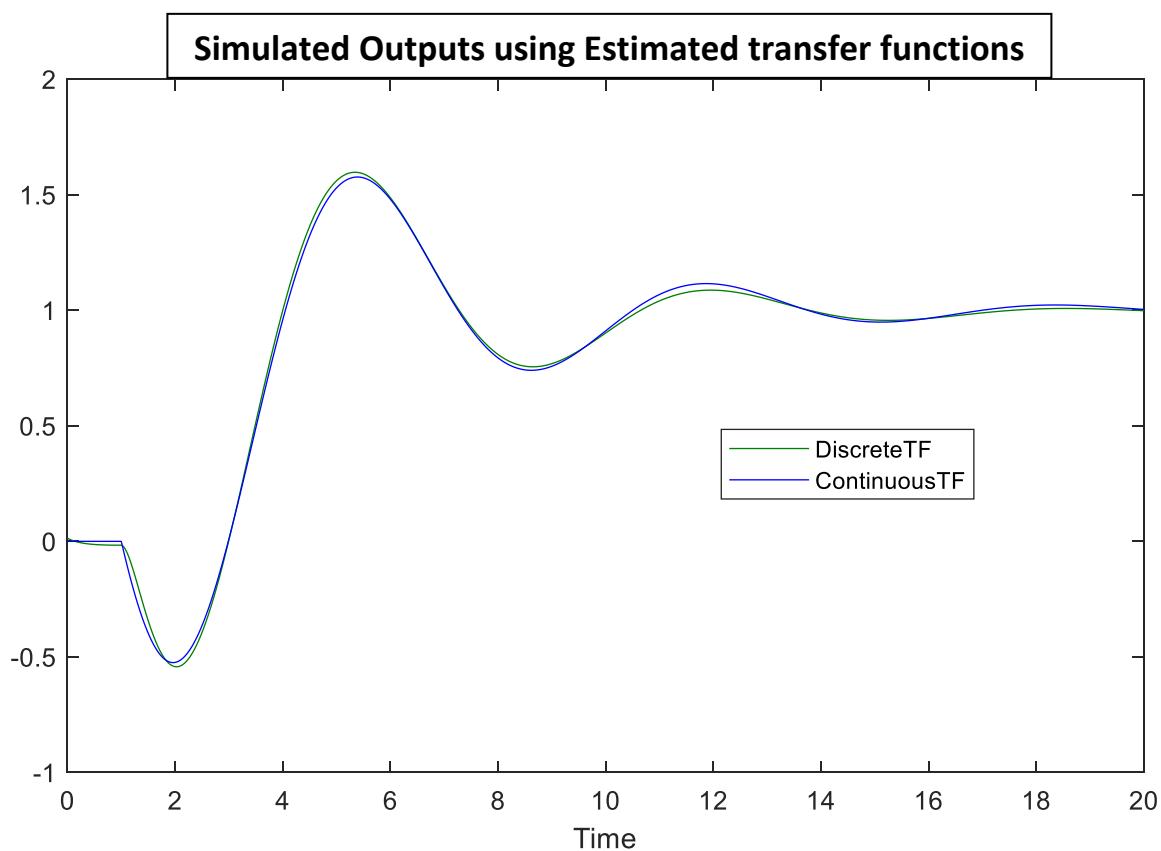
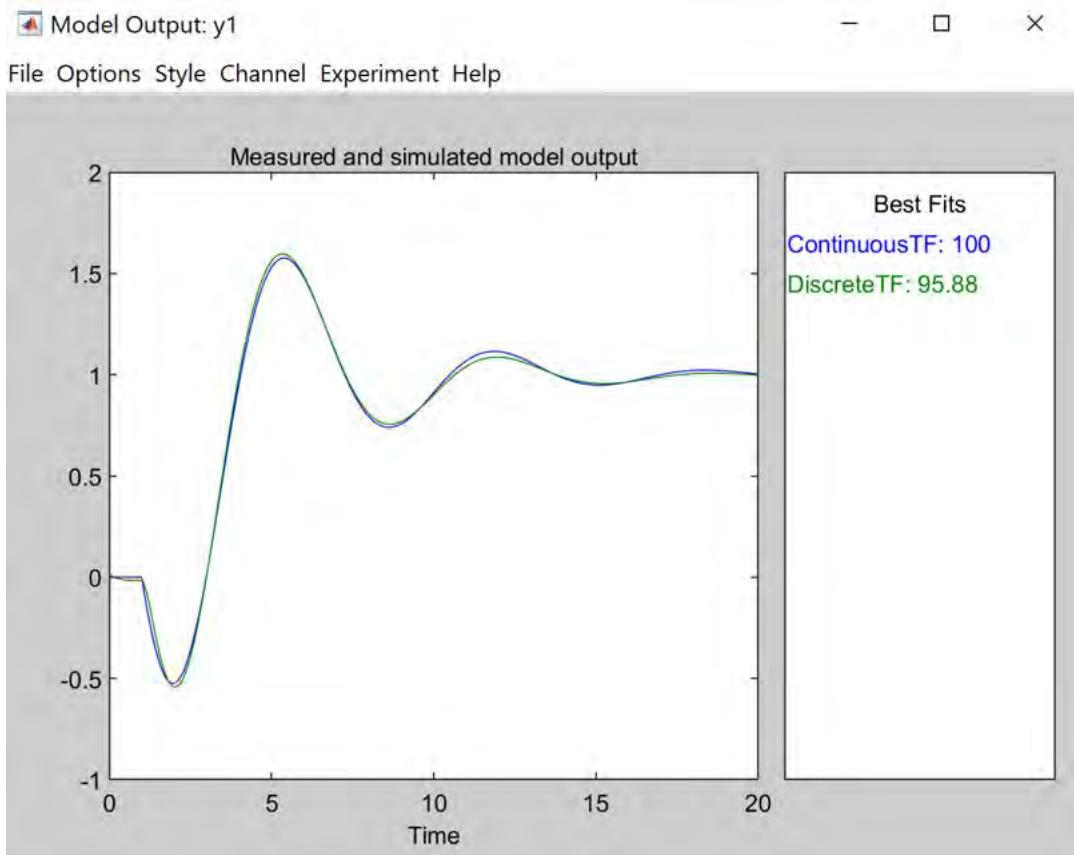
### Estimation Progress

Initializing model parameters...  
 Initializing using 'arx' method...  
 Initialization complete.

Algorithm: Nonlinear least squares with automatically chosen line search method

Iteration	Cost	step	Norm of optimality	First-order Expected	Improvement (%) Achieved	Bisections
0	0.00042201	-	3.86e+05	95.3	-	-
1	0.000402784	1.98	1.08e+06	95.3	80.5	4
2	0.000129131	0.241	6.4e+05	90.8	47.9	2
3	0.0000455207	0.0241	2.07e+05	65.9	59.4	1
4	0.0000455262	0.0246	1.48e+05	13	13.2	0
5	0.0000452313	0.000204	2.56e+04	0.606	0.646	0
6	0.0000452282	0.000365	2.1e+03	0.00649	0.00677	0

## Estimation for discrete transfer function



## b. MATLAB Commands for converting to discrete form for comparison

```
>> C=tf([-1 -0.9 1],[1 1.5 1.5 1])
```

C =

$$-\frac{s^2 - 0.9 s + 1}{s^3 + 1.5 s^2 + 1.5 s + 1}$$

$$s^3 + 1.5 s^2 + 1.5 s + 1$$

Continuous-time transfer function.

```
>> D=c2d(C,0.01)
```

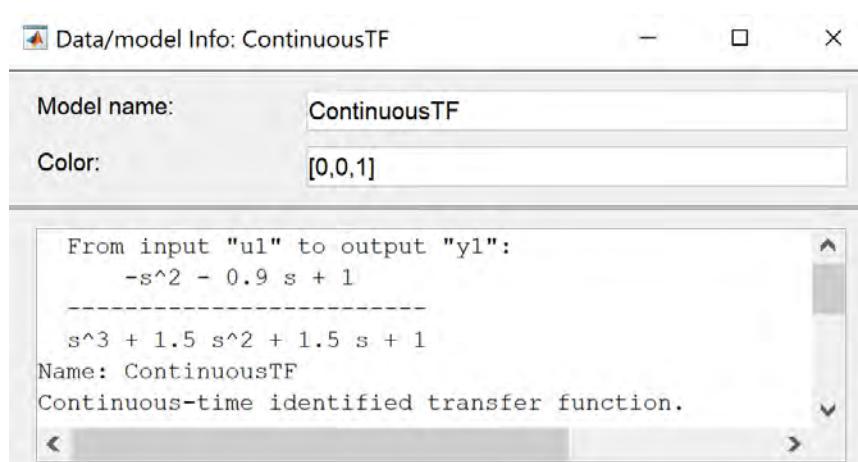
D =

$$-\frac{0.00997 z^2 + 0.01985 z - 0.00988}{z^3 - 2.985 z^2 + 2.97 z - 0.9851}$$

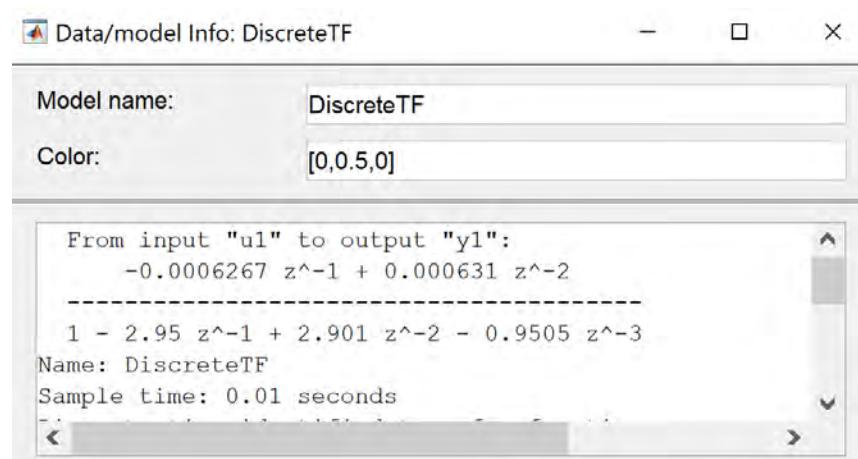
$$z^3 - 2.985 z^2 + 2.97 z - 0.9851$$

Sample time: 0.01 seconds

Discrete-time transfer function.

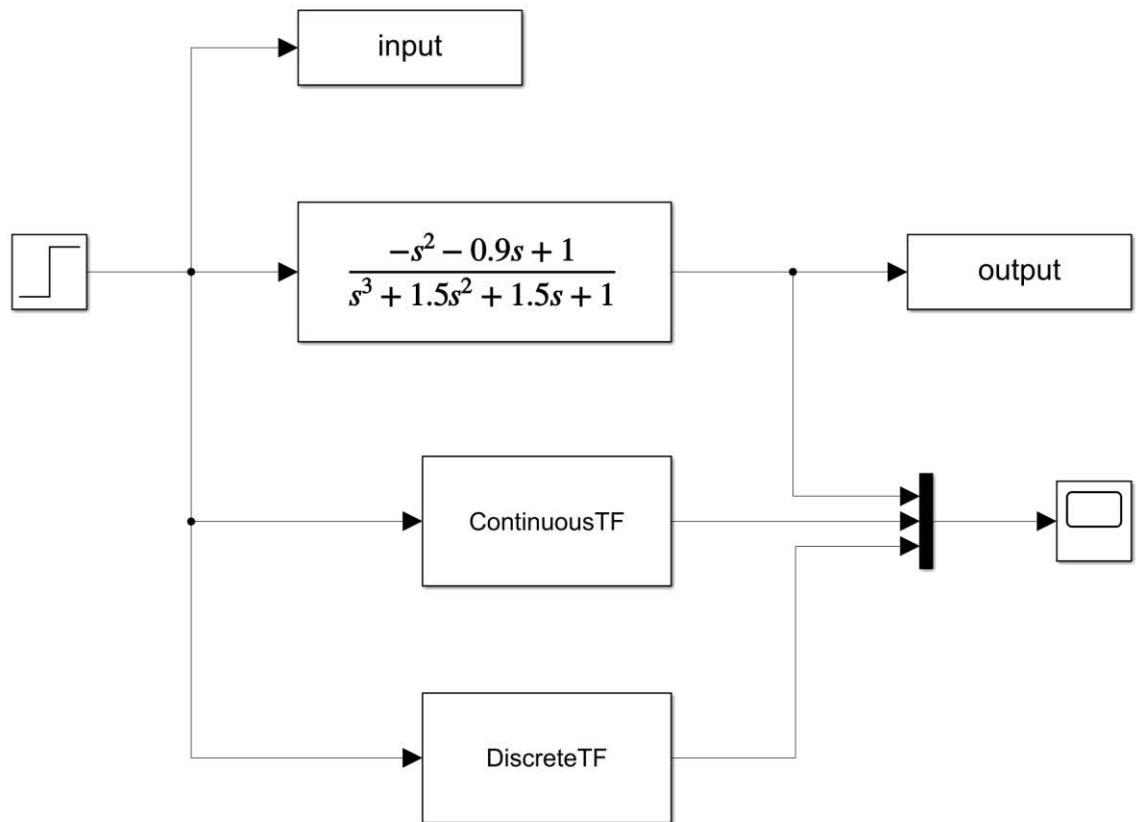


Continuous Transfer Function

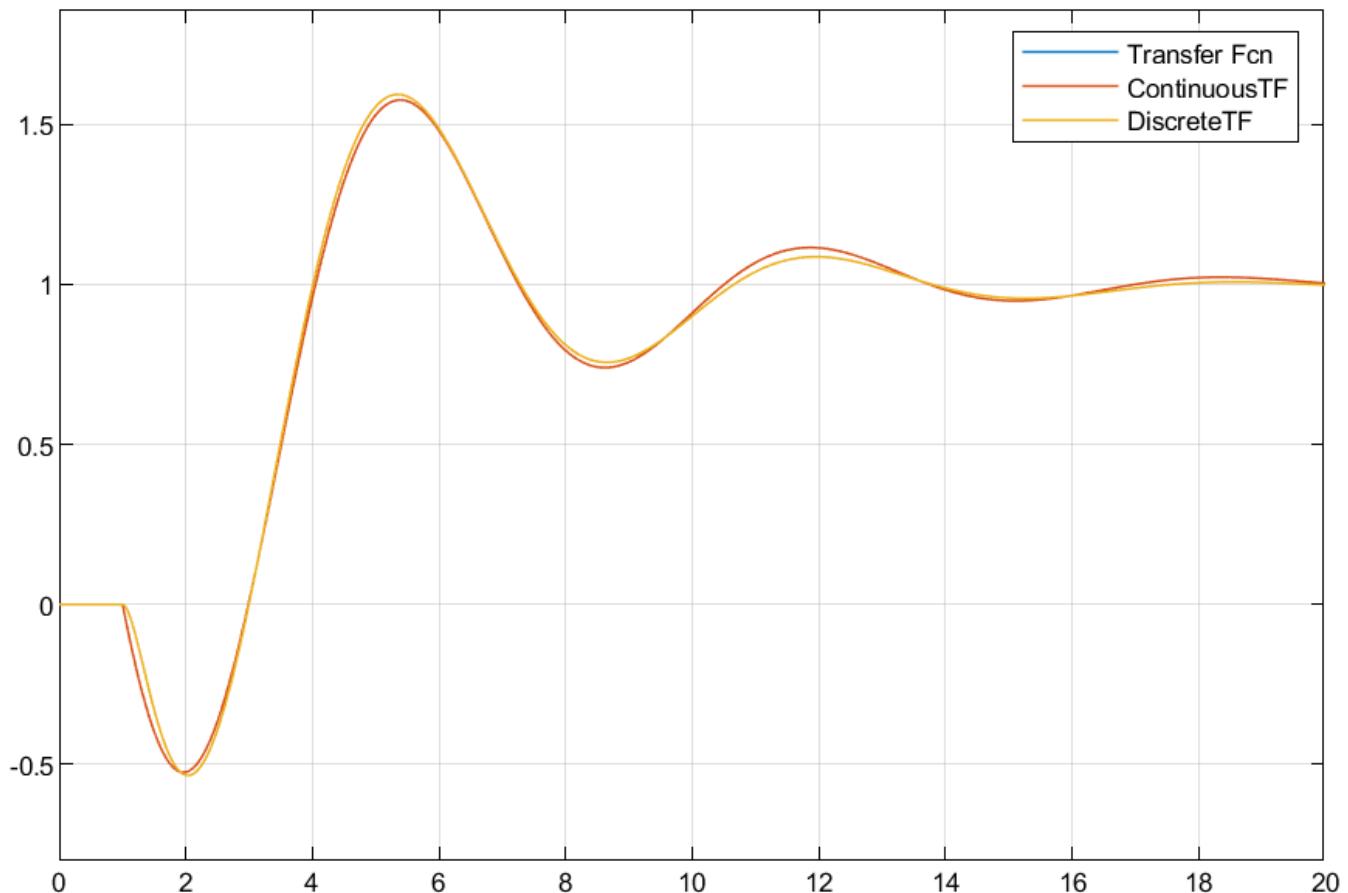


Discrete Transfer function

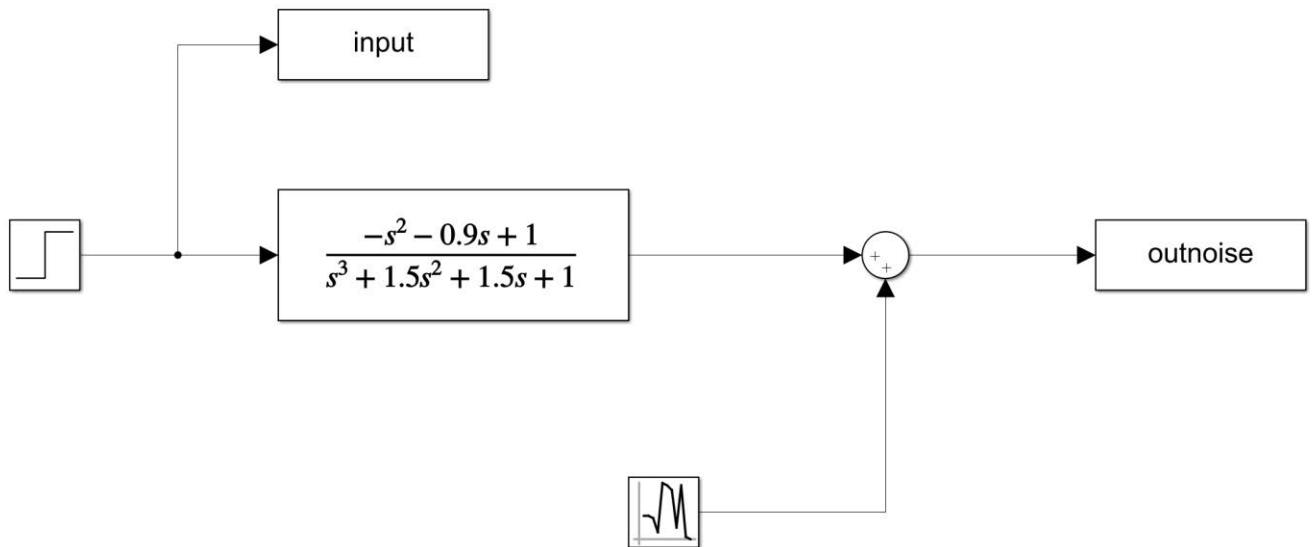
**Conclusion** - The estimated continuous transfer function is exactly the same as the original system values i.e., poles and zeros are same. Denominator for the estimated discrete transfer function is very similar to A(z) but the numerator is different.



c.

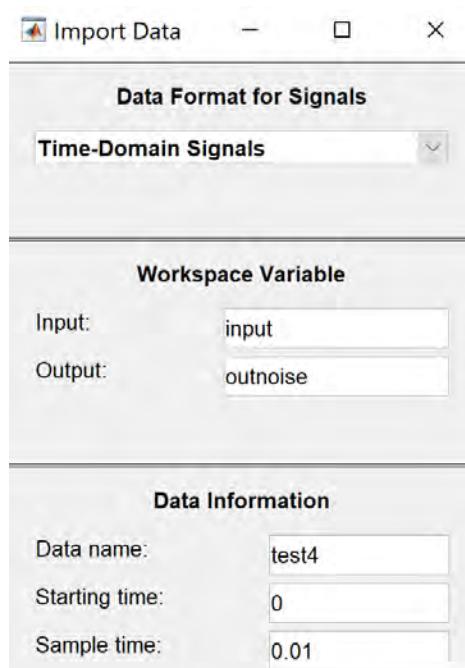


Comparison plots derived from Scope

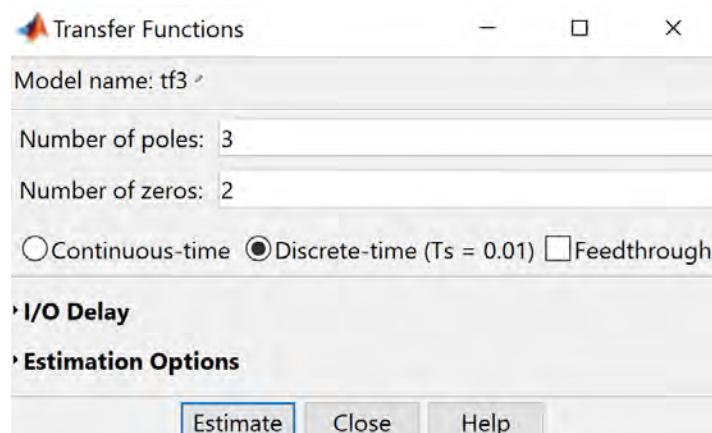
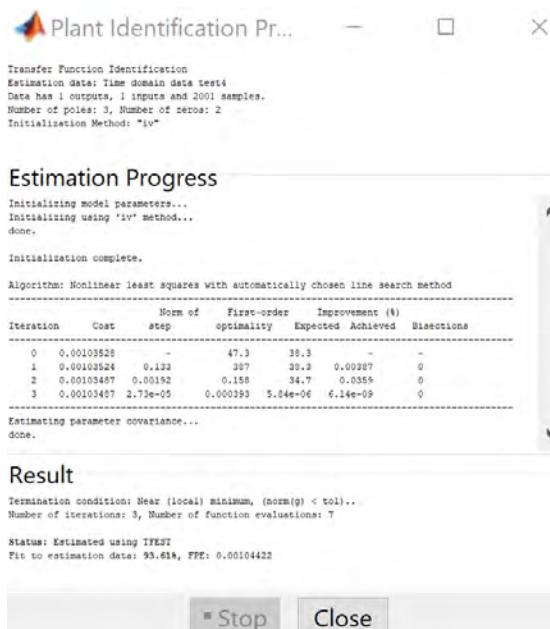
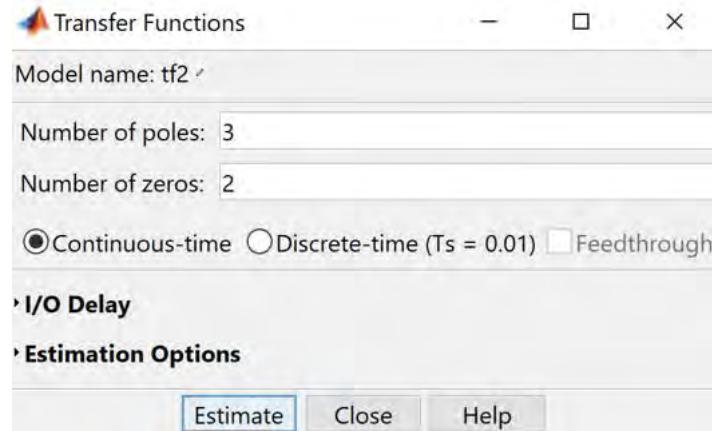


d.

### Simulink Block Diagram with Noise



Importing data with noise from workspace



## Plant Identification Pr...

```
Transfer Function Identification
Estimation data: Time domain data test4
Data has 1 outputs, 1 inputs and 2001 samples.
Number of poles: 3, Number of zeros: 2
Initialization Method: "iv"
Transfer Function Identification
Estimation data: Time domain data test4
Data has 1 outputs, 1 inputs and 2001 samples.
Number of poles: 3, Number of zeros: 2
```

### Estimation Progress

```
Initializing model parameters...
Initializing using 'iv' method...
done.

Initialization complete.

Algorithm: Nonlinear least squares with automatically chosen line search method
-----  
Iteration      Cost      step      Norm of First-order      Optimality      Improvement (%)      Expected      Achieved      Bisections  
-----  
0   0.00103828    -        47.3      38.3      -          -          -  
1   0.00103824  0.133      38.7      38.3  0.00387    0  
2   0.00103827  0.00152    0.158      34.7  0.0359     0  
3   0.00103827  3.73e-05  0.000393  5.84e-04  8.14e-05    0  
-----  
Estimating parameter covariance...
done.  
Initialization complete.

Algorithm: Nonlinear least squares with automatically chosen line search method
-----  
Iteration      Cost      step      Norm of First-order      Optimality      Improvement (%)      Expected      Achieved      Bisections  
-----  
0   25.0809    -        9.07e+06      99.6      -          -          -  
1   1.28049   0.977  8.95e+05      99.6    94.9      1  
2   0.12797   0.241  7.06e+04     91.1      90       0  
3   0.105059   0.409  1.30e+05     29.2     17.9      3  
4   0.0710687   0.341  3.36e+05      35      32.4      3  
5   0.0552307   0.121  4e+05       47.0     16.6      3  
6   0.0281728   0.037  9.00e+03     52.3     52.4      1  
7   0.0234144   0.002  2.41e+05     75.9     76.2      0  
8   0.02348141   0.220  1.738e+04     67.9     58.6      0  
9   0.02342173   0.61  2.79e+04     12.2     6.46      0  
10  0.0219166   2.07  4.84e+04     8.98     2.23      4  
11  0.02111192   0.95  7.79e+04     14.5     3.64      4  
12  0.00158059   0.829  7.31e+05     13.2     28.6      2  
13  0.00138521   0.0446  6.43e+05     5.27     8.15      0  
14  0.00137043   0.00443  4.69e+04     0.875    1.07      0  
15  0.00136524   0.00235  2.04e+03     0.0633  0.0866      0  
16  0.00136909  0.000783   664     0.00776  0.0107      0  
17  0.00136907  0.000281   238     0.00103  0.0014      0  
-----
```

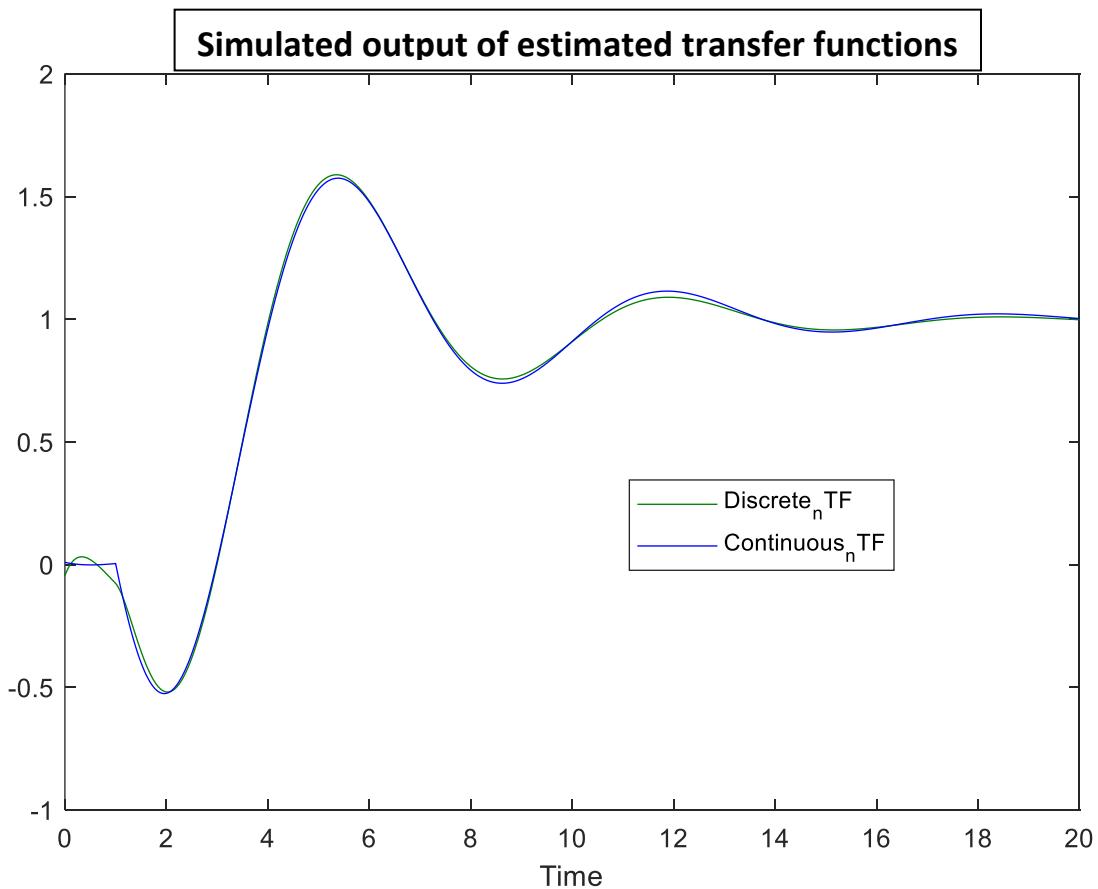
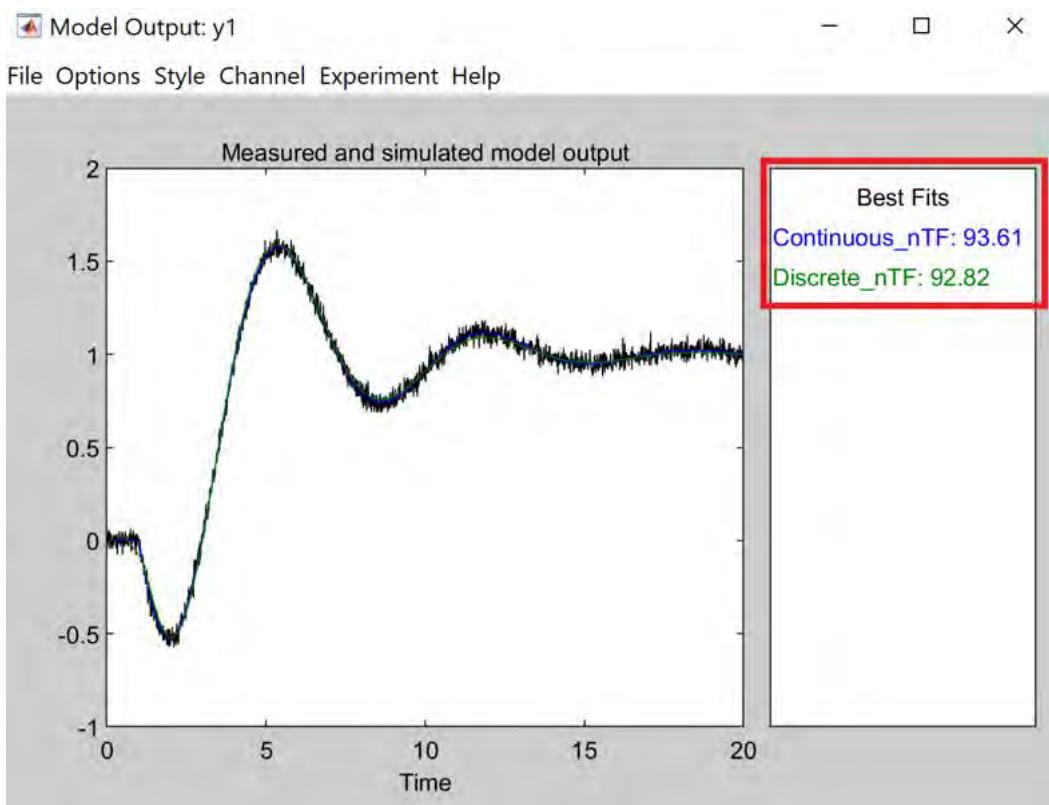
### Result

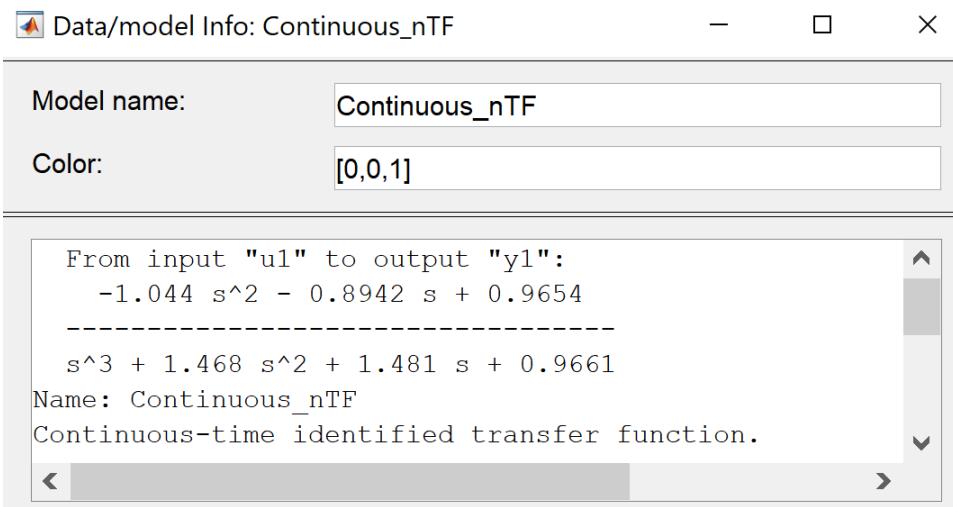
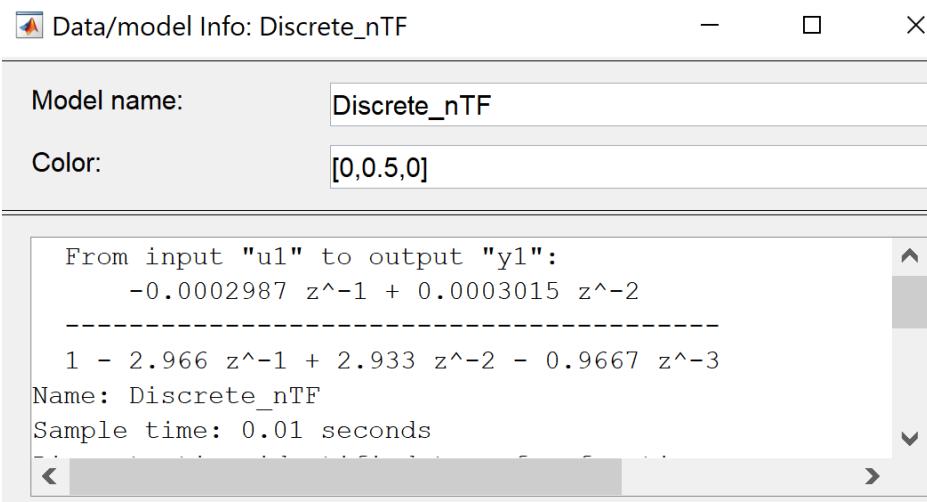
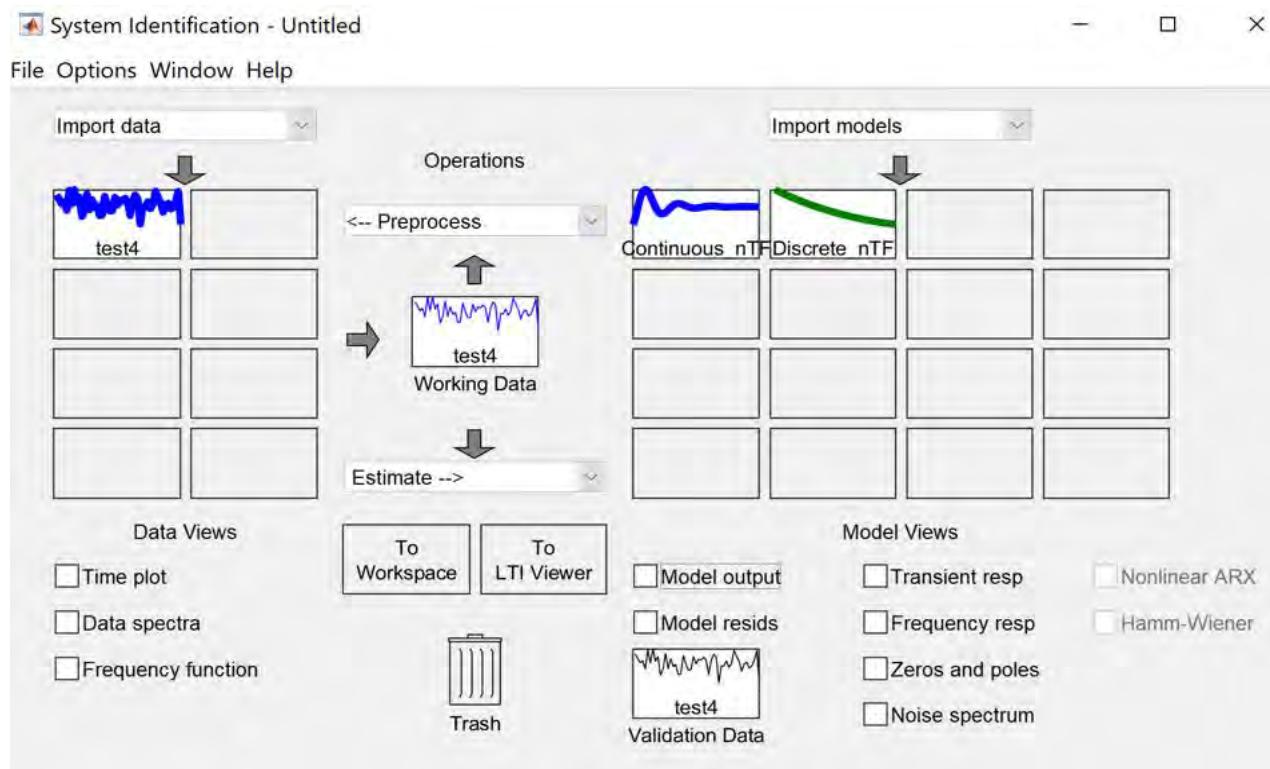
```
Termination condition: Near (local) minimum, (norm(g) < tol)..
Number of iterations: 3, Number of function evaluations: 7

Status: Estimated using TTEST
Fit to estimation data: 93.61%, FPE: 0.00104422
Termination condition: Near (local) minimum, (norm(g) < tol)..
Number of iterations: 17, Number of function evaluations: 56

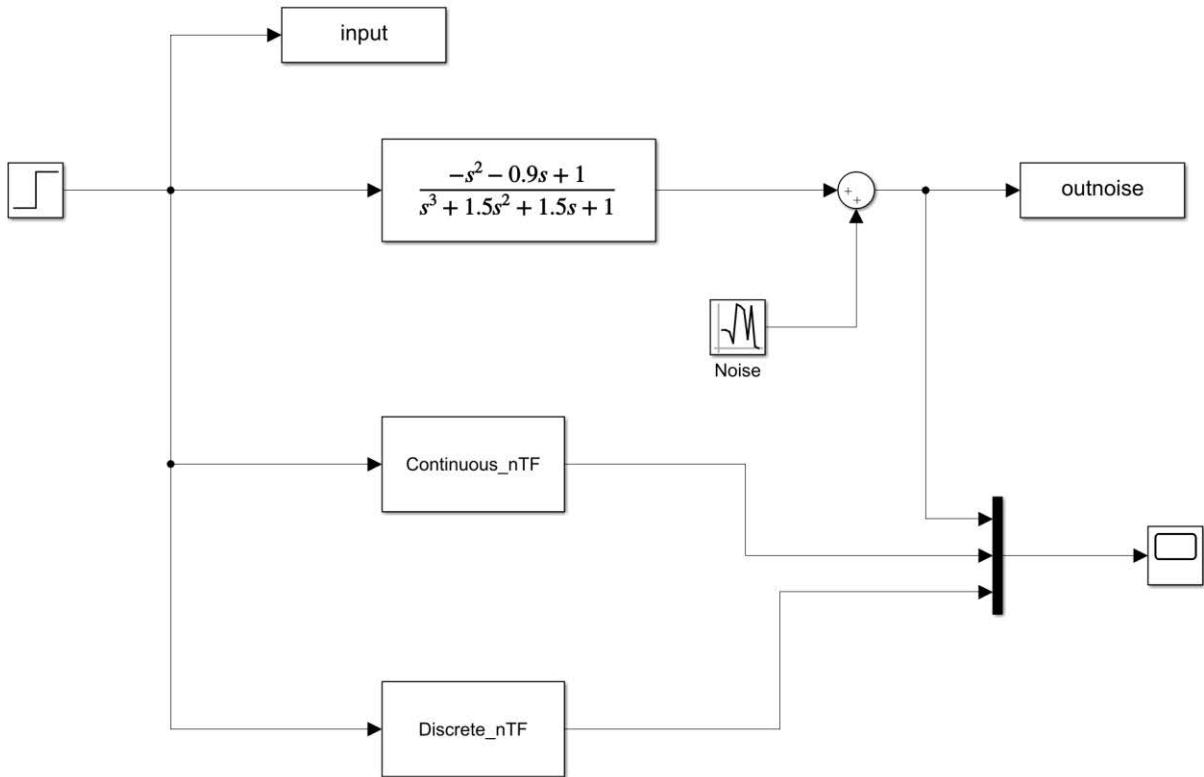
Status: Estimated using TTEST
Fit to estimation data: 92.82%, FPE: 0.00131786
```

## Setting parameters for estimation and estimation process data

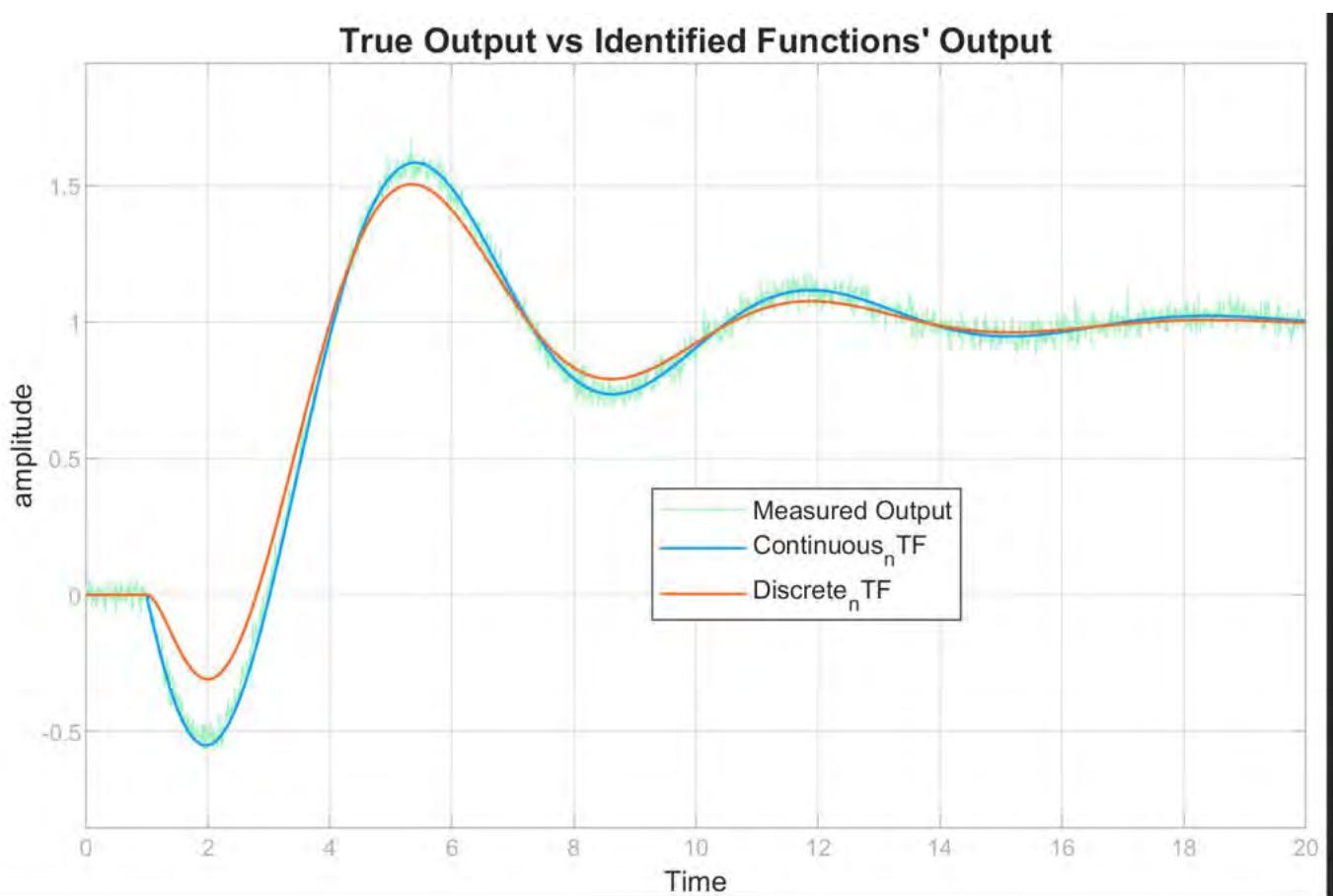


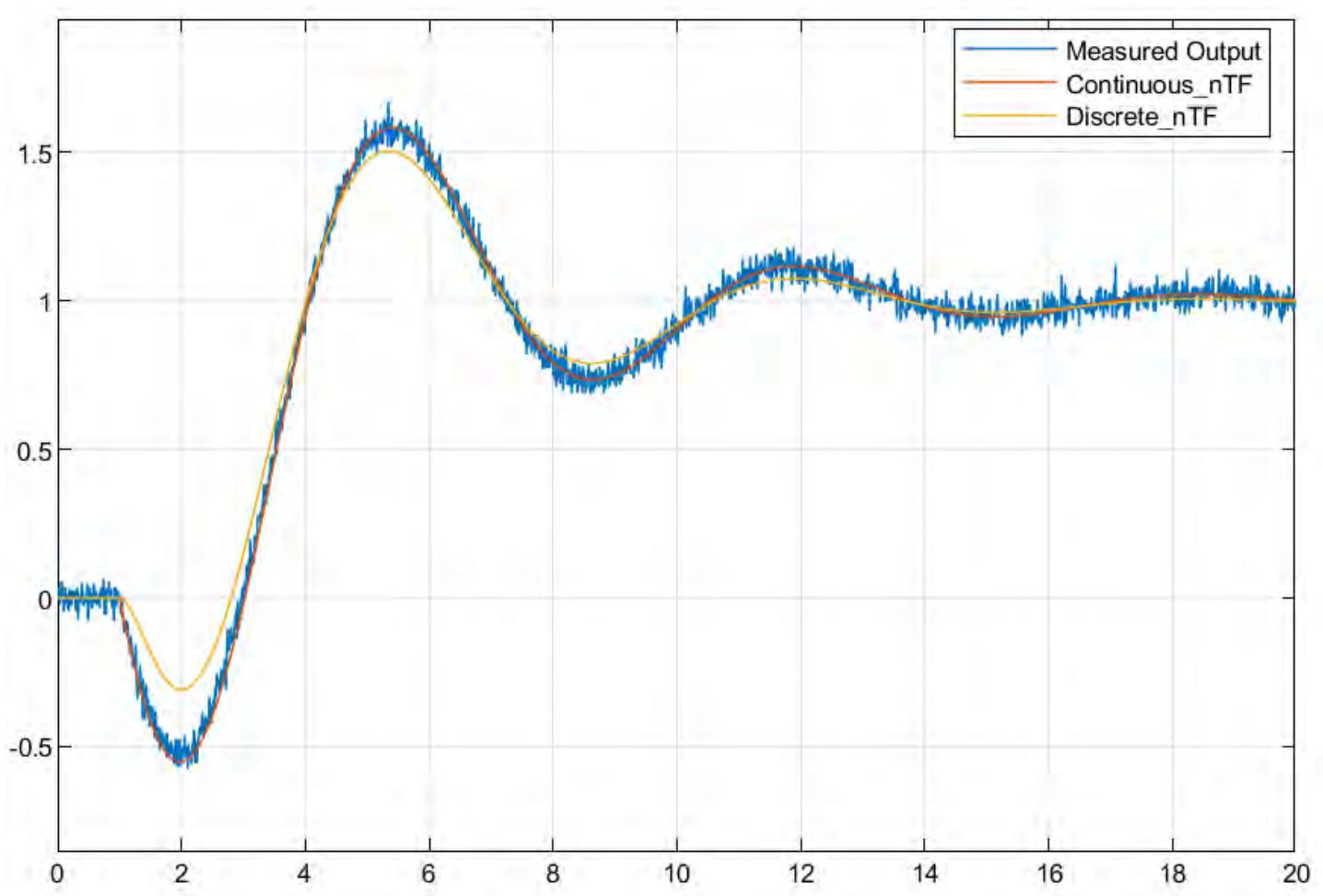


**Comparison of poles and zeros of the identified transfer function with the true values**



**Simulink Block Diagram**

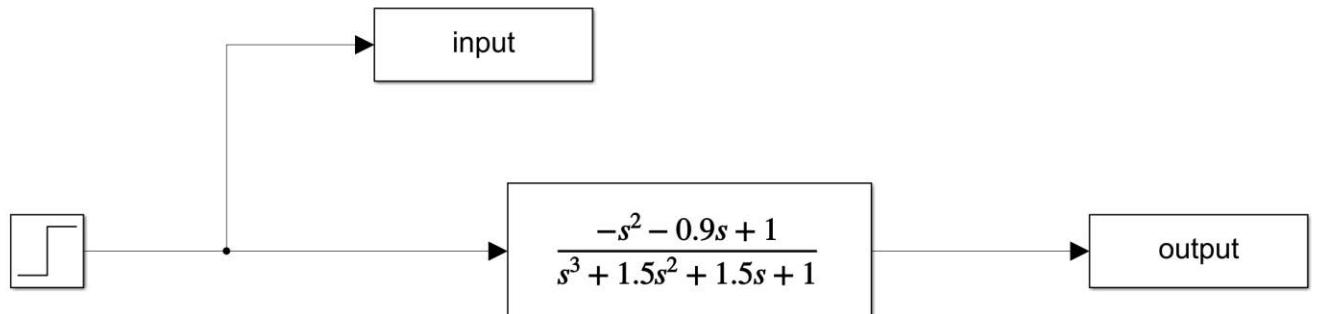




**Conclusion** – The continuous transfer function is similar to the original transfer function i.e., the poles and zeros are similar which can be confirmed using different commands such as `damp` or `pole`. The denominator of the discrete transfer function derived from the original transfer function is also similar to  $A(z)$  as observed in the previous parts.

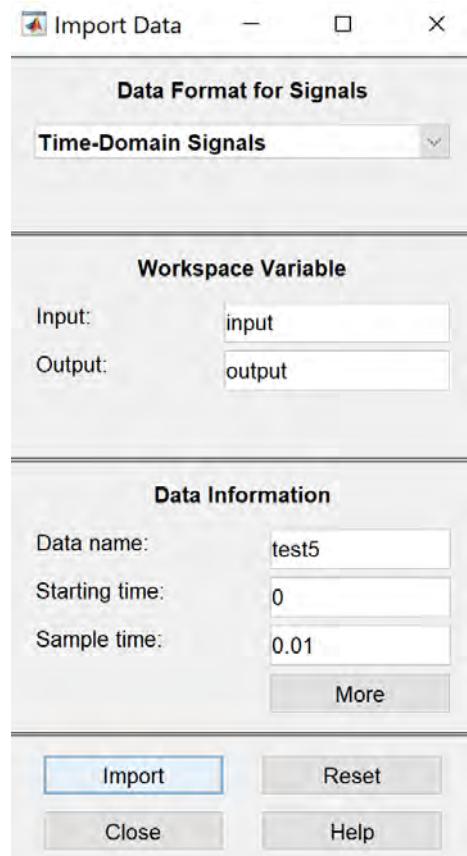
The system identification toolbox does a good job at calculating the transfer functions despite noise. It has an efficiency similar to the ARMAX model used in the previous problem. The continuous transfer function and discrete transfer function denominator are very similar to the original system.

## Problem - 3

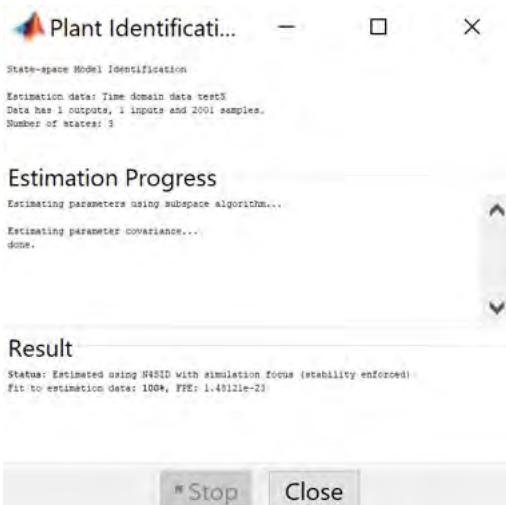
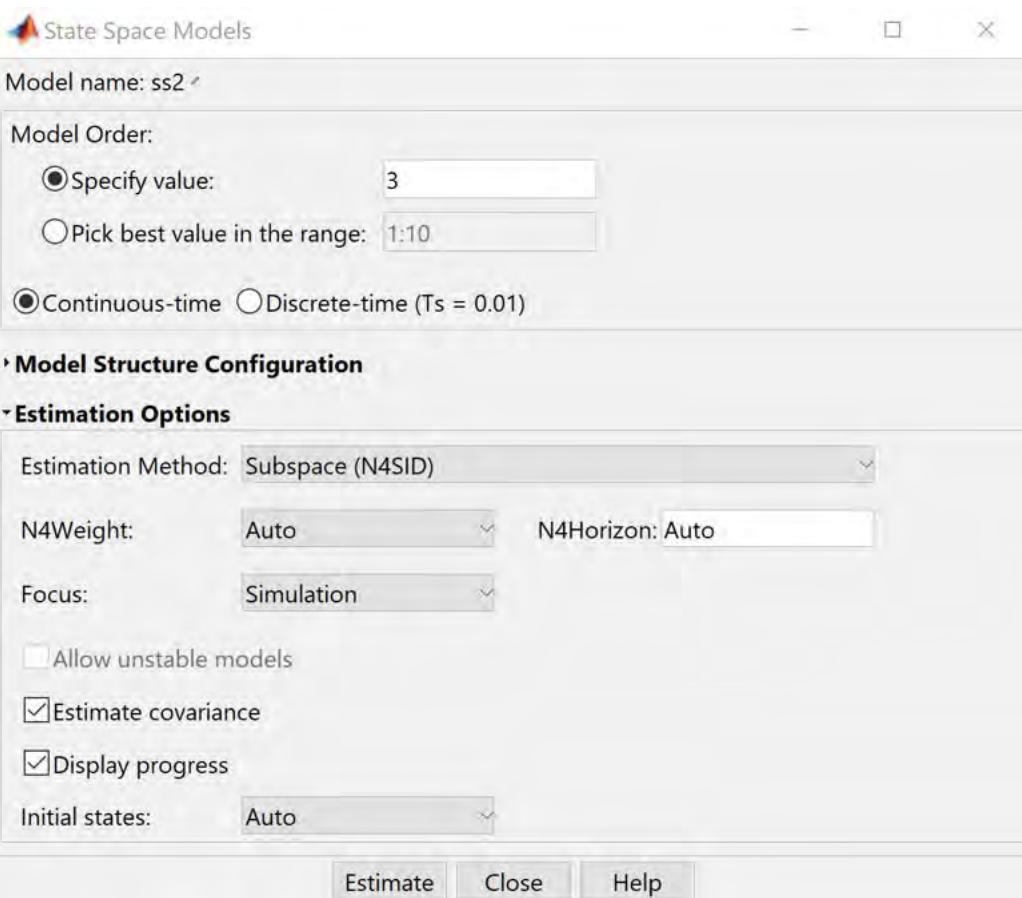


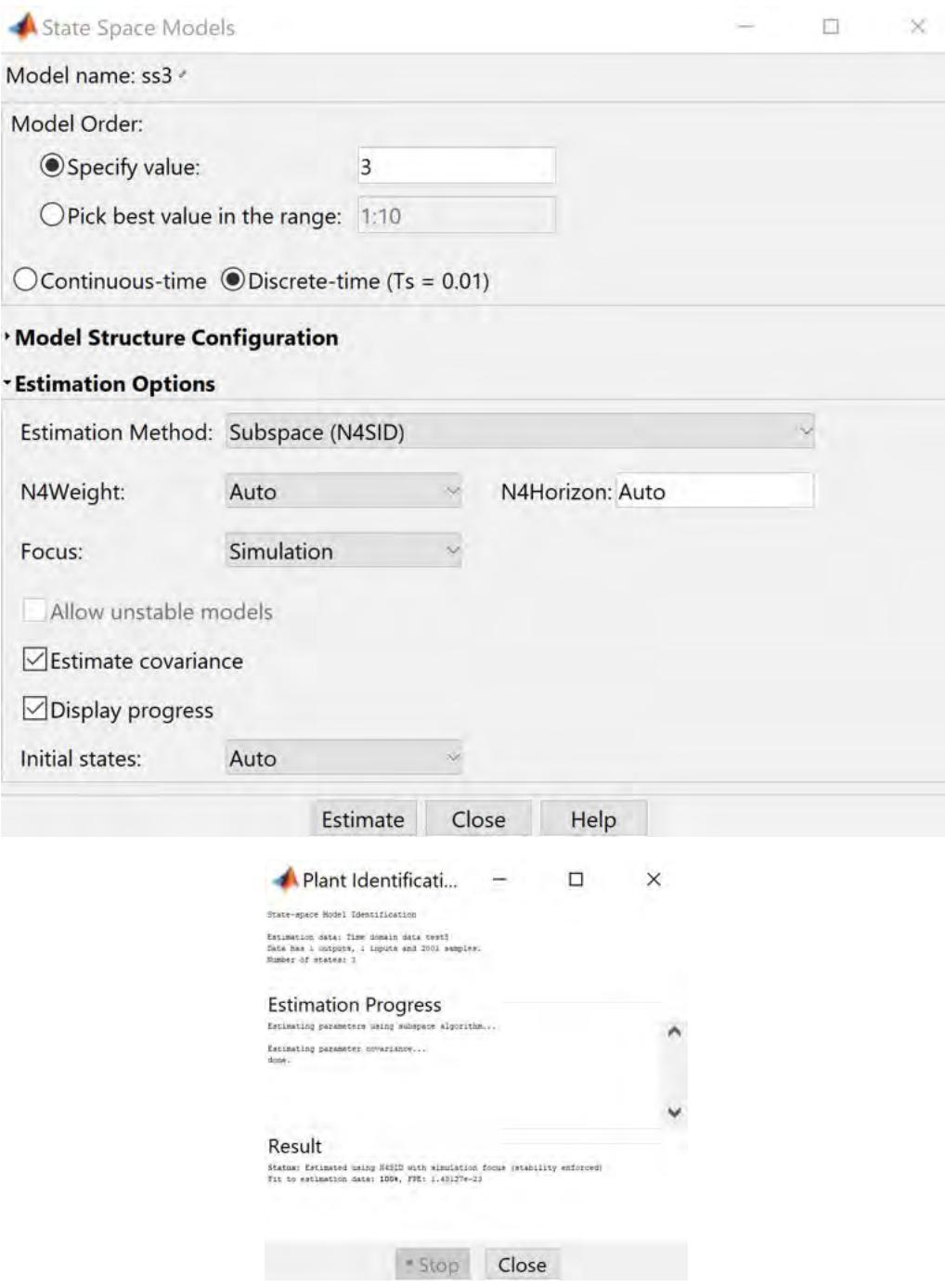
a.

Simulink Block Diagram

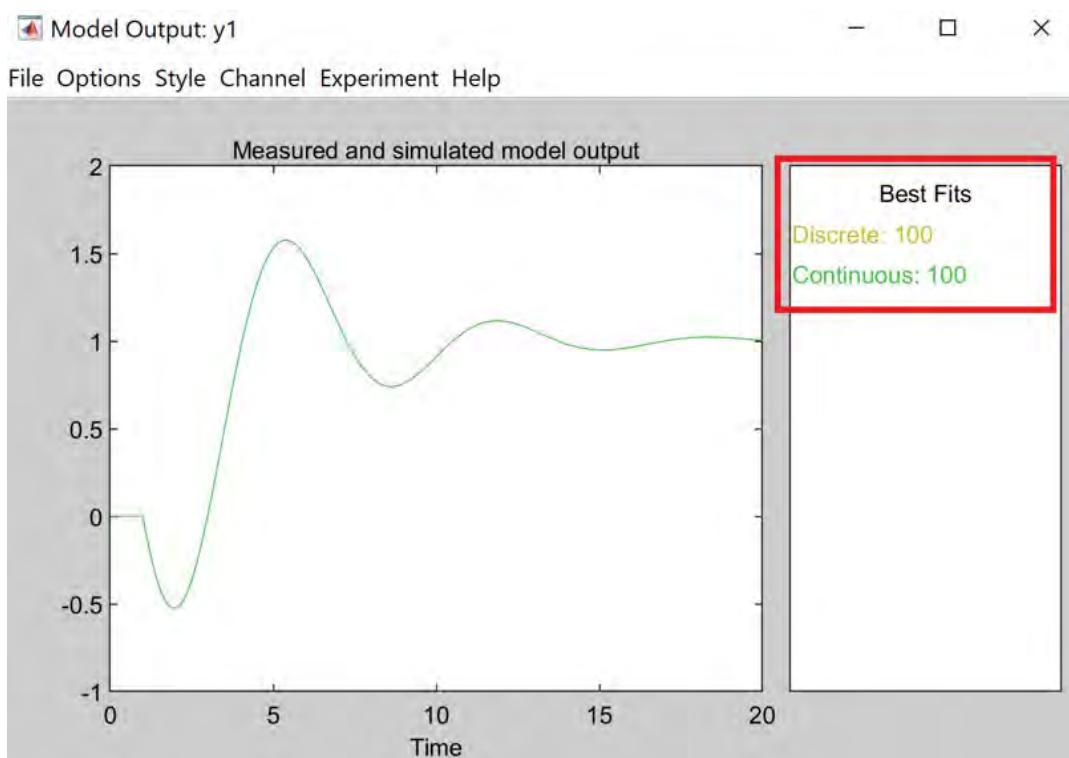
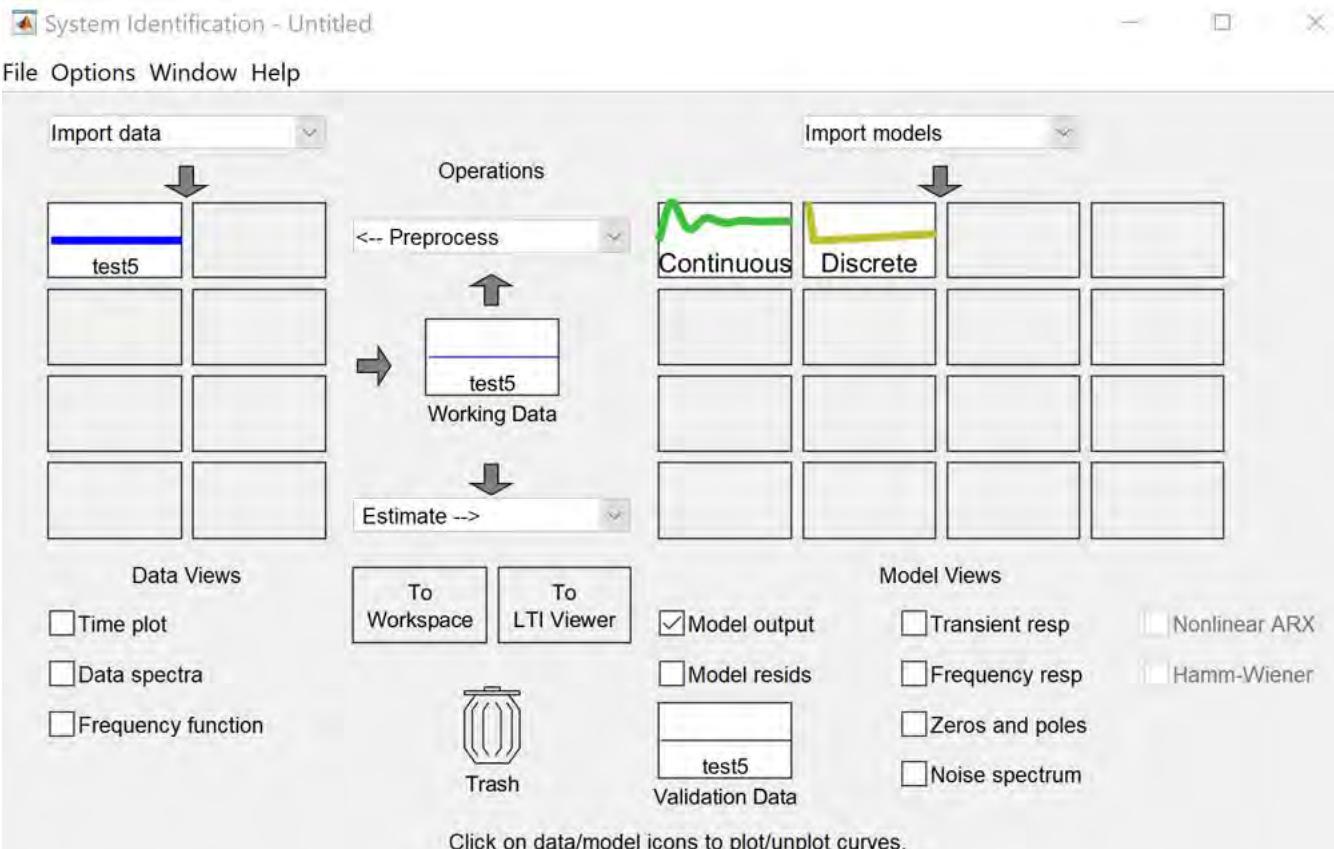


Importing data





**Estimating continuous and discrete time models with 3<sup>rd</sup> order state space functions**



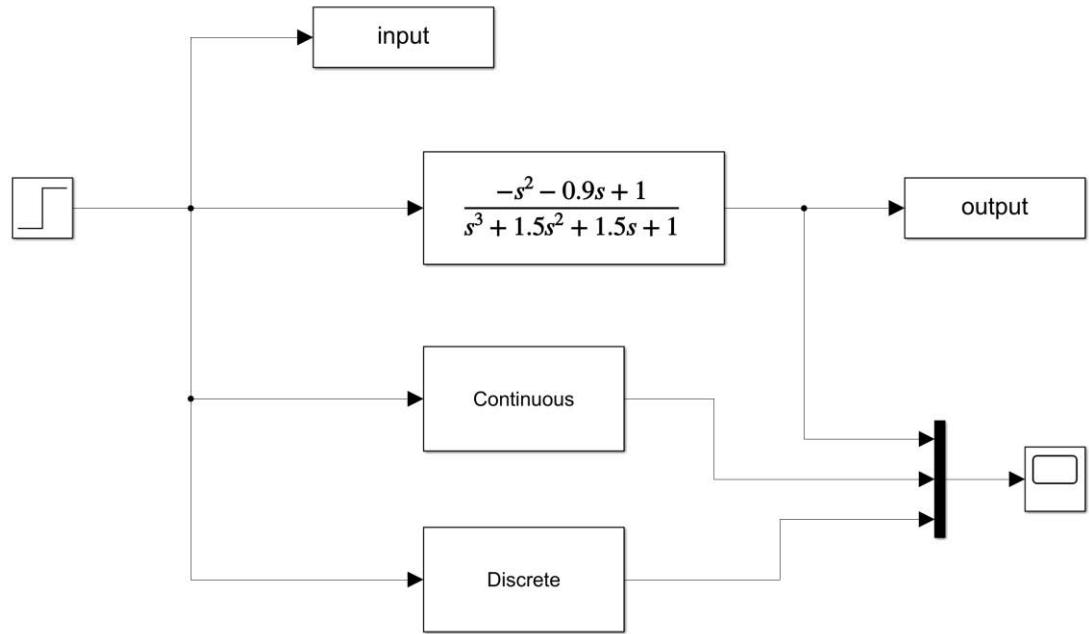
Both discrete and continuous estimations provide perfect results with a 100 percent fit.

## b. MATLAB commands to compare transfer functions

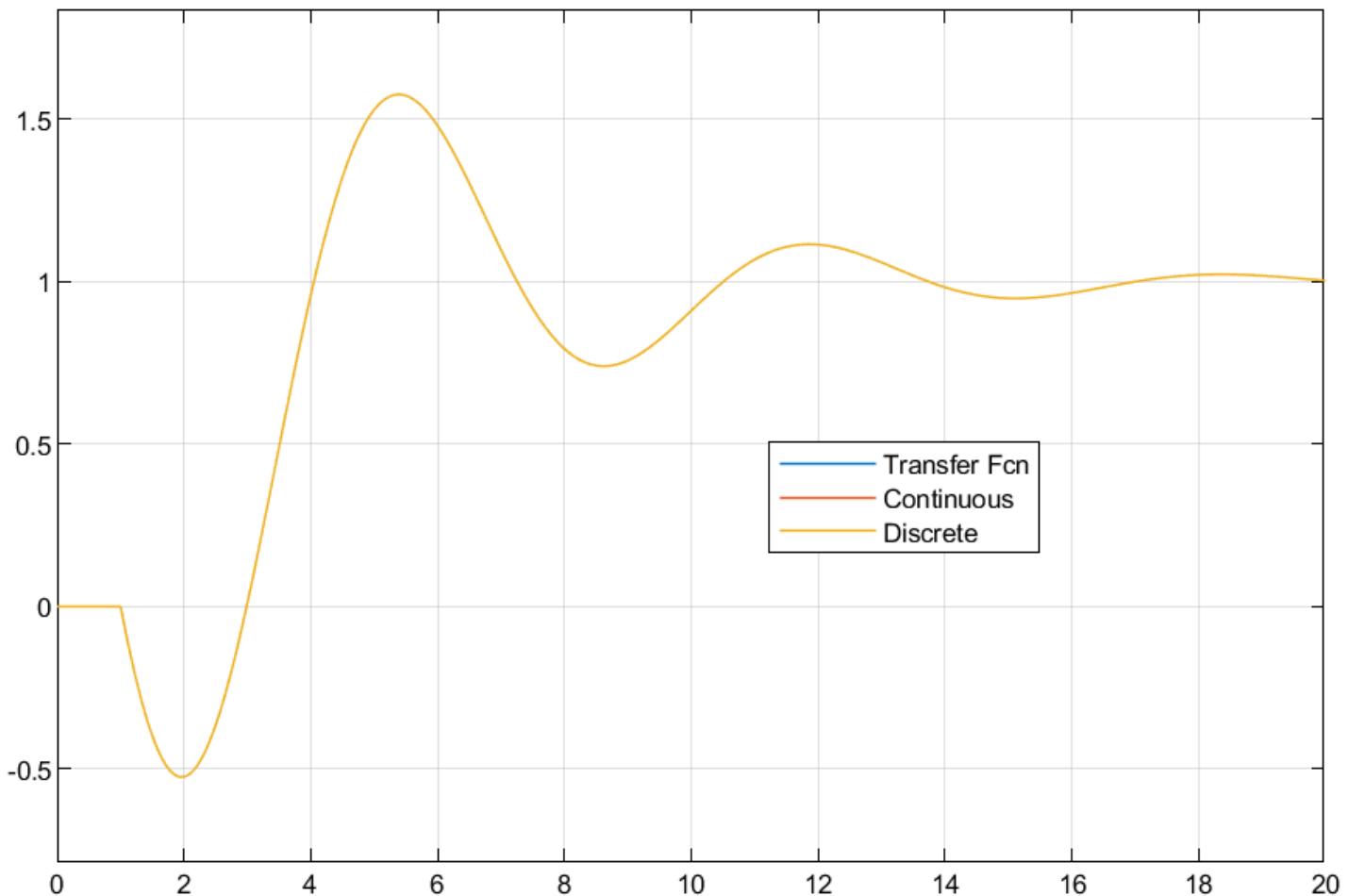
```
AC=[ -1.499 -0.8319 -0.001454; 0.9955 -0.004427 0.8192; -0.001322 -0.8184 0.003508]
BC=[0.2601;-0.1717;-0.007281]
CC=[5.836 15.21 -12.98]
DC=0
[NC,DC]=ss2tf(AC,BC,CC,DC)
CTF=tf(NC,DC)
CTF =
-0.9991 s^2 - 0.9003 s + 1
-----
s^3 + 1.5 s^2 + 1.5 s + 1
Continuous-time transfer function.
```

```
AD=[ 0.9851 -0.008257 -4.833e-05;0.009881 0.9999 0.008191;-5.366e-05 -0.008184 1]
BD=[0.002589;-0.001704;-6.583e-05]
CD=[5.836 15.21 -12.98]
DD=0
[ND,DD]=ss2tf(AD,BD,CD,DD)
DTF=tf(ND,DD)
DiTF=c2d(DTF,0.01)
DiTF =
-0.0001 z^2 + 0.0002021 z - 0.000102
-----
z^3 - 3.03 z^2 + 3.06 z - 1.03
Sample time: 0.01 seconds
Discrete-time transfer function.
```

**Conclusion** – The continuous transfer function obtained from the estimated state space model is very similar to original transfer function i.e., the poles and zeros are also similar. The discrete transfer function obtained from the estimated discrete state space model has an A(z) very similar to the discrete transfer function obtained from the original transfer function.



c.



## Now estimating with 4<sup>th</sup> order

Import Data

Data Format for Signals  
Time-Domain Signals

Workspace Variable  
Input: input  
Output: output

Data Information  
Data name: test6  
Starting time: 0  
Sample time: 0.01  
More

State Space Models

Model name: ss2

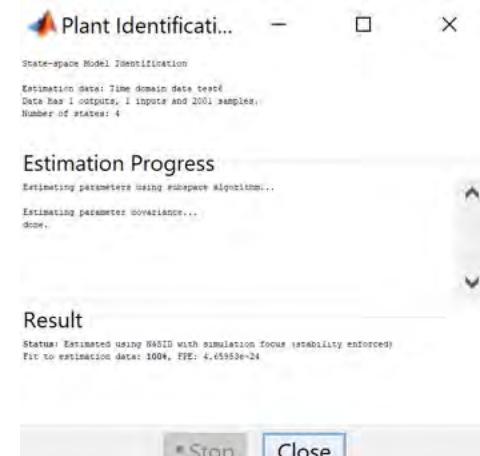
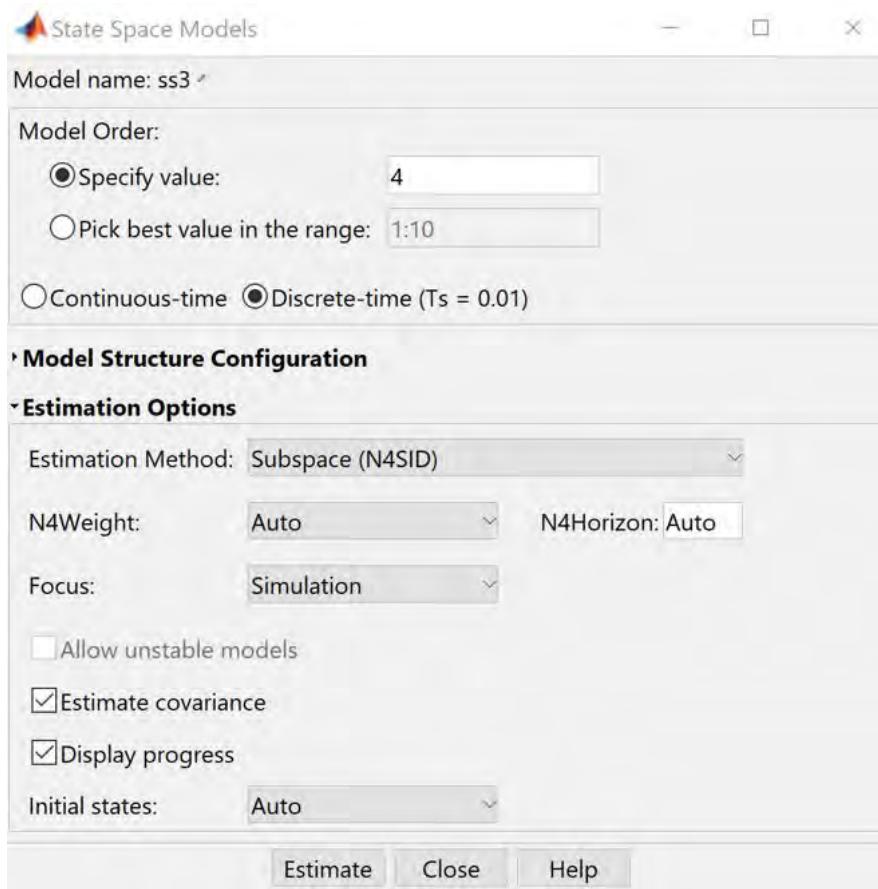
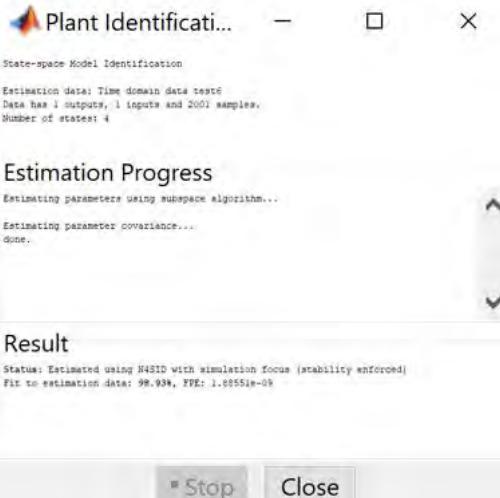
Model Order:  
 Specify value: 4  
 Pick best value in the range: 1:10  
 Continuous-time    Discrete-time (Ts = 0.01)

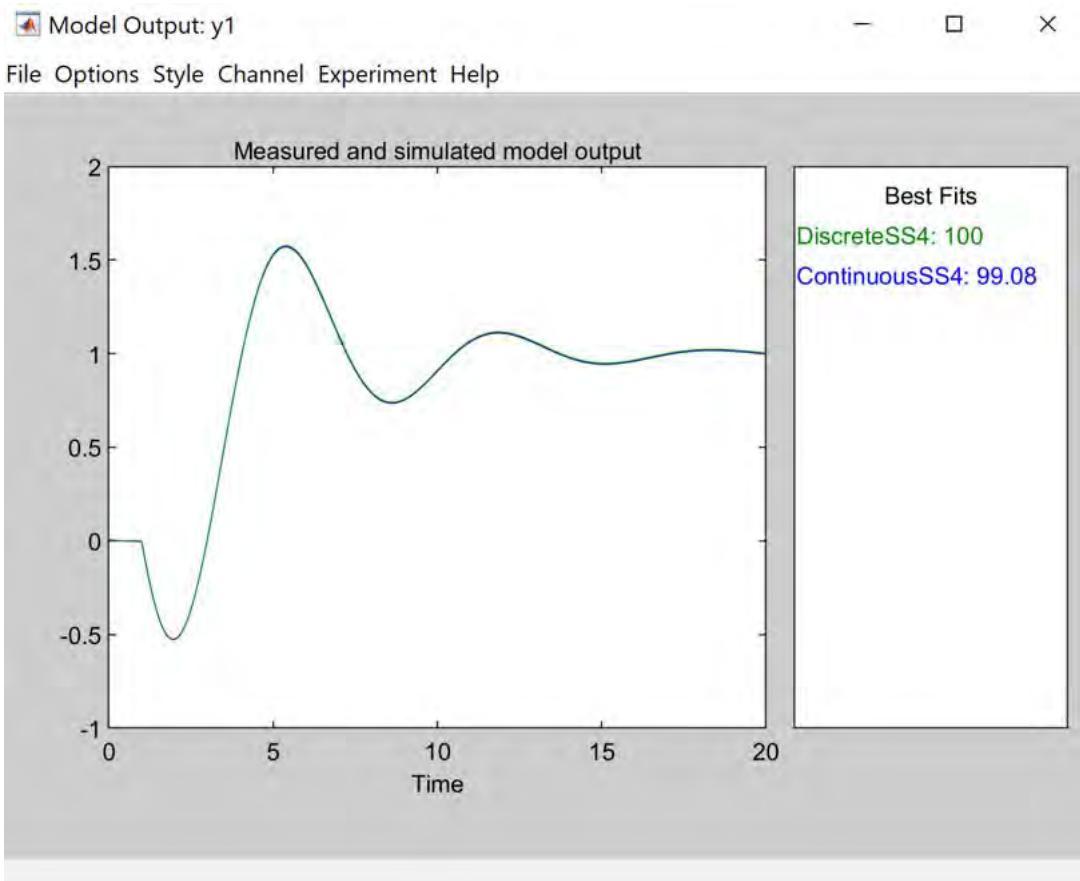
Model Structure Configuration

Estimation Options

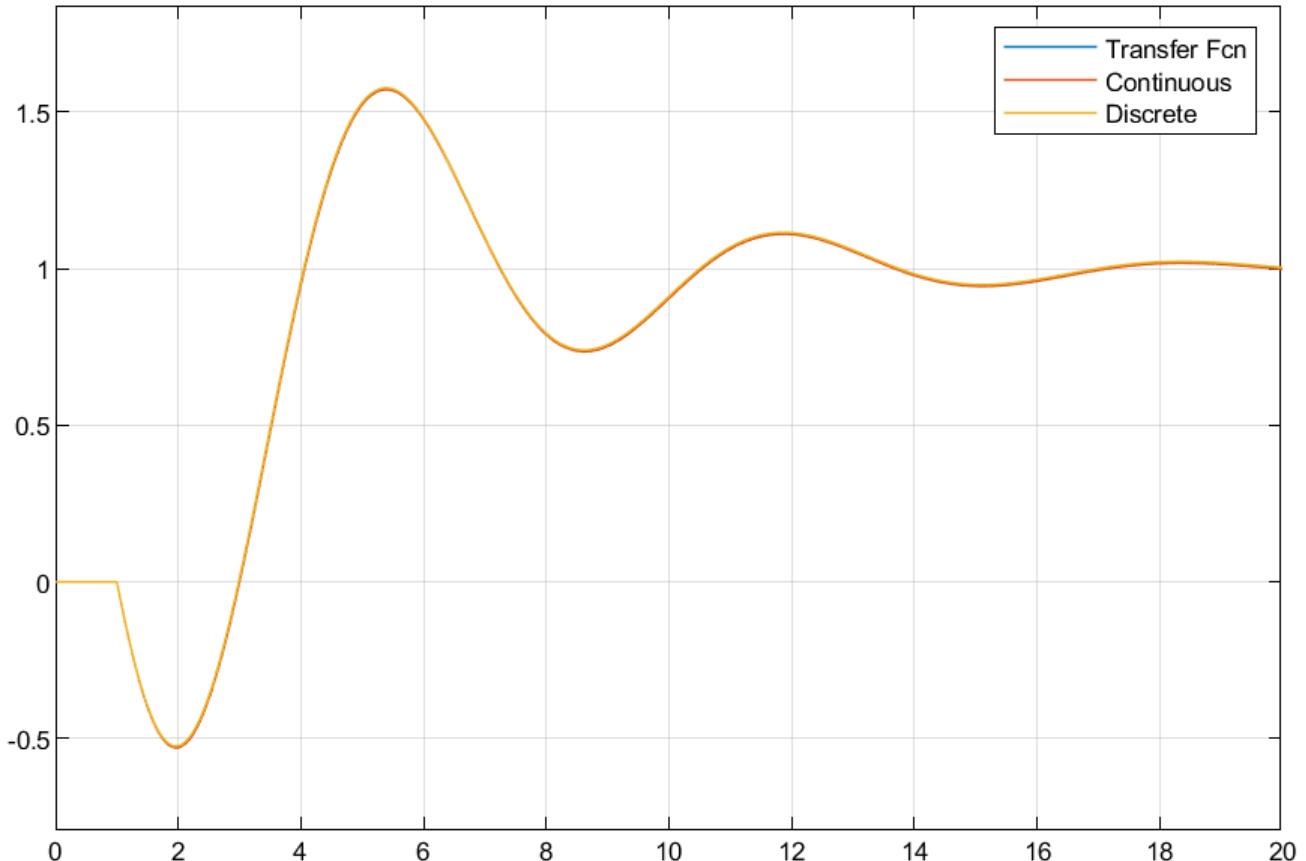
Estimation Method: Subspace (N4SID)  
N4Weight: Auto   N4Horizon: Auto  
Focus: Simulation  
 Allow unstable models  
 Estimate covariance  
 Display progress  
Initial states: Auto

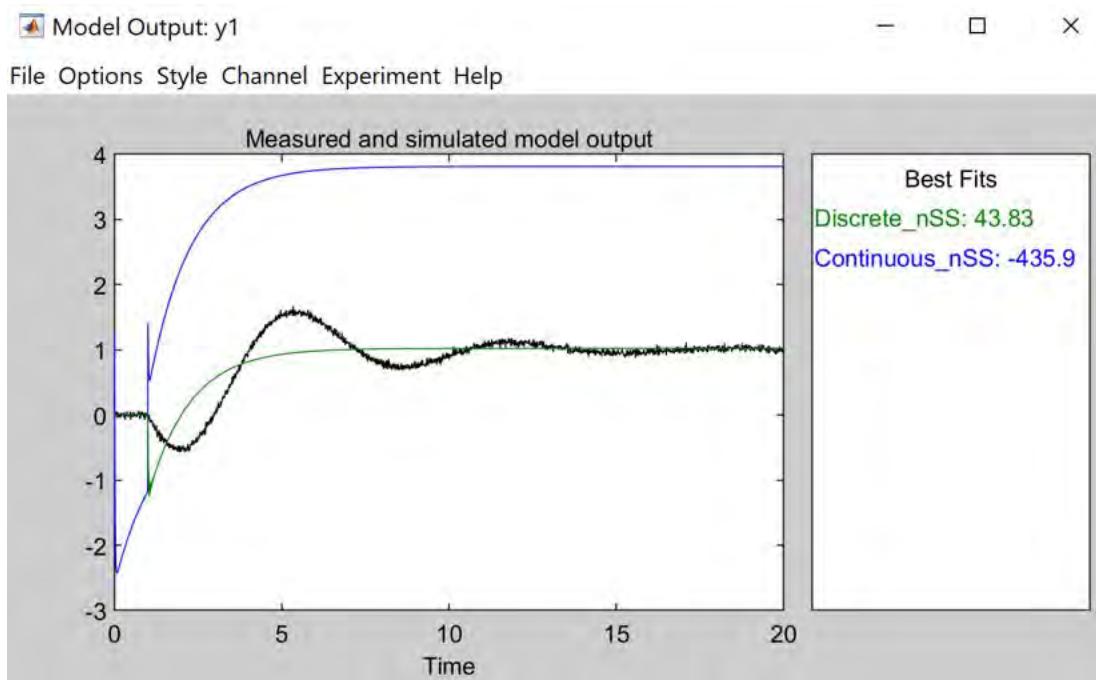
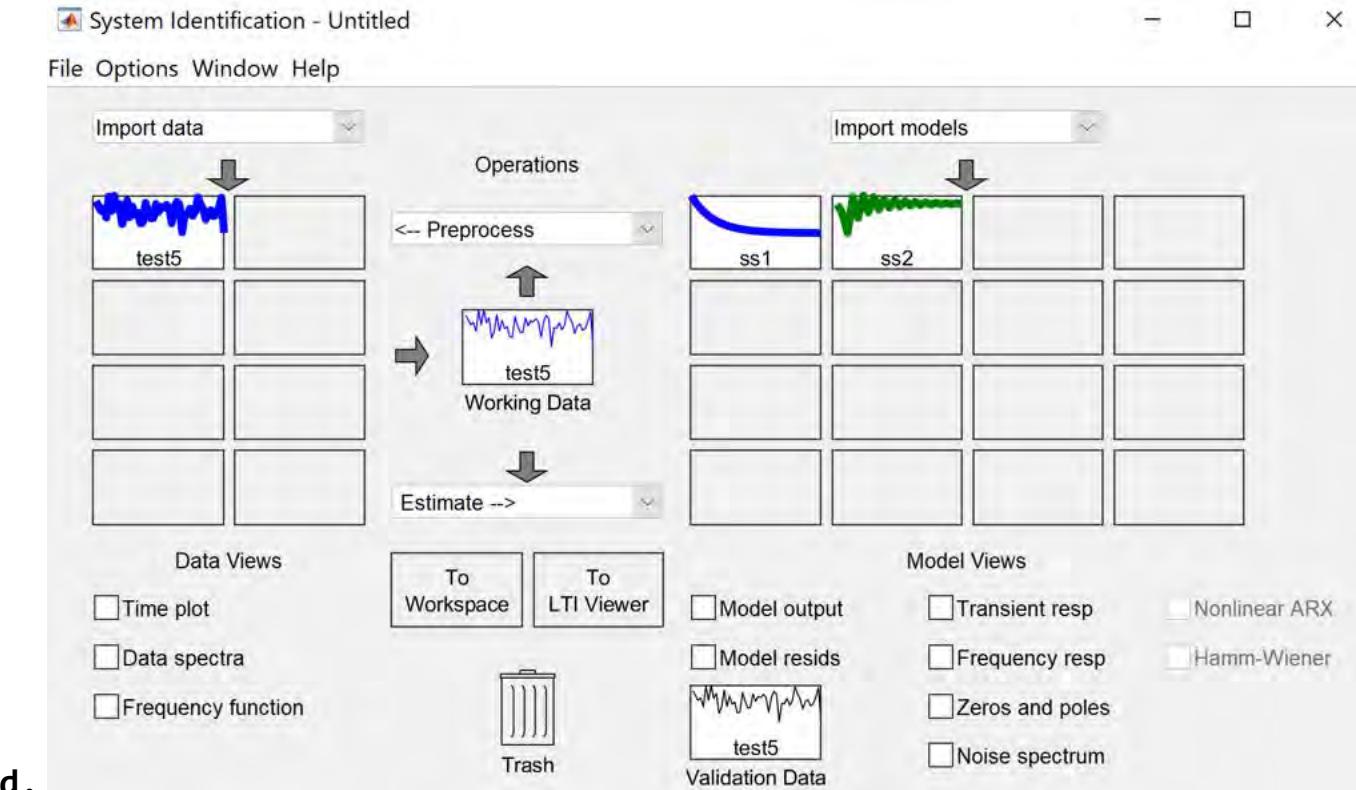
Estimate   Close   Help

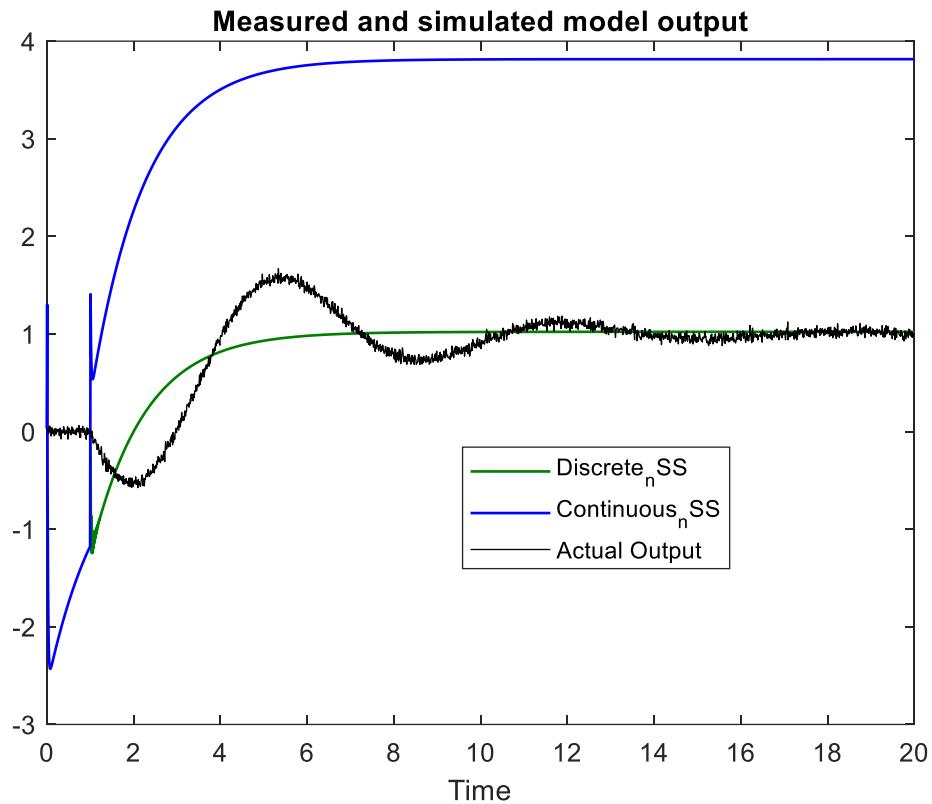




It's observed that the fit for continuous state space model is slightly lower than before







Noise completely throws off the state space estimation, therefore, I used a range.

### MATLAB Code for getting the transfer functions

```
ACN=[ -0.5403 109.8 2.181;-31.24 -2531 506.7;-1.027 -320.2 -1.48]
```

```
BCN=[ -15.7;744.5;49.96]
```

```
CCN=[18.84 11.55 7.518]
```

```
DCN=0
```

```
[NCN,DCN]=ss2tf(ACN,BCN,CCN,DCN)
```

```
CTFN=tf(NCN,DCN)
```

```
CTFN =
```

$$\frac{8679 s^2 + 2.673e5 s + 5.245e5}{s^3 + 2533 s^2 + 1.708e5 s + 1.358e5}$$

Continuous-time transfer function.

**Note - DCN is repeated but by the time it is repeated, the first value becomes irrelevant**

```
ADN=[0.9844 0.05965 0.1715;-0.01898 -0.8875 0.2828;0.02005 -0.1792 0.5349]
```

```
BDN=[0.1091;0.491;-0.2851]
```

```
CDN=[18.66 0.5385 9.019]
```

```
DDN=0
```

```

[NTDN,DTDN]=ss2tf(ADN,BDN,CDN,DDN)
DTFN=tf(NTDN,DTDN)
DTFNF=c2d(DTFN,0.01)
DTFNF =

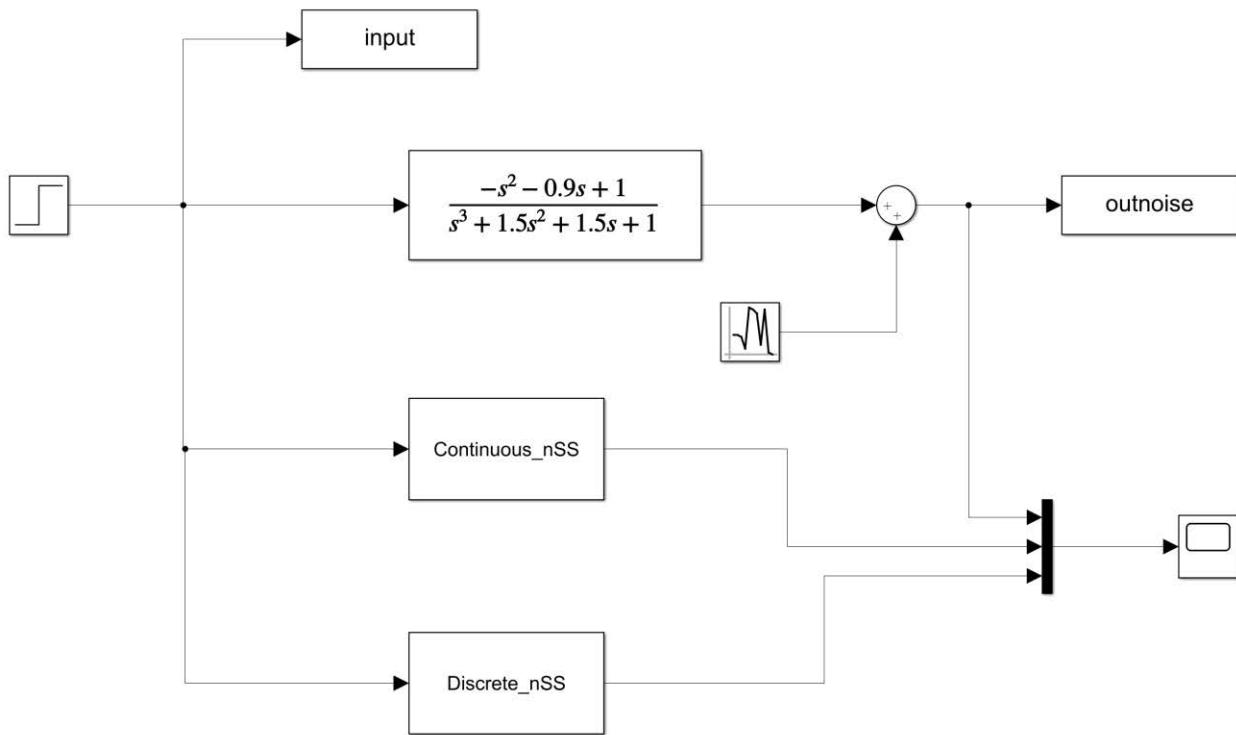
```

$$\frac{-0.002751 z^2 + 0.00544 z - 0.002688}{z^3 - 3.006 z^2 + 3.013 z - 1.006}$$

Sample time: 0.01 seconds

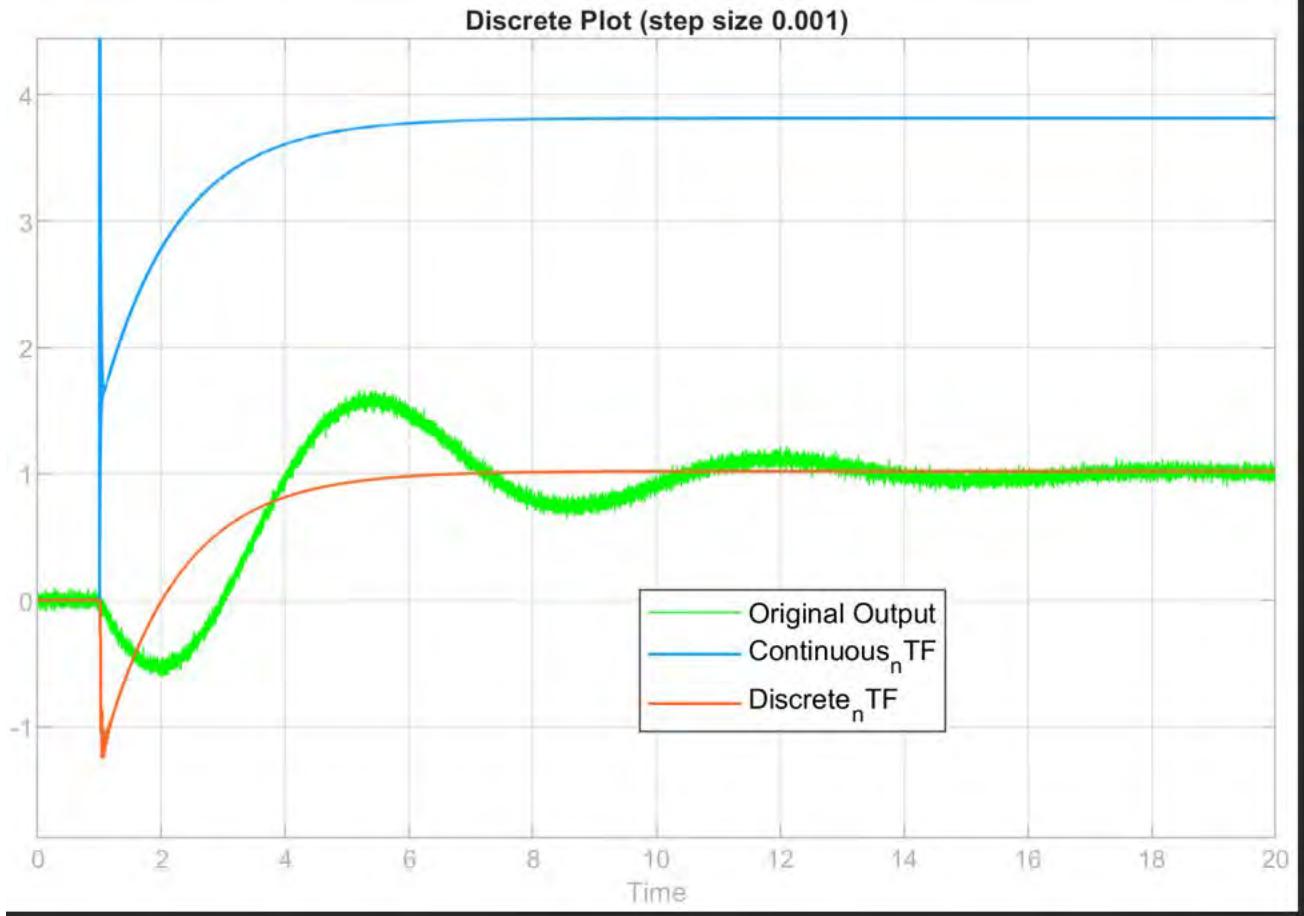
Discrete-time transfer function.

**Conclusion** – The continuous transfer function is completely different from the original as expected from the fit values i.e., the poles and zeros are different. The denominator of the discrete transfer function remains similar to the discrete transfer function values derived from the original transfer function. However, despite that, it is only able to attain a fit value of 43.83

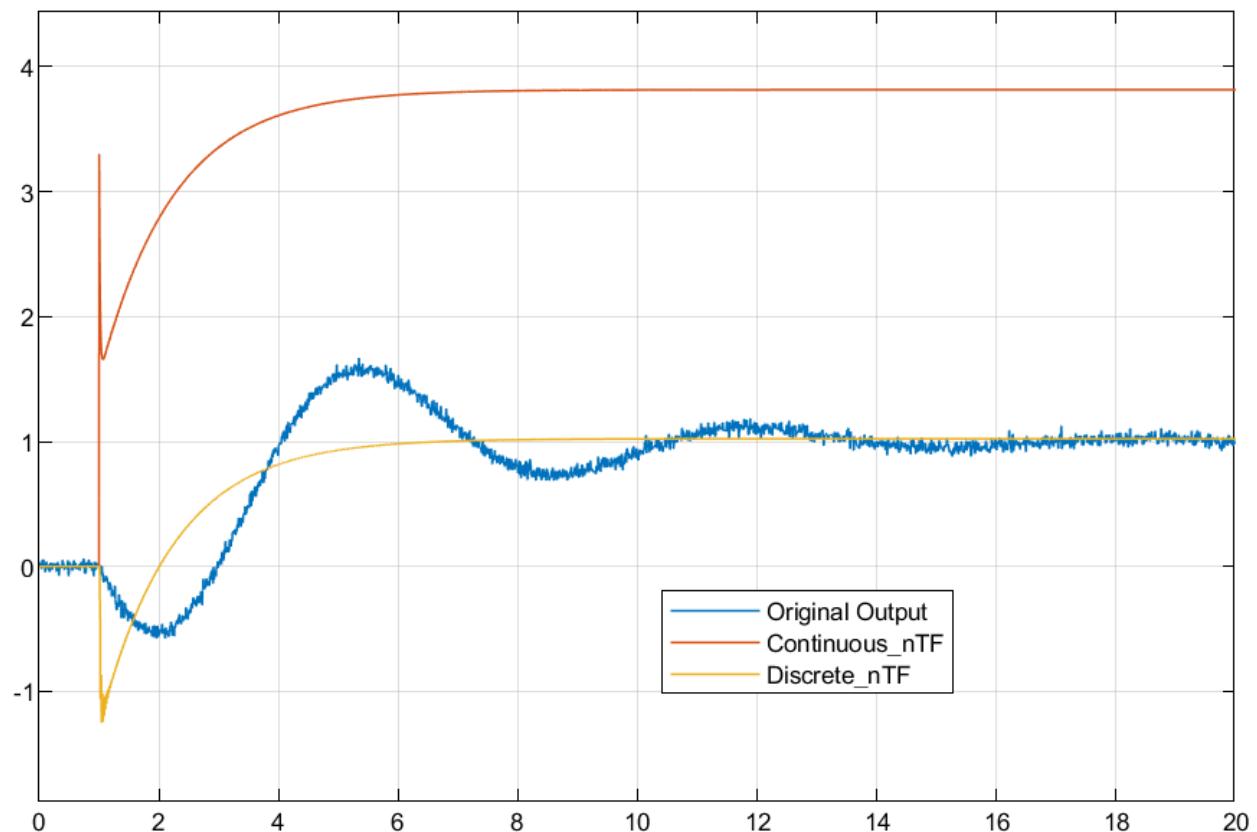


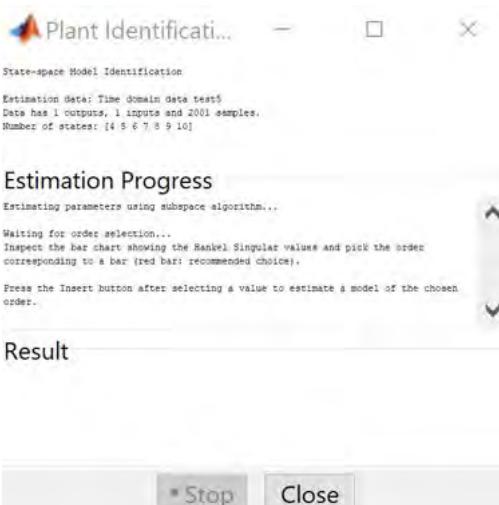
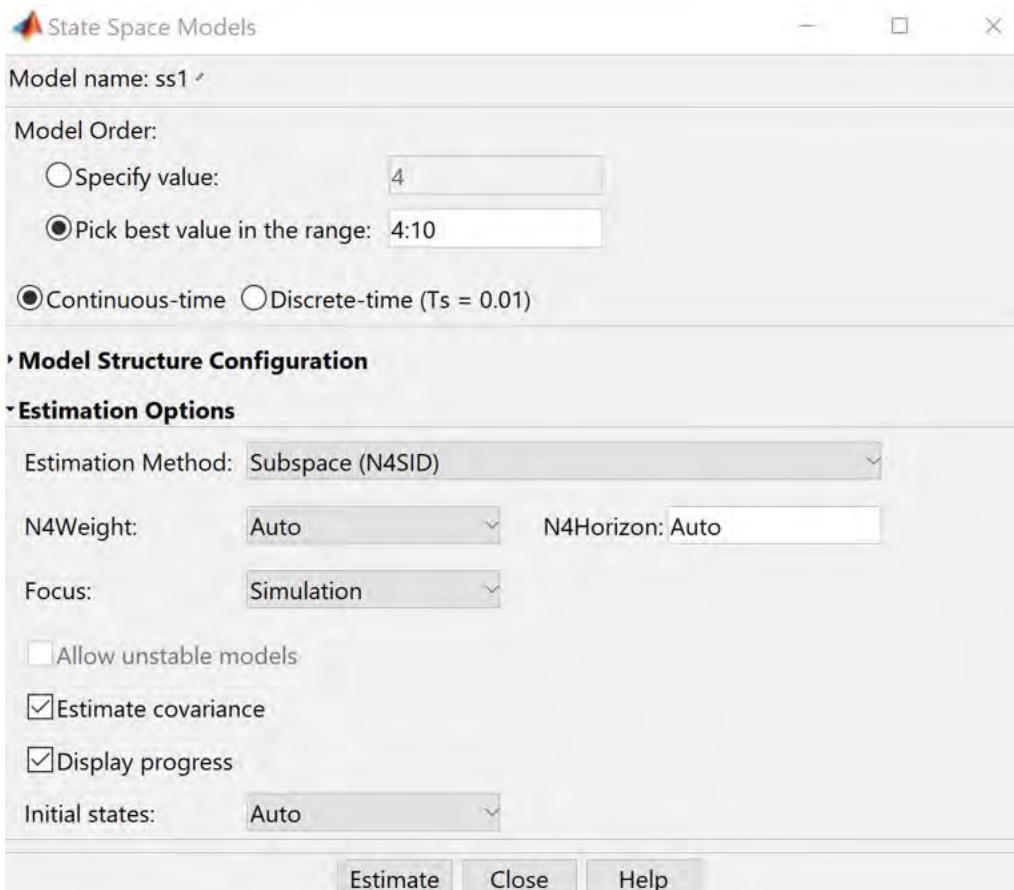
Simulink Block Diagram

The simulation failed to run using fixed step size of 0.01 and requested me to either reduce the step size or reduce error tolerances. I changed it to variable step and received the outputs. Then I also reduced the step size to 0.001 and received the same outputs on the scope plots. Both are shown below.

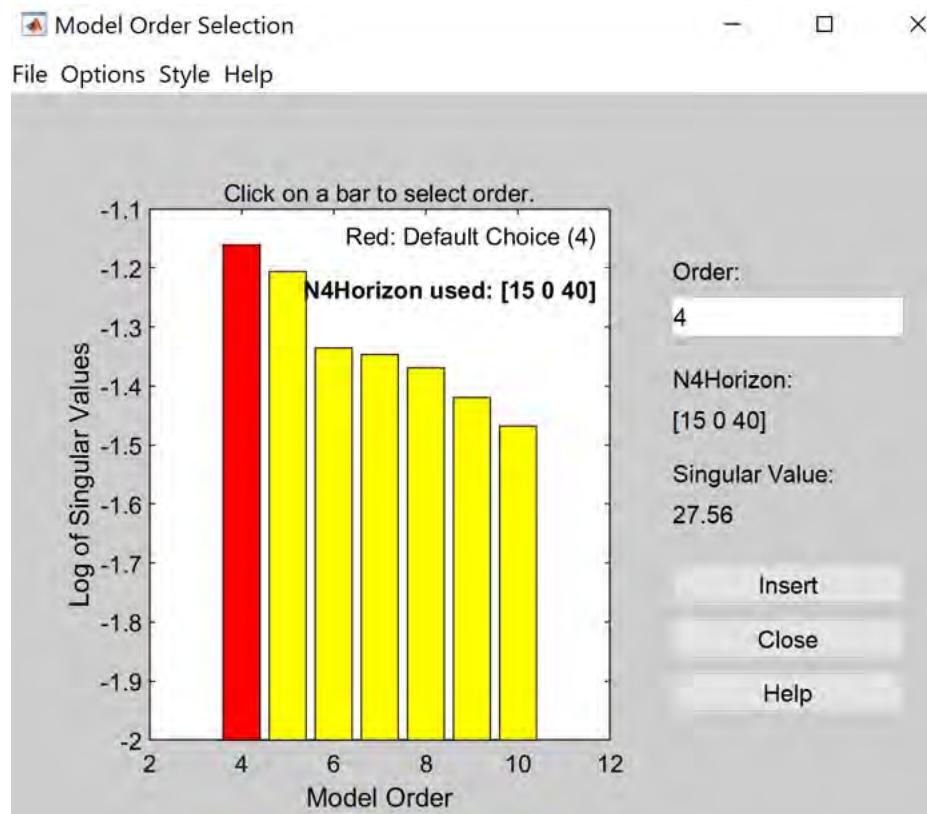


**Plot using variable time step settings in Simulink**

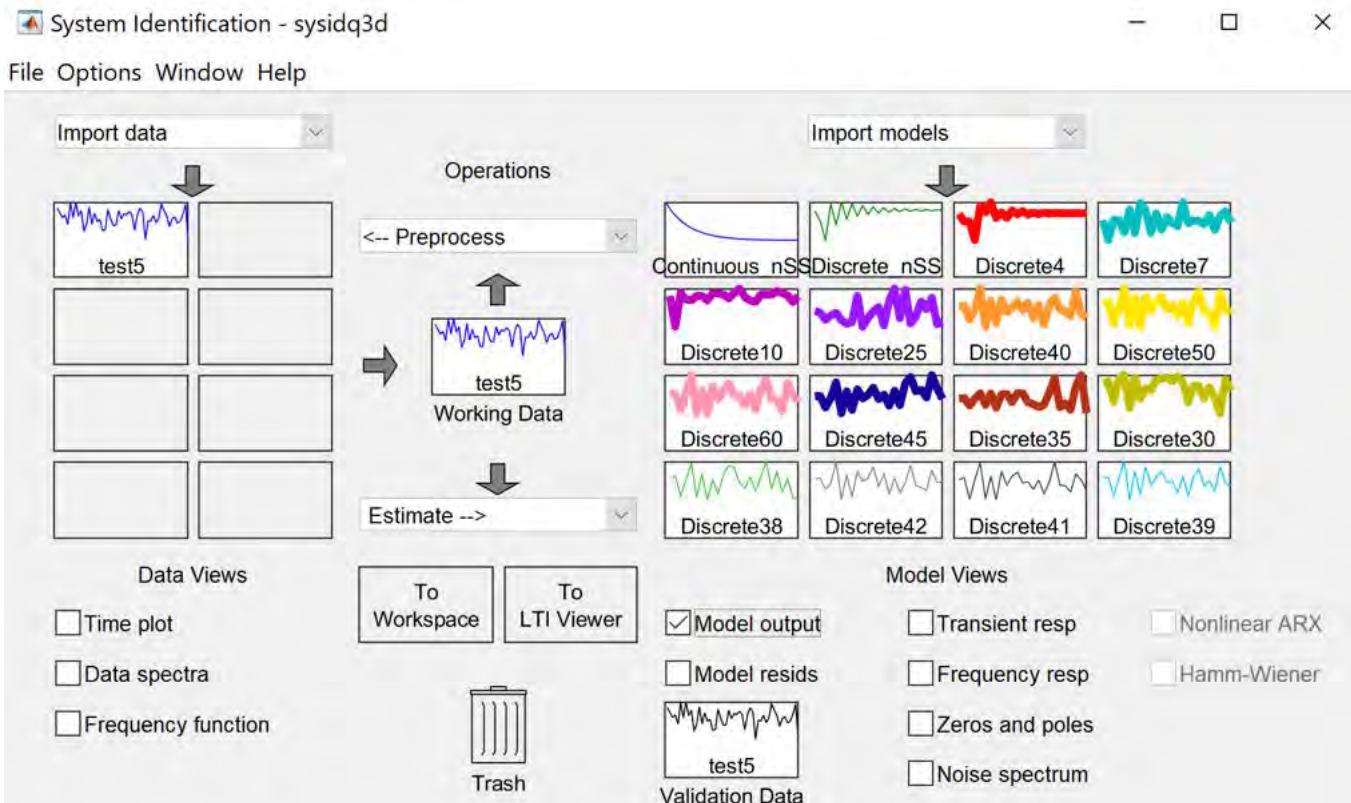


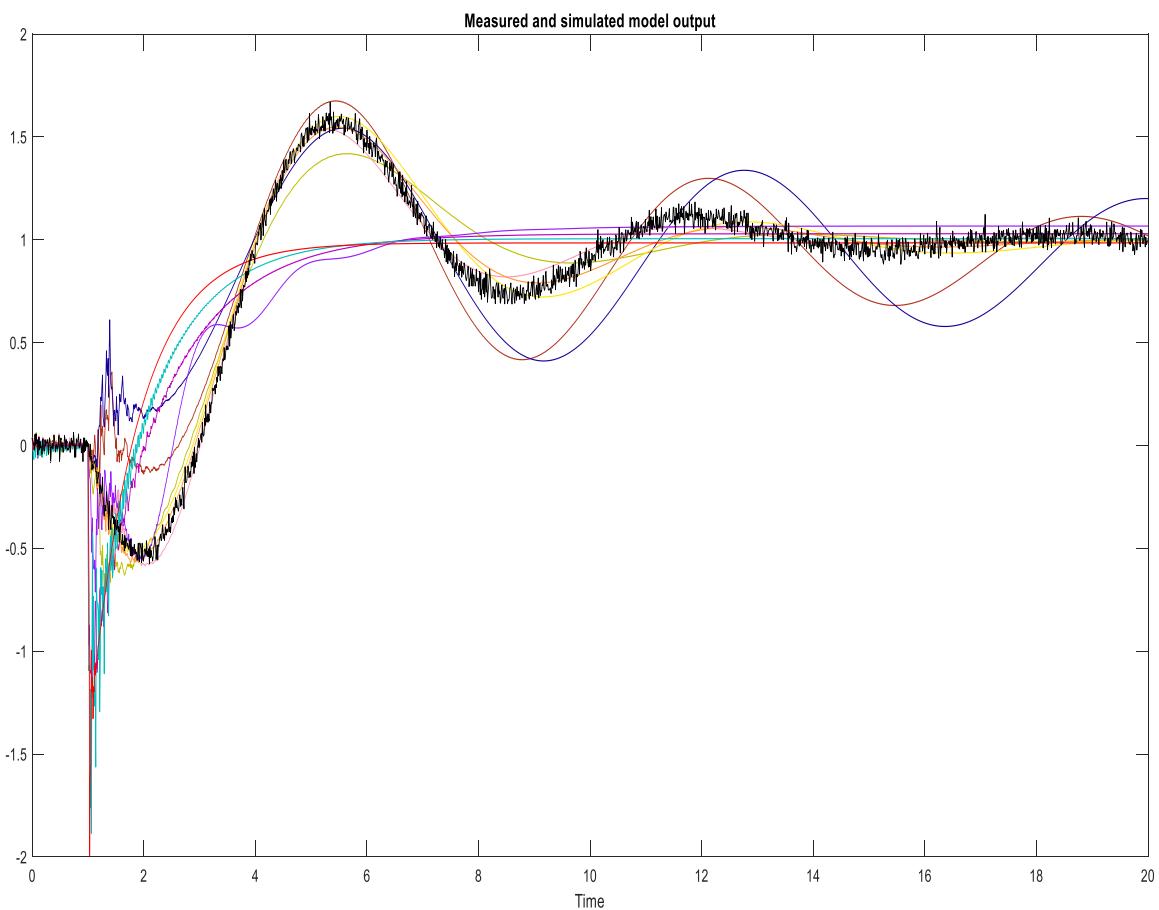
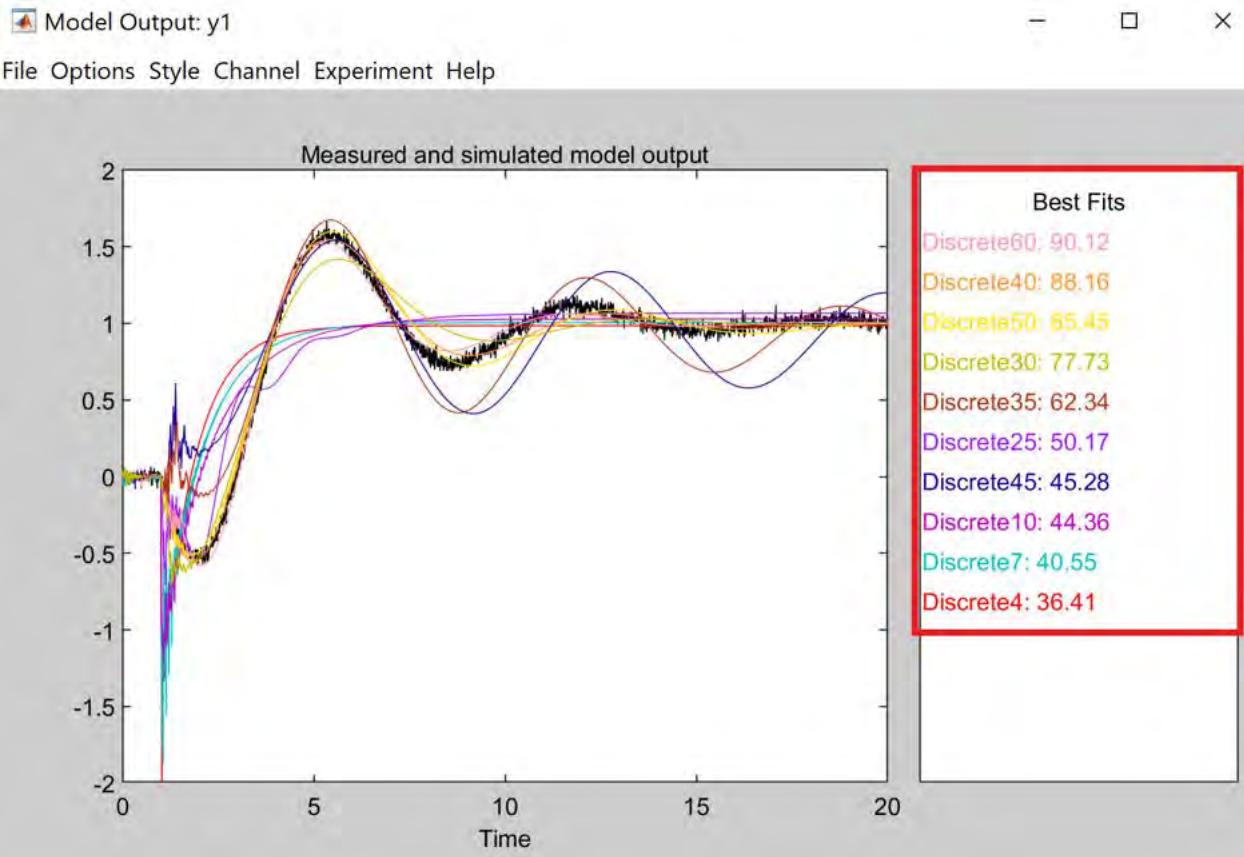


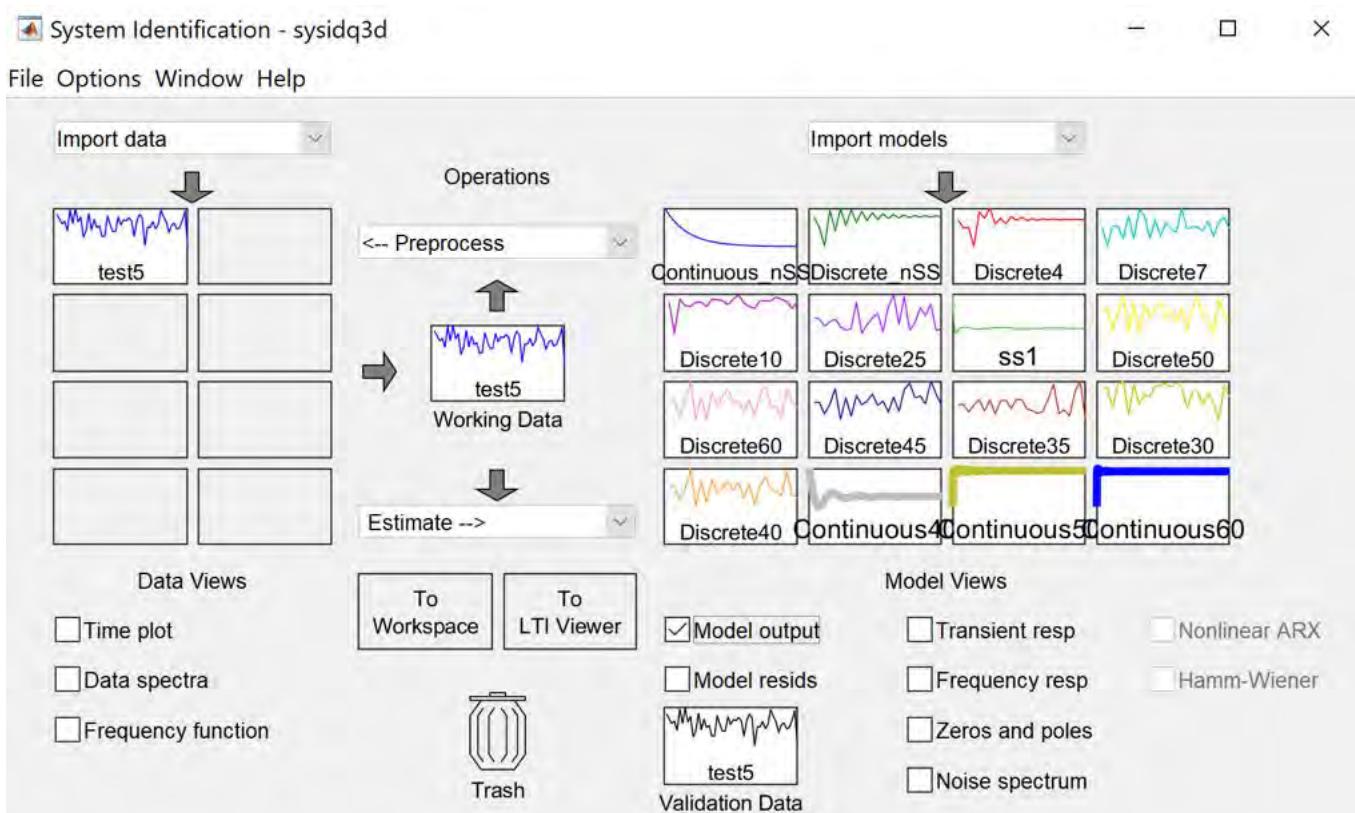
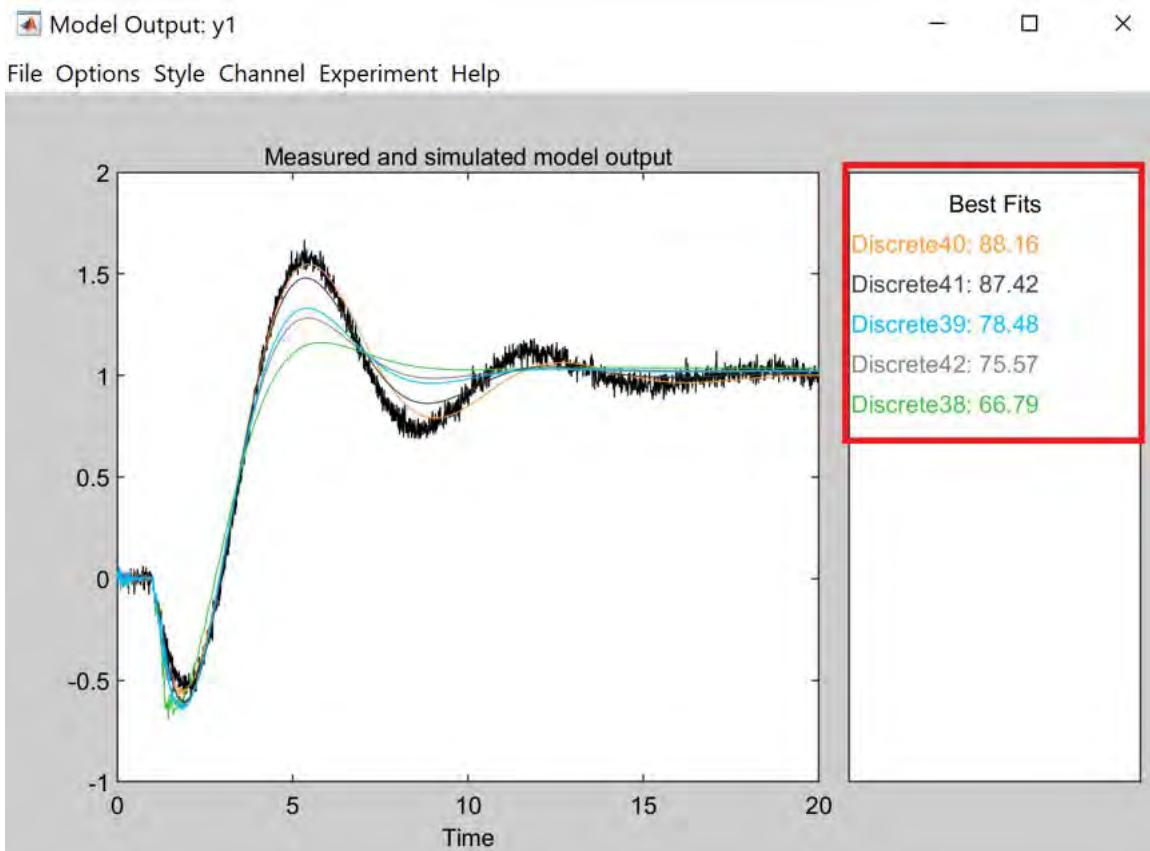
\* Stop Close

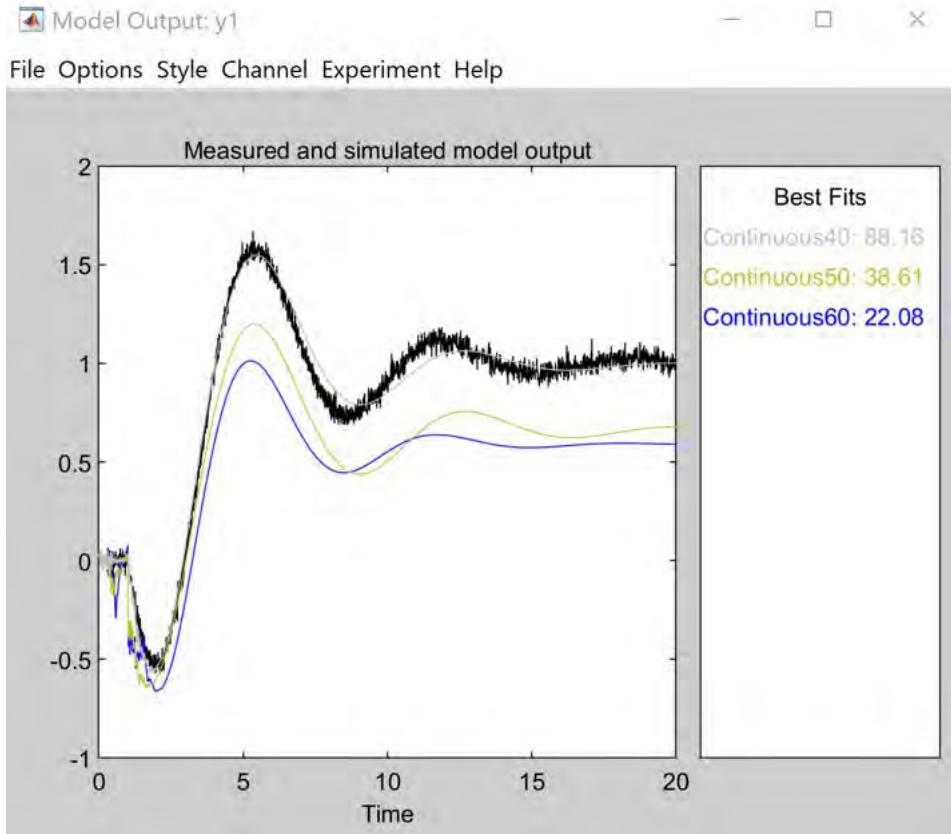


Using the range, the suggested order was 4 which lowered the fit value to 36.41 so I reverted back to increasing the order of the estimated function to see the effects on estimated output. I didn't experiment much with the continuous estimation as it took too much time for estimation process.









**Conclusion** – Using the hit and trial method, I observed two things for discrete estimation. Firstly, it takes a model of order 60 to take the fit value above 90. Secondly, there is a peak in fit value for the order 40 which might be due to the way the system identification toolbox is set up in MATLAB as explained by Dr. Niestroy during class.

The hit and trial method took a long time with continuous estimation once the order reached values of 20. However, the peak was observed at order 40 similar to what was seen in the case of discrete estimation. But the fits don't improve as the order is increased till 60.

Note: There were many other orders I ran the continuous function with but the documentation started getting messy so I only kept the ones that were important.

- e. Noise definitely hurts the estimation as the fits are reduced. For the state space model estimation, noise is highly detrimental as it takes an order of 60 to get the fit value over 90 for the discrete estimation model

## **Problem - 4**

For the noise free data, all the continuous estimation models produced a fit of 100 whereas for the discrete estimation, the state space model produced the best fit at 100 while the other two were around the value of 95.

For the noisy data, the best fits were produced by the transfer function method and the ARMAX method at fit values around 92.

To develop a model from noisy data, the best approach is definitely to try different methods. However, I would first try the ARMAX and transfer function methods as they appear to fit the data in the best way.

**EE5327, Fall 2020**  
**Homework 5: Neural Networks**  
**Due 11/19/2020**

For Problem 1, implement the following system in Simulink and assume a damping ratio (zeta) of 0.5, a fixed step integration with a time step of 0.01 sec, and a unit step input occurring at 1 second, with a simulation stop time of 15 seconds. Generate a training dataset of (time, input, omega) as inputs and y as the output for values of omega 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, and 3.5 rad/sec.

$$y(s) = \frac{1}{s^2 + 2s + 1} u(s)$$

**Problem 1) Neural Network Estimator – single variable (60 points)**

- Use the nnstart tool Fitting App to fit four, single hidden-layer networks with 10, 15, 20, and 25 nodes in the hidden layer, respectively. Save each as either an m-file function or Simulink block. Generate comparisons of the neural network fits with the training data. Make any observations about the size of the network and the quality of fit it produces.
- Now test the fit model by testing the neural network output with the true system output for omega values 0.75, 1.25, 1.75, 2.25, 2.75, and 3.25 rad/sec. Make any observations about the size of the network and the quality of fit it produces for this test data or if any frequencies fit better than others.
- Finally, comment on the overall ability of the neural network to work or not work for this application and the best size for the network, in your opinion.
- Let's see if the network can extrapolate. Set the natural frequency to 0.25 rad/sec, run the system, and compare the output of the true system with the output of your best neural network. Does the network do a good job when operating outside the training limits in this case? Repeat for a natural frequency of 3.75 rad/sec.

**Problem 2) Neural Network Estimator – noise effects (40 points)**

For Problem 2, repeat the first three steps (a through c but not d) of Problem 1 but add zero mean noise with variance 0.001 to the output data used for training. Comment on the effects of noise on the ability of the network to model these data.

**ATUL SHROTRIYA**

**UTA ID - 1001812437**

**Neural Networks**

**Homework - 5**

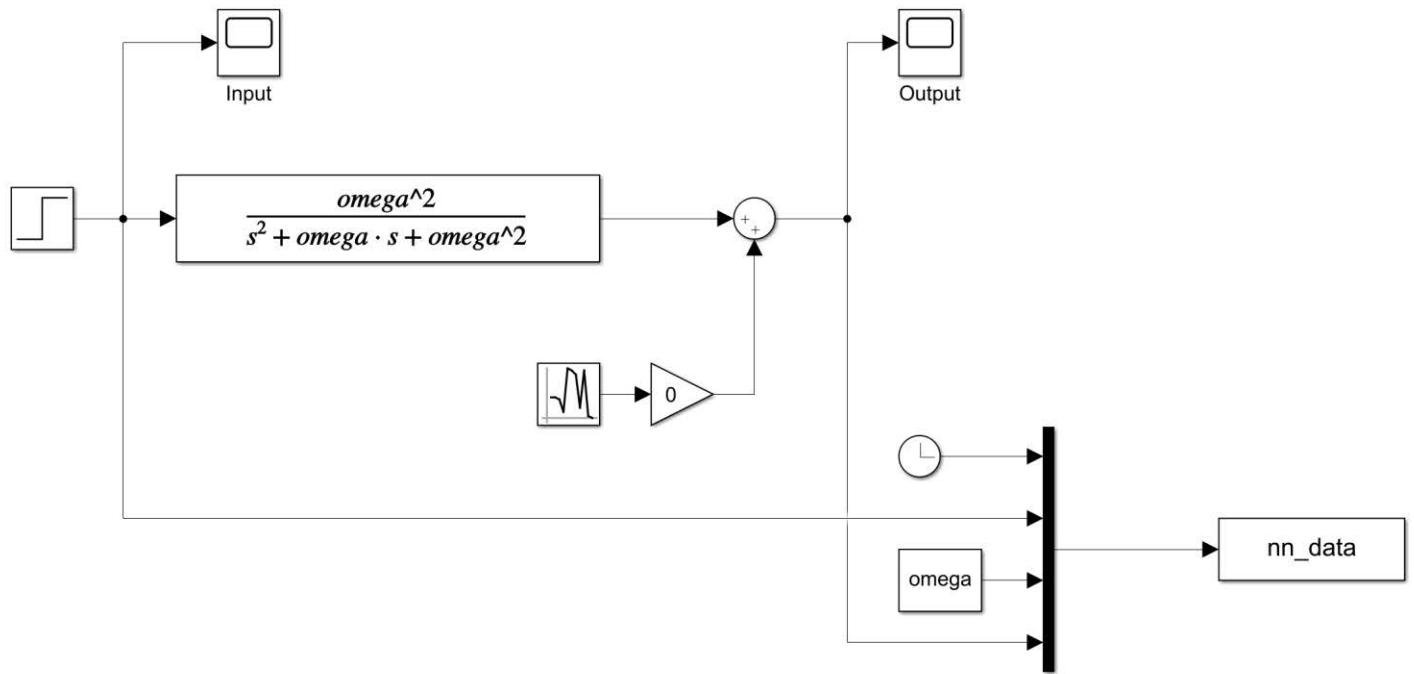
**Submitted - November 19, 2020**

**Course:**

**System Identification and Estimation**

## Problem - 1:

a.

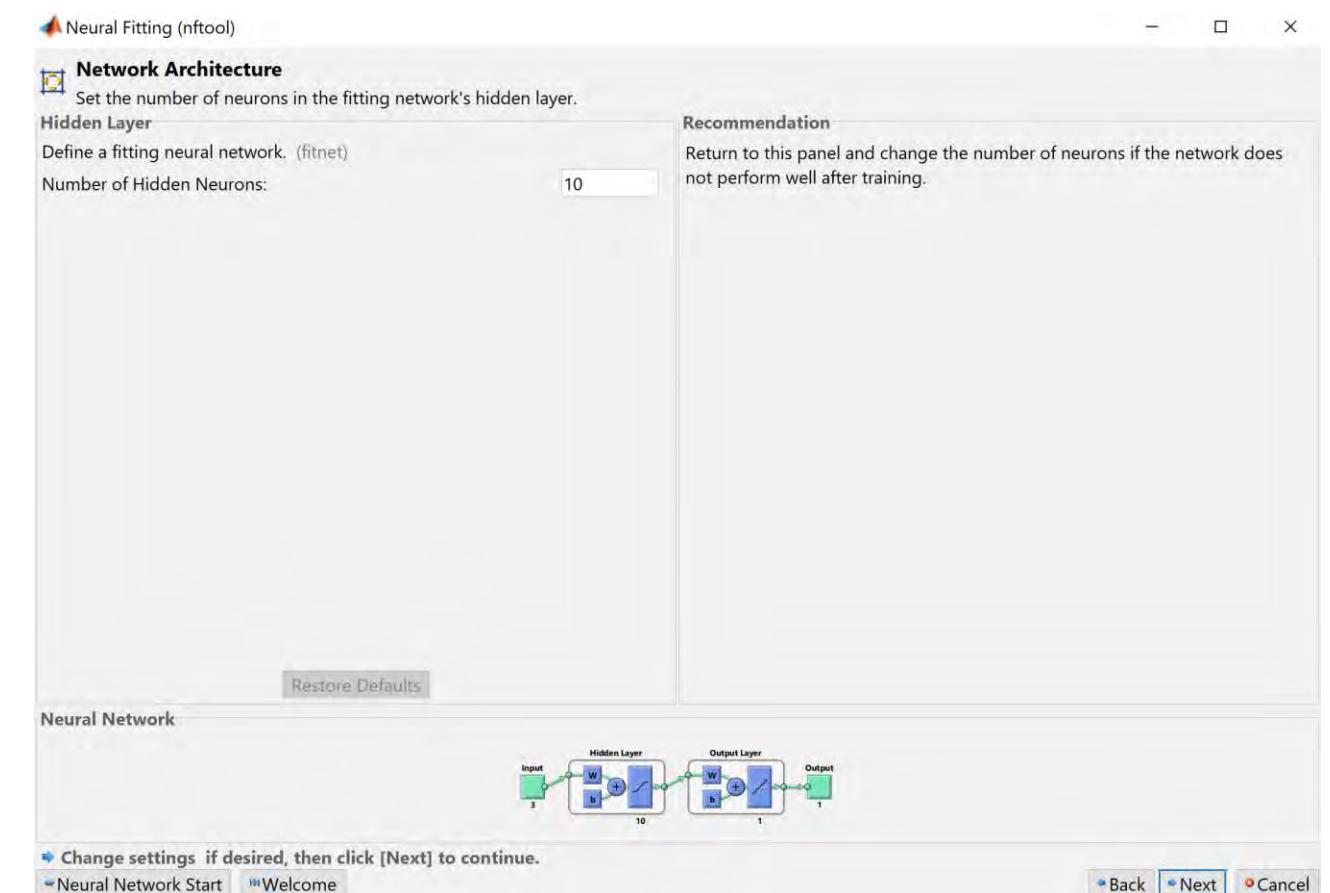
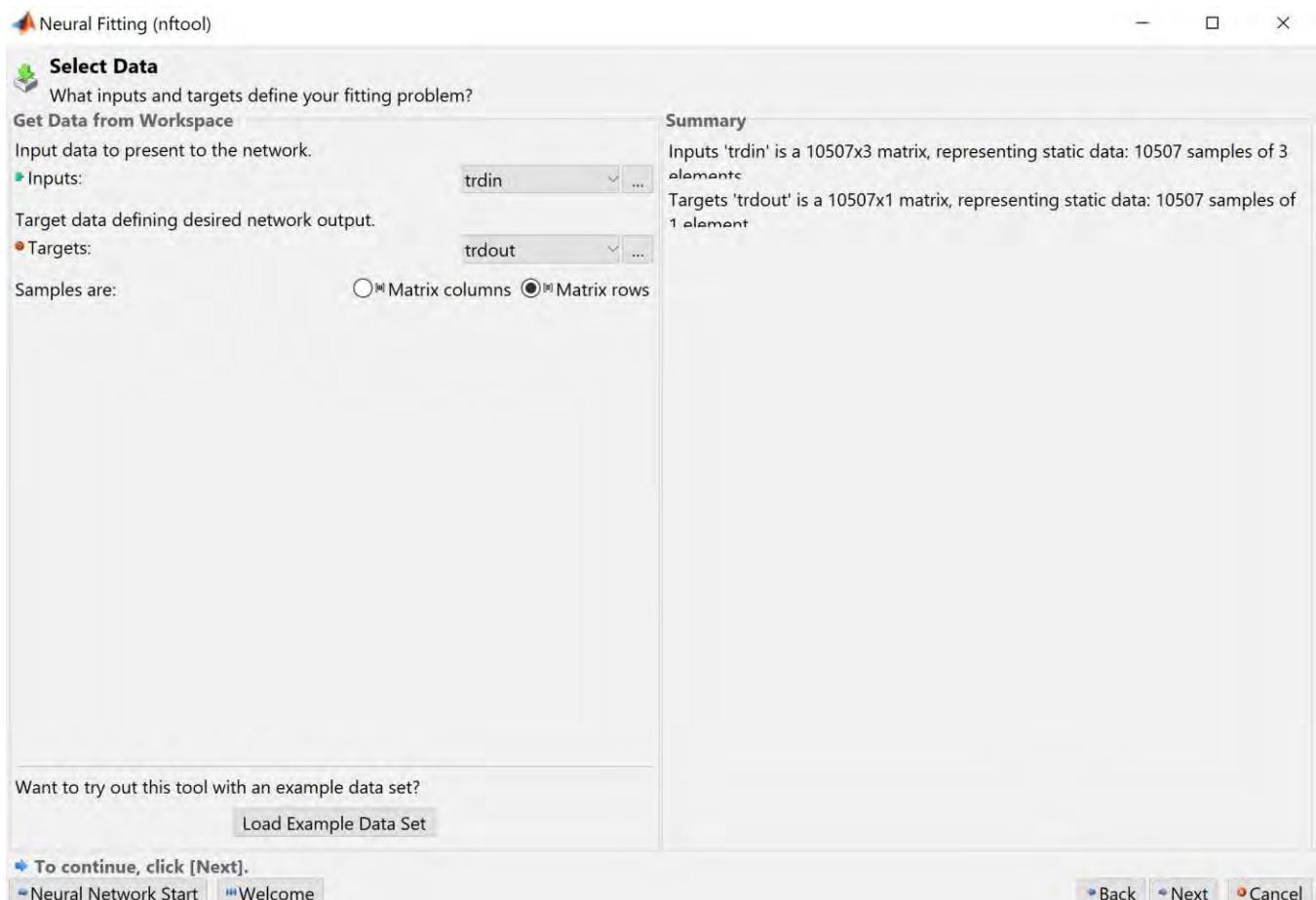


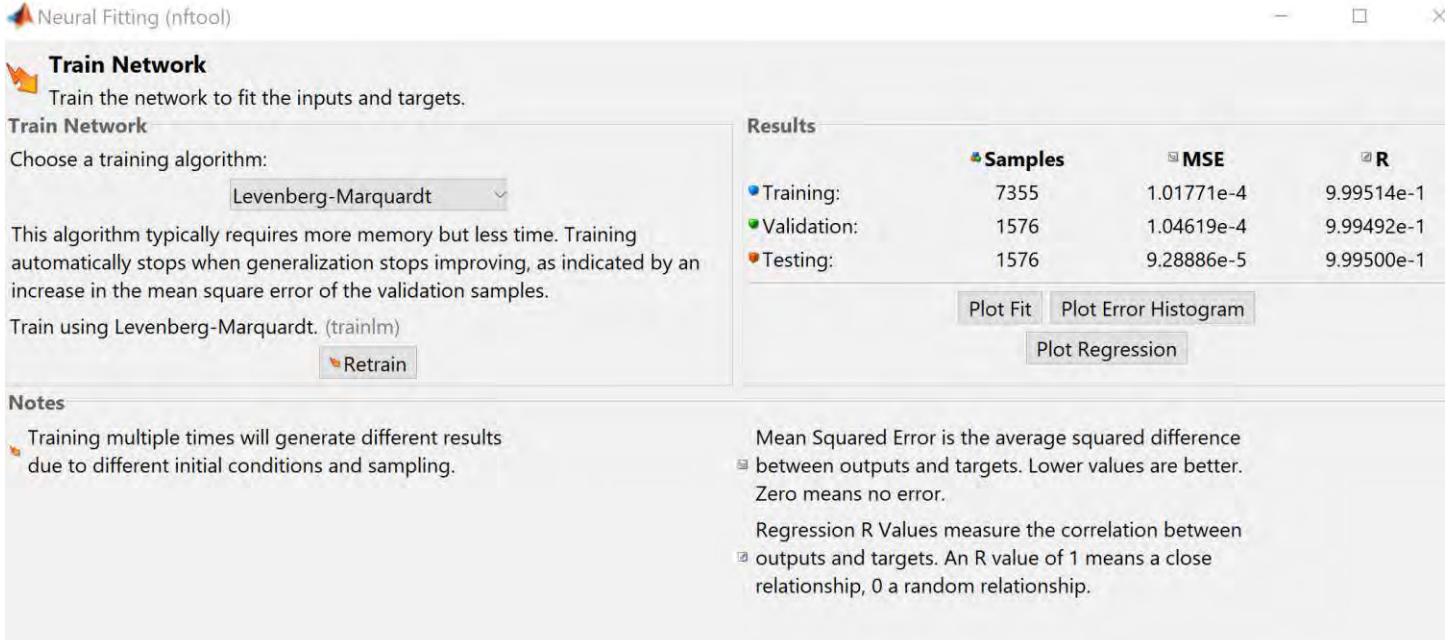
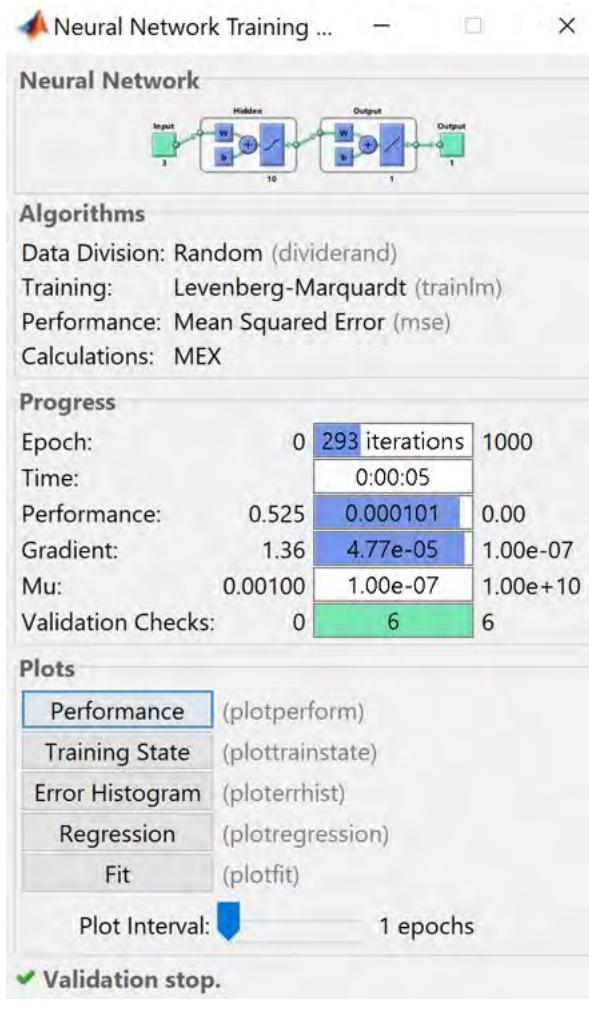
Simulink Diagram to generate training data

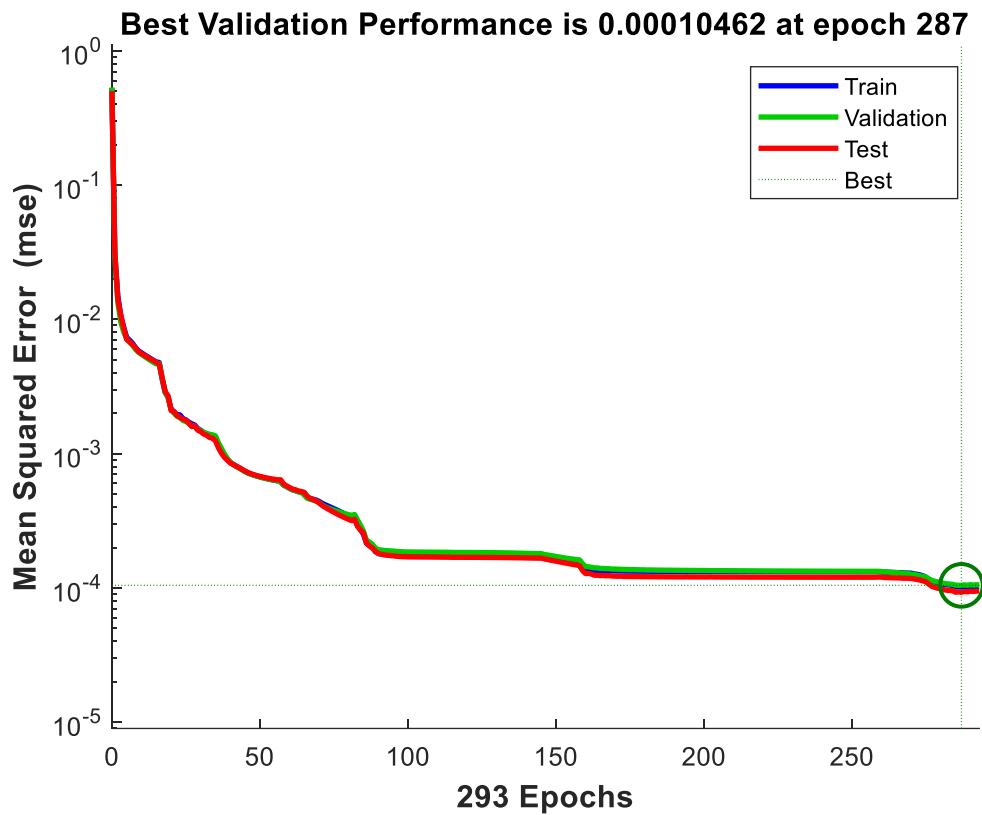
### MATLAB Commands to arrange the data

```
omega=0.5 %run the simulation every time a new omega is defined  
trdata=nn_data  
omega=1  
trdata=[trdata;nn_data]  
omega=1.5  
trdata=[trdata;nn_data]  
omega=2  
trdata=[trdata;nn_data]  
omega=2.5  
trdata=[trdata;nn_data]  
omega=3  
trdata=[trdata;nn_data]  
omega=3.5  
trdata=[trdata;nn_data]  
  
trdin=trdata(:,1:3)  
trdout=trdata(:,4)
```

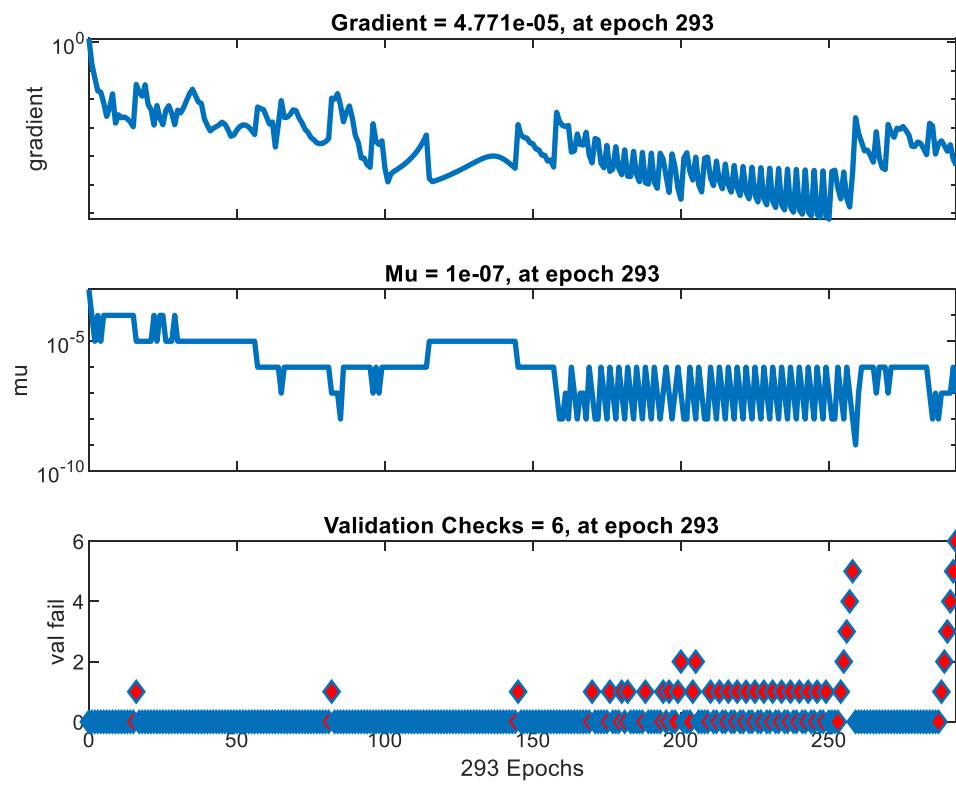
## Fitting with 10 neurons

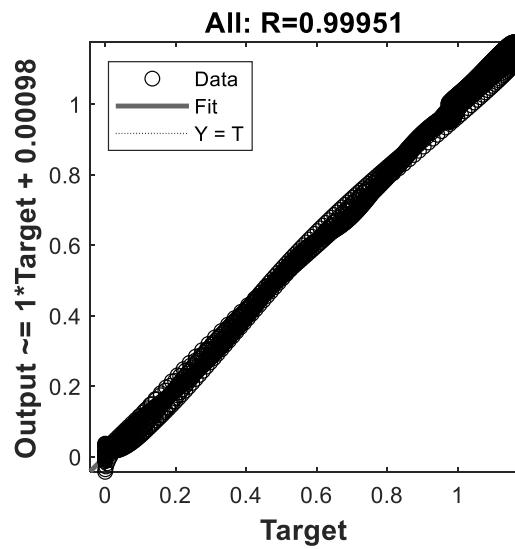
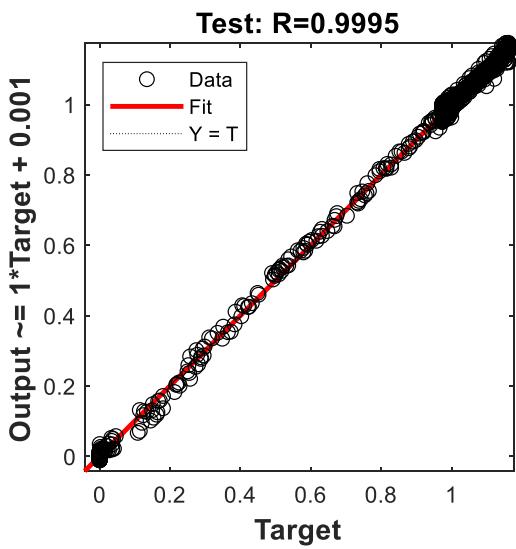
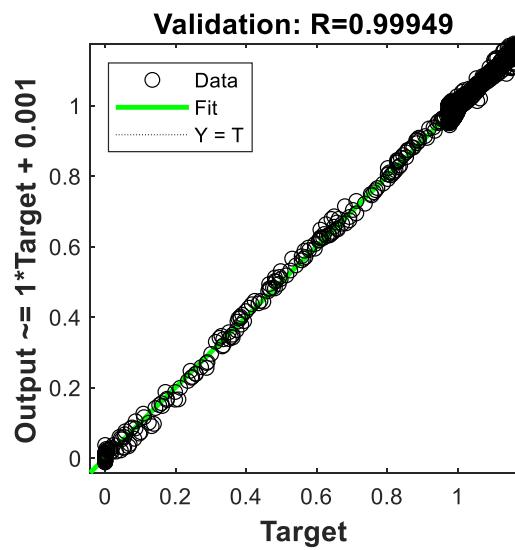
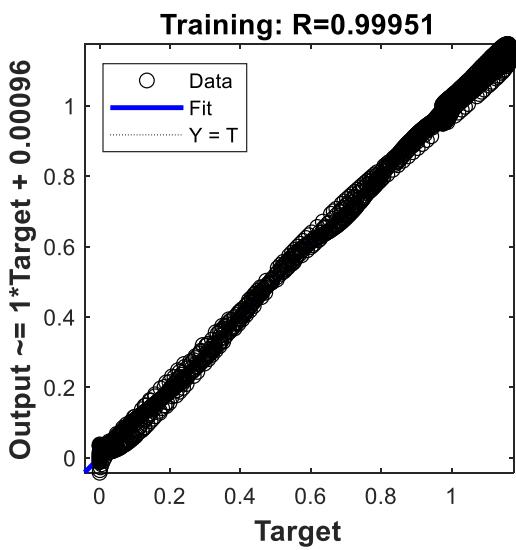
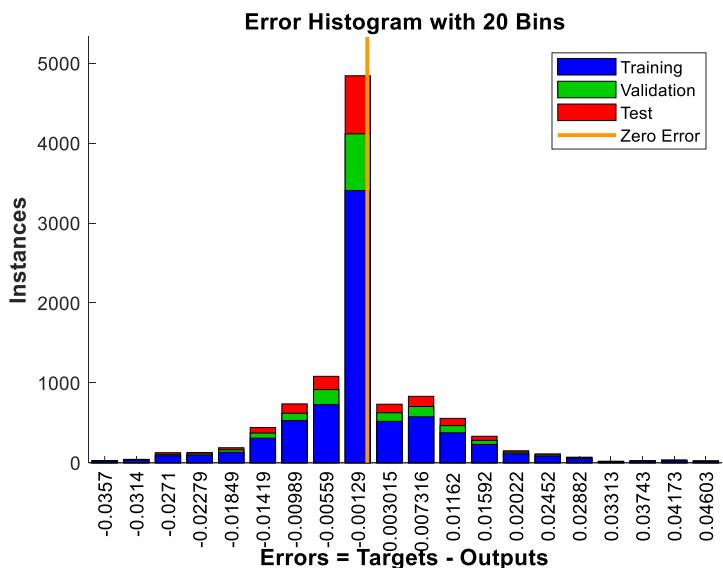


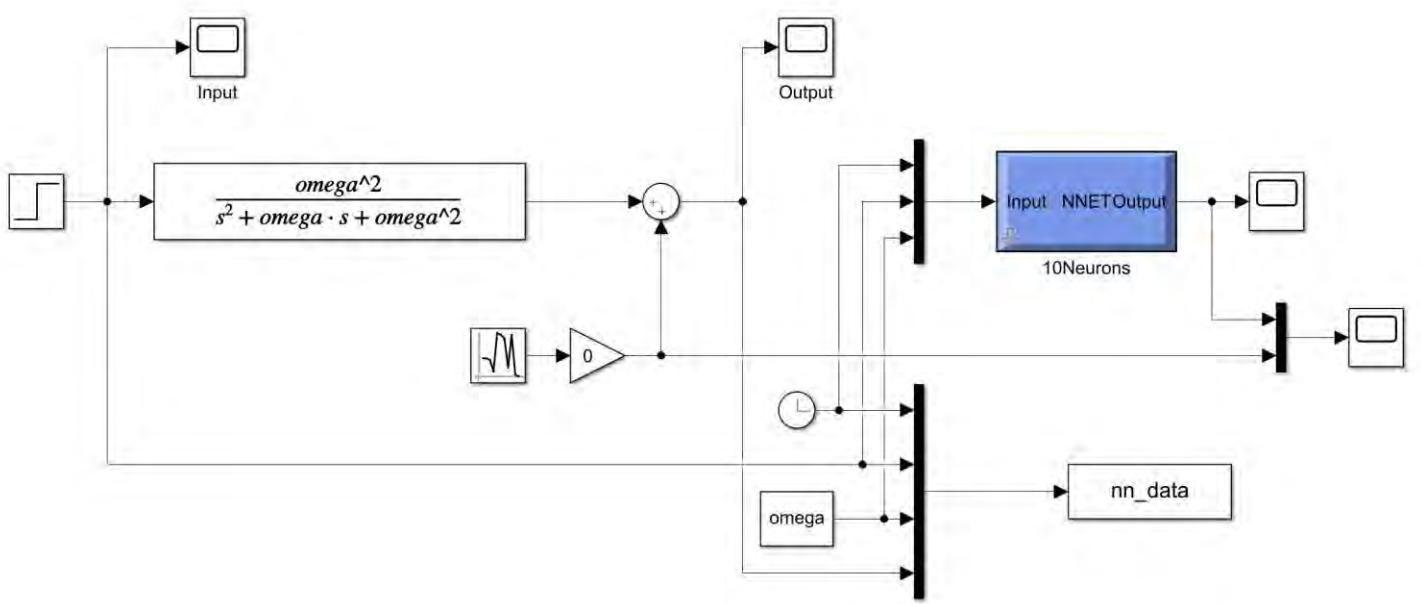




### Training State

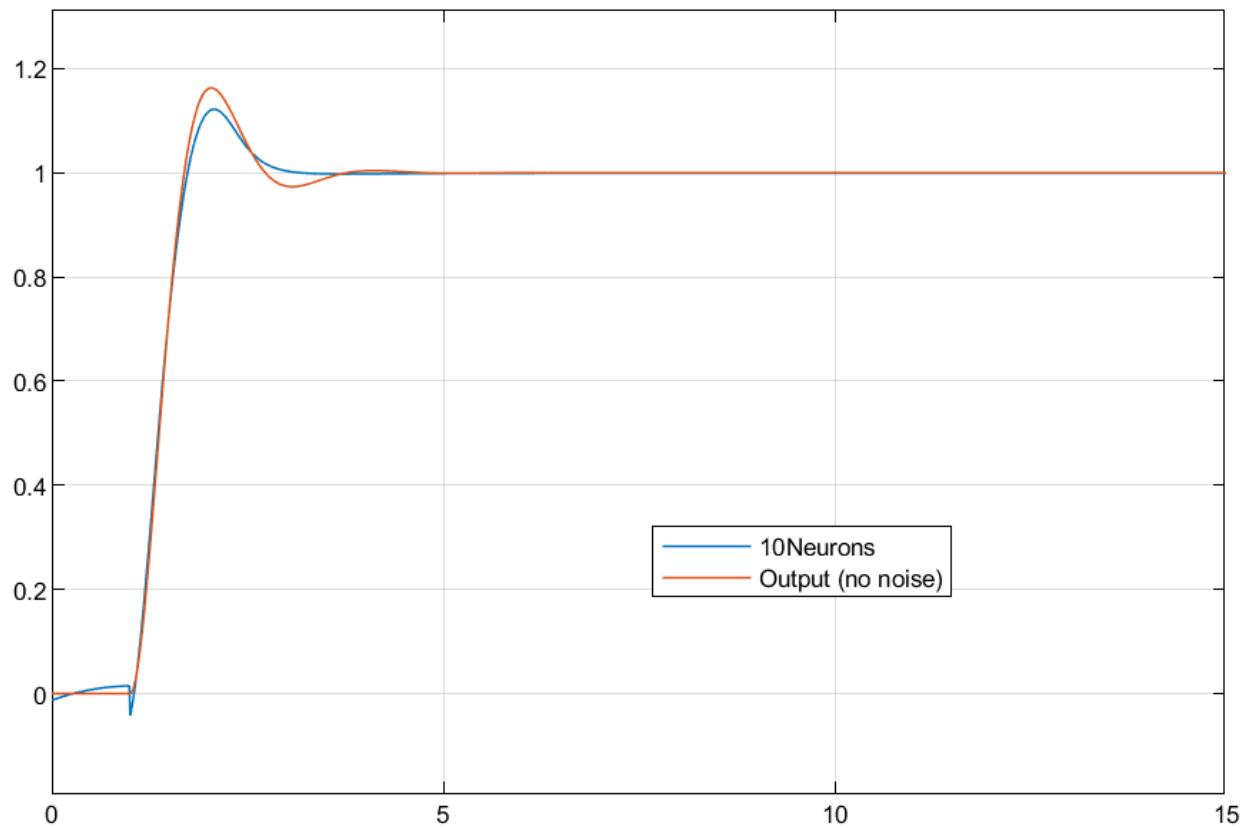




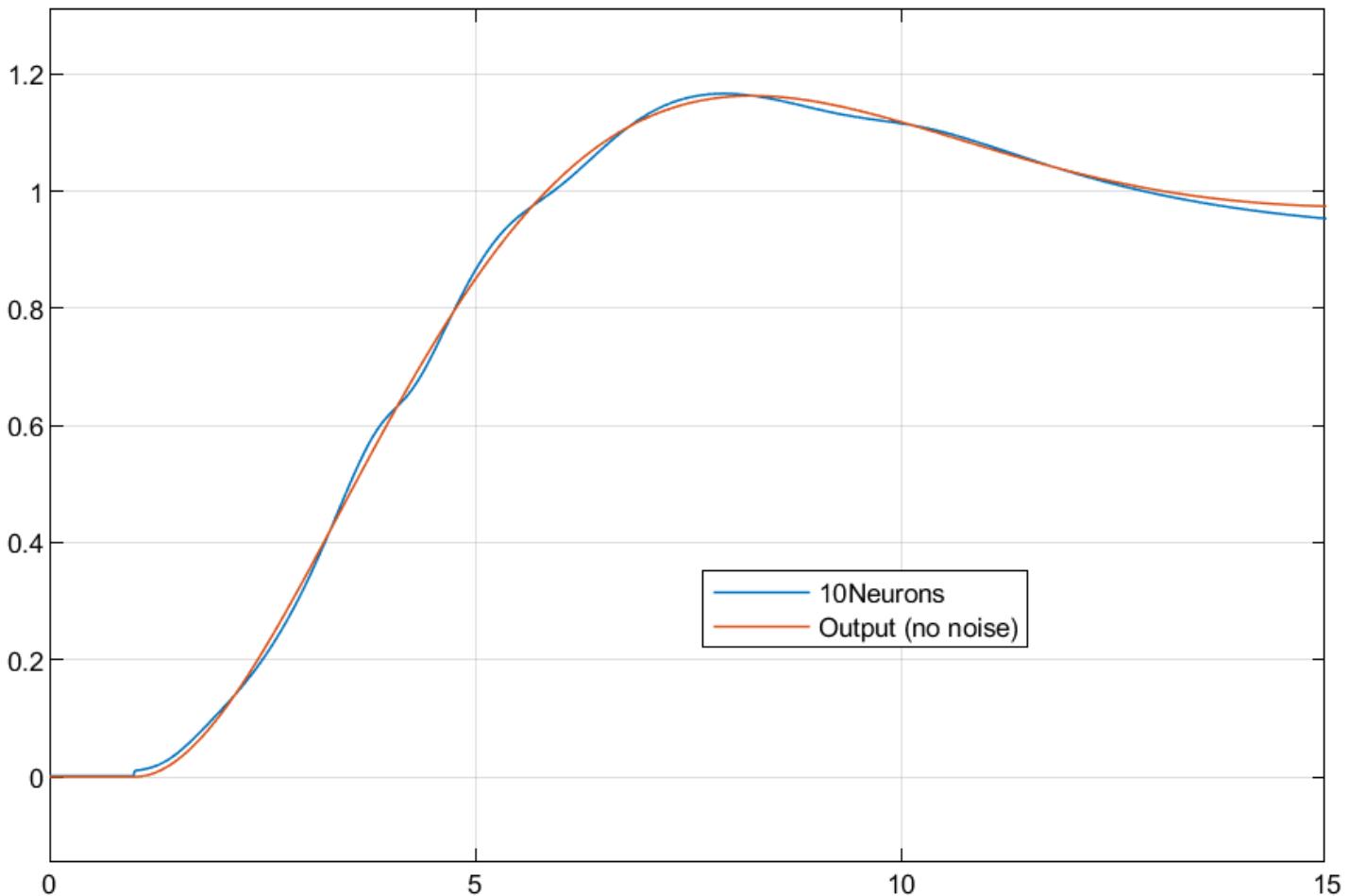


**Simulink Diagram**

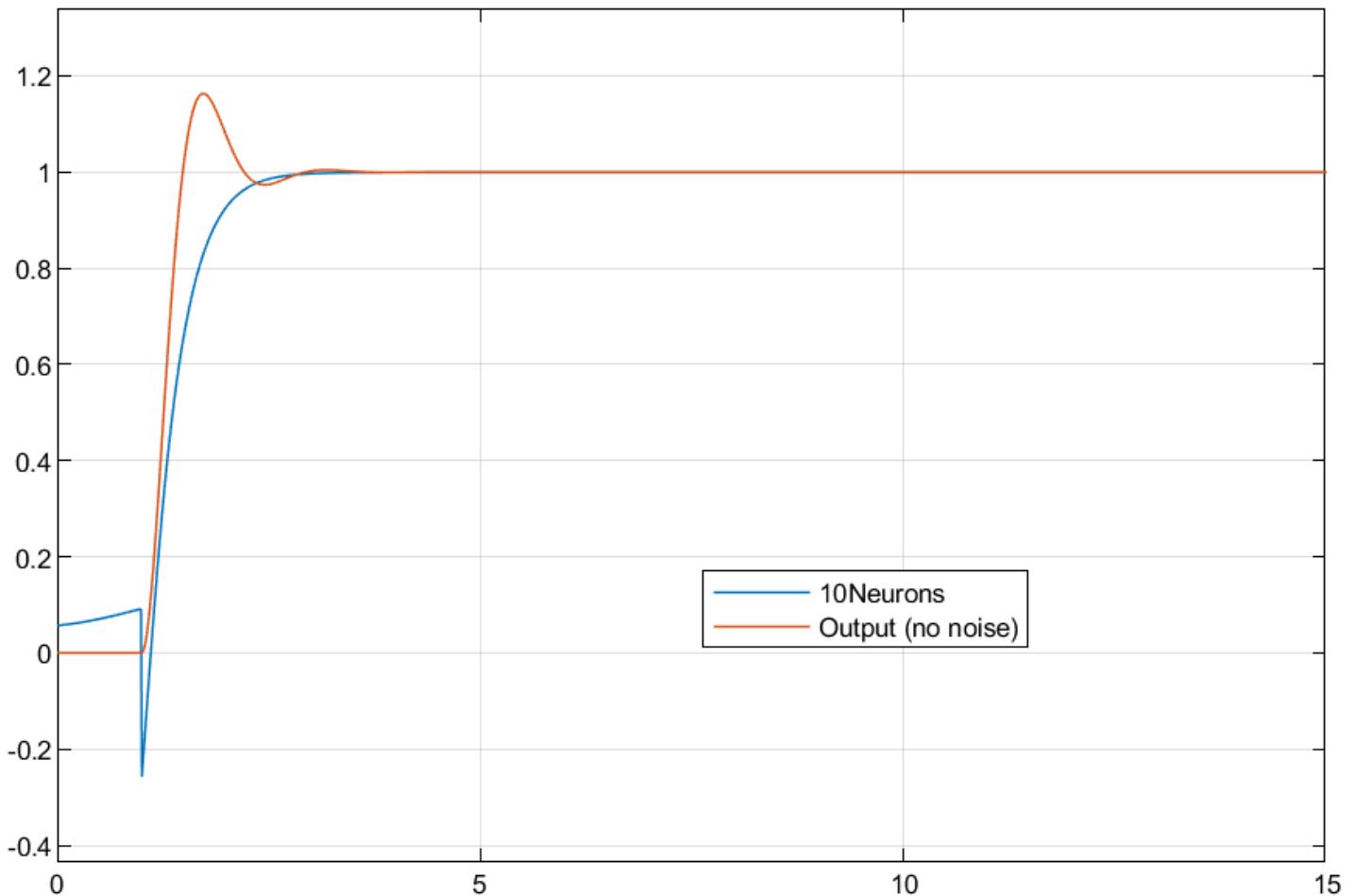
**Output comparison**



It is observed that the neural network approximation is slightly different at the start. The value of omega is at the last defined value of 3.5. On running the simulation again with  $\omega = 0.5$ , the following graph is obtained

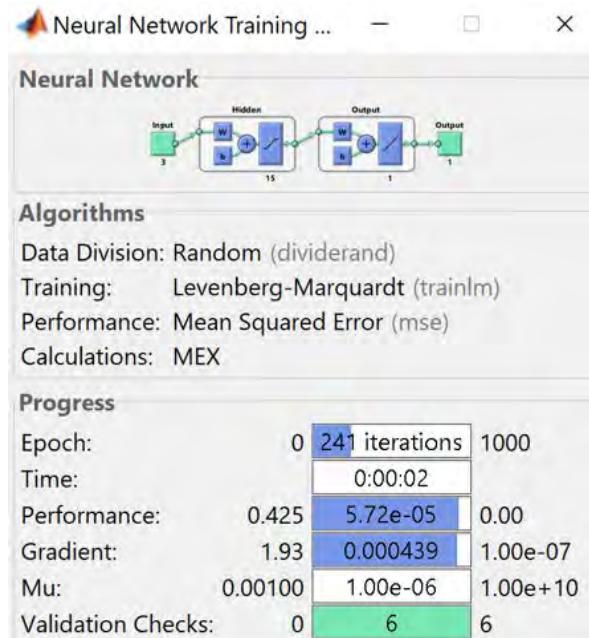


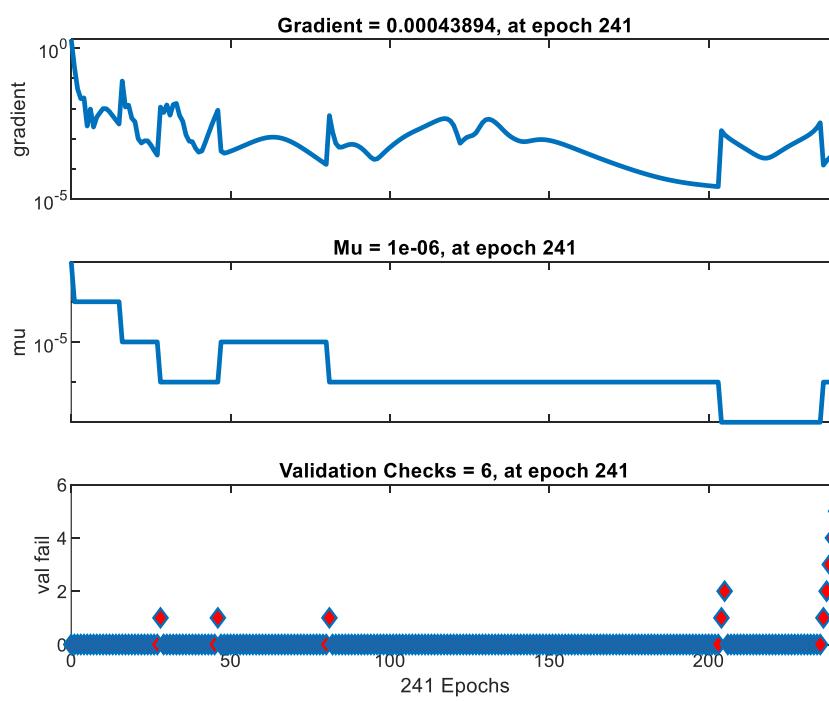
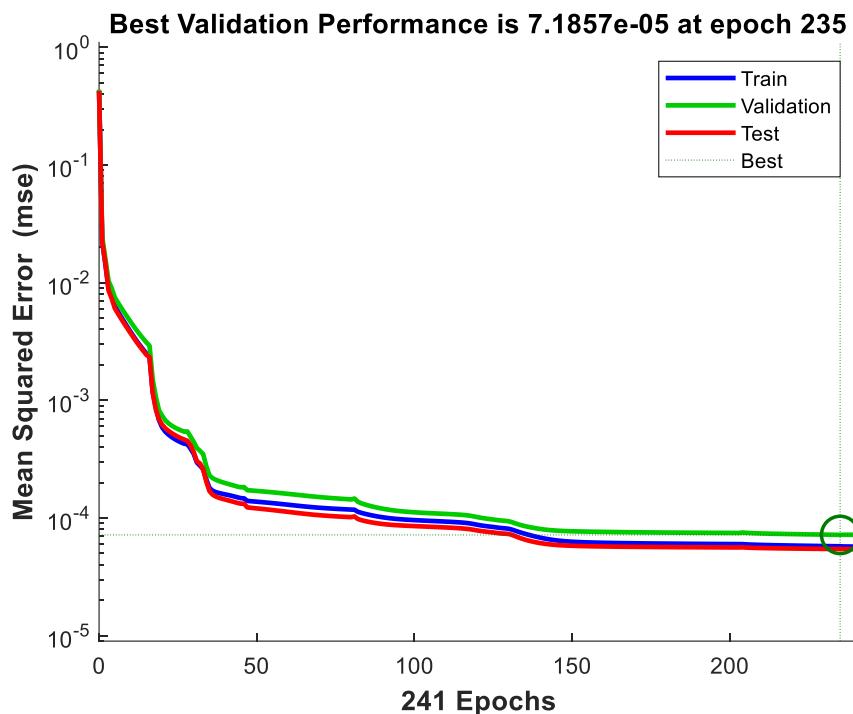
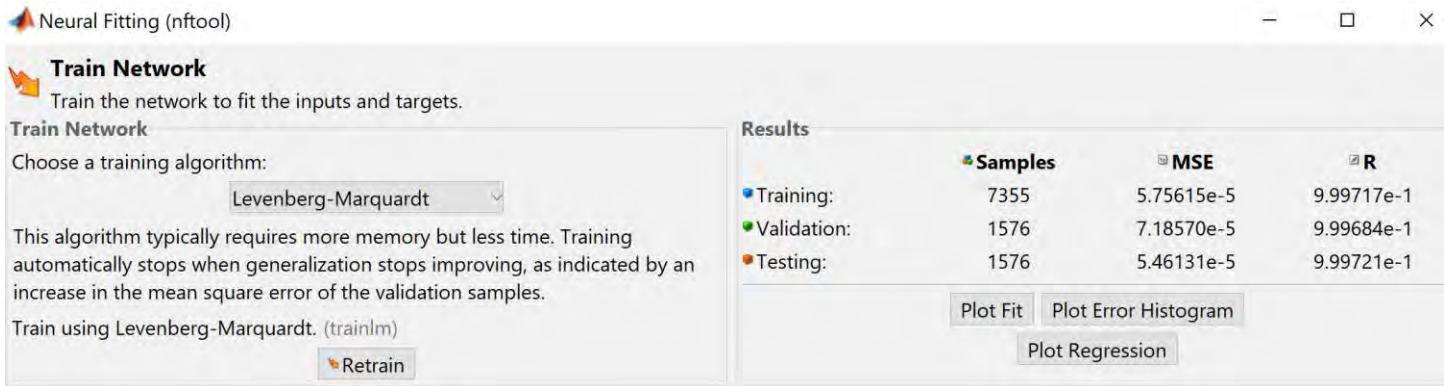
Although the mismatch is still present, we can see that the approximation is very good. Now checking the algorithm for extrapolation, we define  $\omega$  as 5 which is far from all training data.

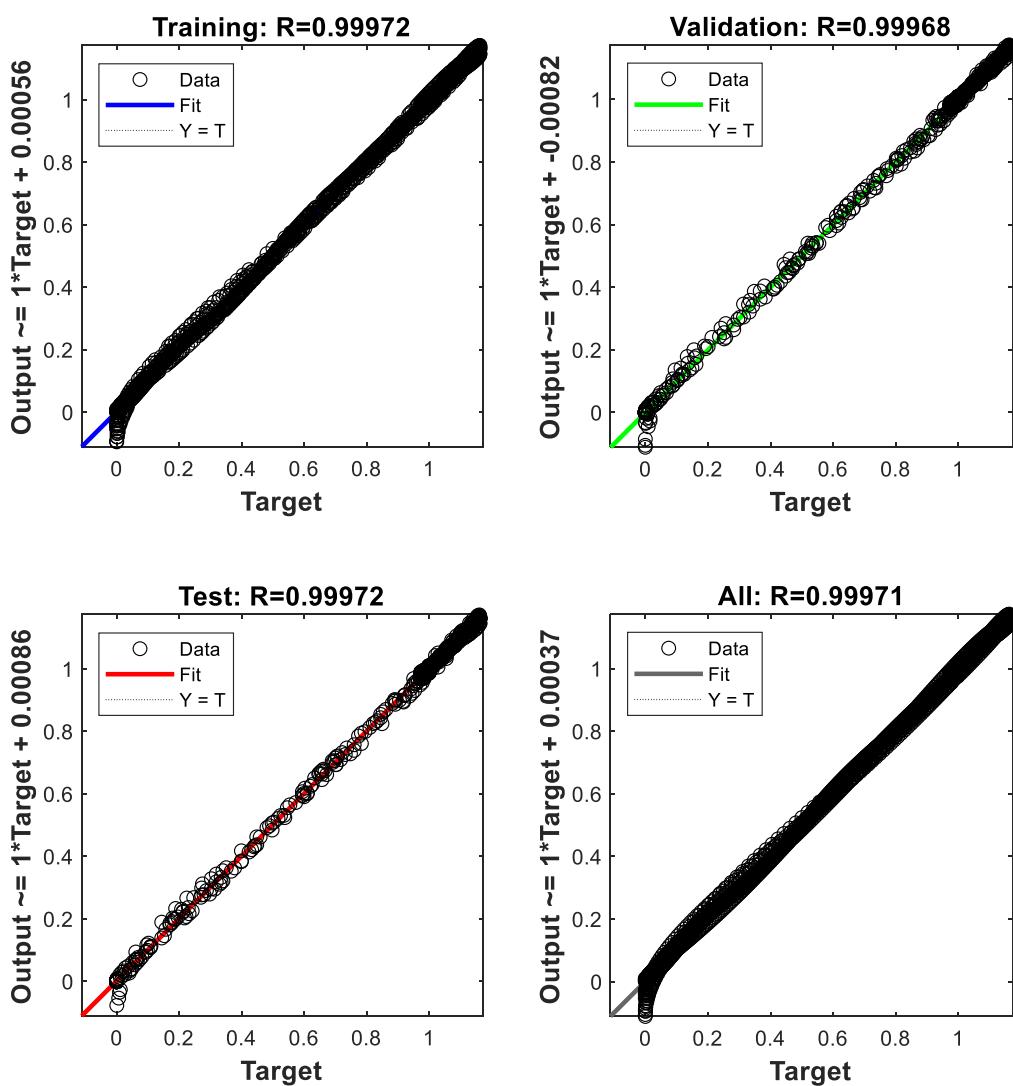
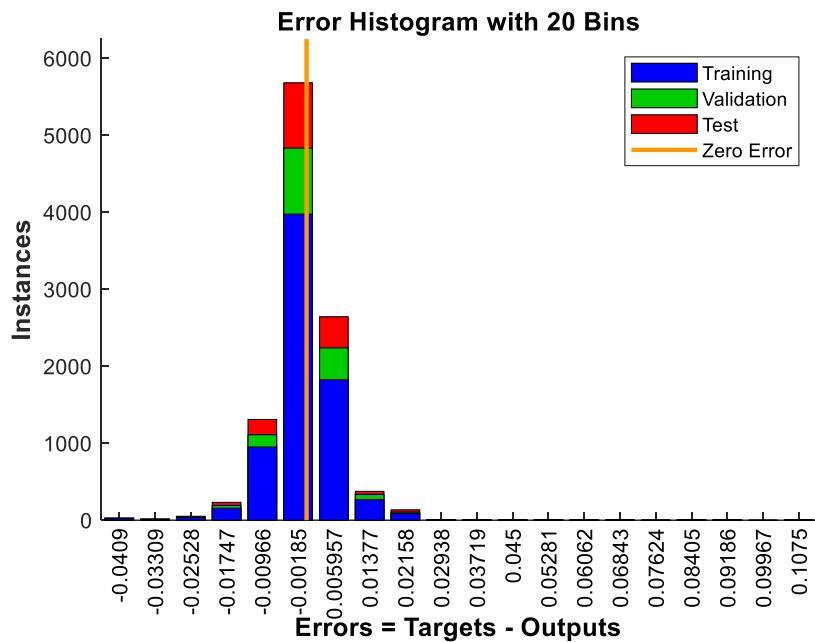


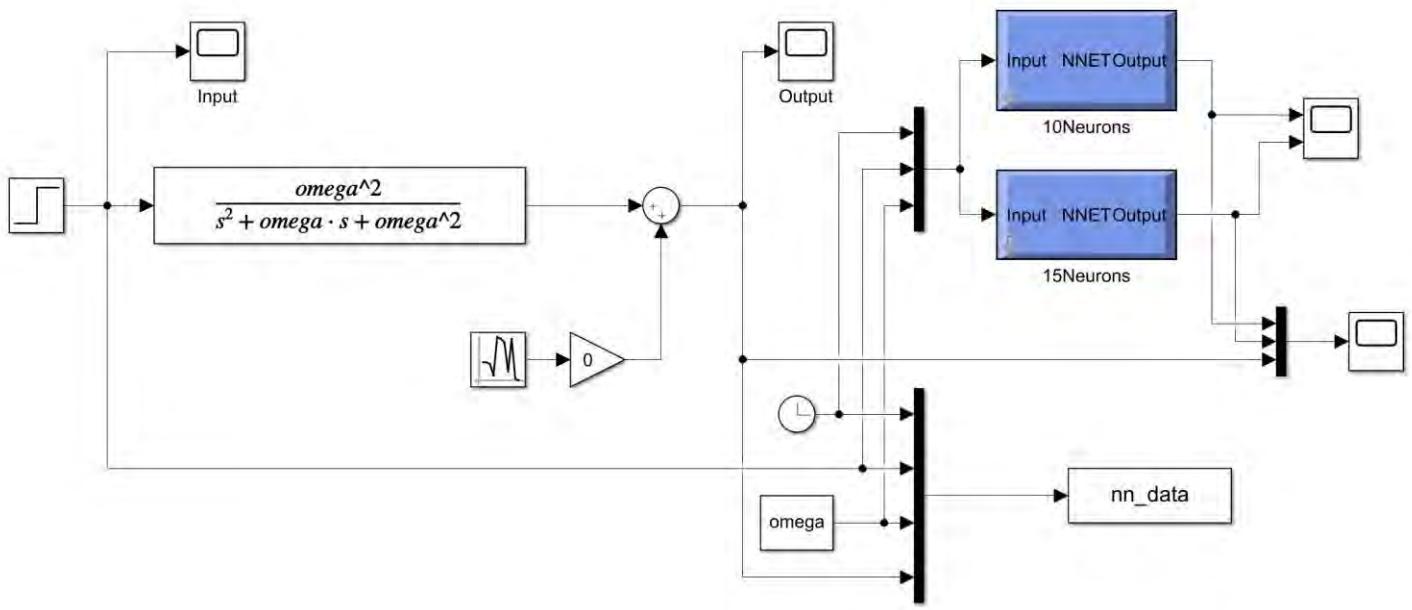
It is observed that the transition state outputs are significantly different although they converge in around 3 seconds which is also the settling time for the system.

### Training with 15 neurons

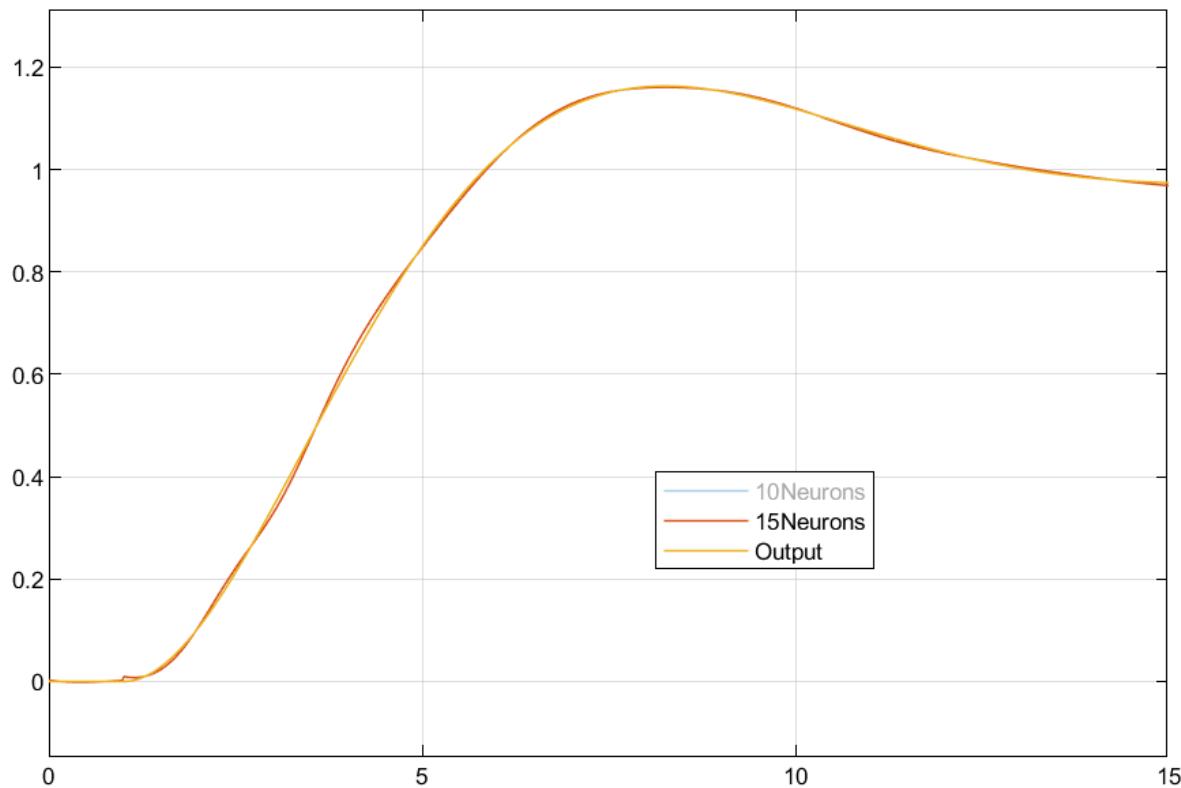




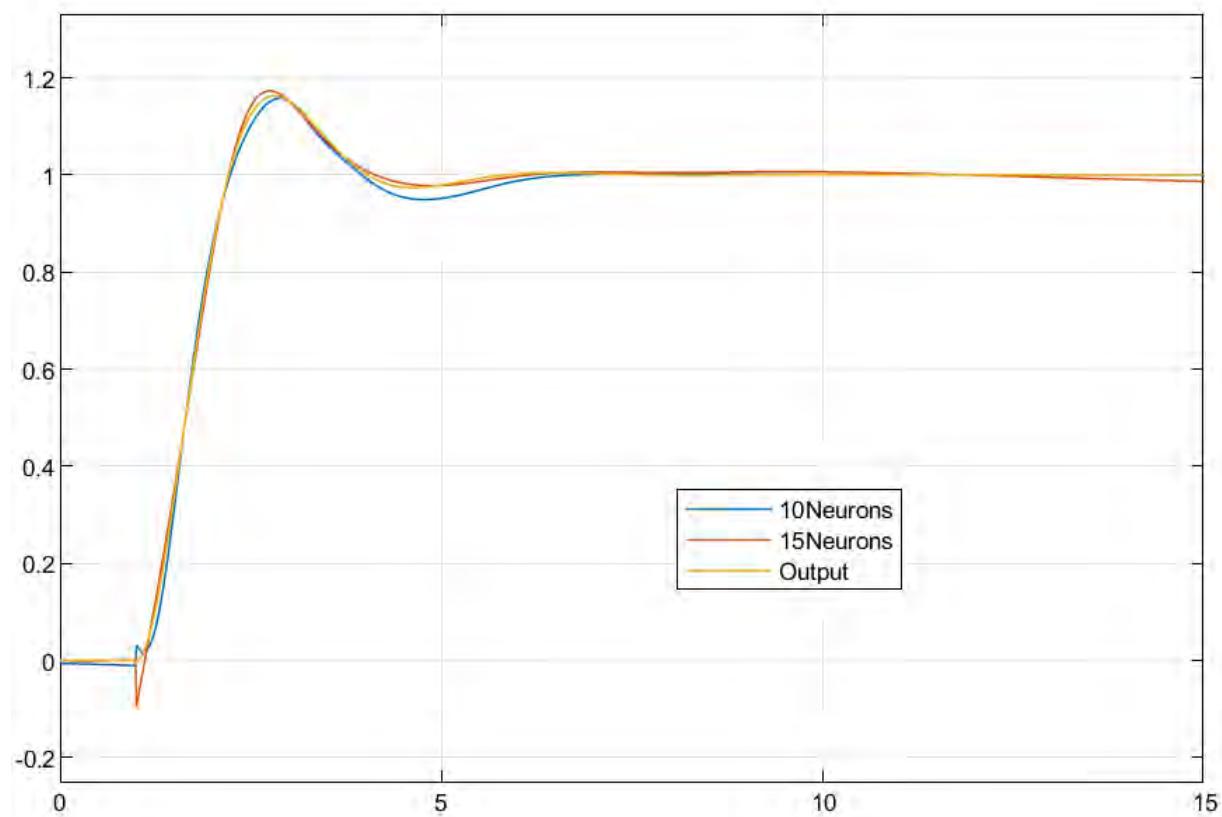




**Simulink Diagram**  
**Output Comparison**

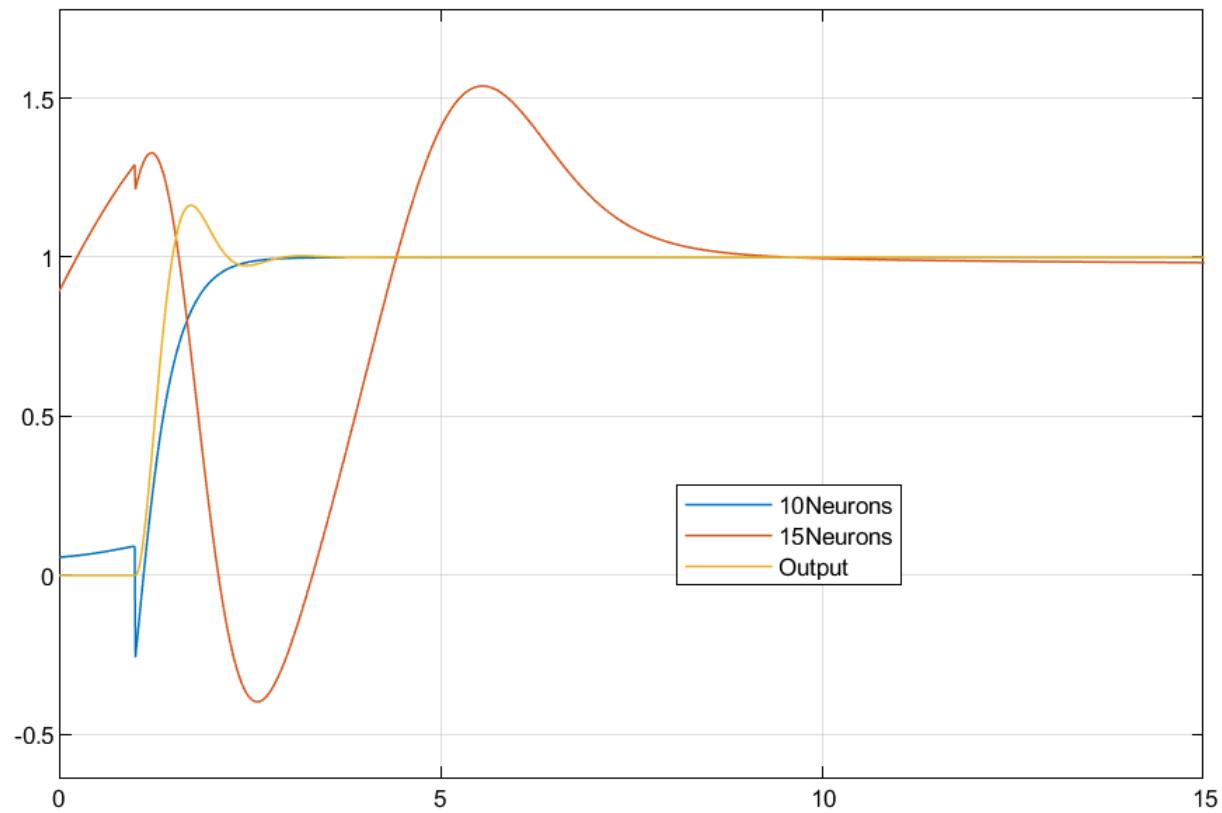


Omega = 0.5



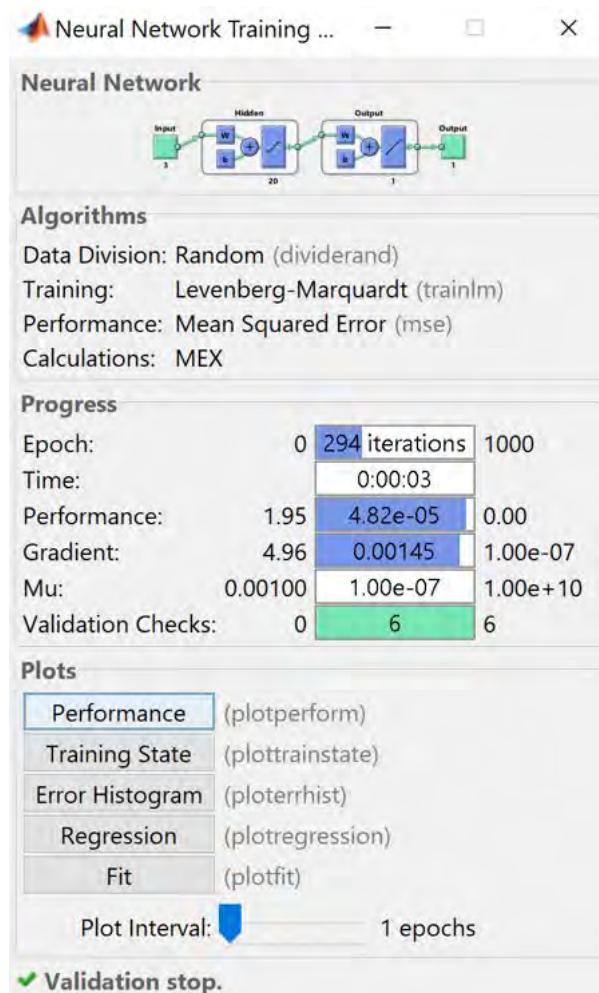
Plot above  $\Omega = 2$

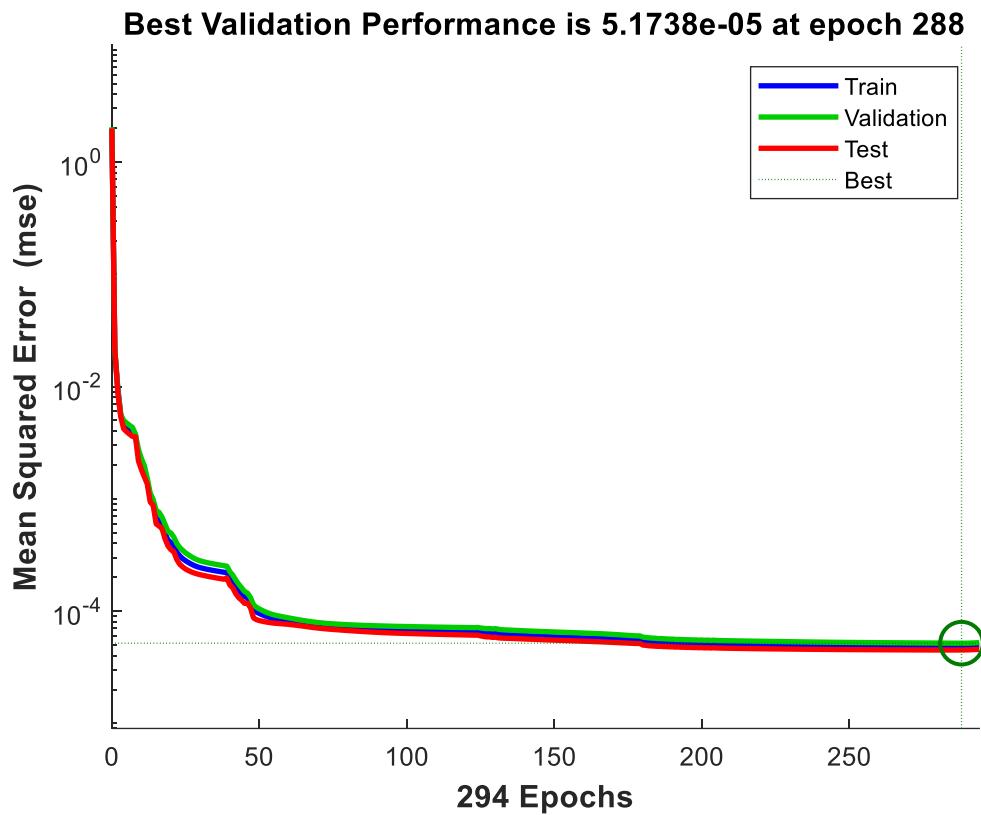
Plot below  $\Omega = 5$



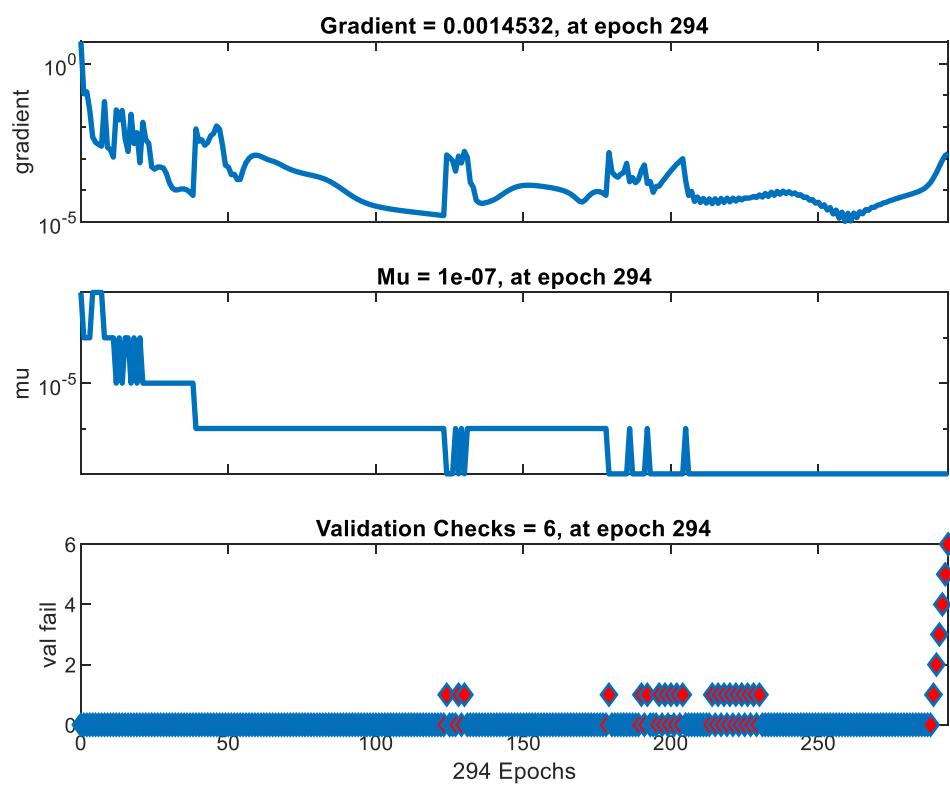
Initial mismatches have increased and the neural network trained with 15 neurons is less efficient at extrapolating  $\Omega$  values as compared to the previous one with 10 neurons.

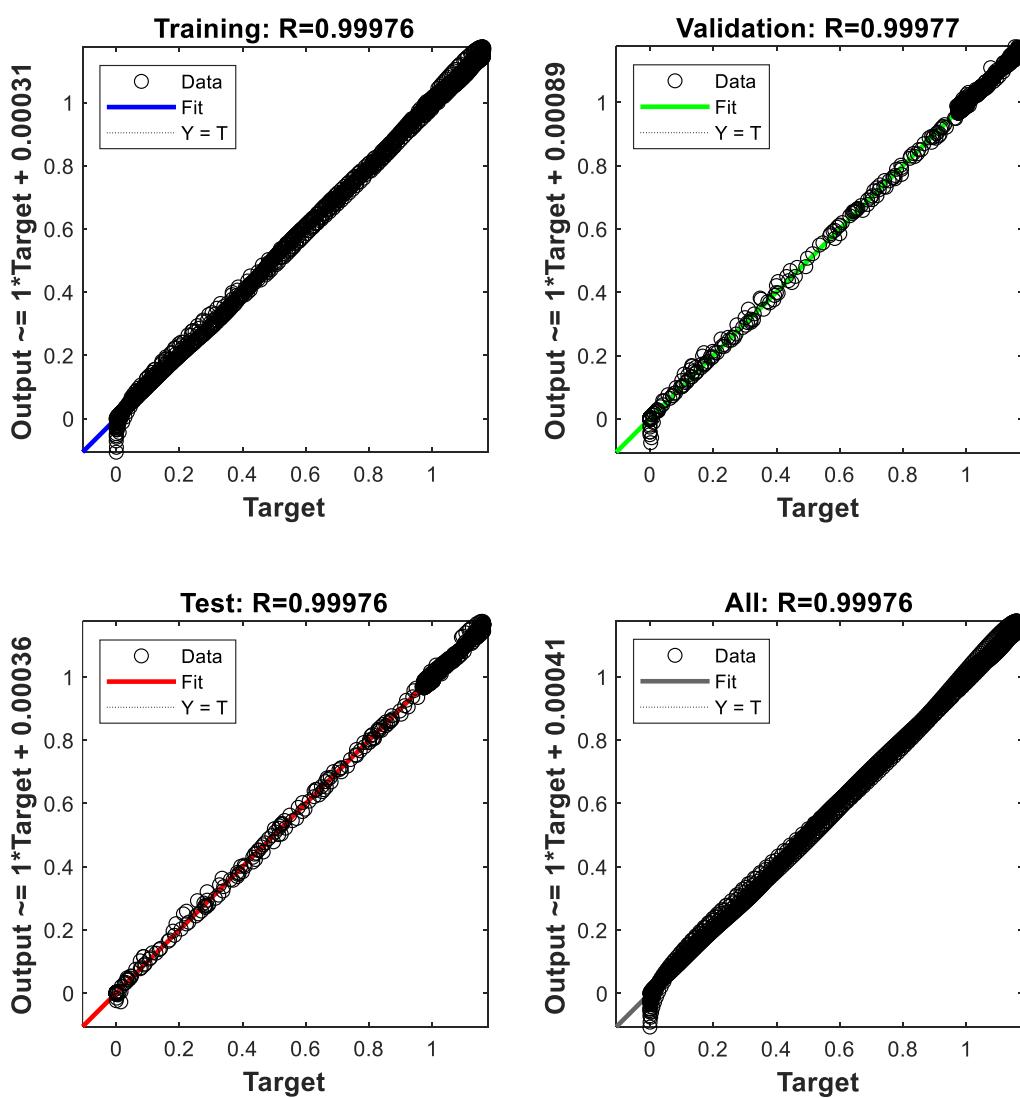
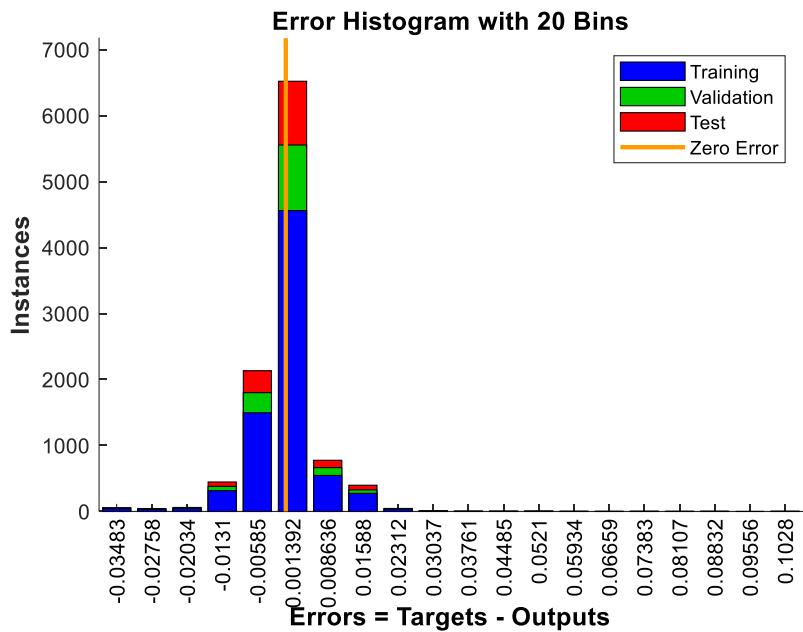
## Training with 20 neurons



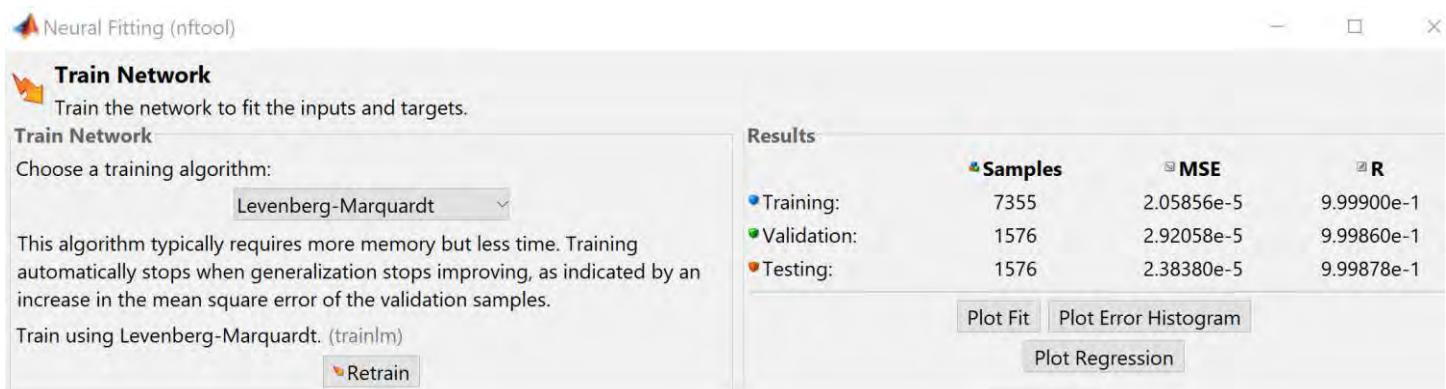


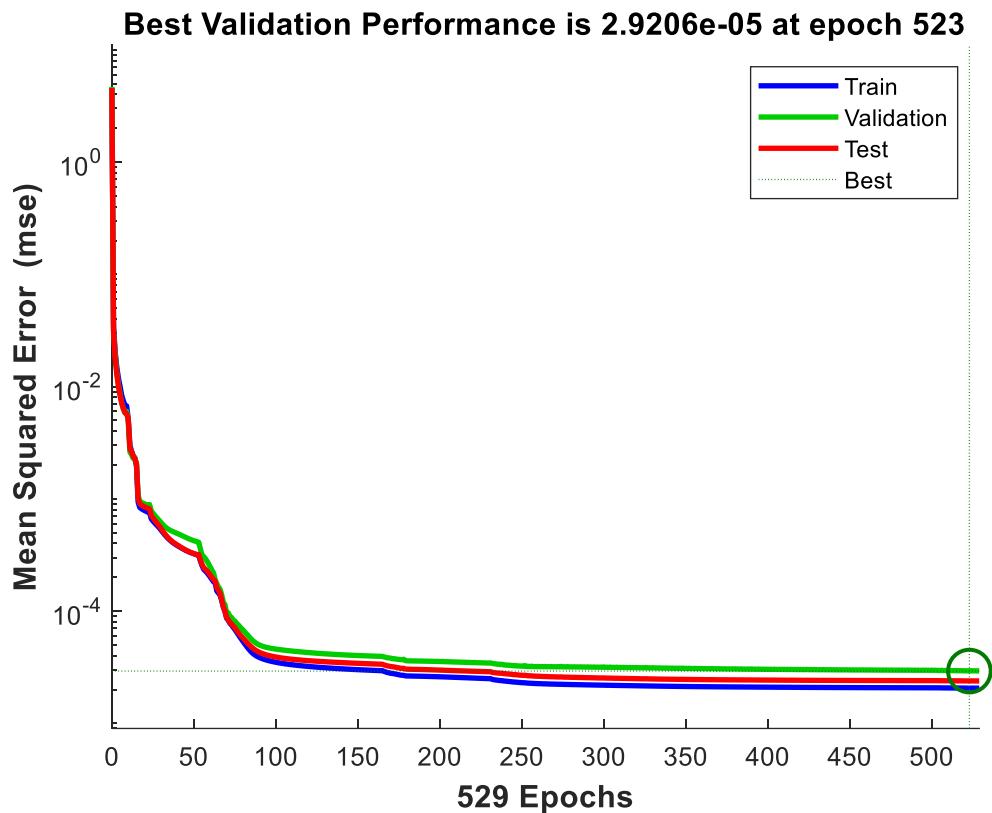
### Training State



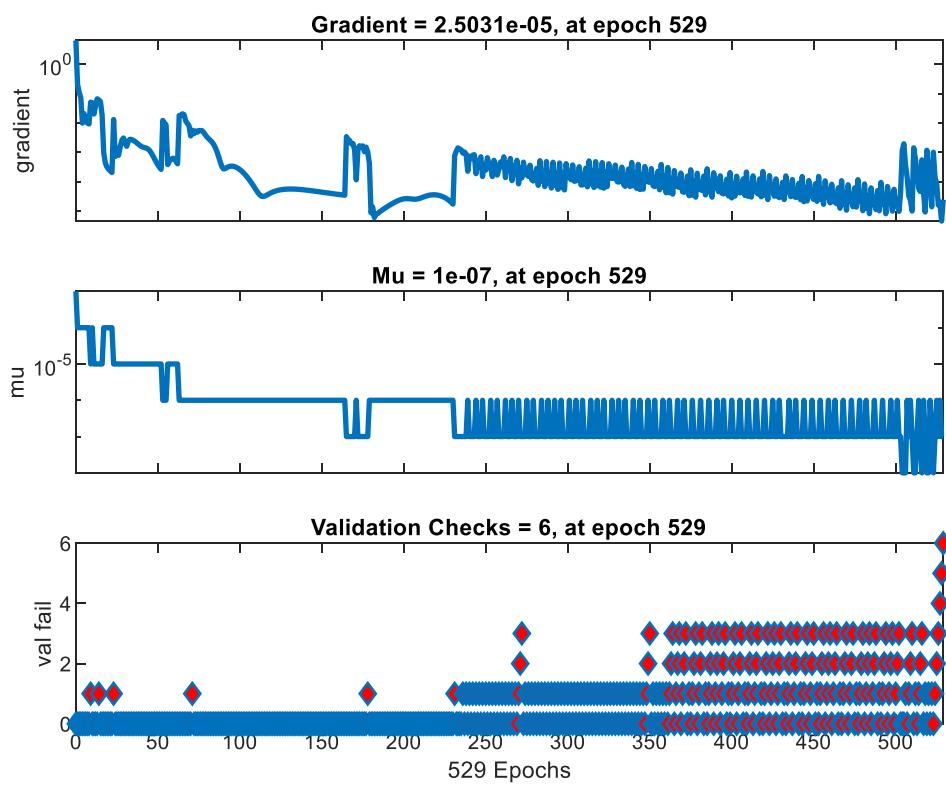


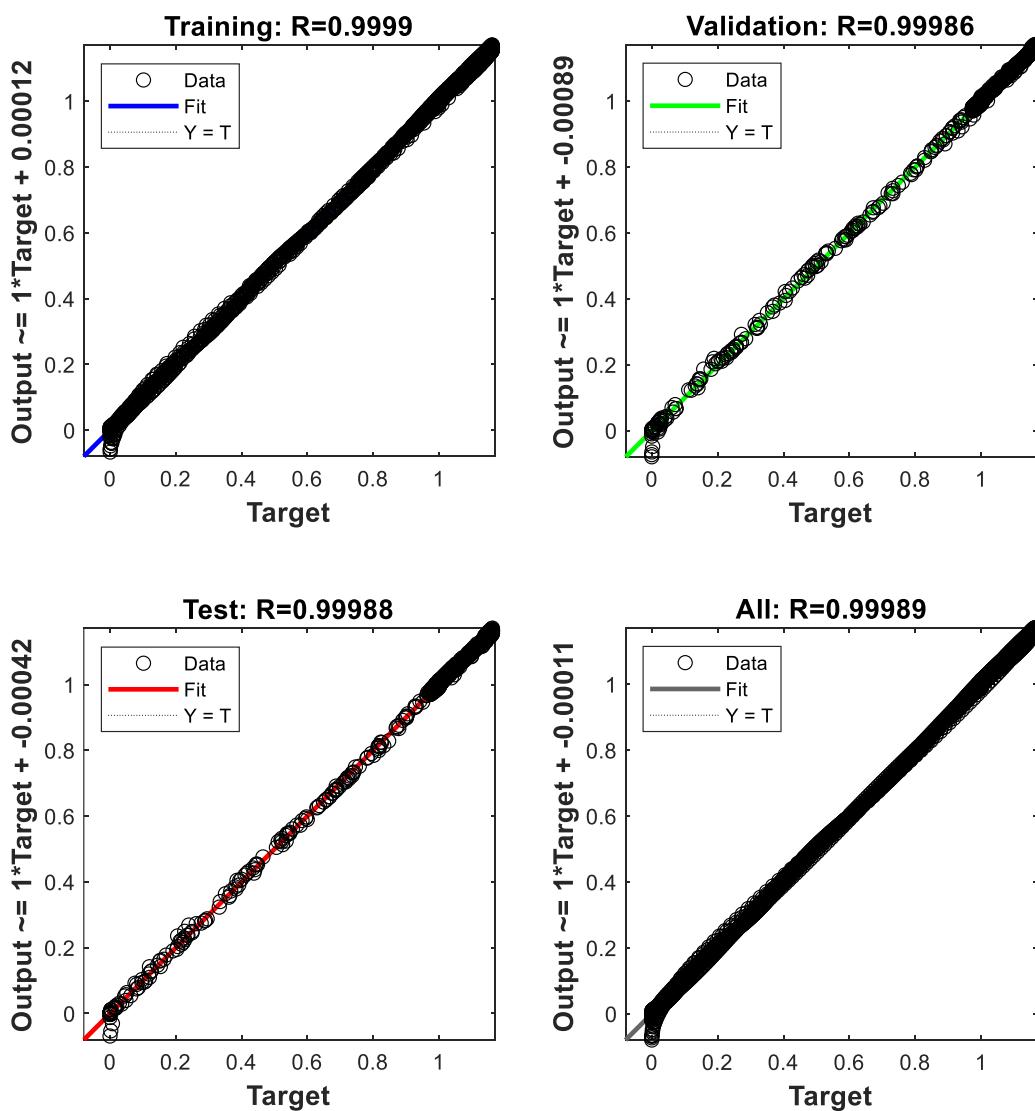
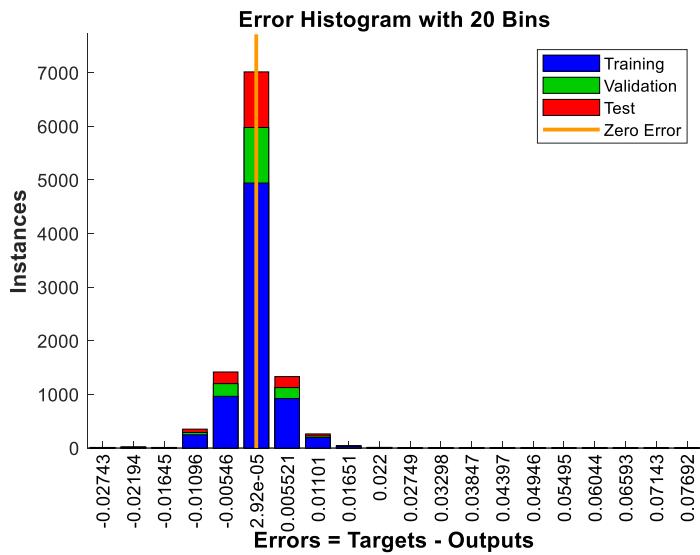
## Training with 25 neurons



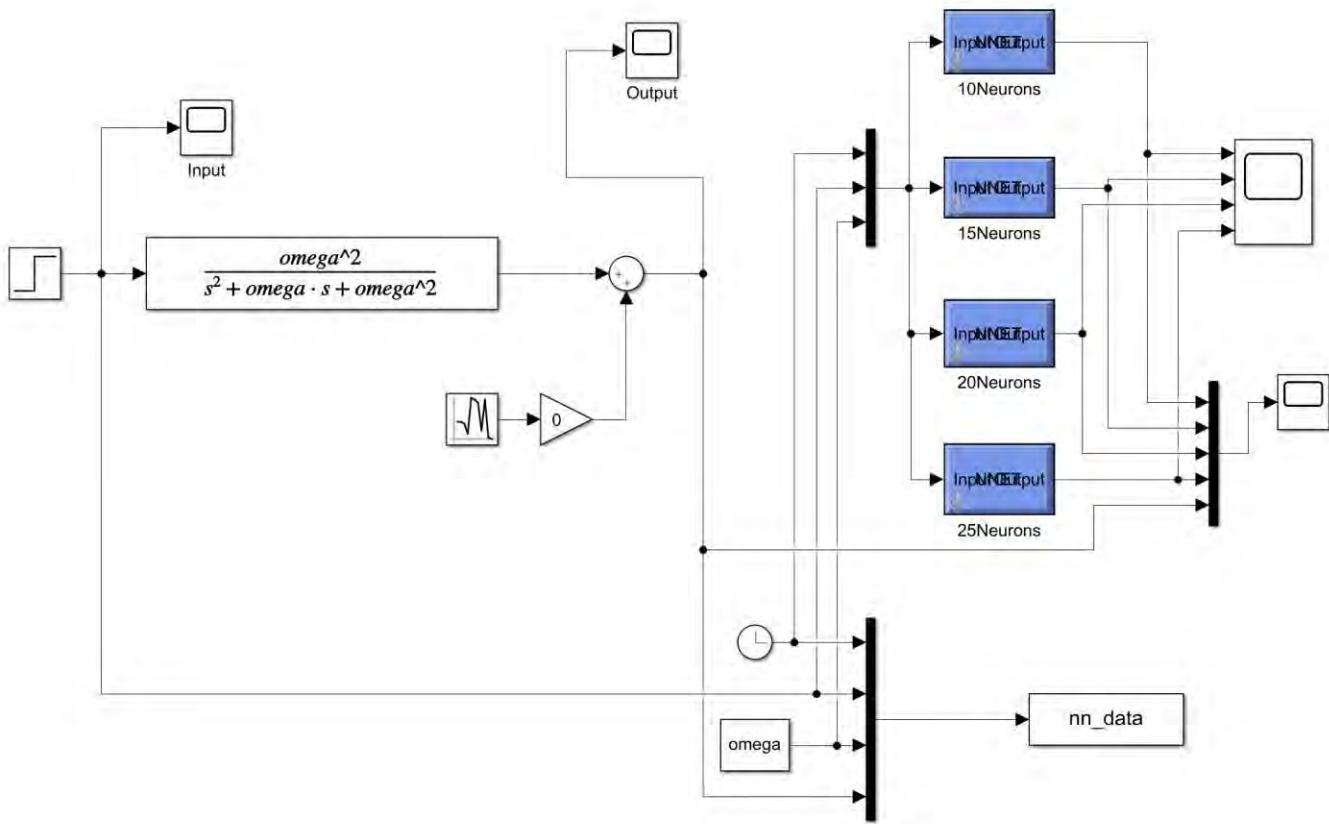


### Training State

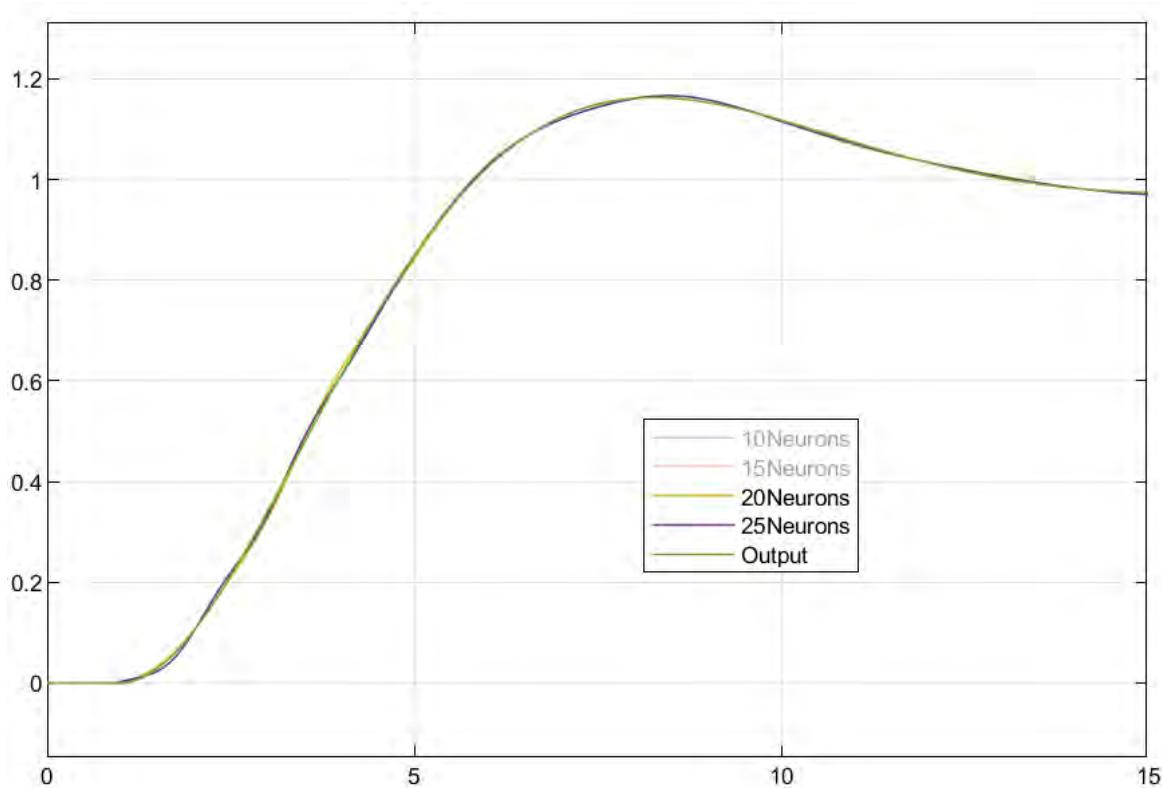




## Output Comparison

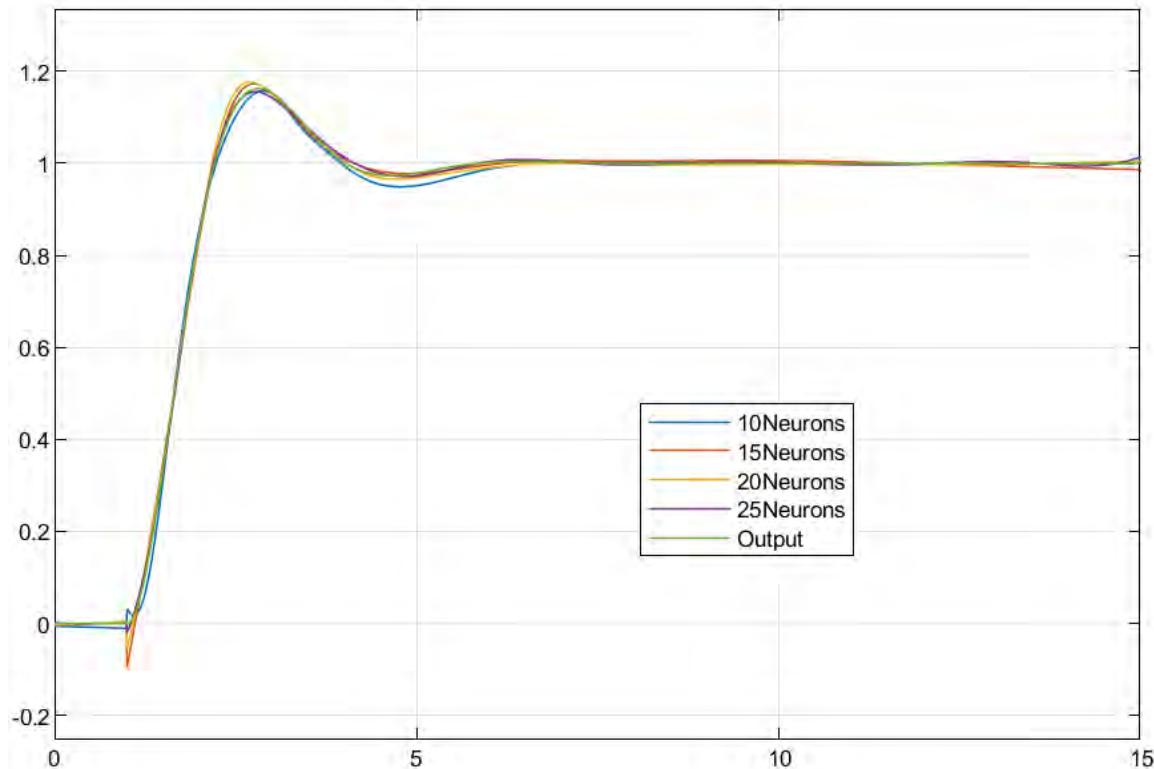


**Simulink Diagram**



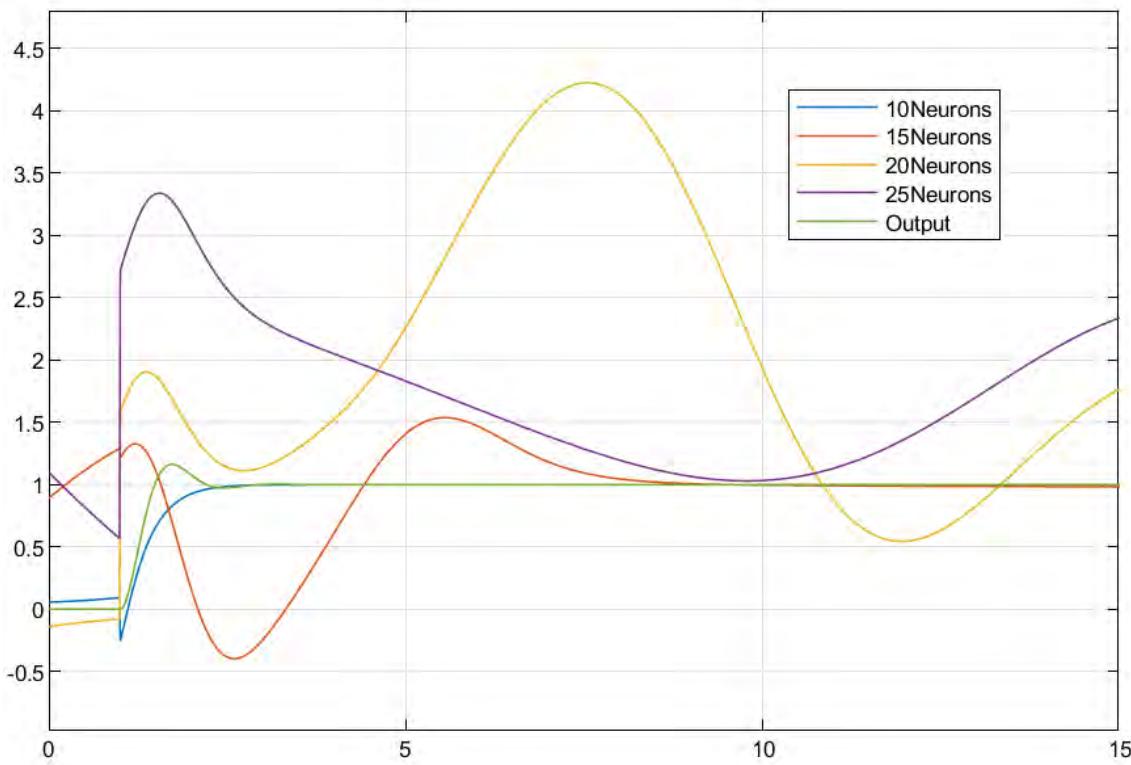
**Plot for  $\omega = 0.5$**

It is observed that as the number of neurons is increased to 20 and 25, the fit becomes better with reduced mismatch in the initial stages for  $\omega = 0.5$ .



Plot above  $\omega = 2$

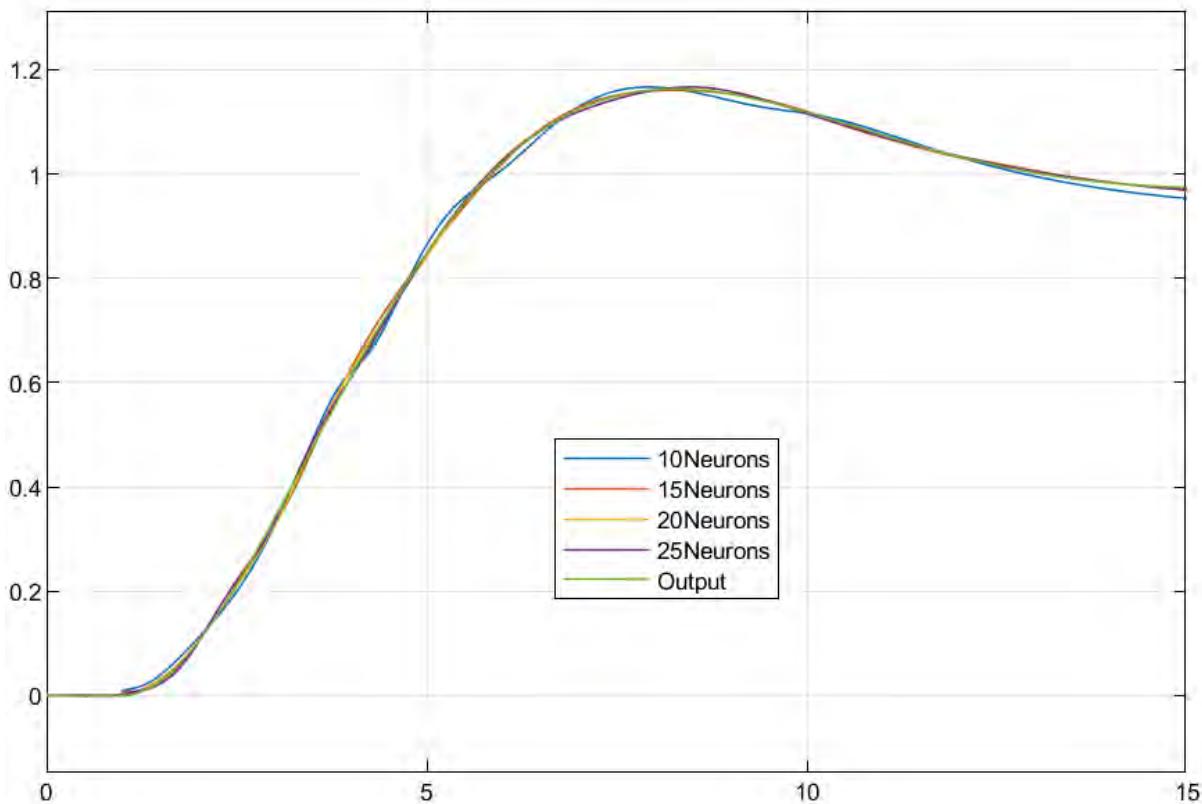
Plot below,  $\omega = 5$



More neurons are good for fitting if much extrapolation is not required. However, since more neurons accommodate more information, they might fit unnecessarily complex functions.

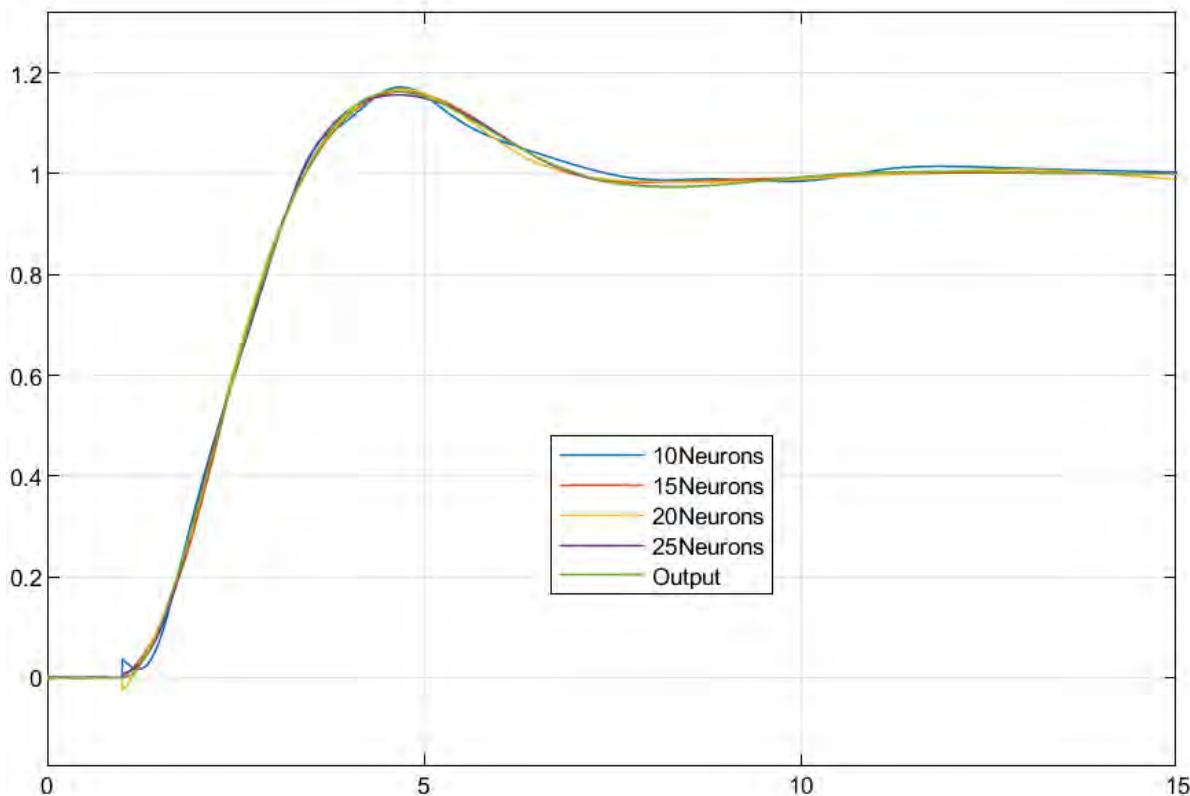
## Extended Comparison

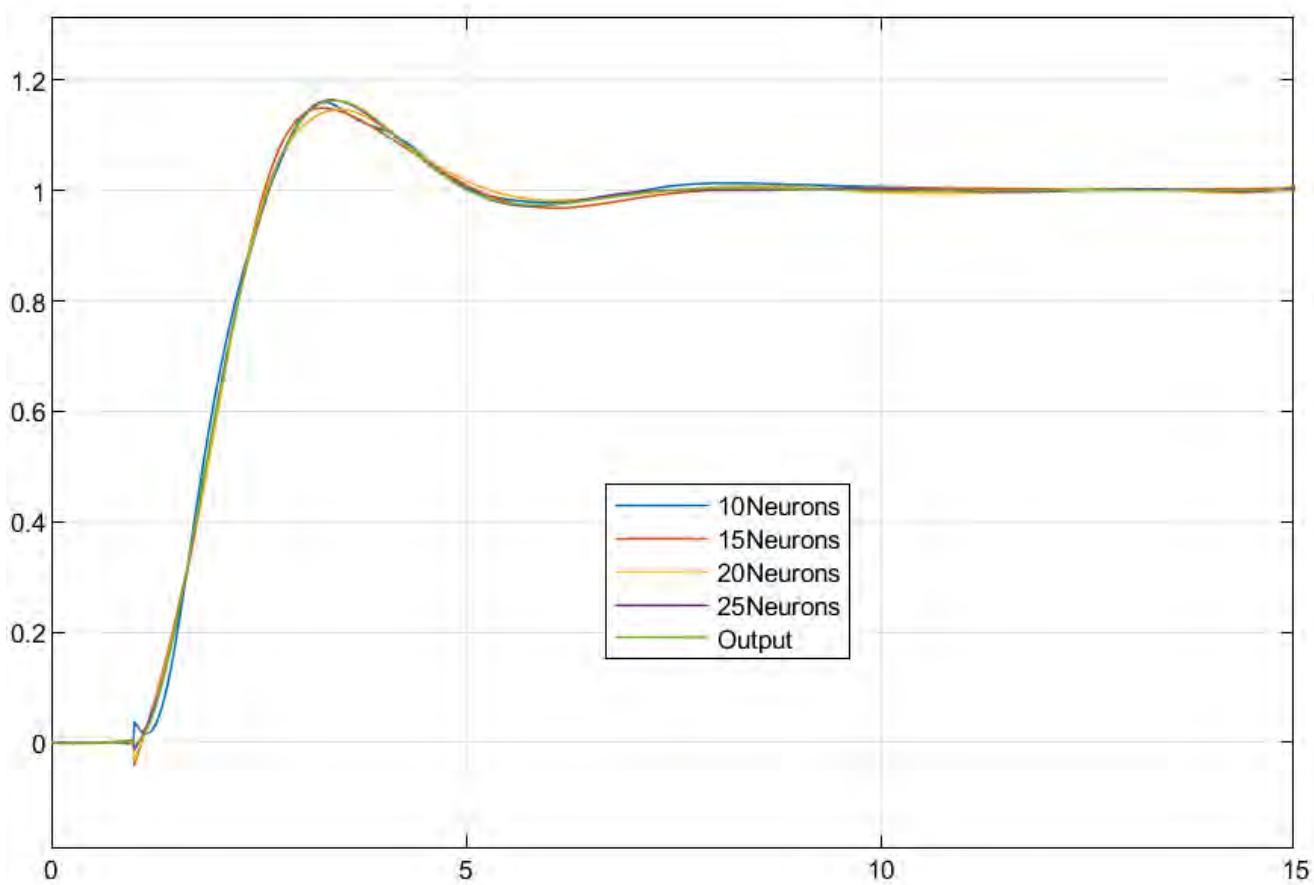
This part merely compares all the fits at all the given omega values for training.



Plot above, Omega = 0.5

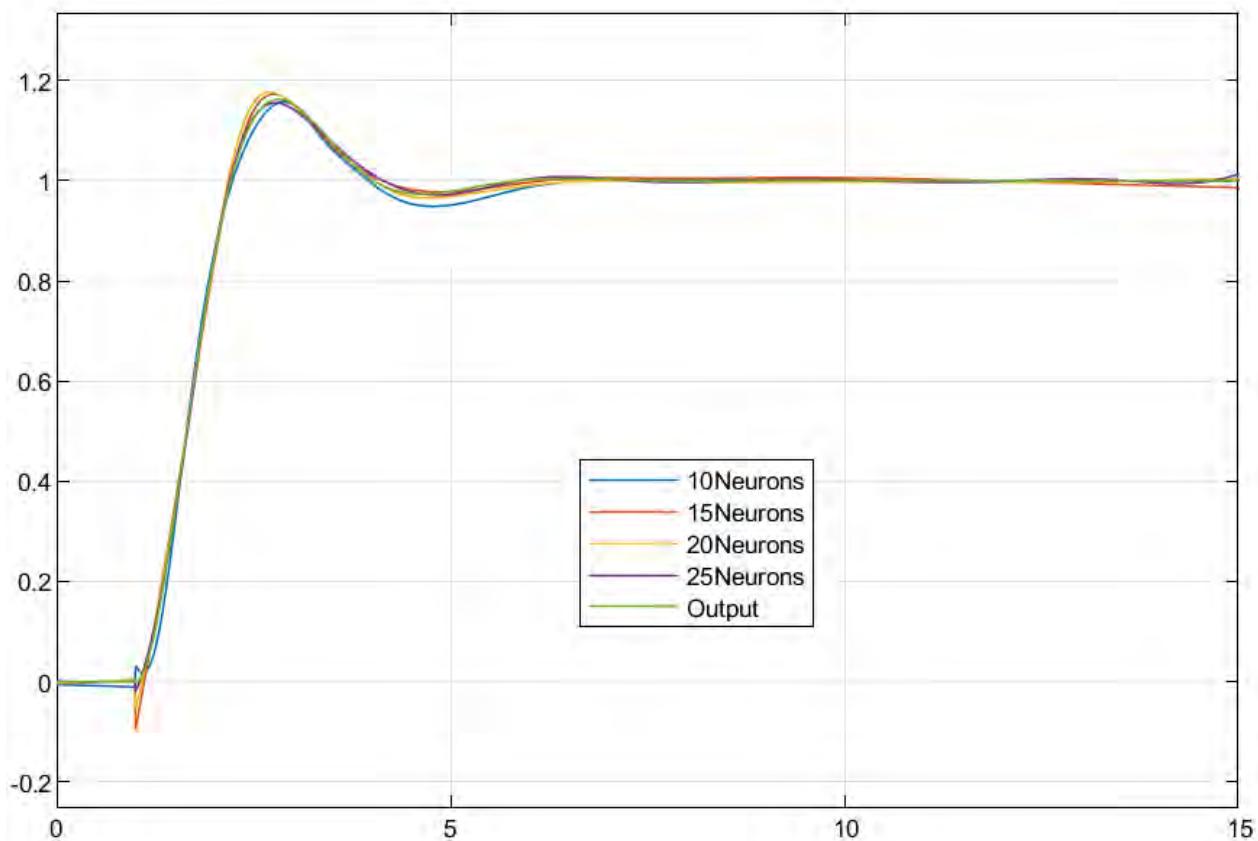
Plot below, Omega = 1

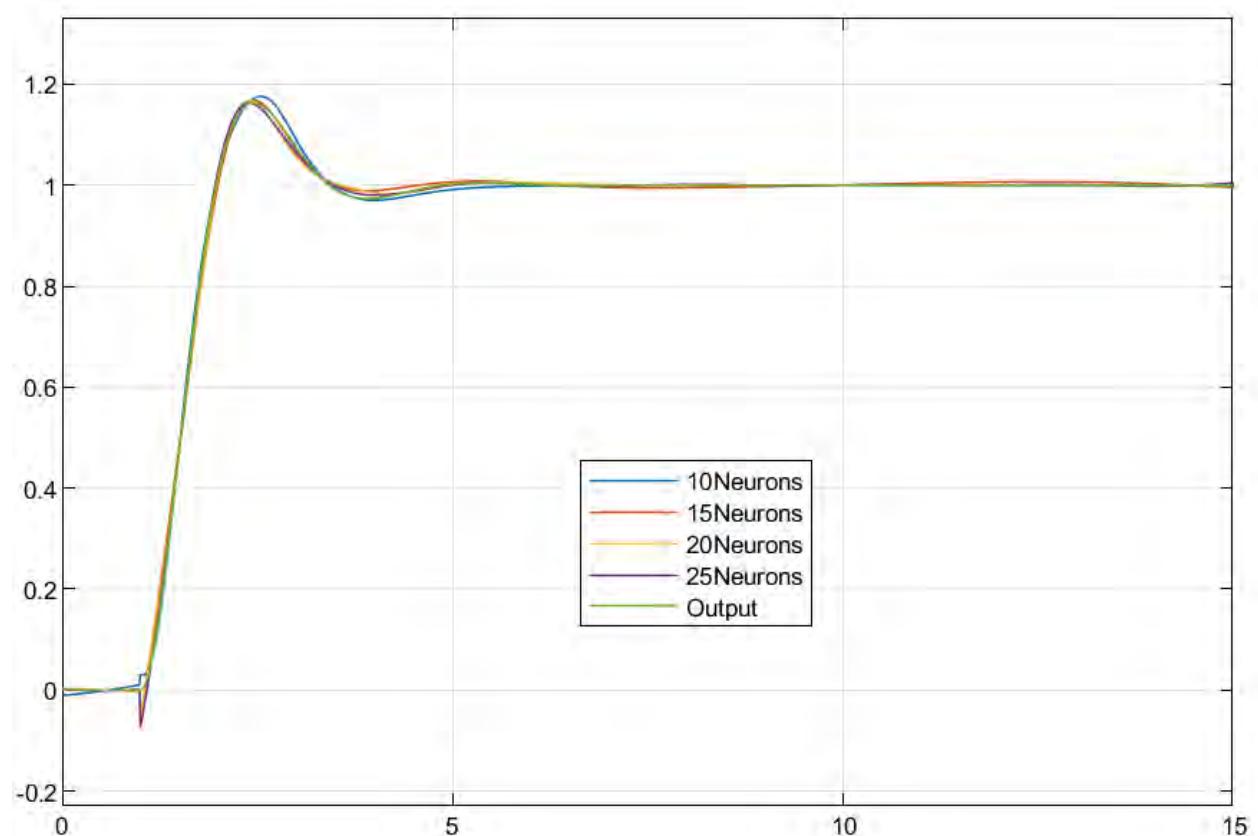




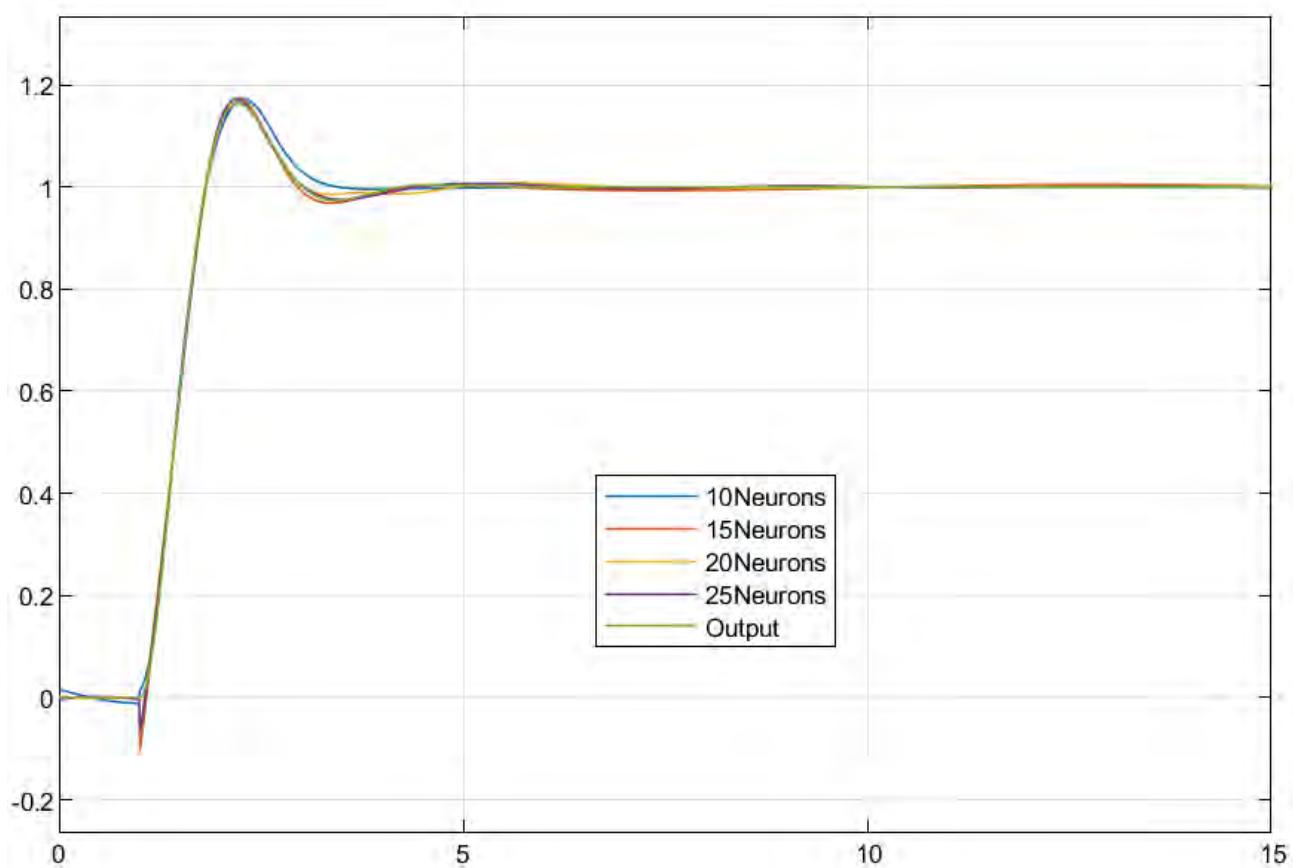
Plot above,  $\omega = 1.5$

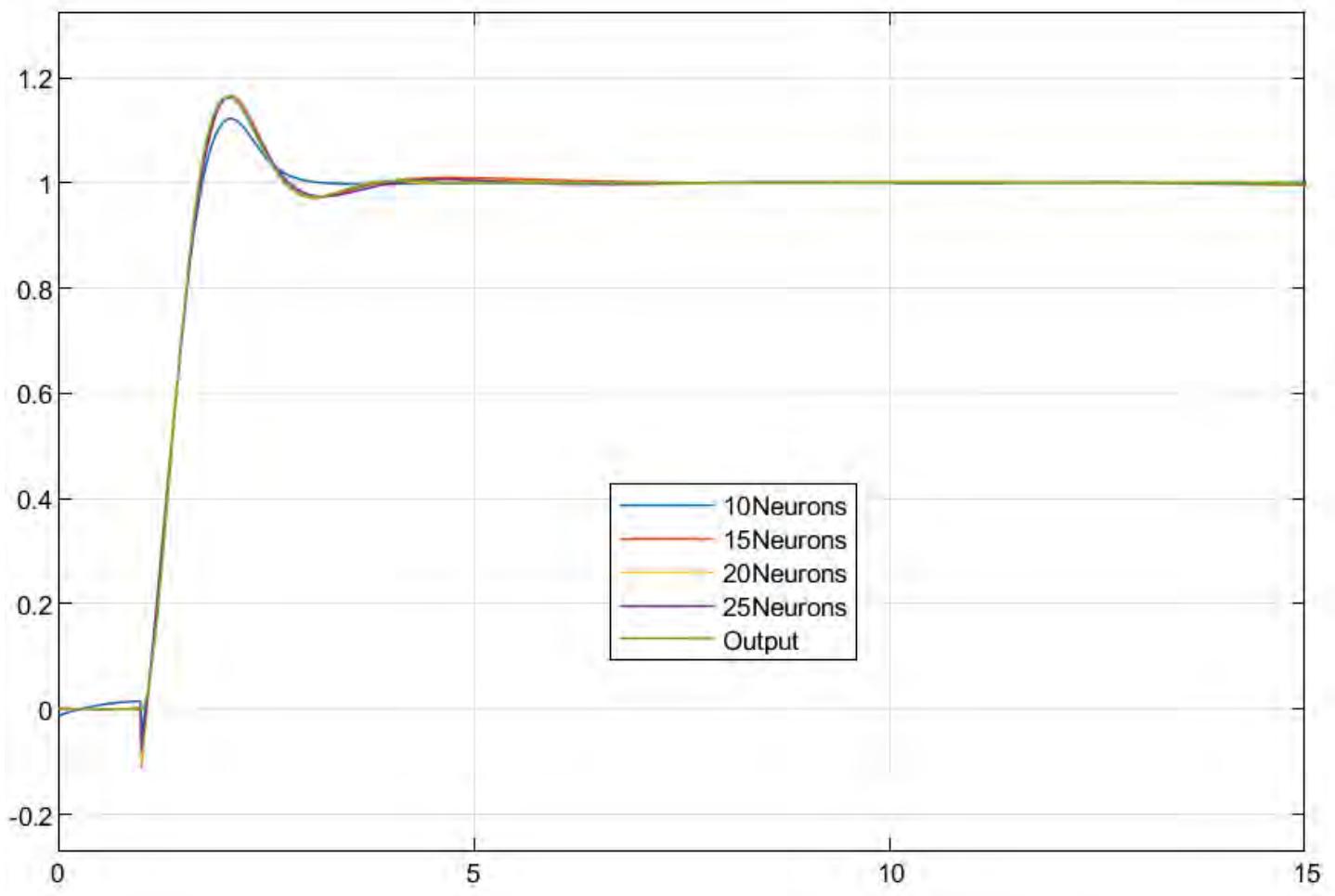
Plot below,  $\omega = 2$





Plot below,  $\omega = 3$



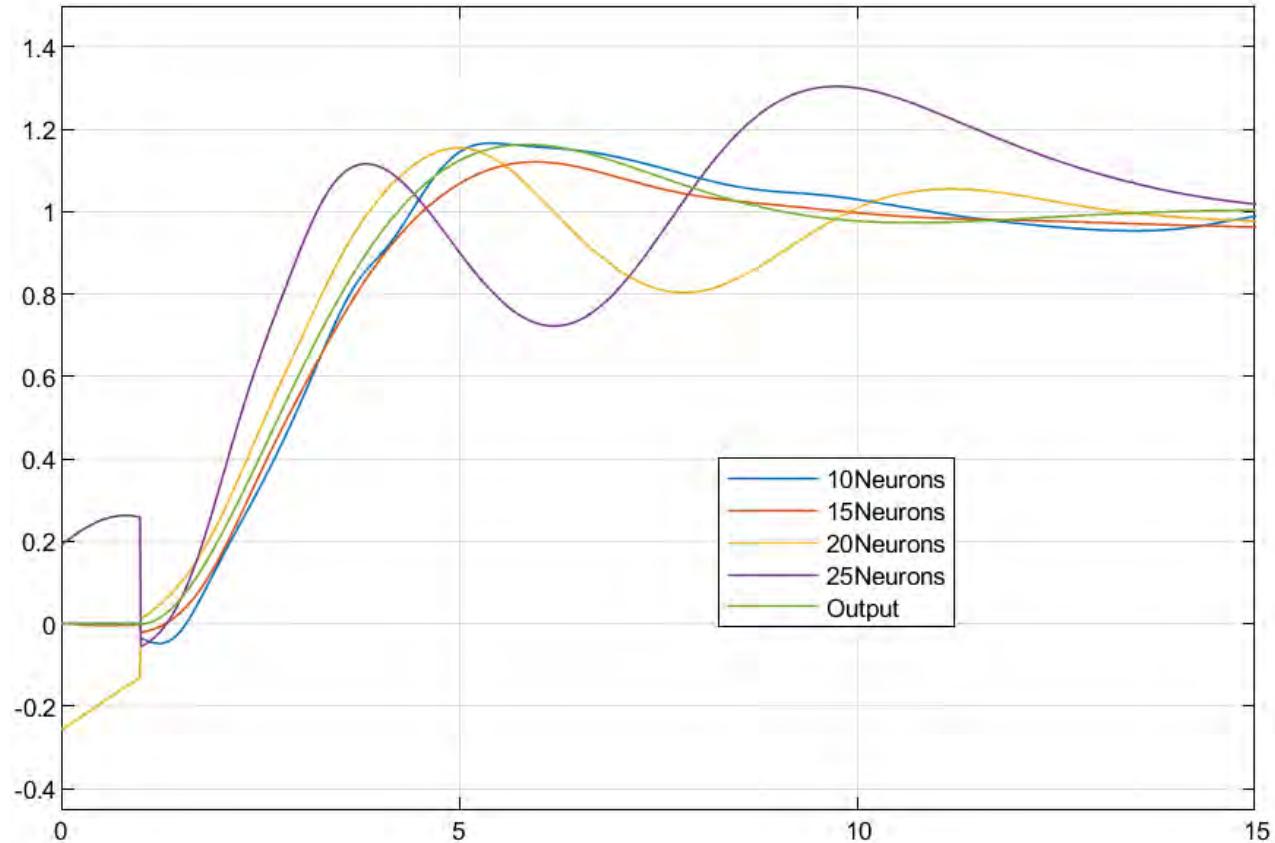


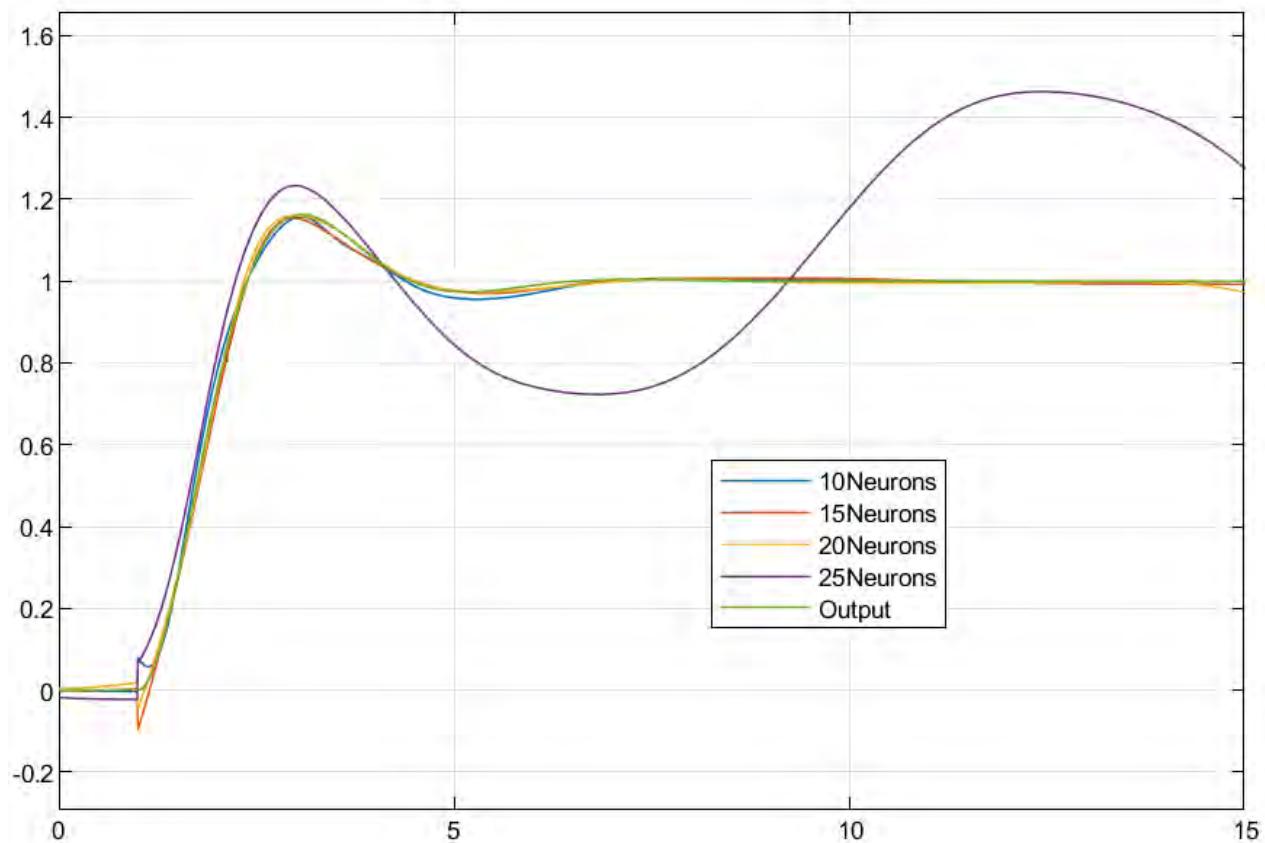
$\Omega = 3.5$

**Conclusion** – From the training and validation data, it is apparent that more neurons produce a better fit. This is supported by the best performance for validation data and the general performance (in the Neural Network Training window) metrics.

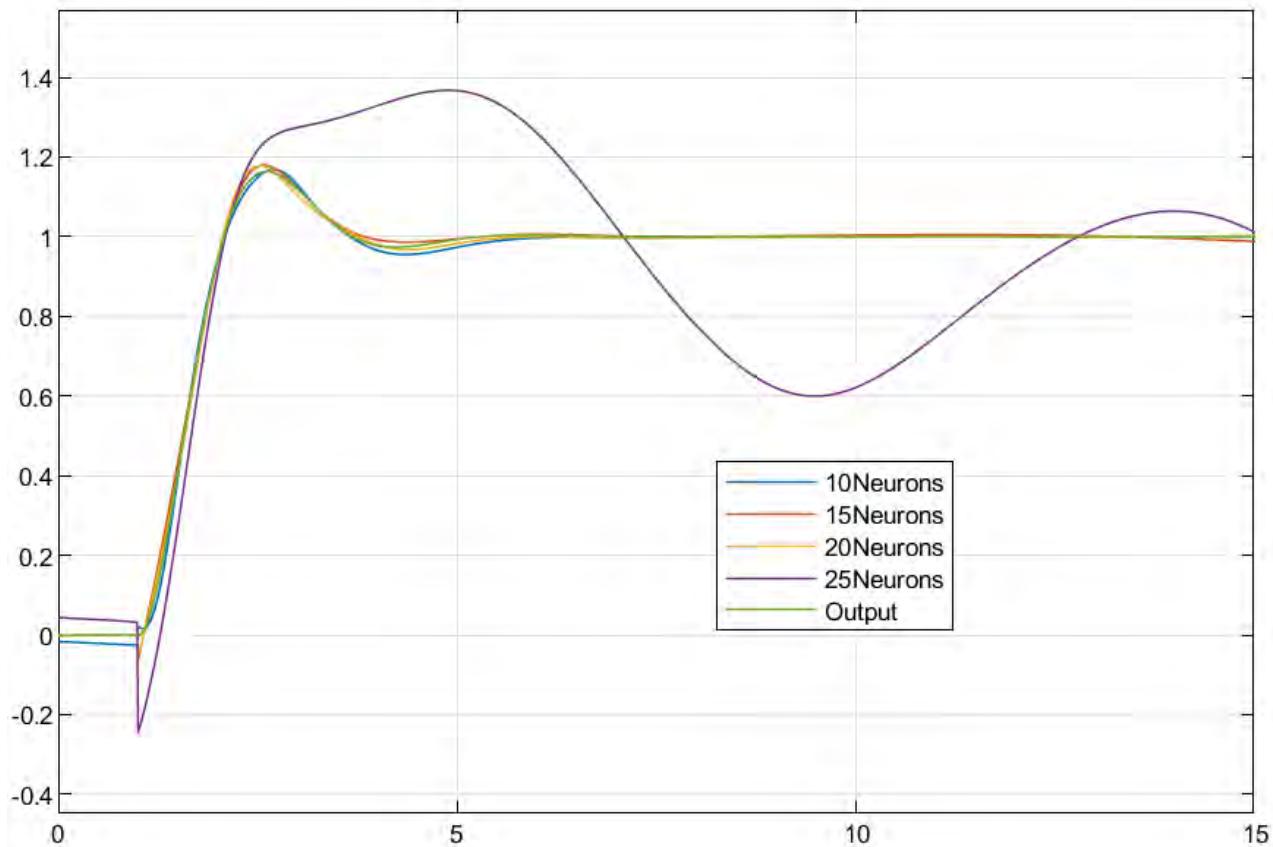
Other than that, the output is slightly erroneous in the transition state but is very good once steady state is reached for almost all neural networks. However, it is notable that the neural network with 10 Neurons goes up just before the step input whereas others dip downwards. The dipping amplitude decreases with the increase of neurons.

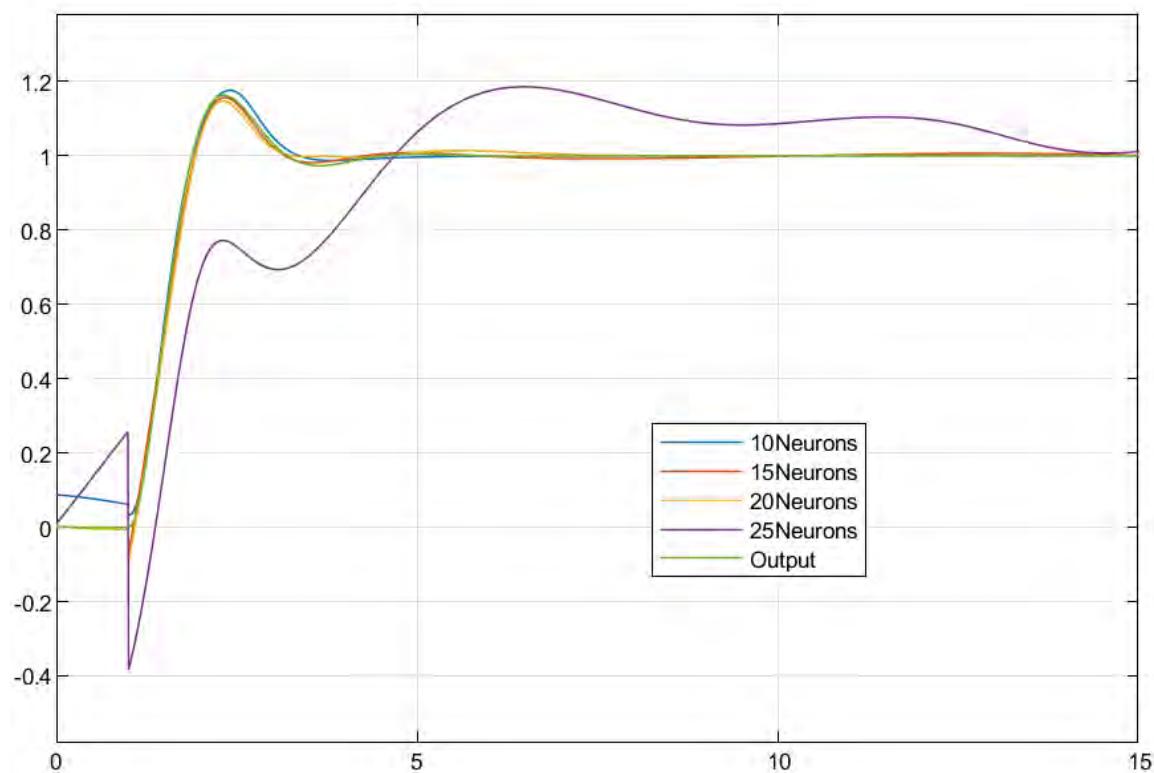
**b.** Testing extrapolation





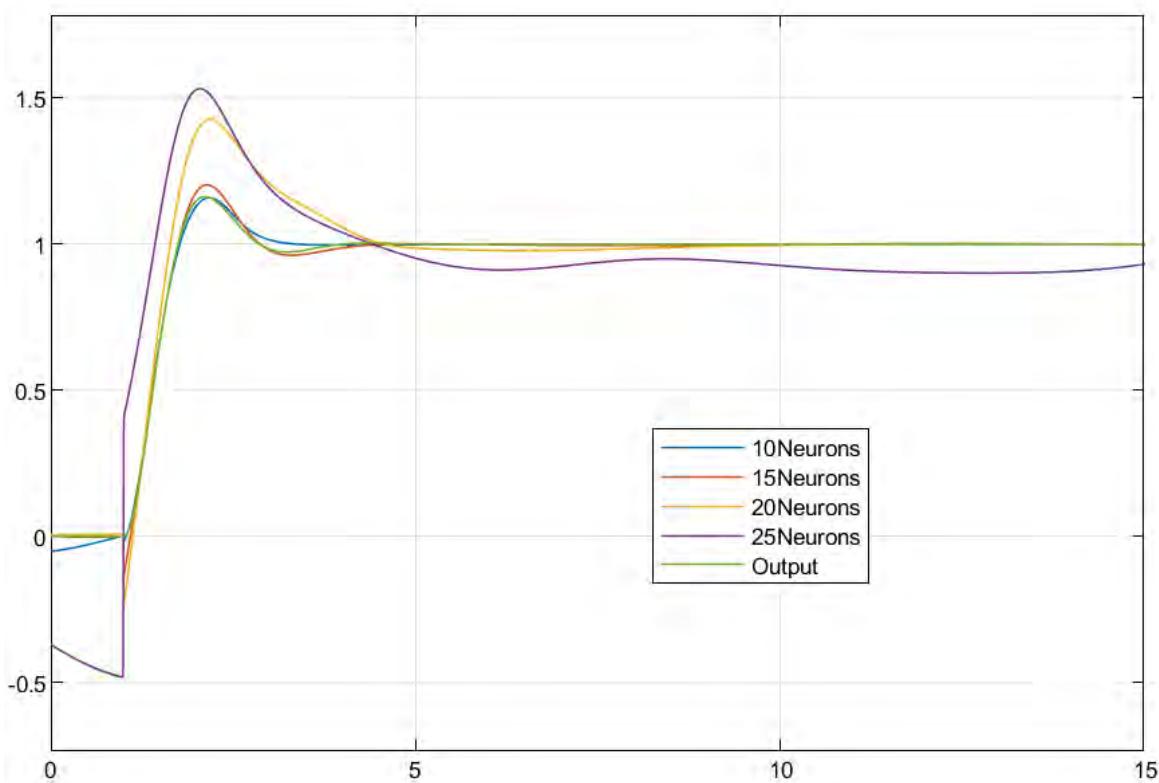
Plot below,  $\omega = 2.25$





Plot above,  $\omega = 2.75$

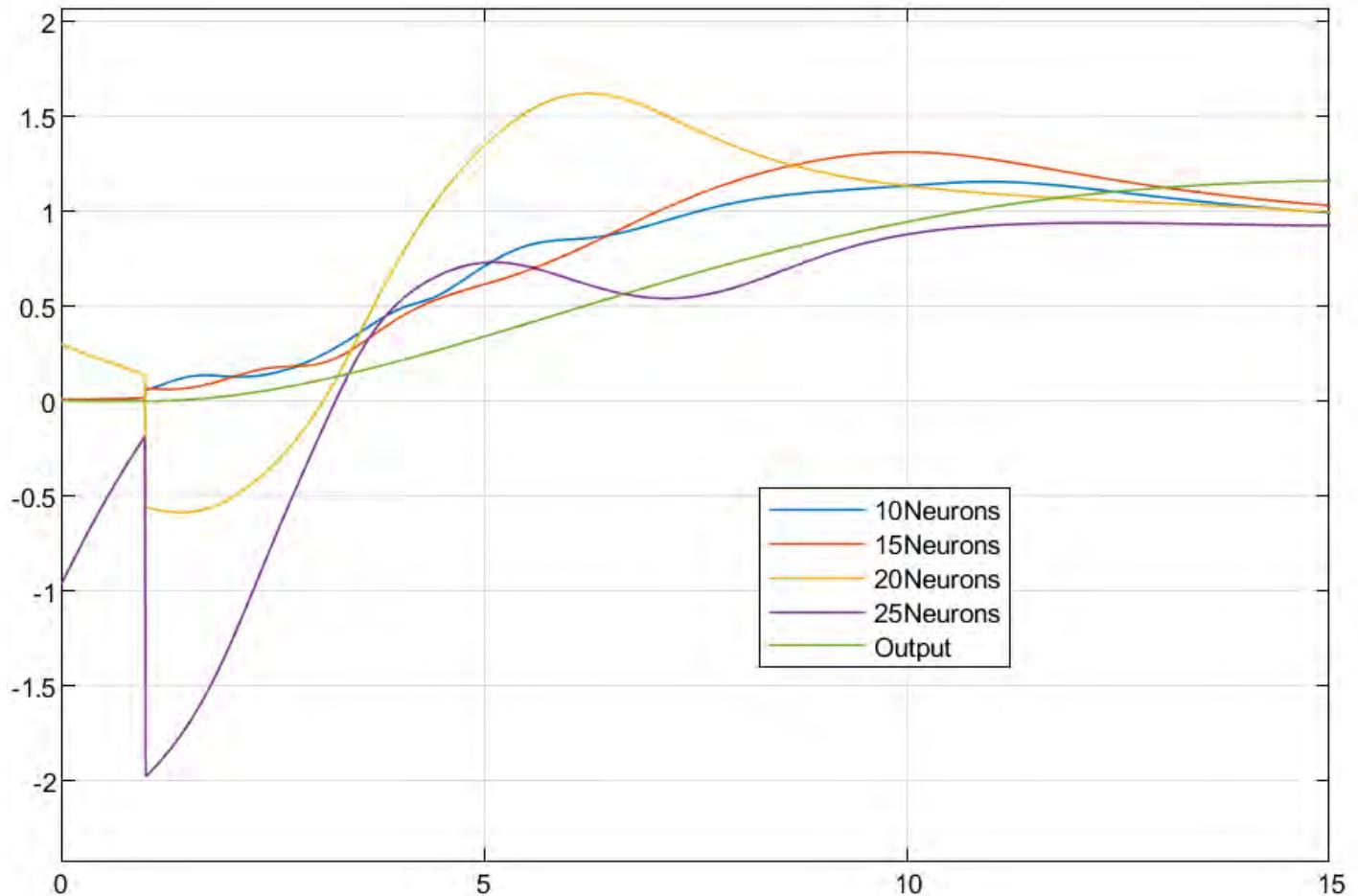
Plot below,  $\omega = 3.25$



**Conclusion** – It is observed that smaller frequencies are tougher to interpolate as compared to larger frequencies. The neural network with 10 neurons does the best job at interpolation as it provides the nearest output at  $\omega = 0.75$ . More number of neurons may cause overfitting which is why more neurons lead to less efficient interpolation.

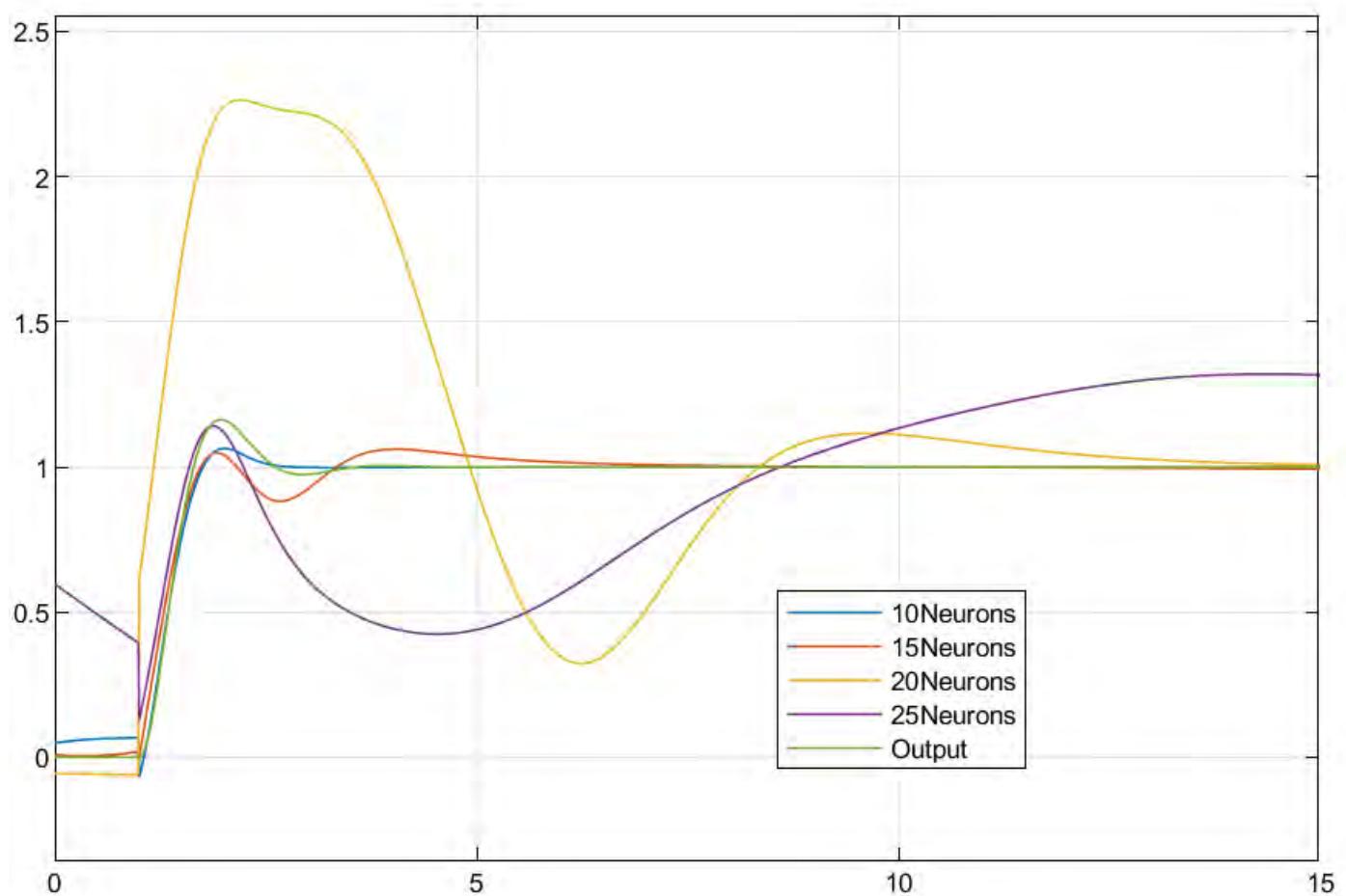
**C.** The neural network with 10 neurons does a good job of interpolating frequency at lower as well as higher frequencies. Thus, **neural networks with around 10 neurons** will work effectively for this system if they are trained using inputs and outputs over all the operating frequencies. If the operating frequency doesn't change or stays constant at regions used for training data then more neurons may be good. Note that more neurons are good at fitting but they are not so good at interpolation for this system. Also, none of the neural networks is really close to the system so they might not be usable if the acceptable errors in output are of the magnitude  $10^{-3}$ .

#### d. Plots for extrapolation



Plot above,  $\omega = 0.25$

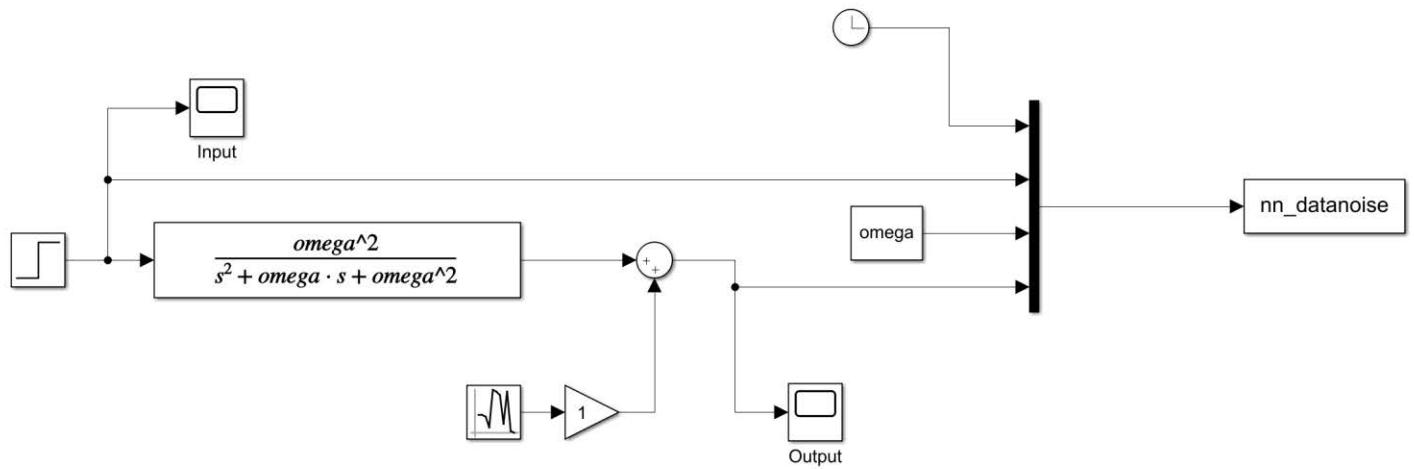
Plot below,  $\omega = 3.75$



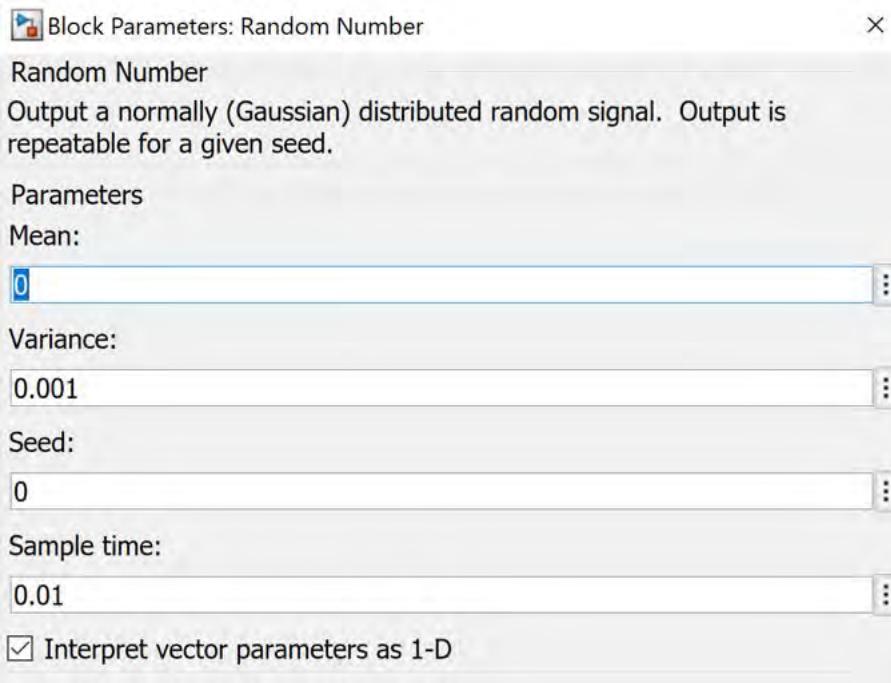
**Conclusion** - It is observed that for both frequencies, the neural network with 10 neurons is the closest to the actual output. While it doesn't do a very good job when operating outside the training limits, it is still better than the rest.

## Problem - 2:

a.



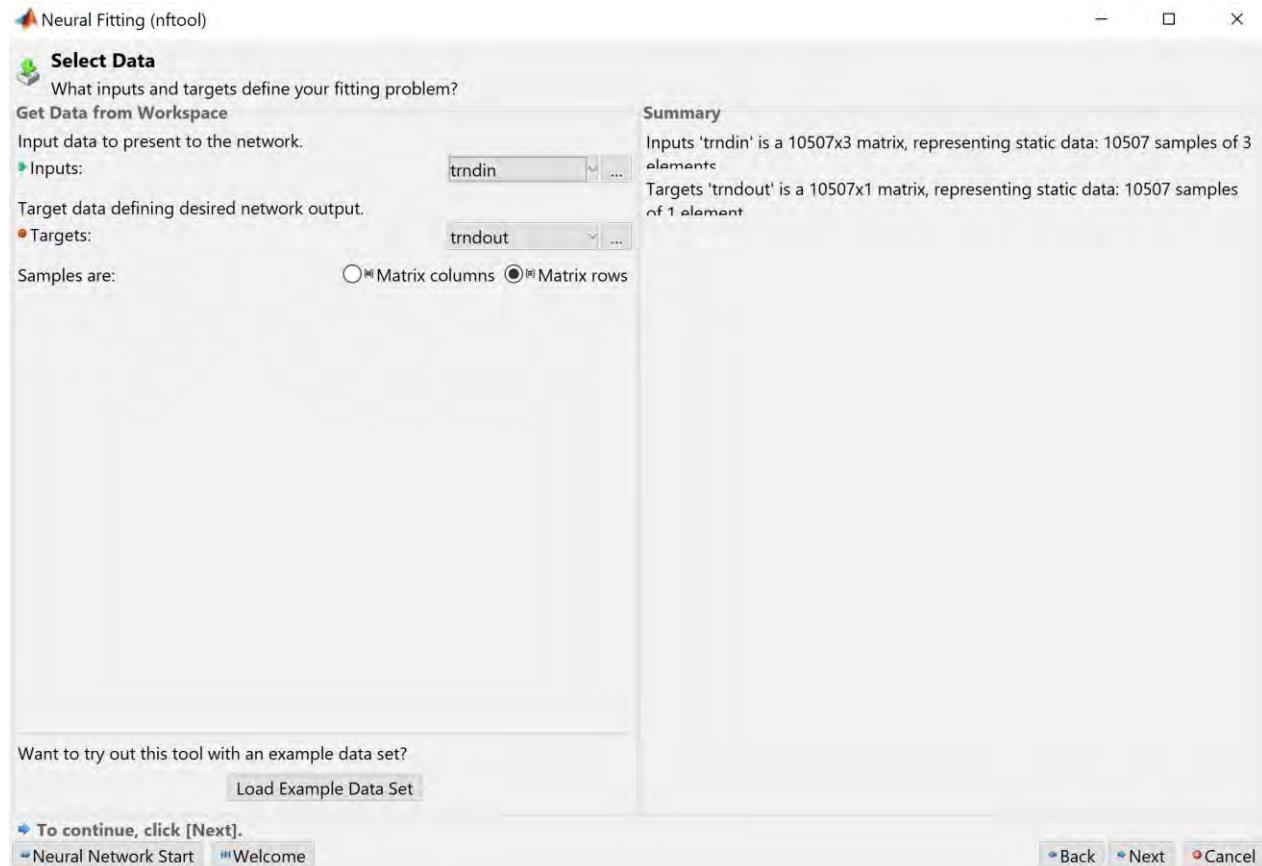
Simulink Diagram



Noise Parameters

## MATLAB Commands to arrange the data

```
omega=0.5 %run simulation after every new omega value  
trndata=nn_datanoise  
omega=1  
trndata=[trndata;nn_datanoise]  
omega=1.5  
trndata=[trndata;nn_datanoise]  
omega=2  
trndata=[trndata;nn_datanoise]  
omega=2.5  
trndata=[trndata;nn_datanoise]  
omega=3  
trndata=[trndata;nn_datanoise]  
omega=3.5  
trndata=[trndata;nn_datanoise]  
trndin=trndata(:,1:3)  
trndout=trndata(:,4)
```



## Neural Network Training ...

### Neural Network



### Algorithms

Data Division: Random (dividerand)  
Training: Levenberg-Marquardt (trainlm)  
Performance: Mean Squared Error (mse)  
Calculations: MEX

### Progress

Epoch:	0	91 iterations	1000
Time:		0:00:01	
Performance:	0.443	0.00128	0.00
Gradient:	1.73	0.00120	1.00e-07
Mu:	0.00100	1.00e-06	1.00e+10
Validation Checks:	0	6	6

### Plots

- Performance (plotperform)
- Training State (plottrainstate)
- Error Histogram (ploterrhist)
- Regression (plotregression)
- Fit (plotfit)

Plot Interval: 1 epochs

✓ Validation stop.

## Neural Fitting (nftool)

### Train Network

Train the network to fit the inputs and targets.

#### Train Network

Choose a training algorithm:

Levenberg-Marquardt

This algorithm typically requires more memory but less time. Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples.

Train using Levenberg-Marquardt. (trainlm)

Retrain

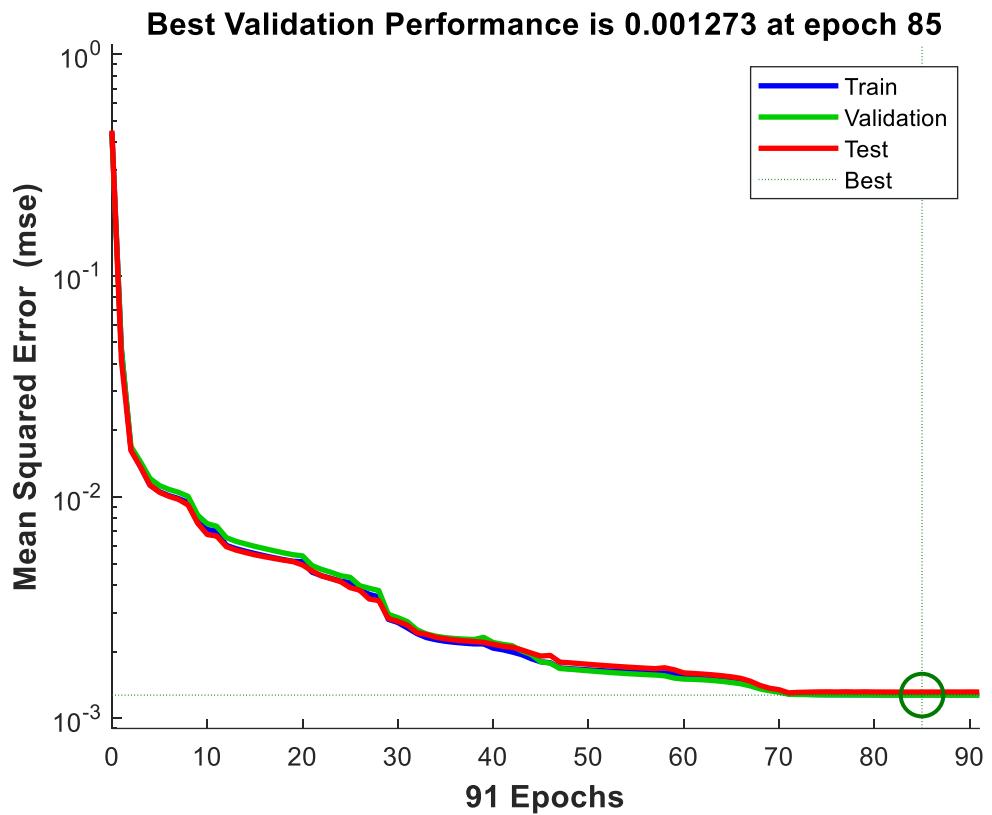
#### Results

	Samples	MSE	R
Training:	7355	1.27695e-3	9.93871e-1
Validation:	1576	1.27302e-3	9.93896e-1
Testing:	1576	1.31431e-3	9.93286e-1

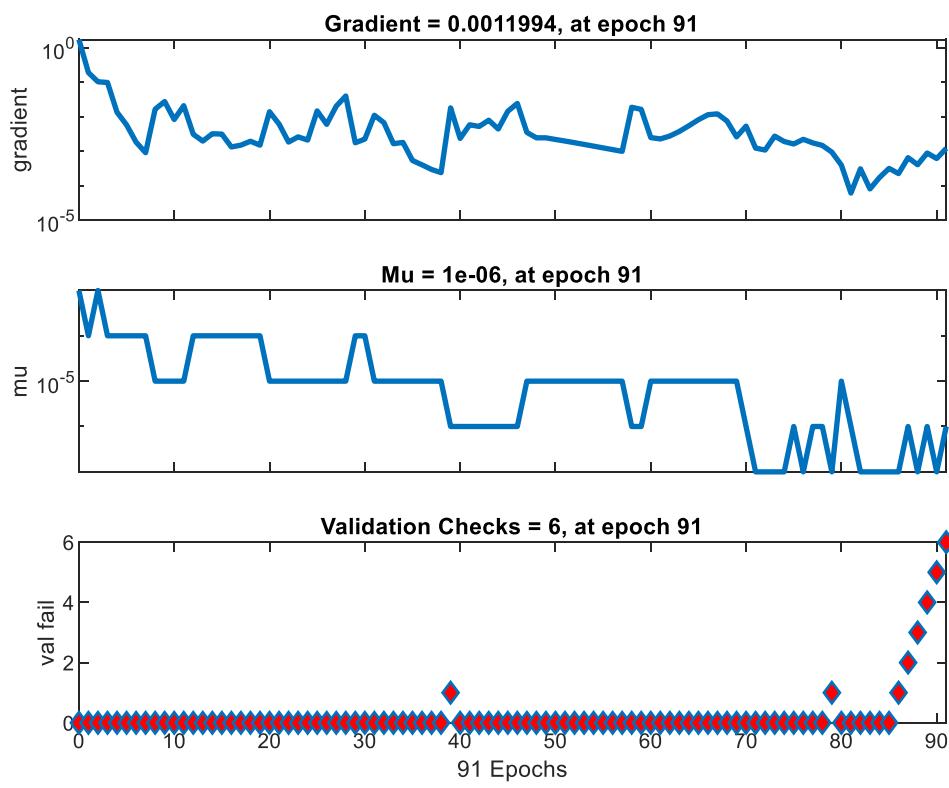
Plot Fit Plot Error Histogram

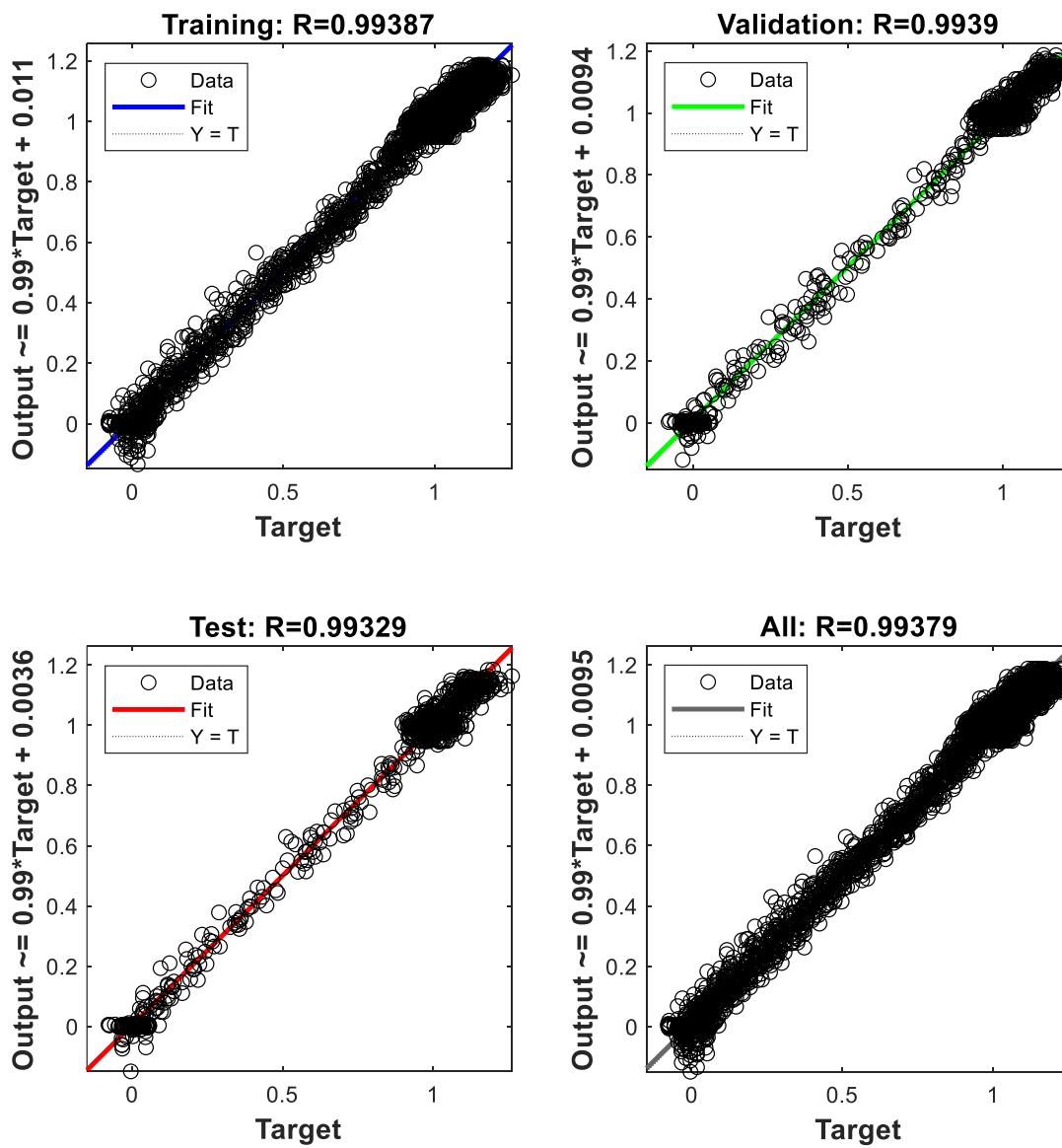
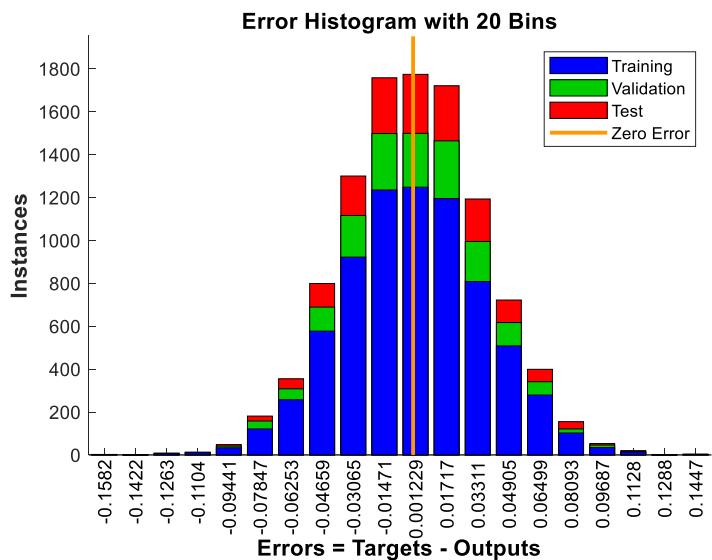
Plot Regression

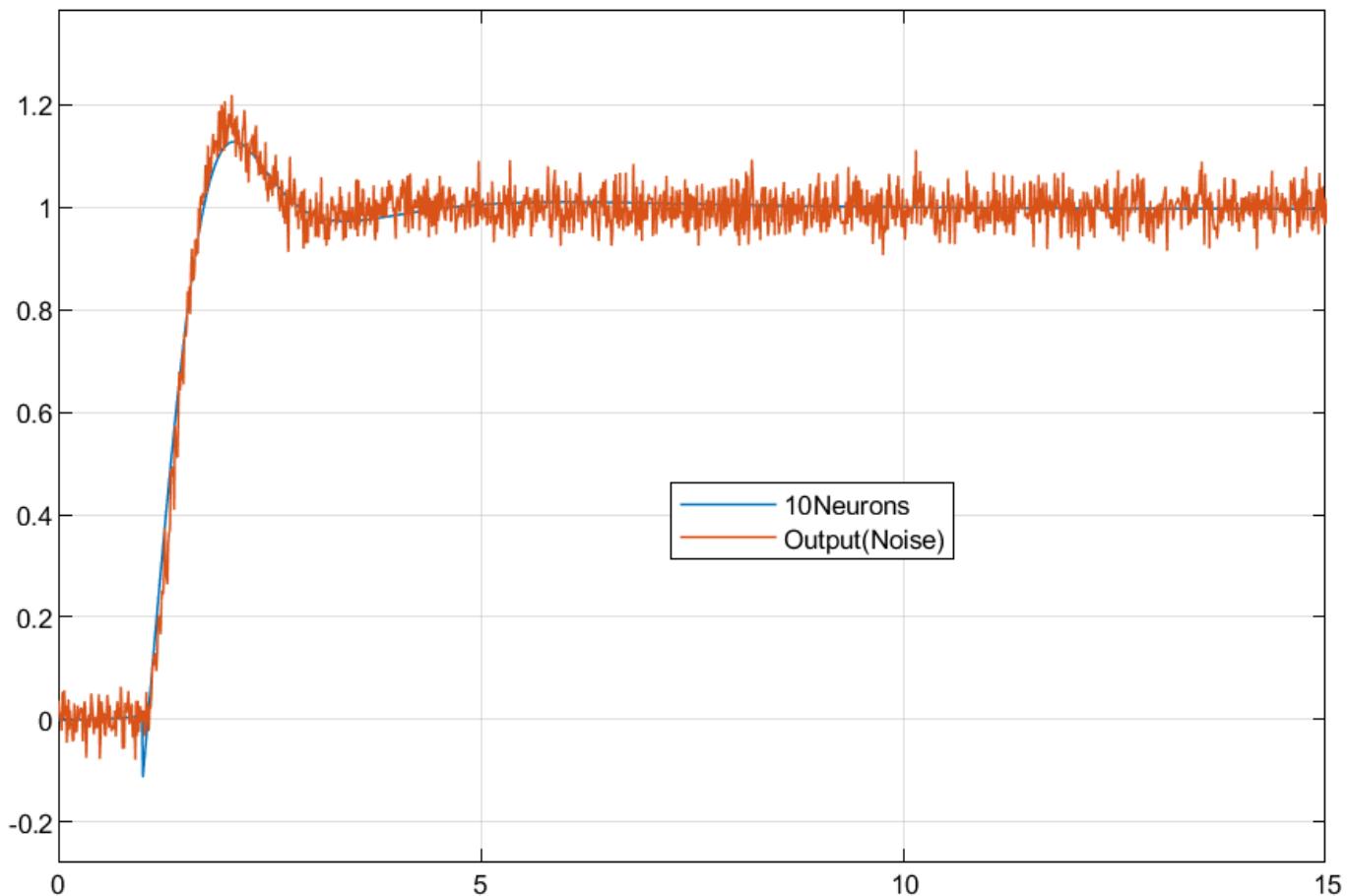
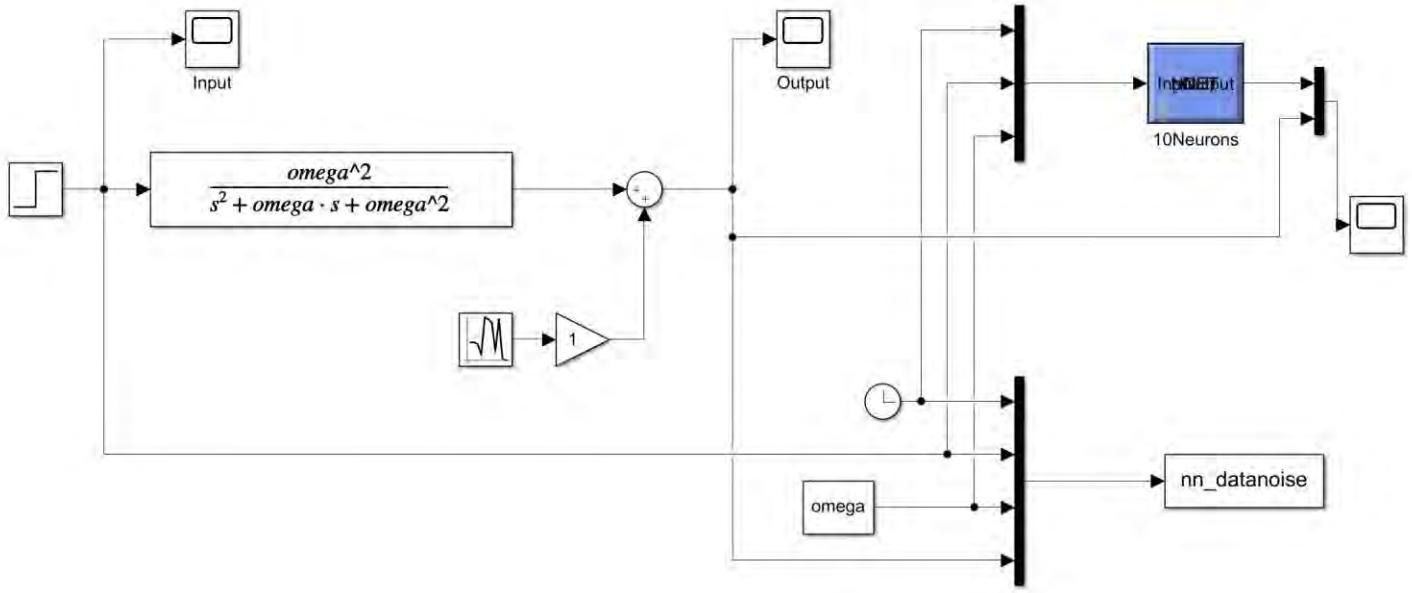
Notes



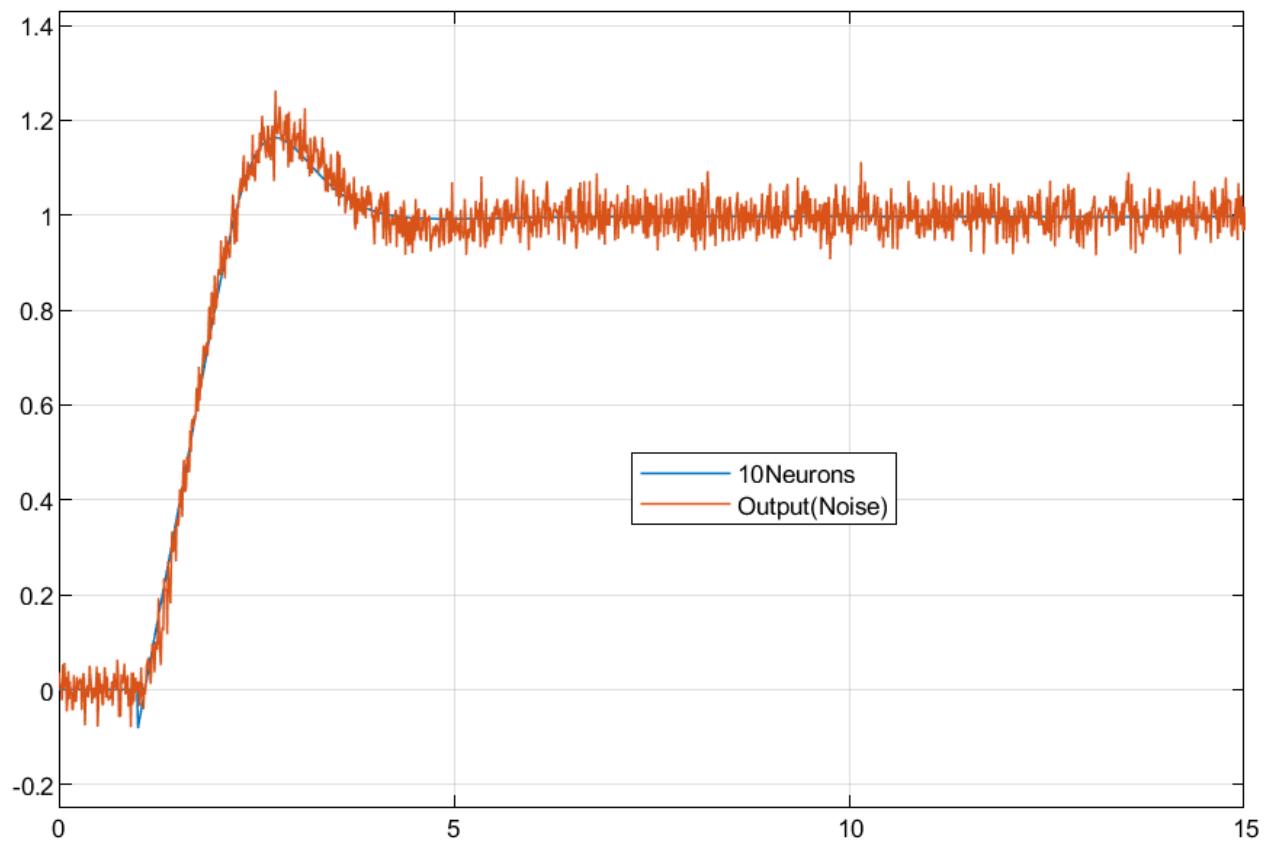
### Training State





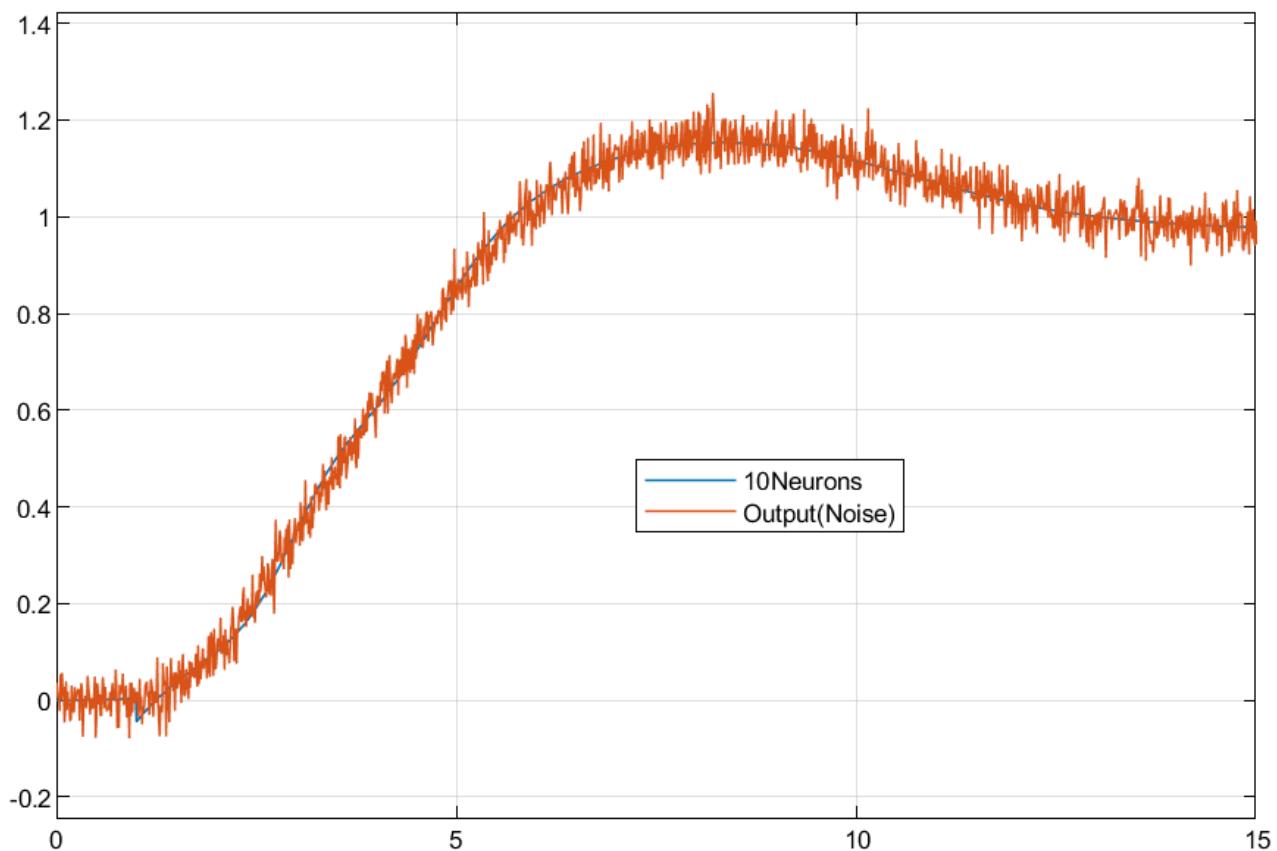


Omega = 3.5

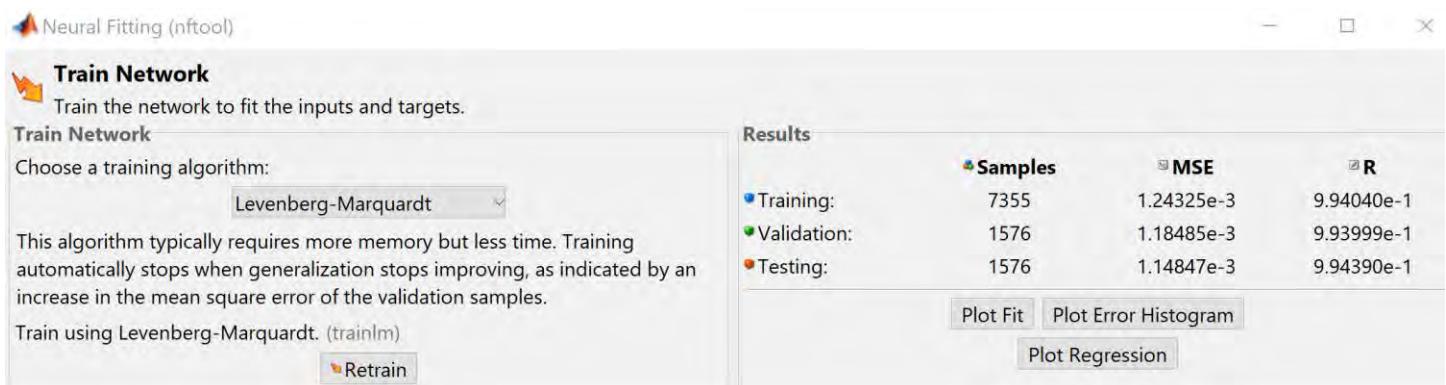
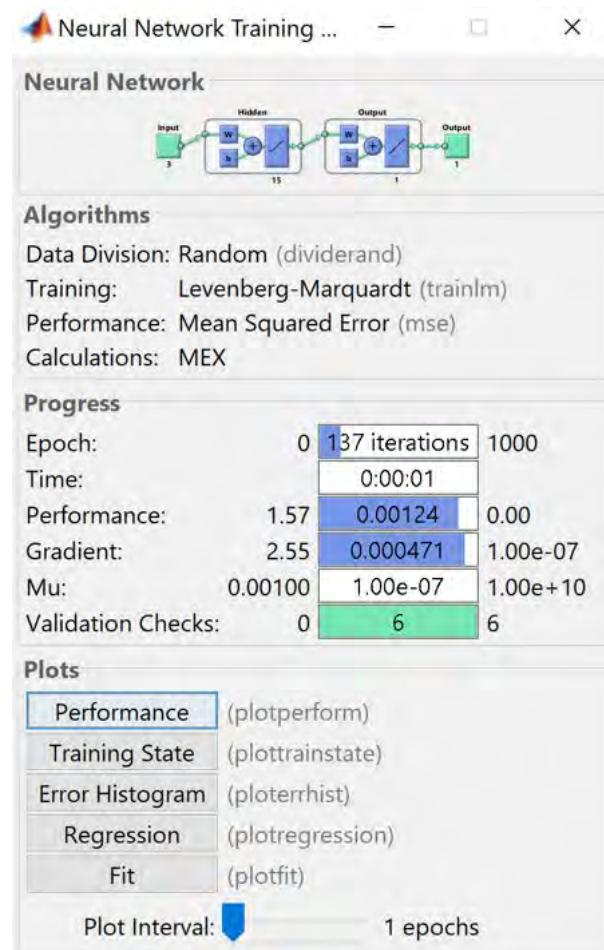


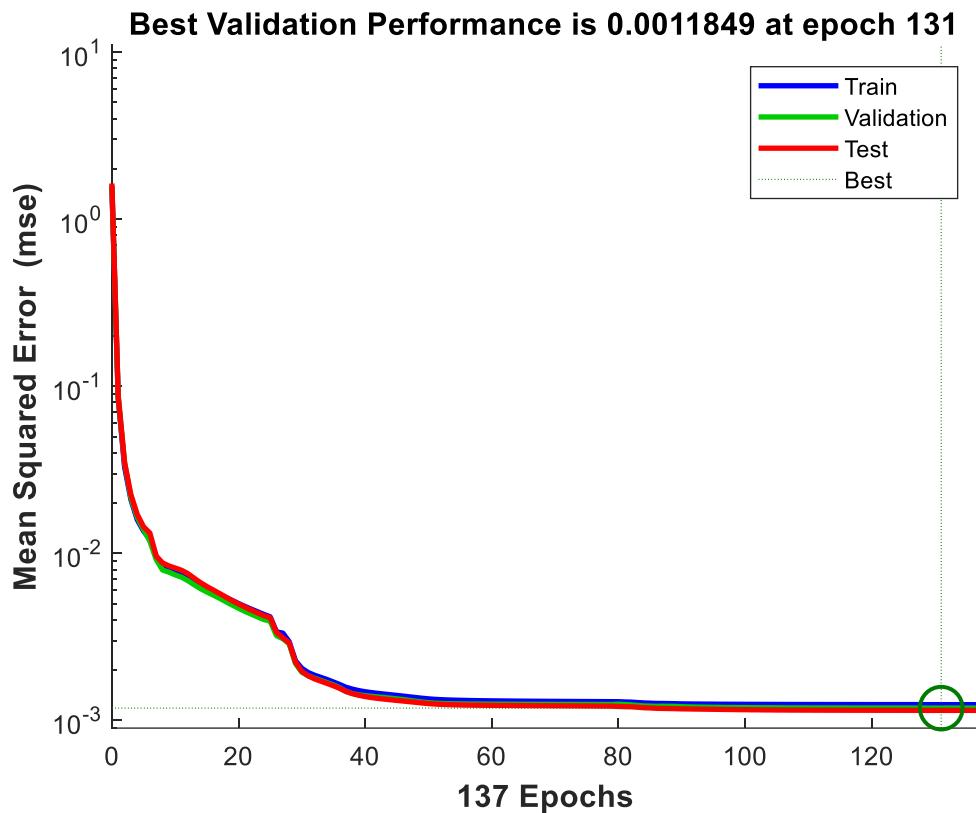
Plot above,  $\omega = 2$

Plot below,  $\omega = 0.5$

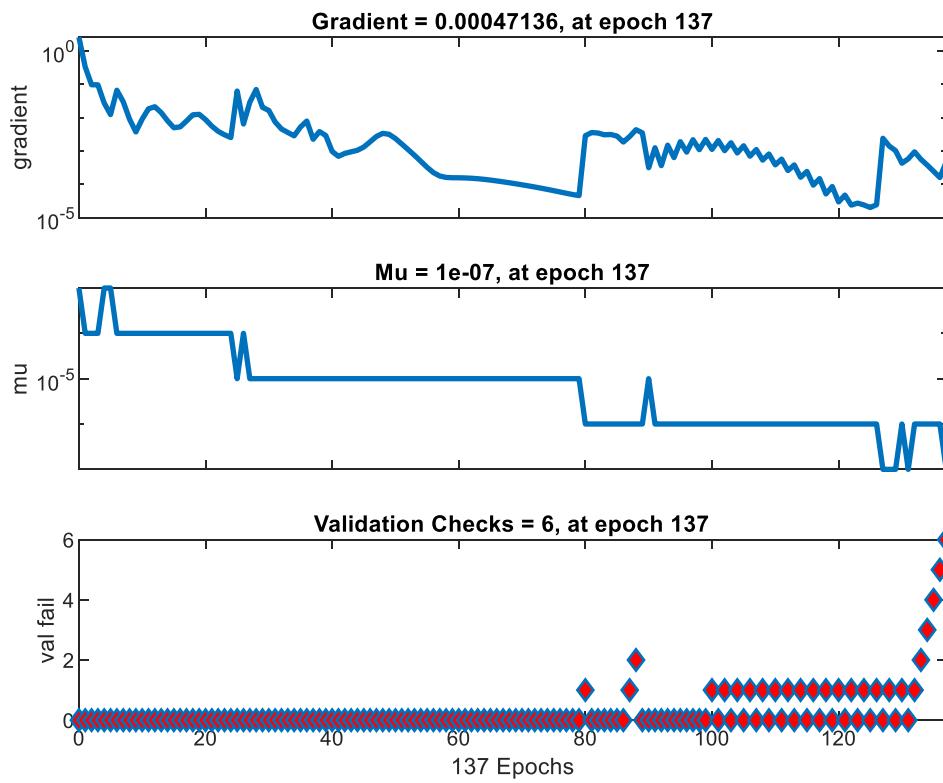


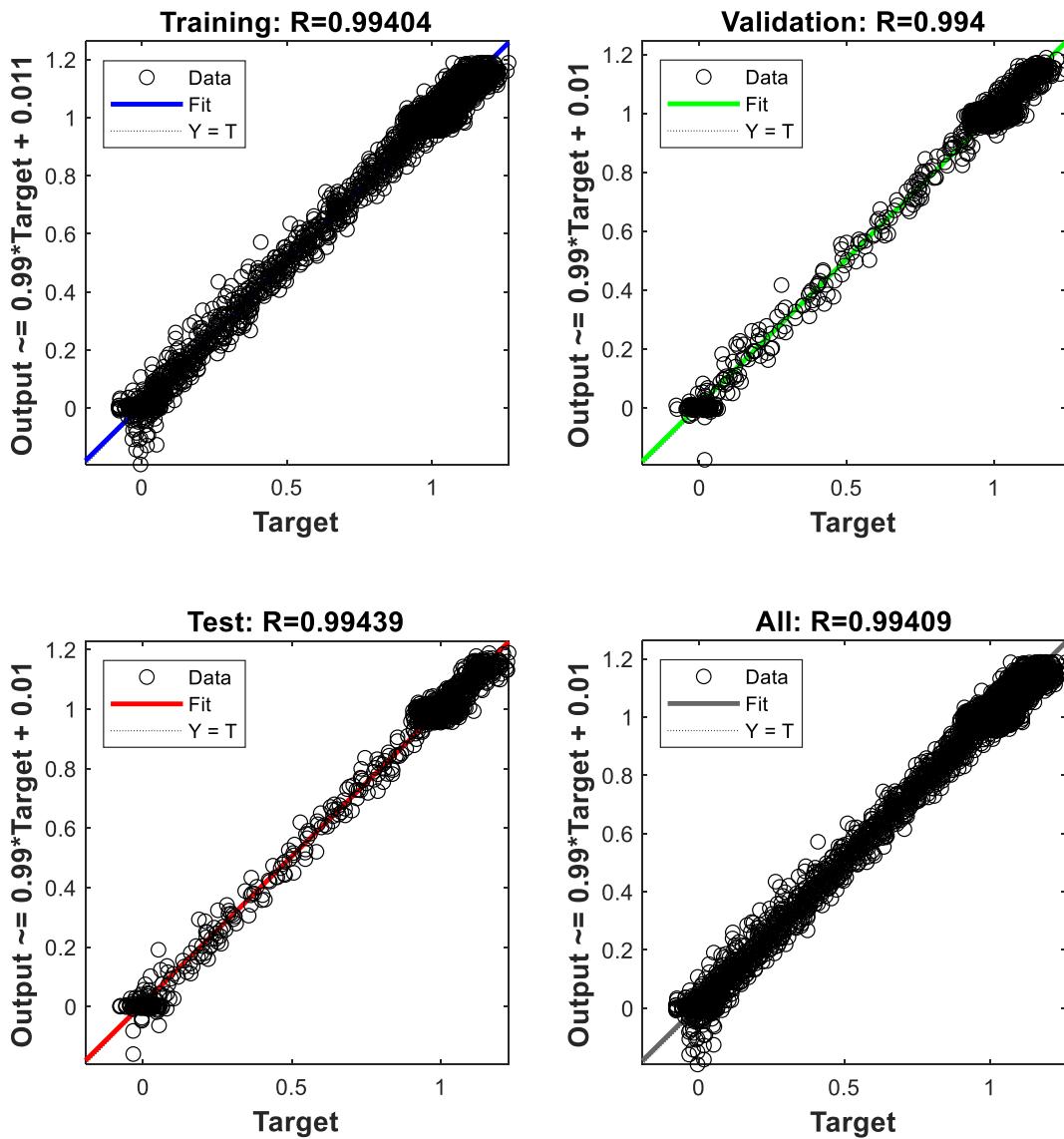
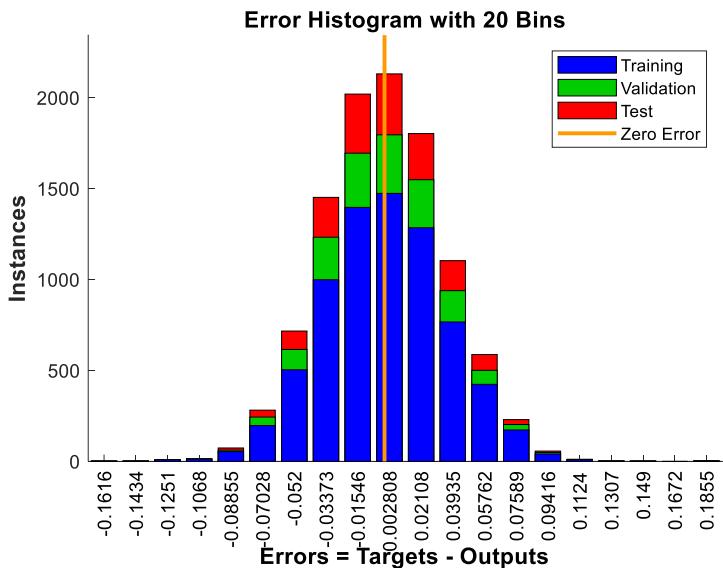
## Training with 15 Neurons



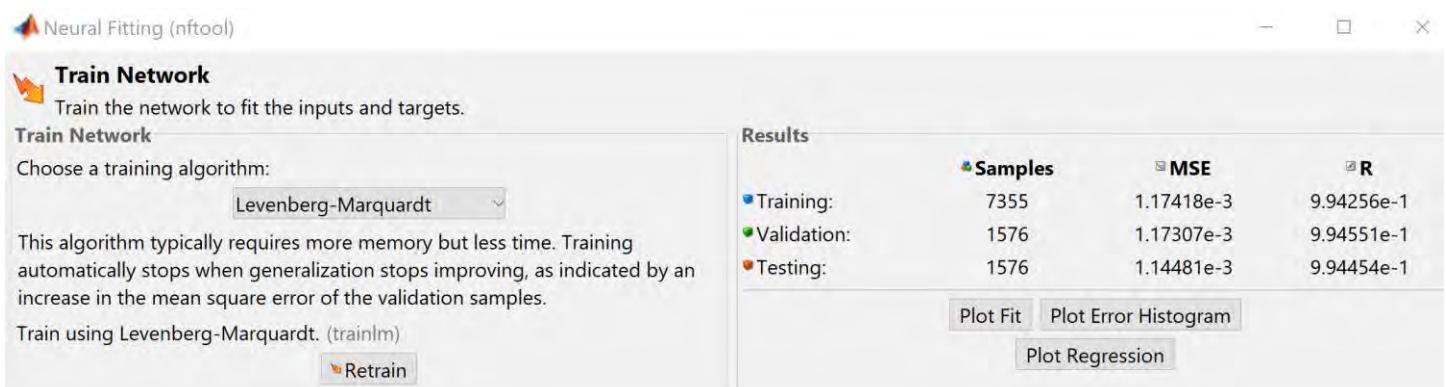
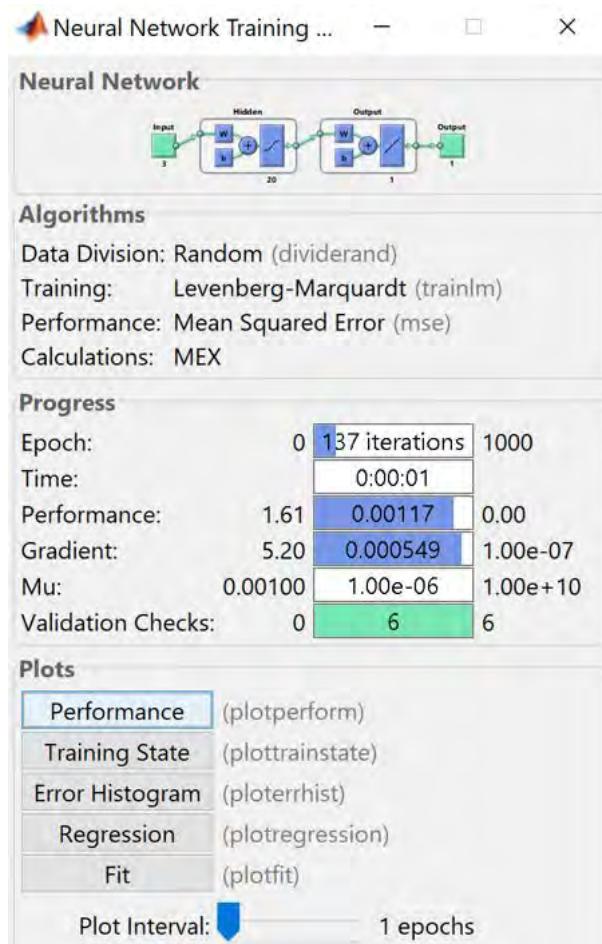


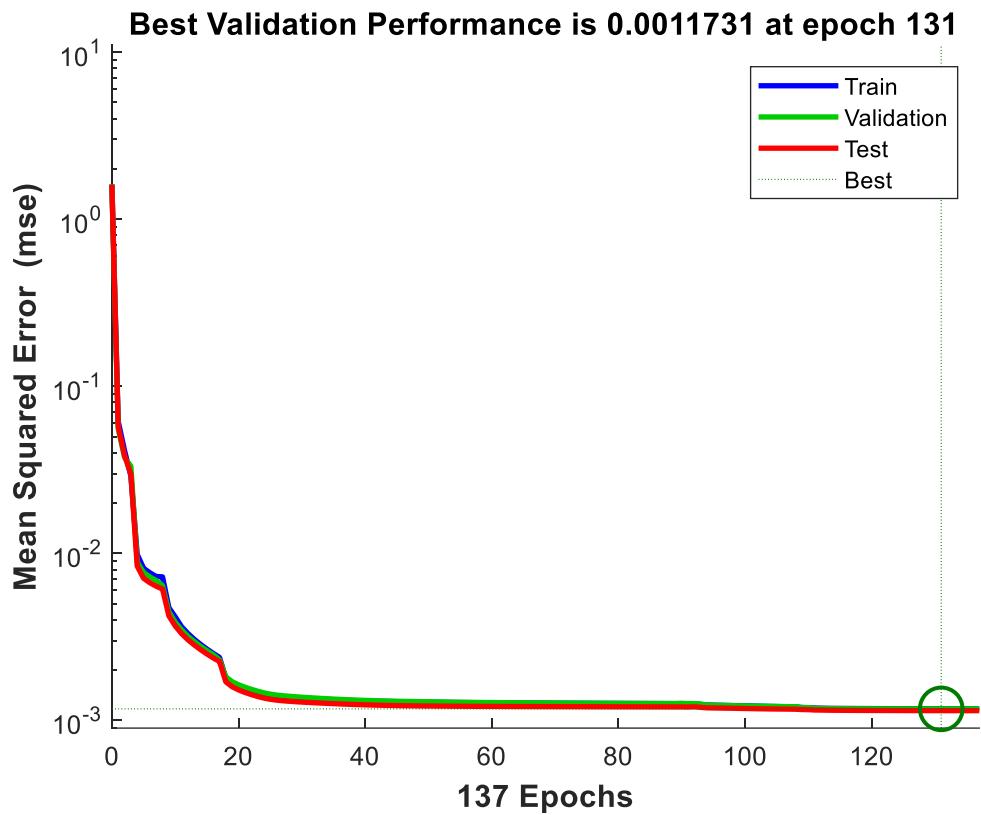
### Training State



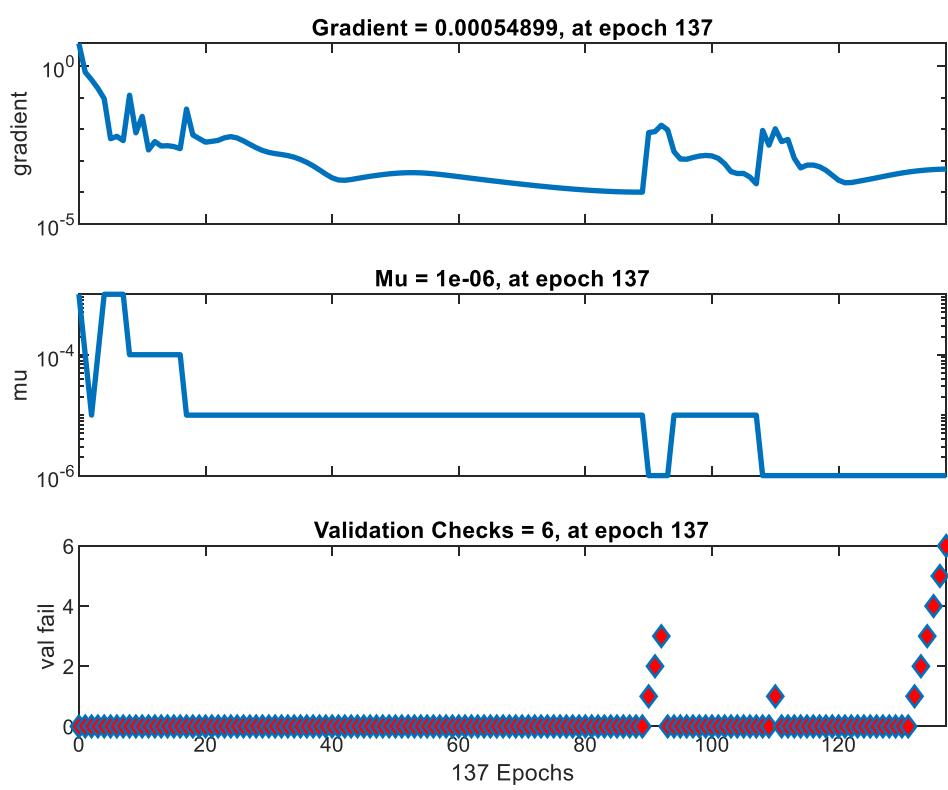


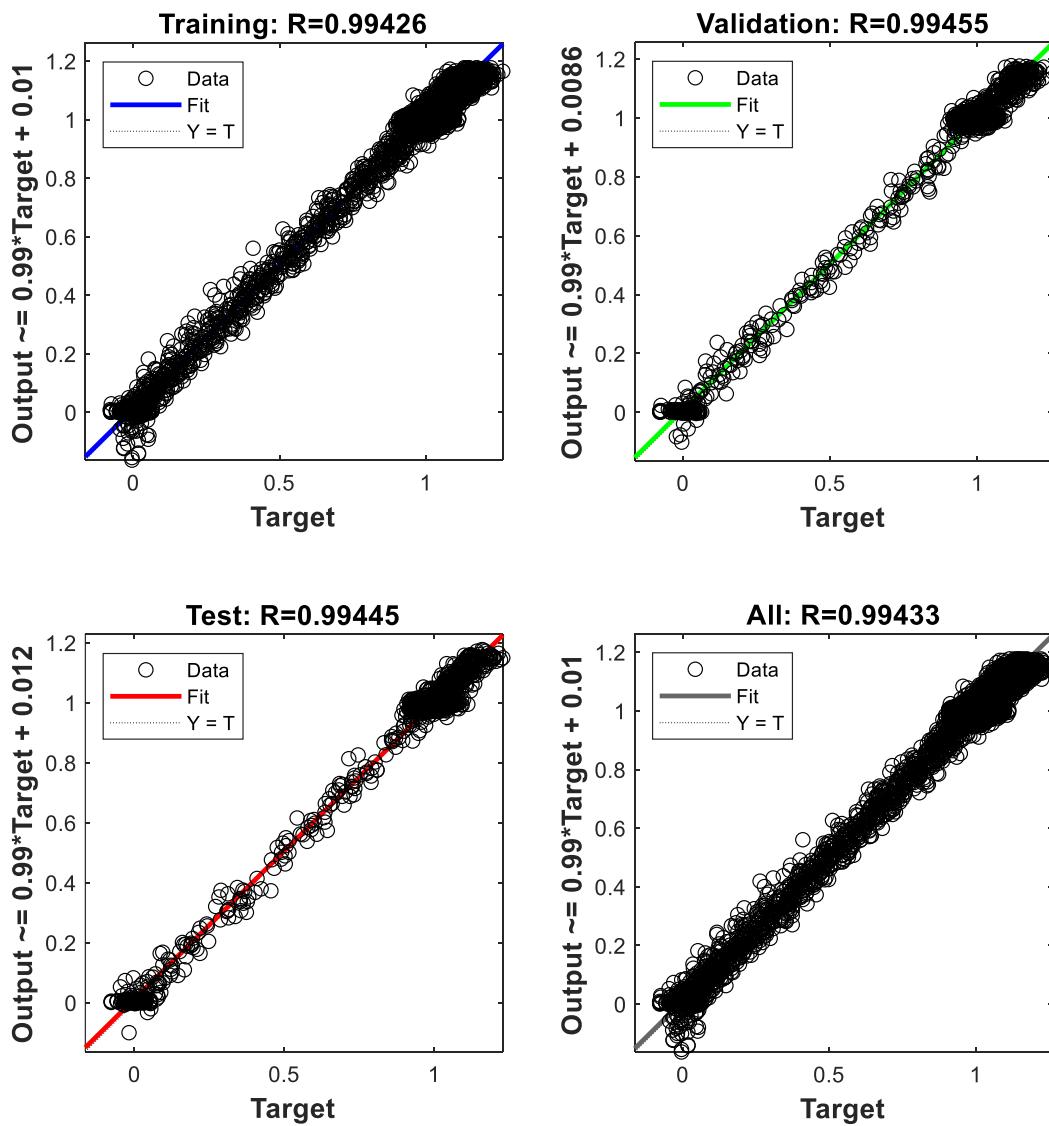
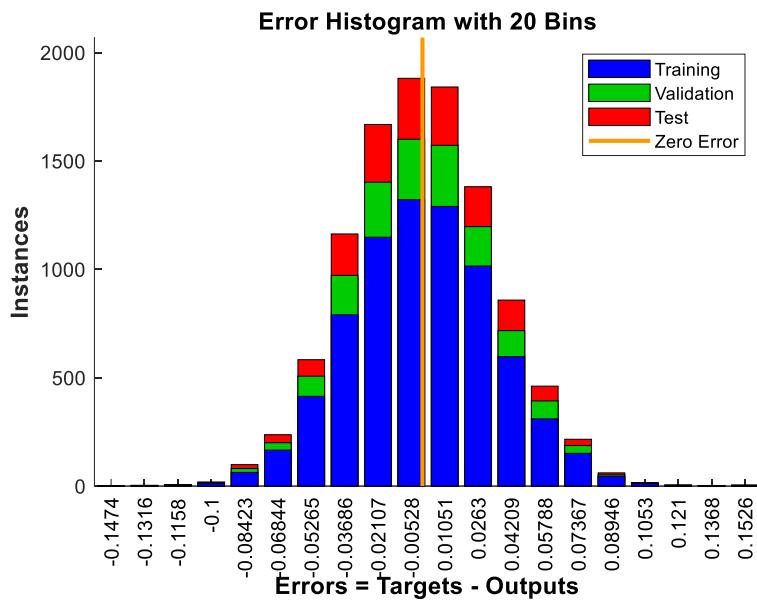
## Training with 20 neurons



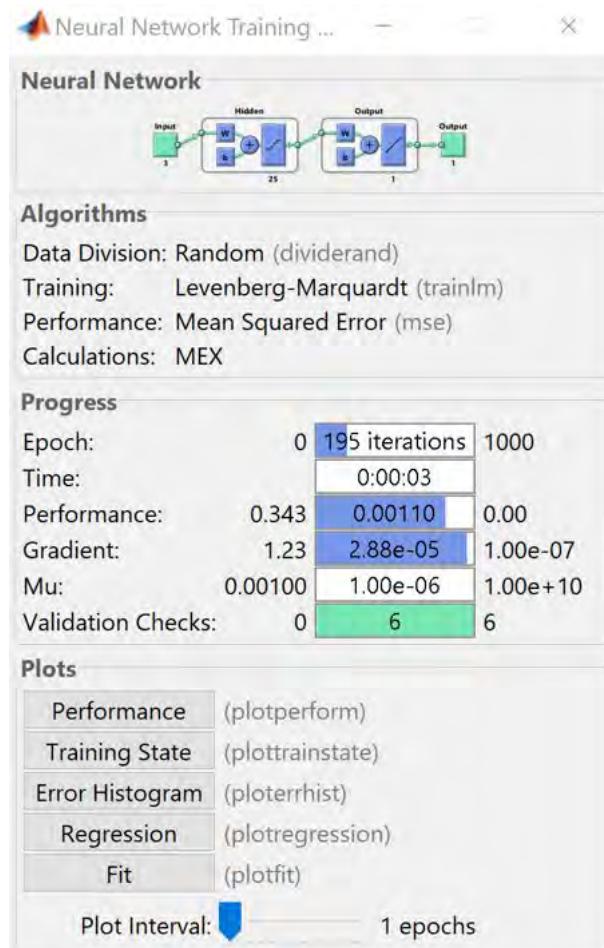


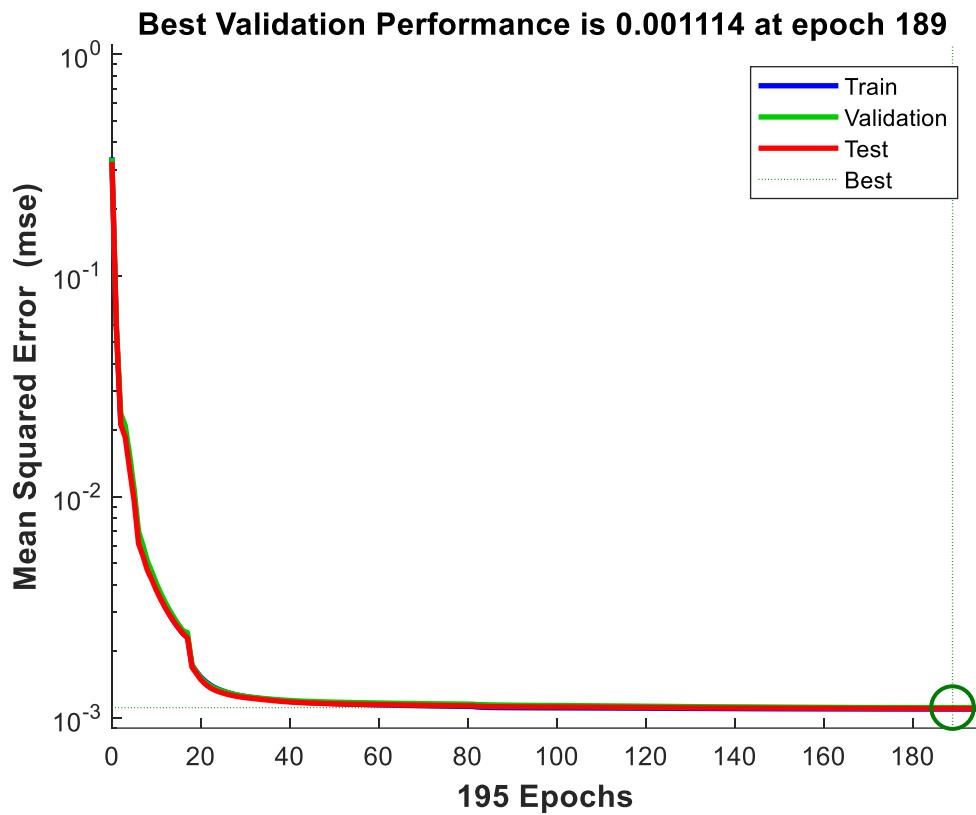
### Training State



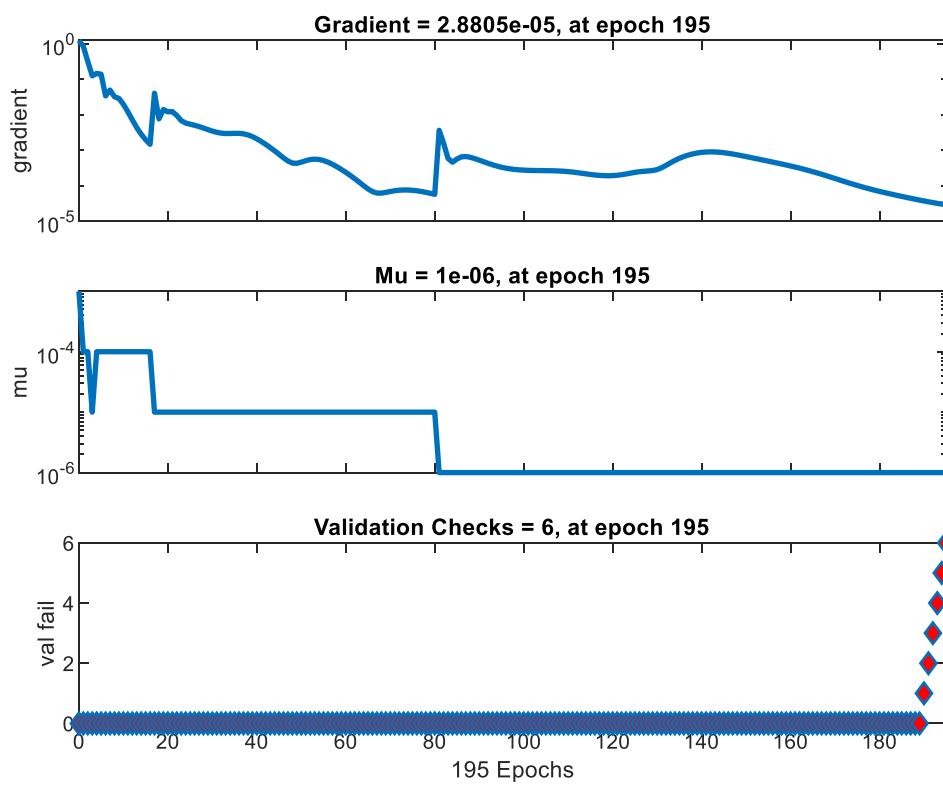


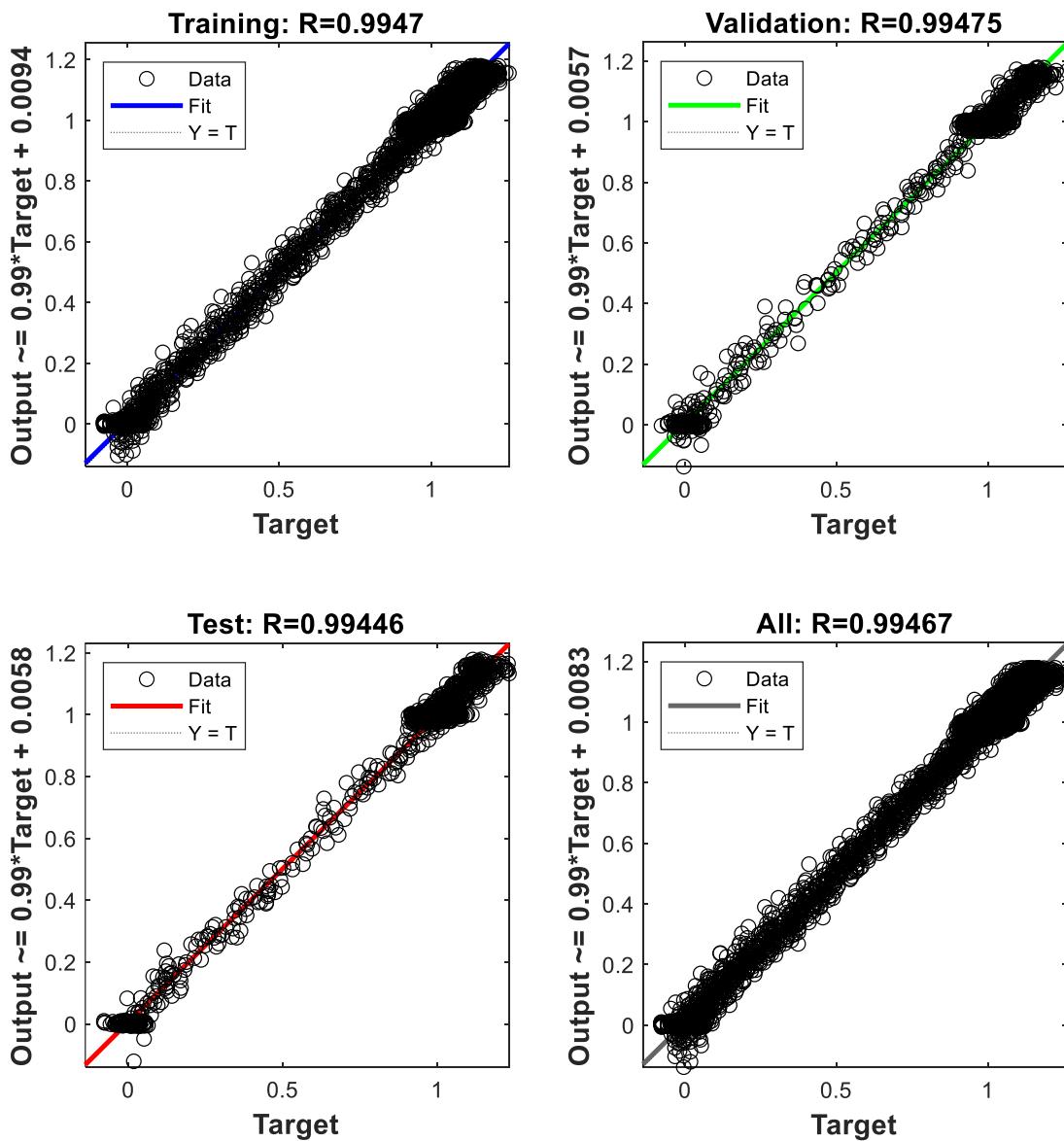
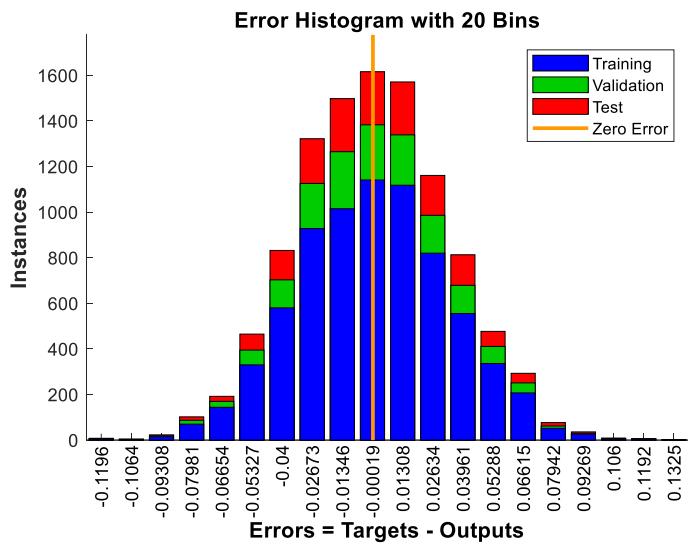
## Training with 25 neurons



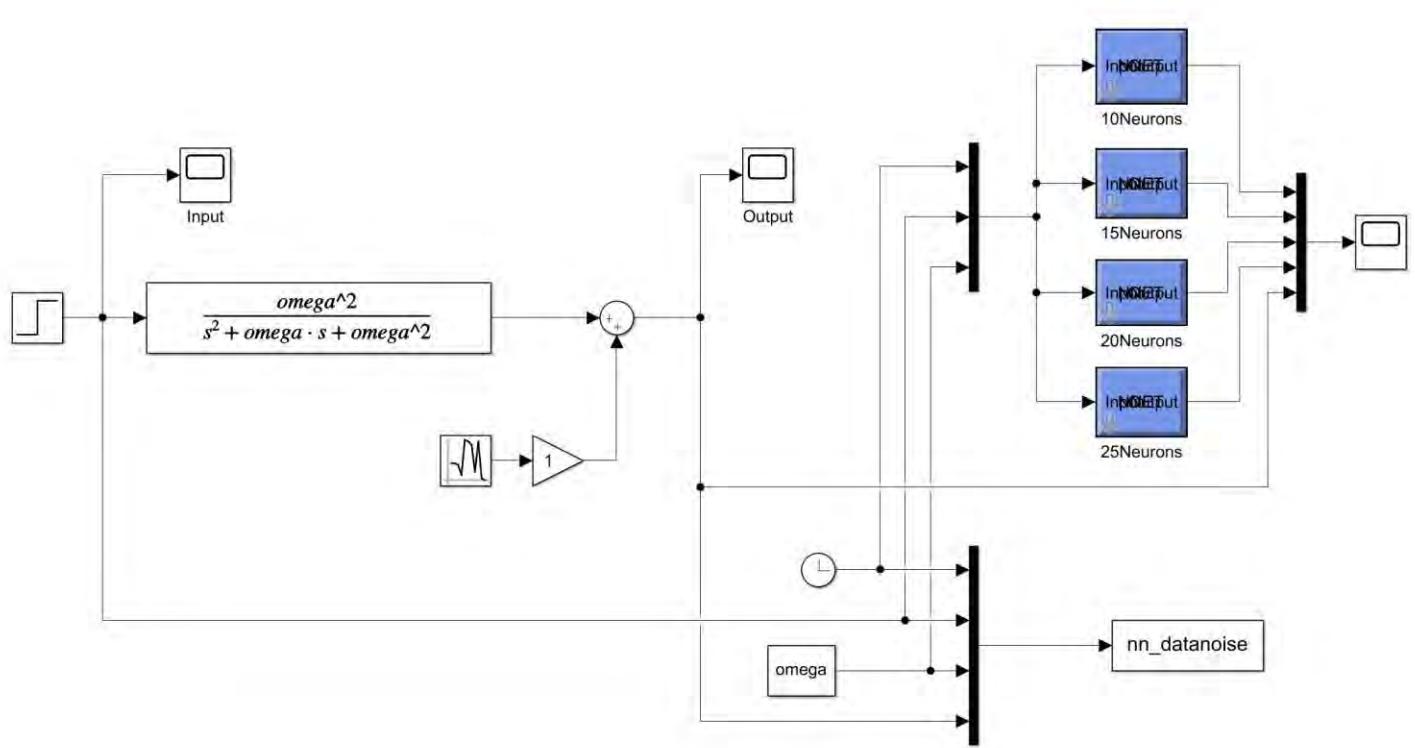


### Training State

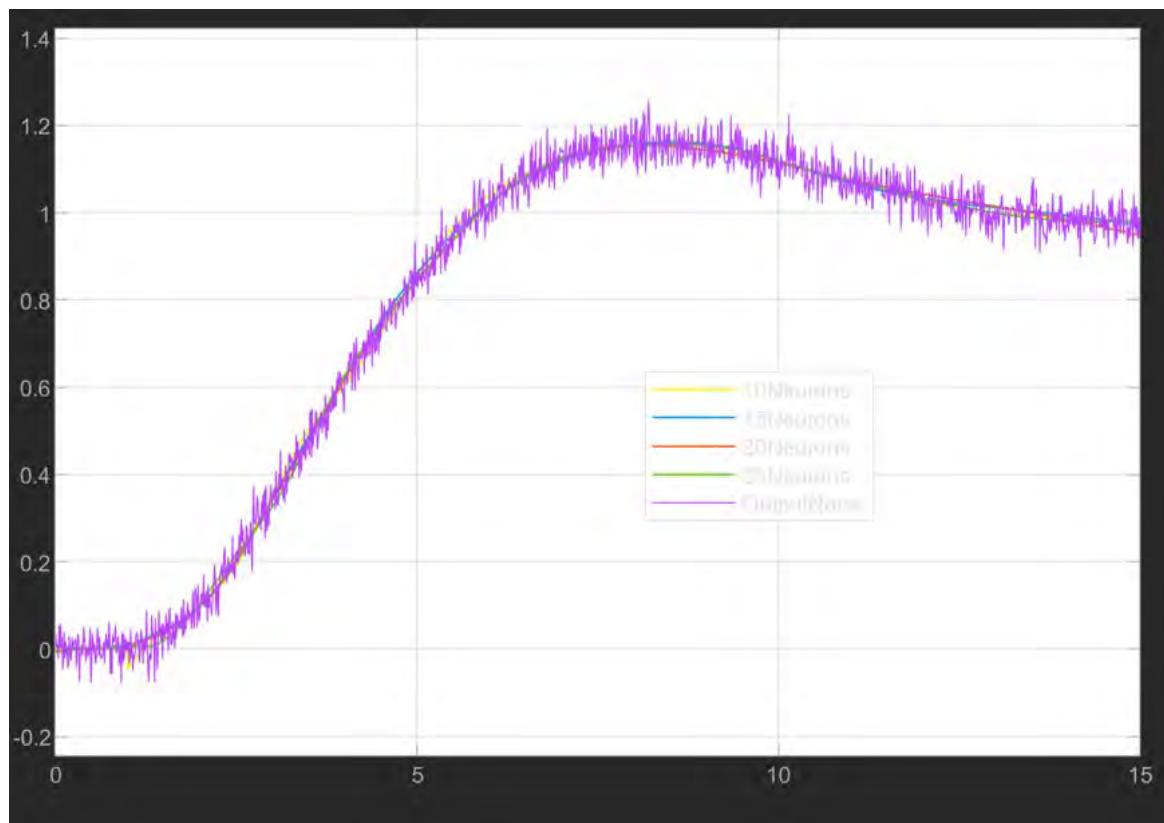




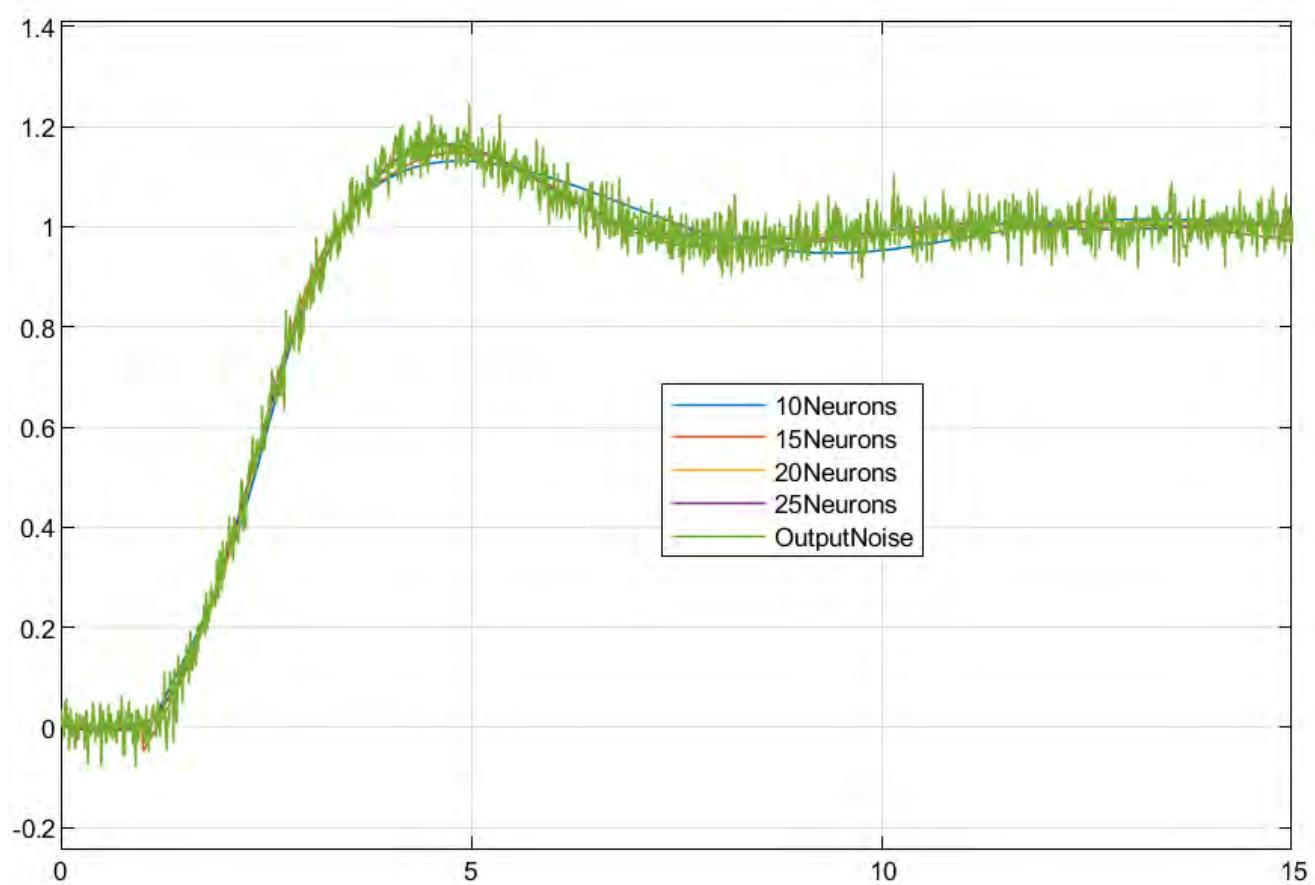
## Output Comparison



**Simulink Diagram**

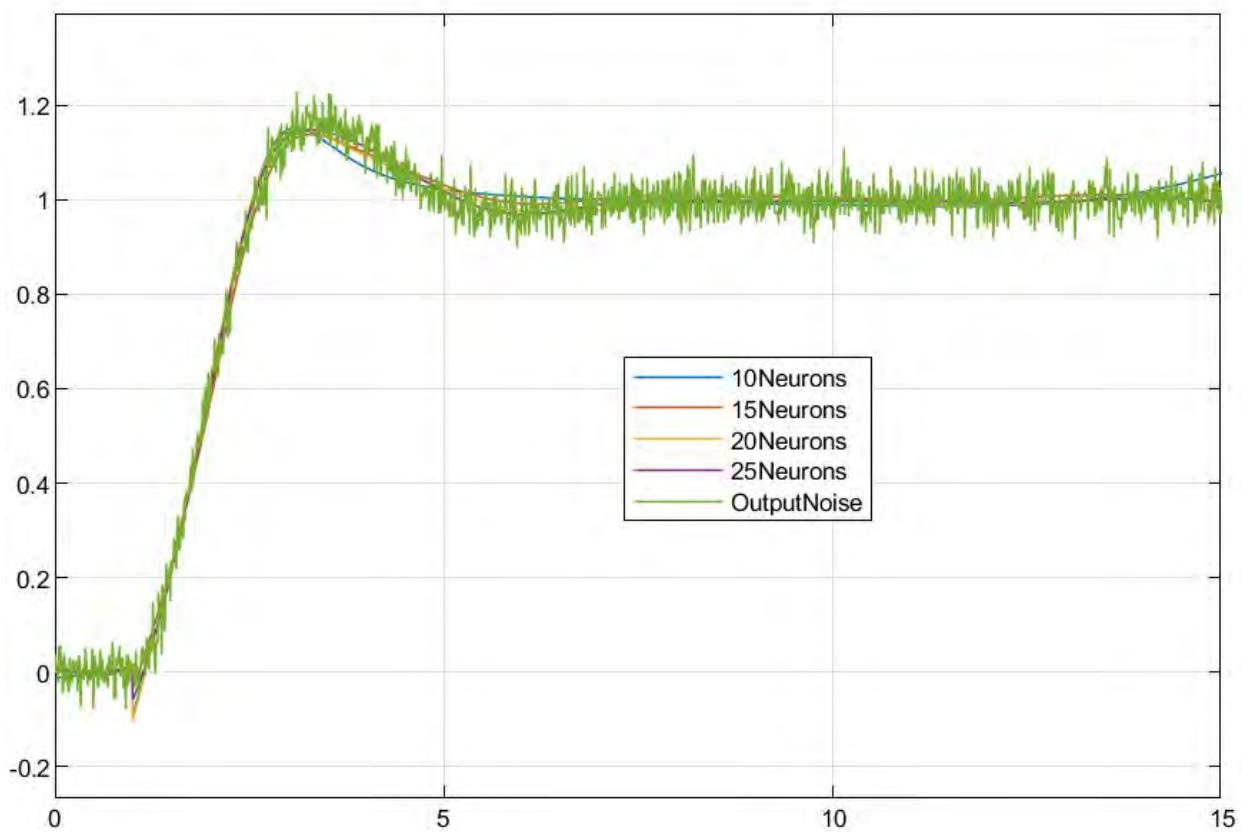


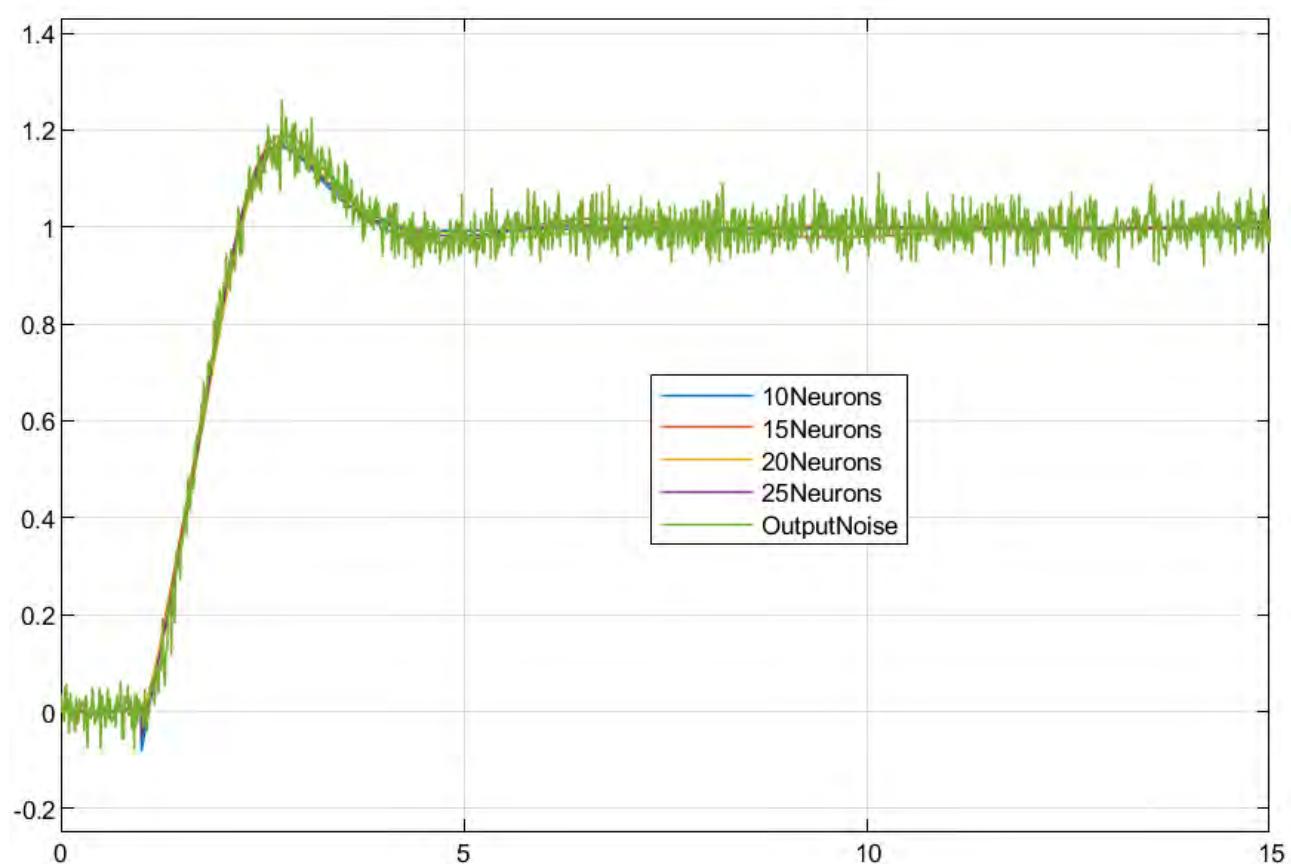
**Omega = 0.5**



Plot above,  $\omega = 1$

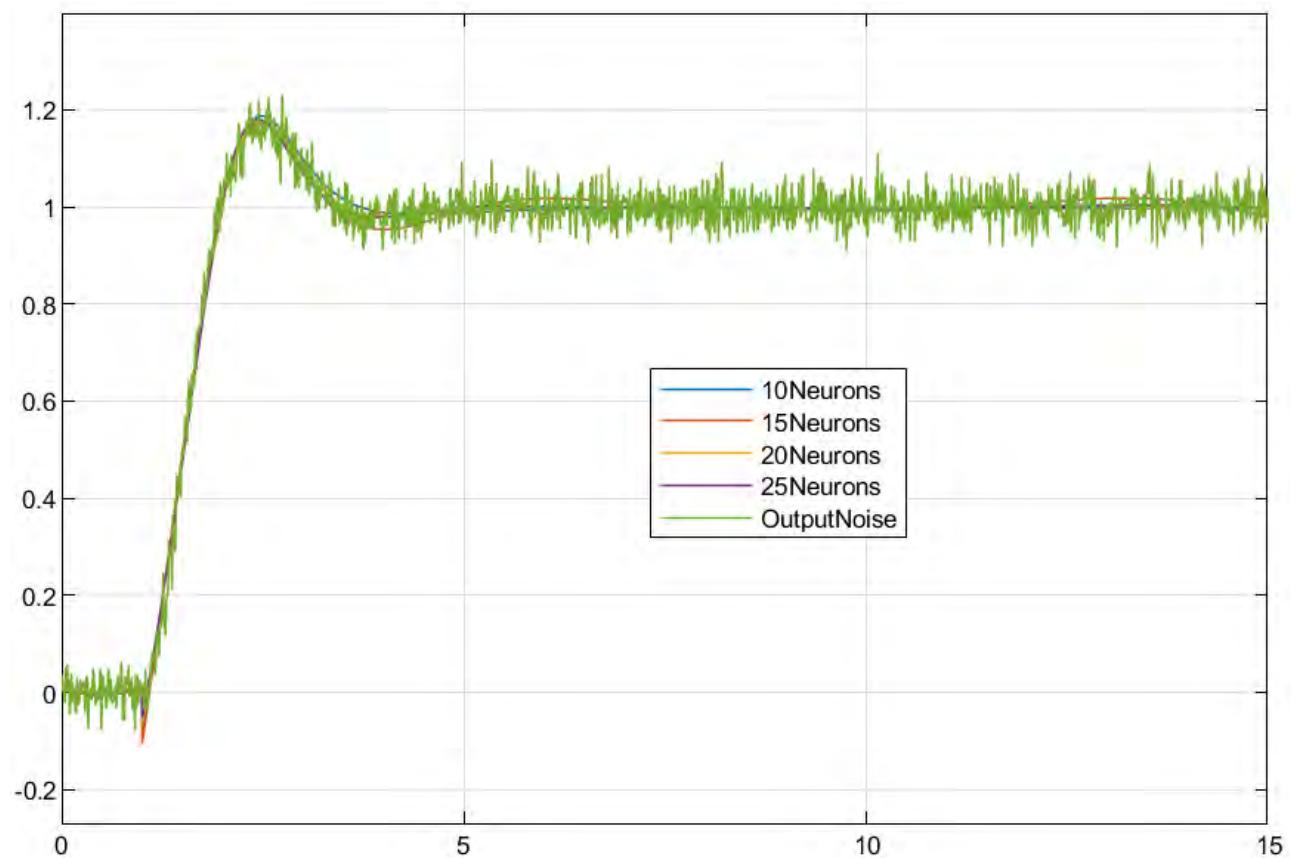
Plot below,  $\omega = 1.5$

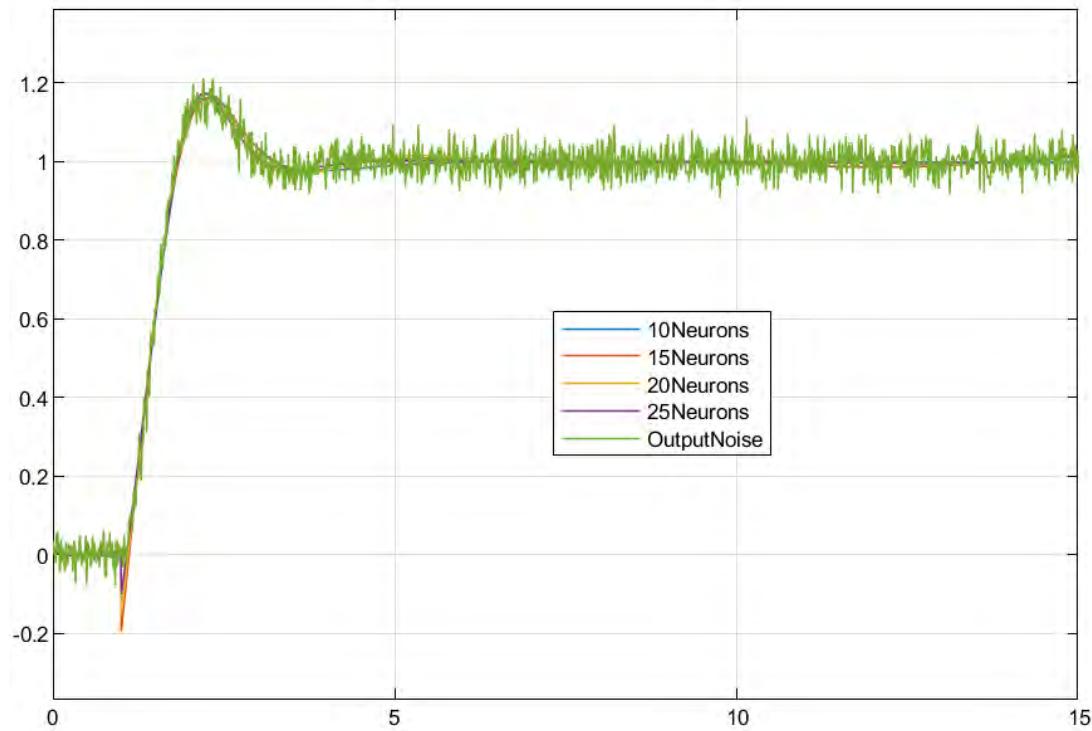




Plot above,  $\omega = 2$

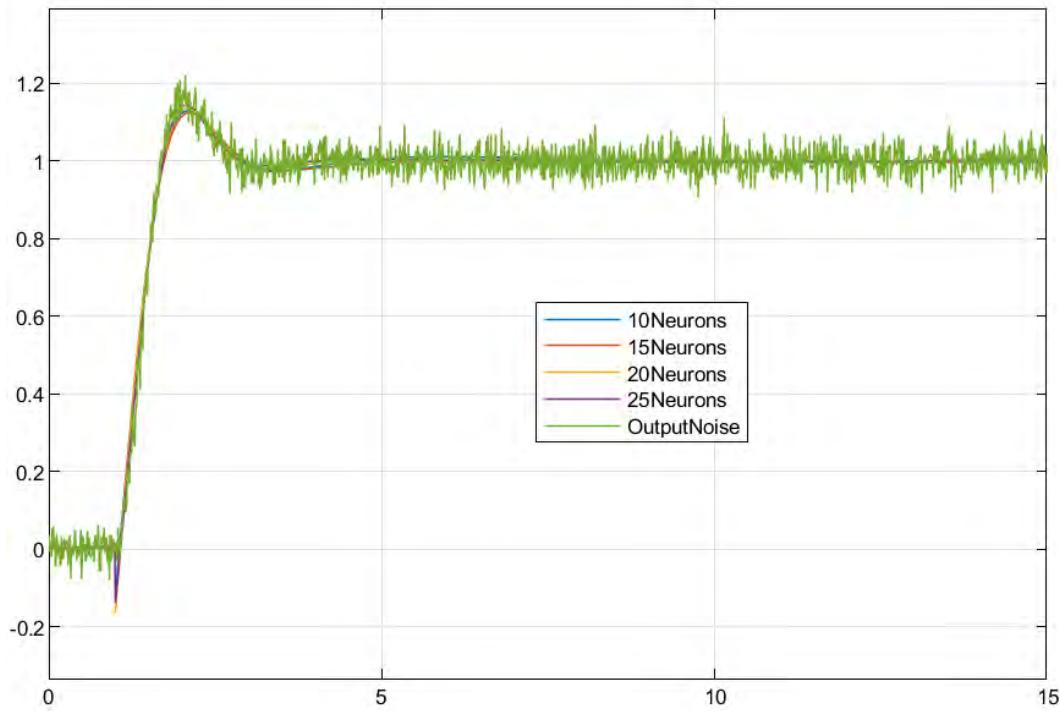
Plot below,  $\omega = 2.5$





Plot above,  $\omega = 3$

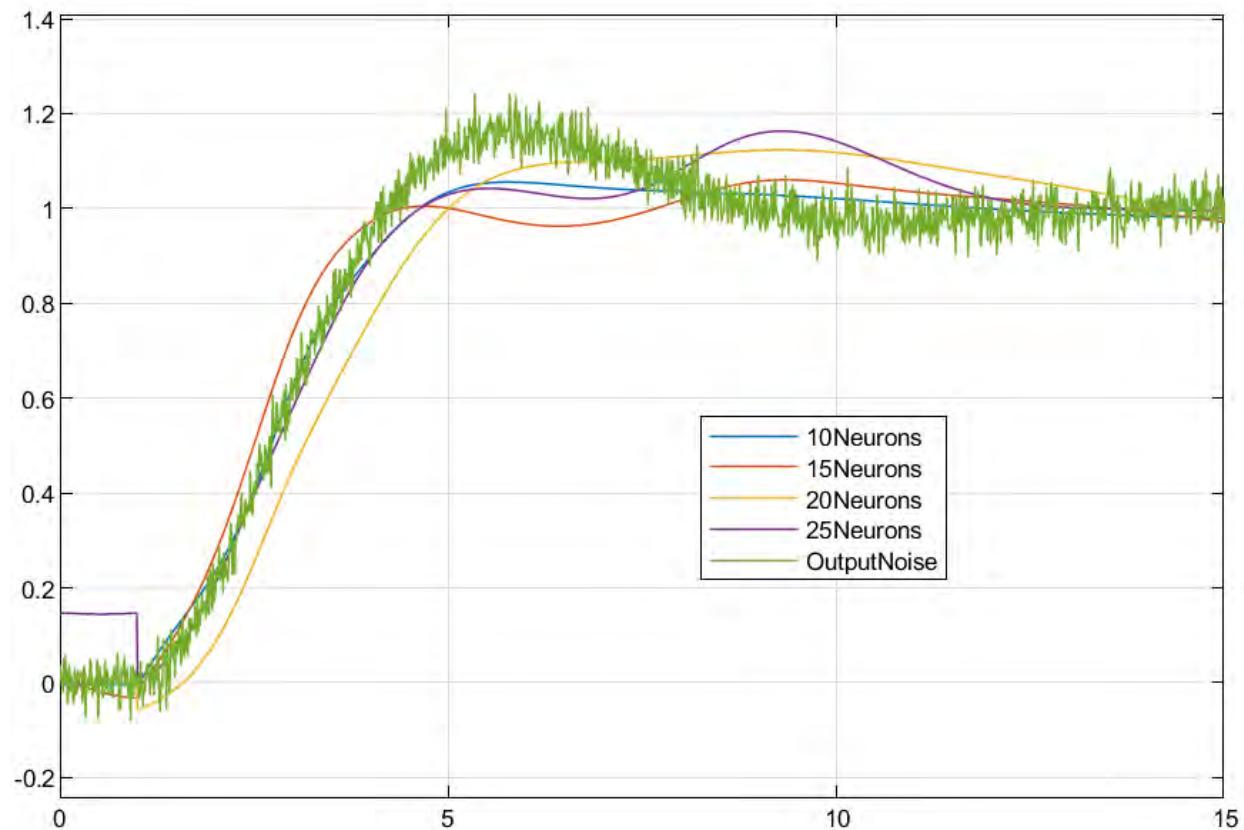
Plot below,  $\omega = 3.5$



**Conclusion** – Based on the validation data performance and plots above, increasing the number of neurons improves the fit. This is supported by the best performance for validation data and the general performance (in the Neural Network Training window) metrics.

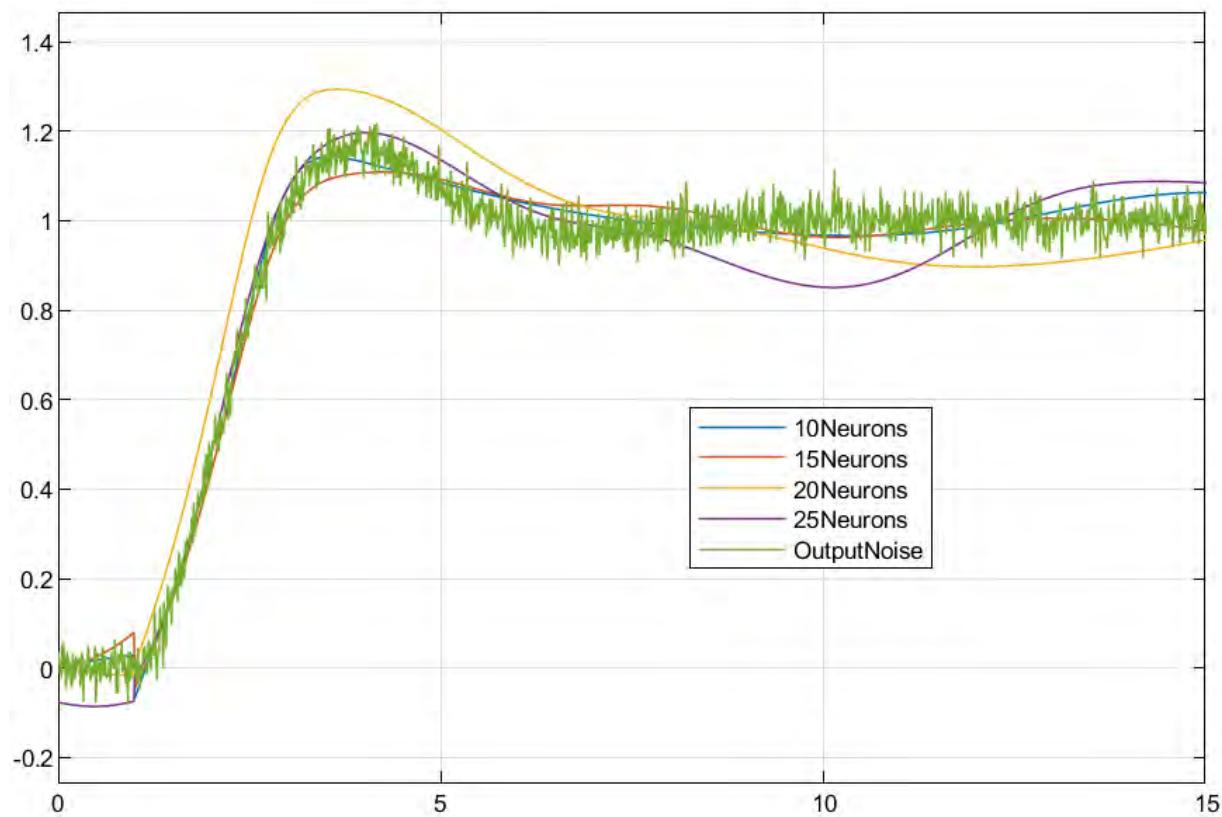
Other than that, the output is slightly erroneous in the transition state but is very good once steady state is reached for almost all neural networks. However, it is notable that other than the neural network with 10 Neurons, the output dips downwards just before the step input. The dipping amplitude decreases with the increase of neurons.

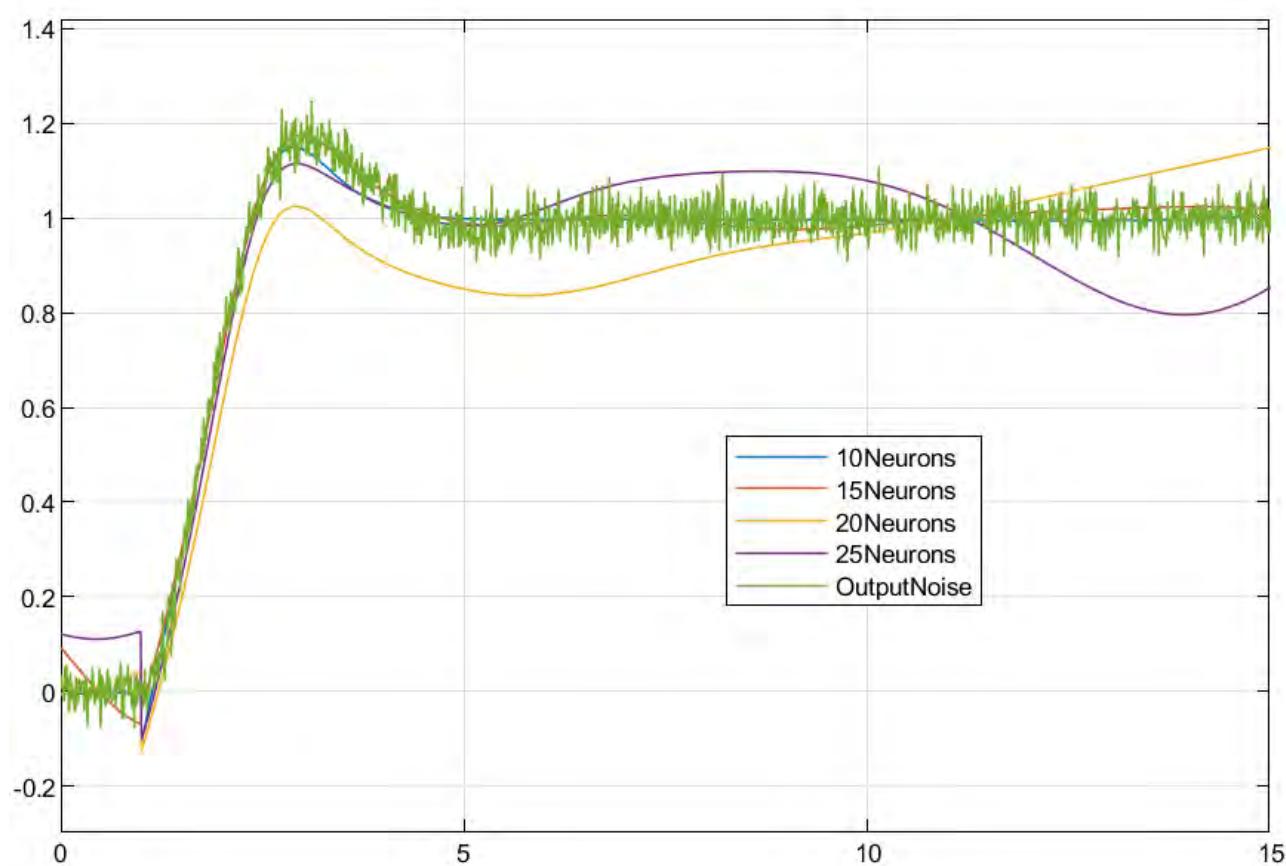
## b. Interpolation



Plot above,  $\omega = 0.75$

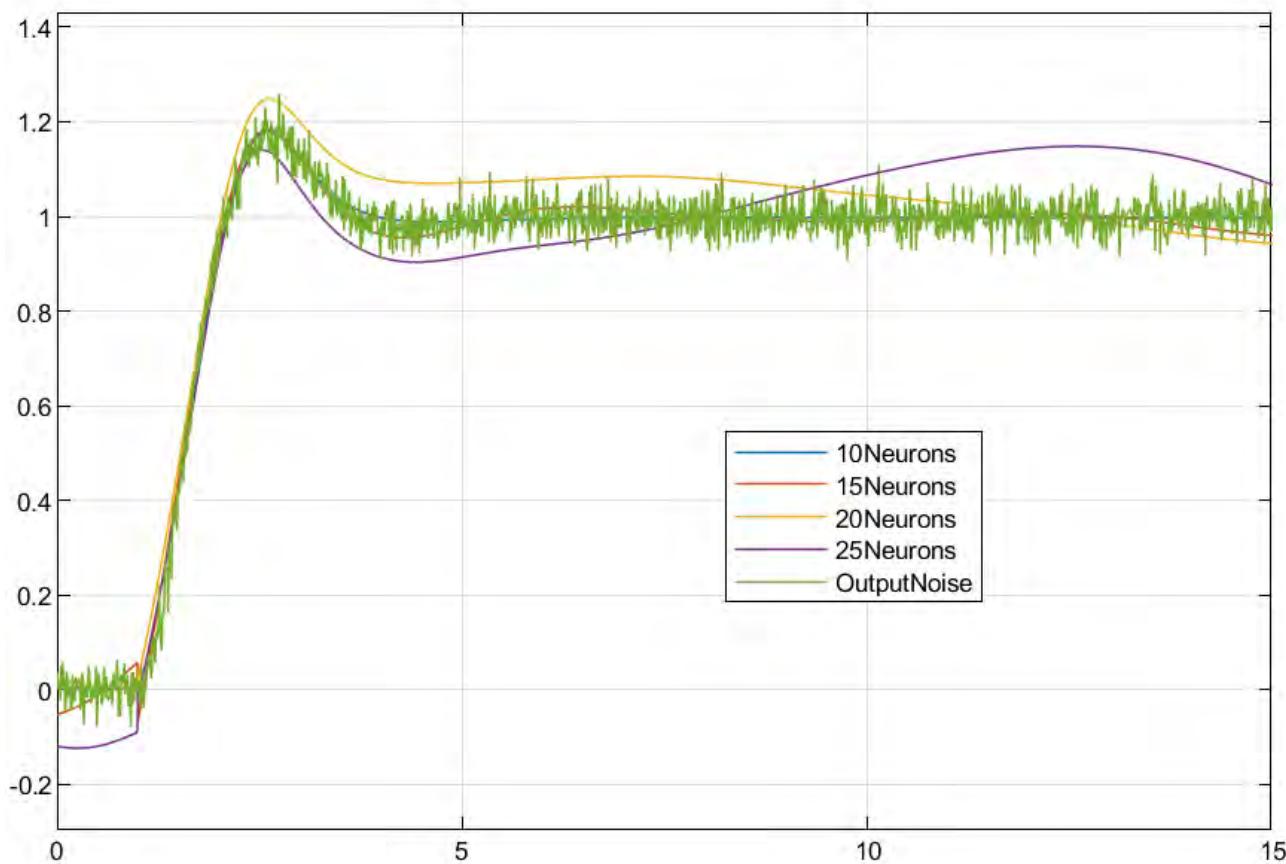
Plot below,  $\omega = 1.25$

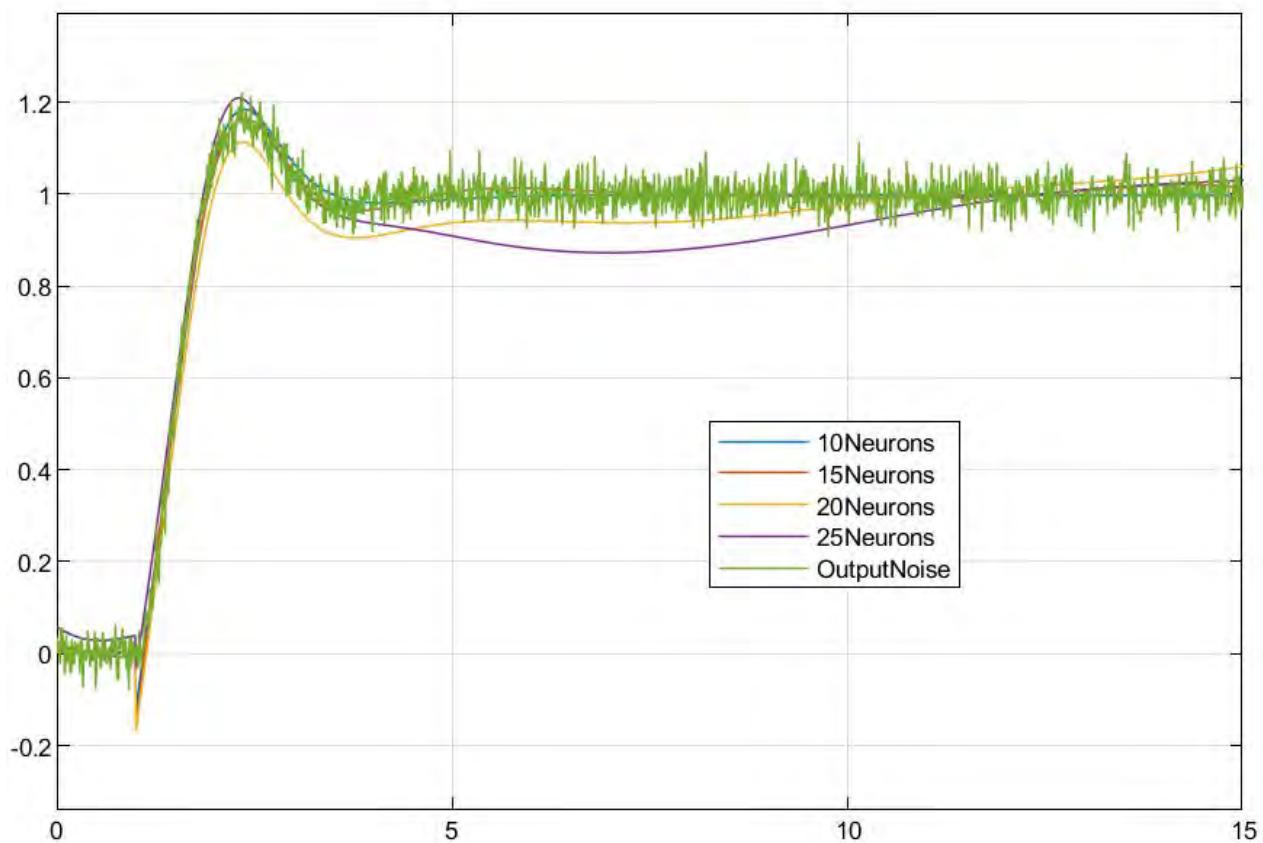




Plot above,  $\omega = 1.75$

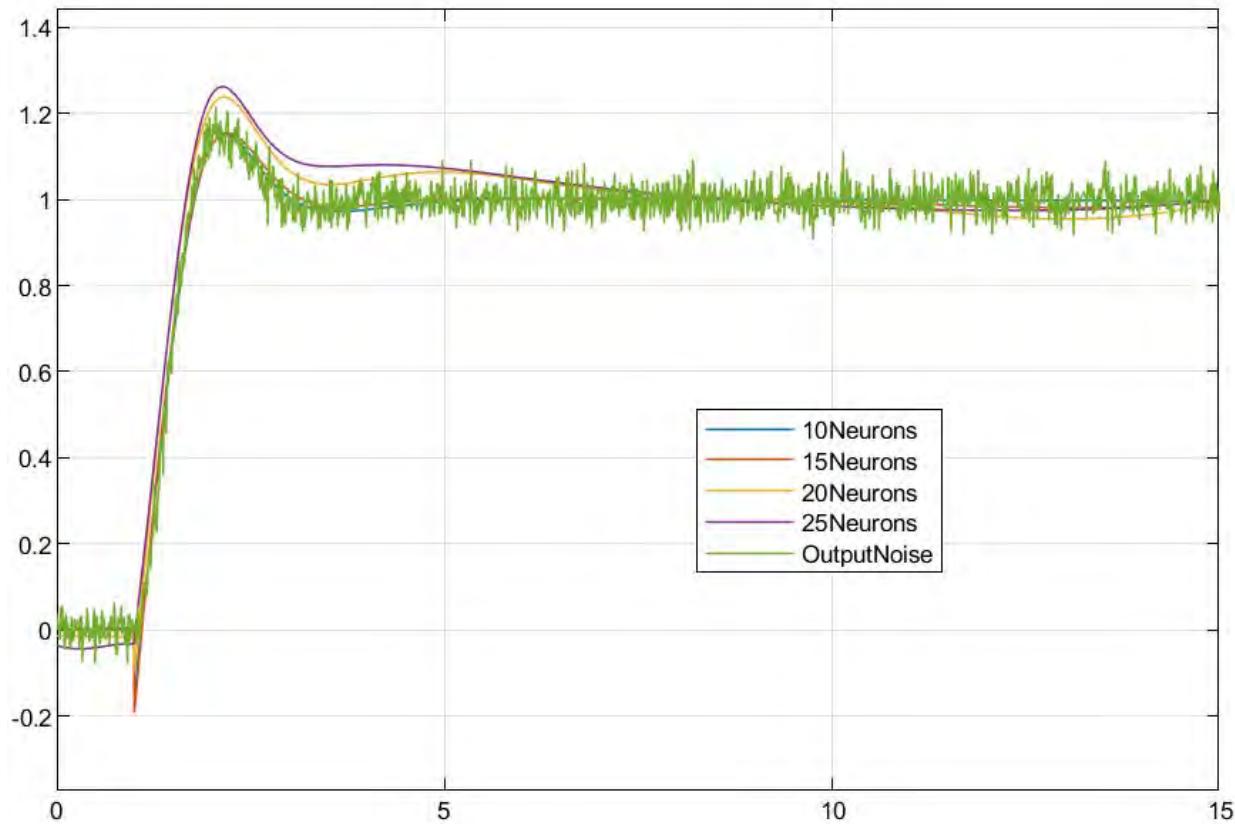
Plot below,  $\omega = 2.25$





Plot above,  $\omega = 2.75$

Plot below,  $\omega = 3.25$



**Conclusion** - Similar to the previous scenario without noise, it is observed that smaller frequencies are tougher to interpolate as compared to larger frequencies. The neural network with 10 neurons does the best job at interpolation as it provides the nearest output at  $\omega = 0.75$ . More number of neurons may cause overfitting which is why more neurons lead to less efficient interpolation.

- C.** The neural network with 10 neurons does a good job of interpolating frequency at lower as well as higher frequencies. Thus, **neural networks with around 10 neurons** will work effectively for this system if they are trained using inputs and outputs over all the operating frequencies. If the operating frequency doesn't change or stays constant at regions used for training data then more neurons may be good. Note that more neurons are good at fitting but they are not so good at interpolation for this system. Also, none of the neural networks is really close to the system so they might not be usable if the acceptable errors in output are less than  $10^{-3}$ . Although noise seems to have no effect on the ability of network to model the data, its effect can be clearly seen when comparing the best validation performance and the mean squared error values. Thus, **noise reduces the fitting ability of the network** for this system.

**EE5327, Fall 2020**  
**Homework 6: Fuzzy Systems**  
**Due 12/1/2020**

As you did for Homework 5, use the same model and data. Recall you implemented the following system in Simulink and assumed a damping ratio (zeta) of 0.5, a fixed step integration with a time step of 0.01 sec, and a unit step input occurring at 1 second, with a simulation stop time of 15 seconds. You generated a training dataset of (time, input, omega) as inputs and y as the output for values of omega 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, and 3.5 rad/sec.

$$y(s) = \frac{s + 2}{s + 5} u(s)$$

**Problem 1) Fuzzy Inference System Estimator – No Noise (60 points)**

- a) Use the neuroFuzzyDesigner tool to fit a few fuzzy inference systems. First, let's see which one has the best fit to the training data. Try the following making note of the final error after 3 epochs of training (to save time):
  - 1) 3-3-3 with *triangular* membership functions and *linear* output.
  - 2) 5-5-5 with *triangular* membership functions and *linear* output
  - 3) 3-3-3 with *generalized bell* membership functions and *linear* output.
  - 4) 5-5-5 with *generalized bell* membership functions and *linear* output.
- b) Export each FIS model to the workspace and include in the Simulink simulation using a Fuzzy Logic Controller block. Generate comparisons of the FIS model with the training data. Make any observations about the type of FIS and the quality of fit it produces.
- c) Now test the FIS by comparing its output with the true system output for omega values 0.75, 1.25, 1.75, 2.25, 2.75, and 3.25 rad/sec. Make any observations about the type of FIS and the quality of fit it produces for this test data.
- d) Finally, comment on the overall ability of the FIS to work or not work for this application. **Note: the model produced by this neural-fuzzy system will likely not be nearly as good as the neural network produced.**

**Problem 2) Fuzzy Inference System as Estimator with Noisy Data (40 points)**

For Problem 2, repeat the steps of Problem 1 but add noise with variance 0.001 to the output data used for training. Comment on the effects of noise on the ability of the fuzzy system to model these data.

**ATUL SHROTRIYA**

**UTA ID - 1001812437**

**Fuzzy Systems**

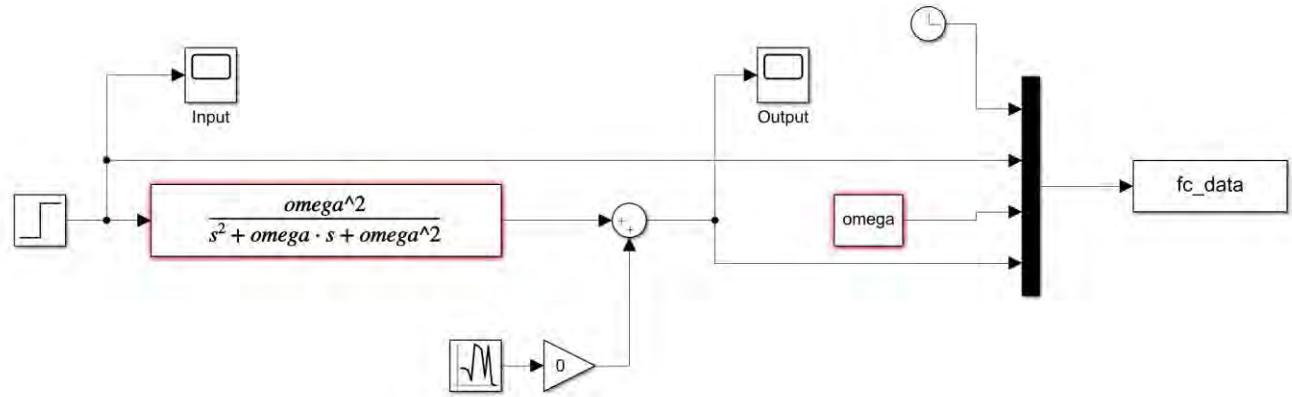
**Homework - 6**

**Submitted - December 1, 2020**

**Course:**

**System Identification and Estimation**

## Problem – 1



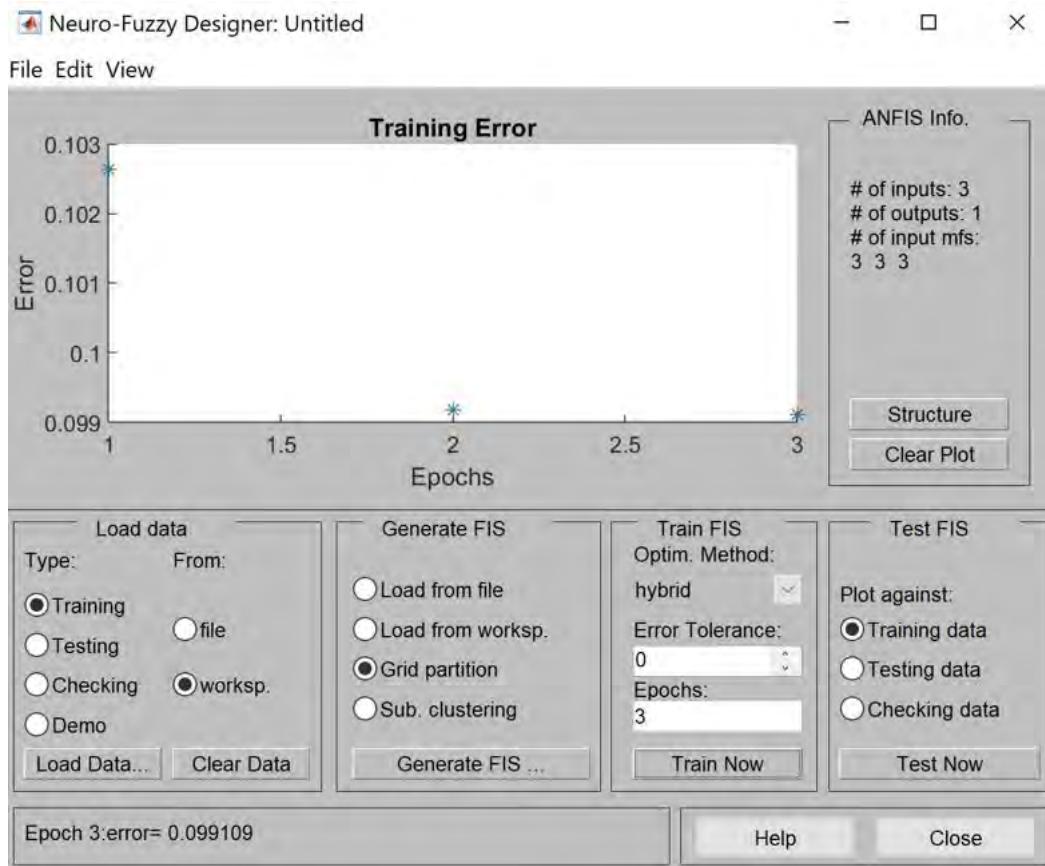
a.

Simulink model to generate training data

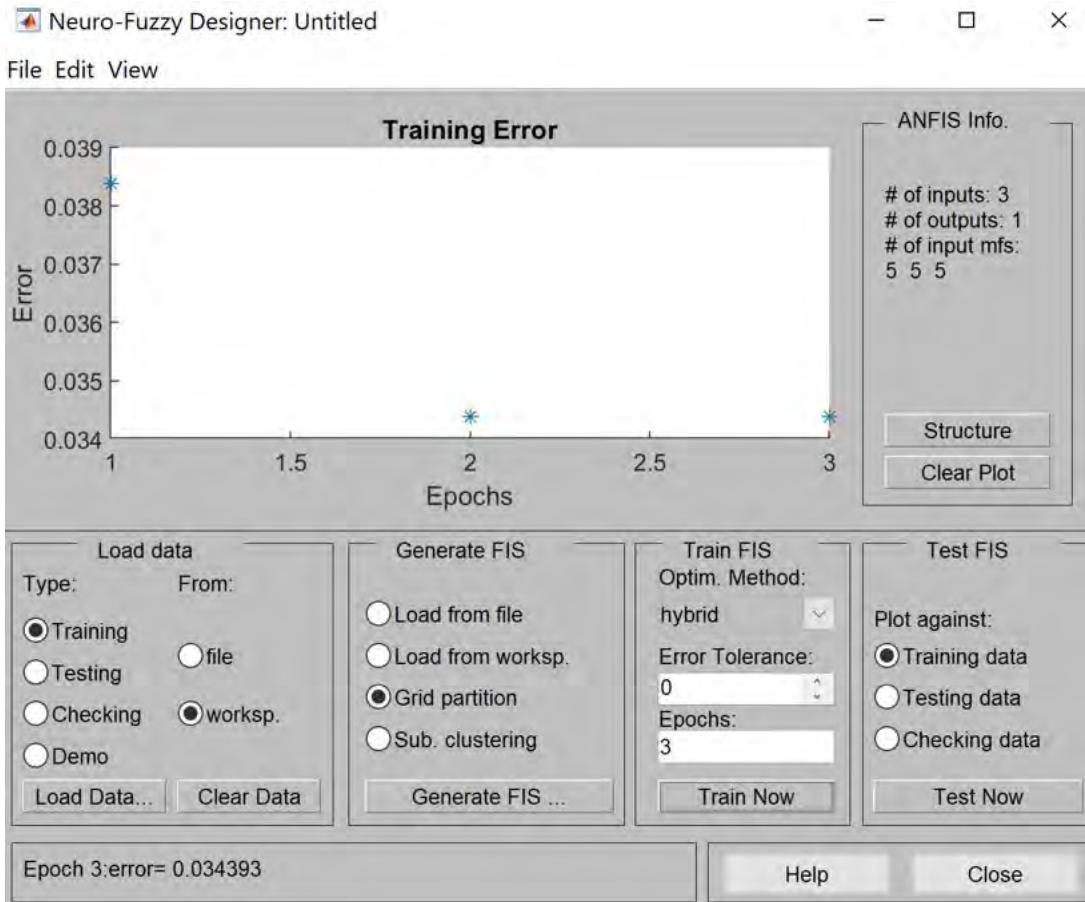
### MATLAB commands for generating the training data set

```
omega=0.5
fc_trdata=fc_data;
omega=1
fc_trdata=[fc_trdata;fc_data]
omega=1.5
fc_trdata=[fc_trdata;fc_data]
omega=2
fc_trdata=[fc_trdata;fc_data]
omega=2.5
fc_trdata=[fc_trdata;fc_data]
omega=3
fc_trdata=[fc_trdata;fc_data]
omega=3.5
fc_trdata=[fc_trdata;fc_data]
```

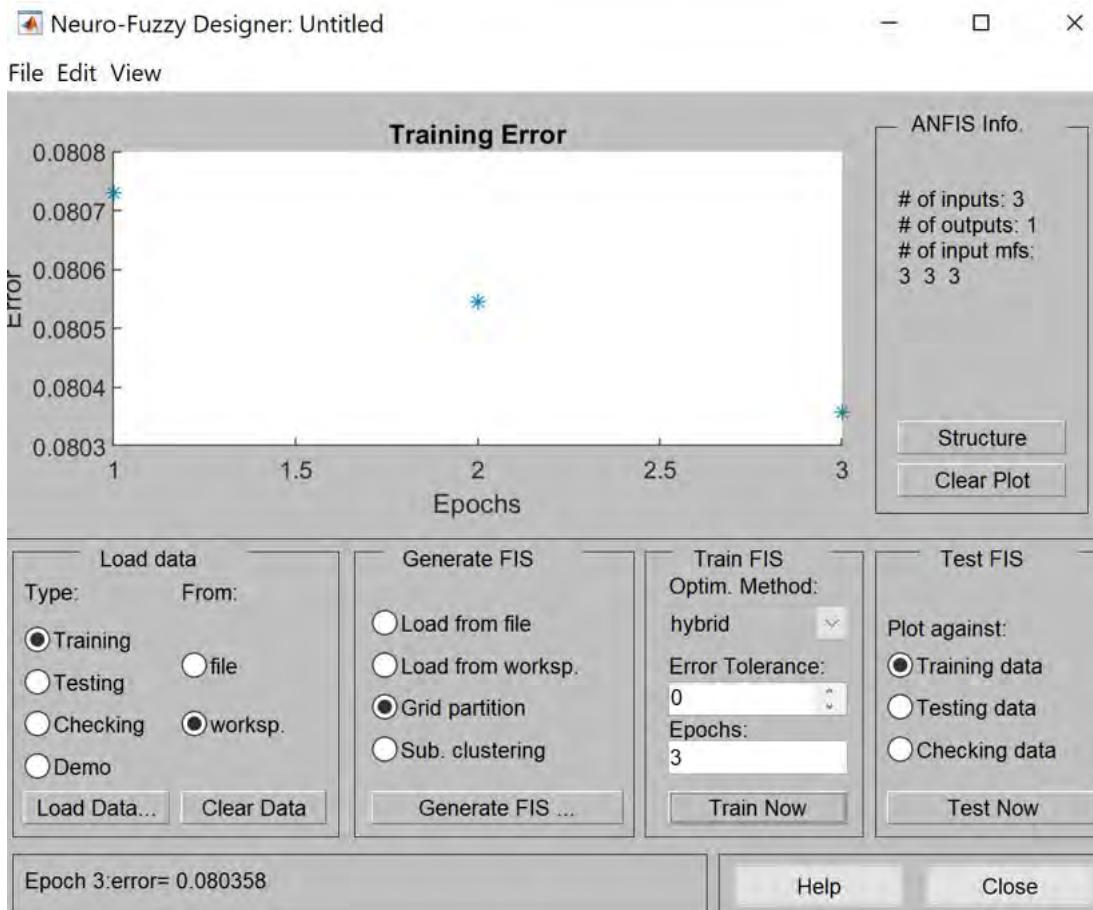
# TMF 333



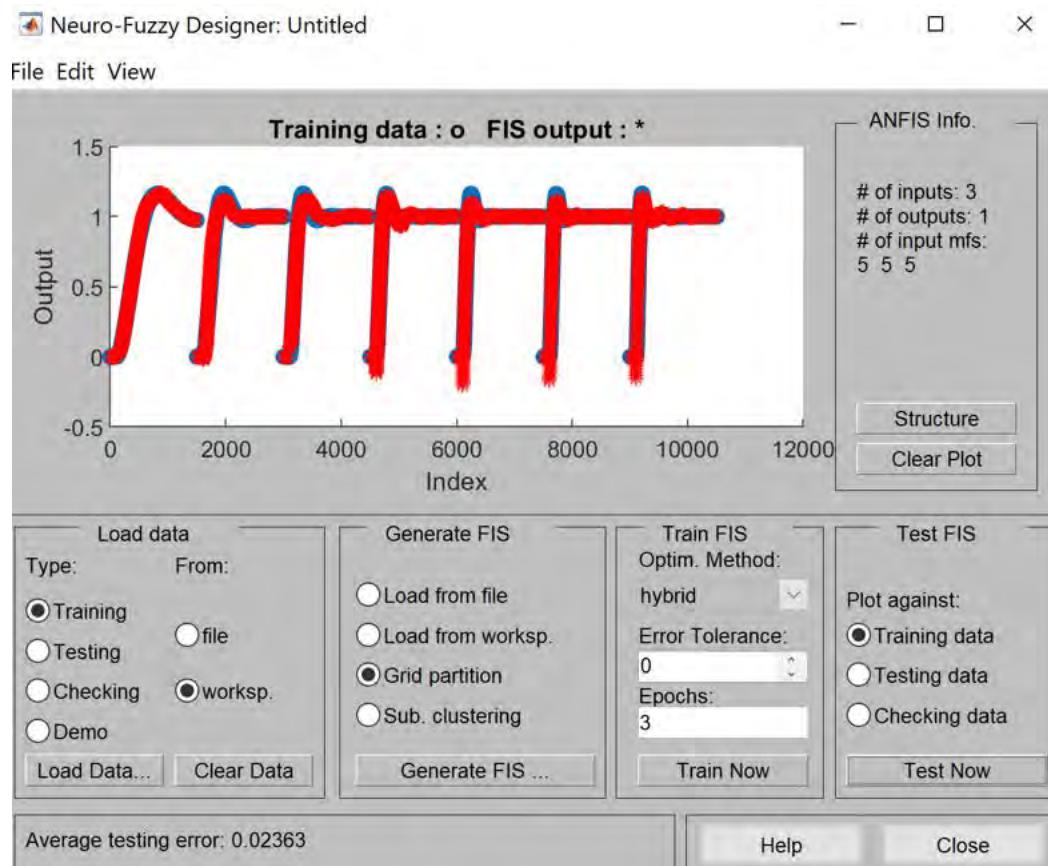
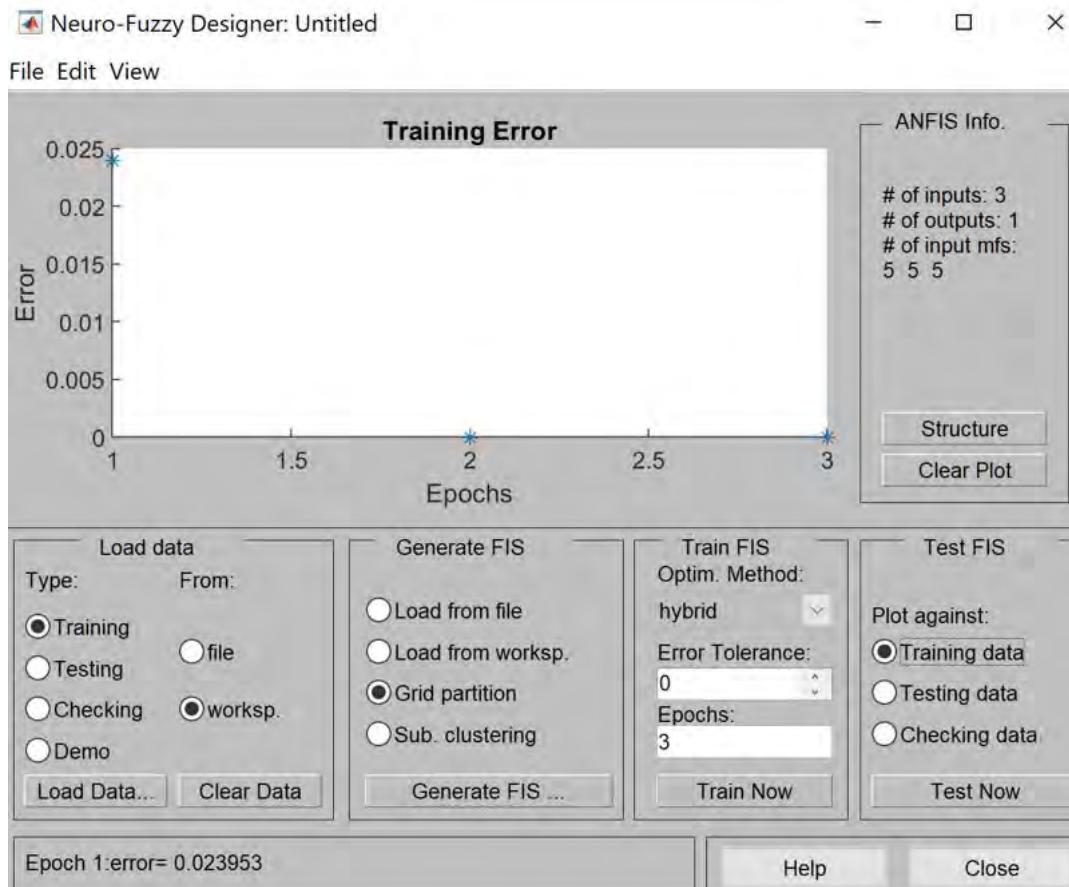
# TMF 555



# GBELL 333

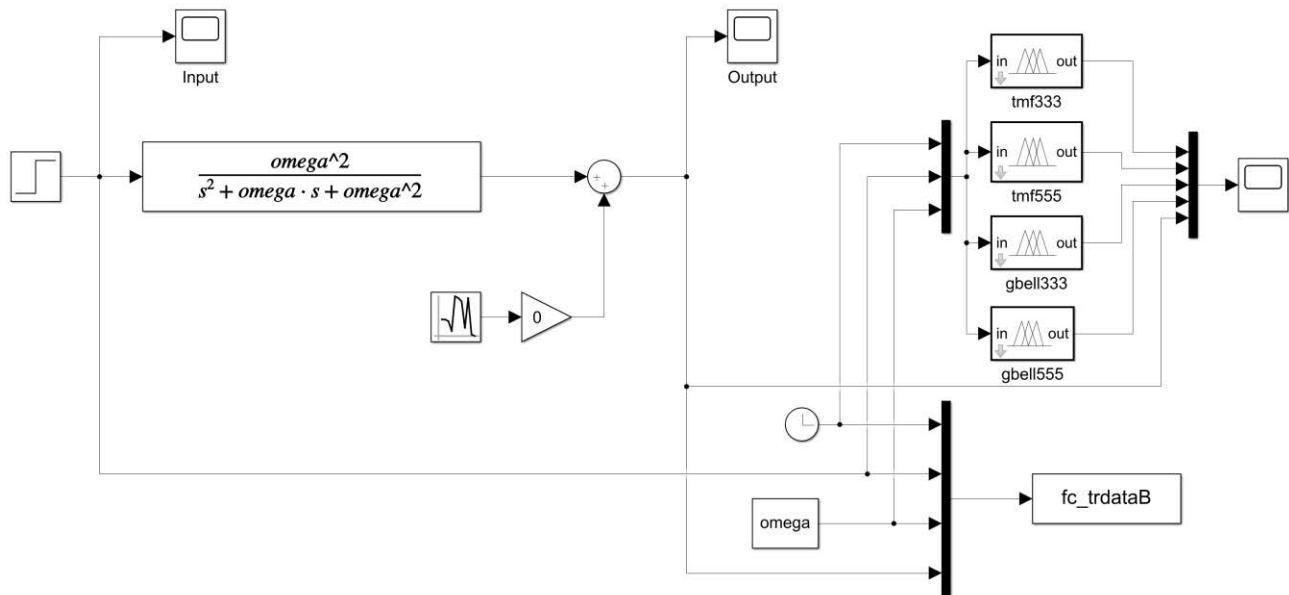


## GBELL 555



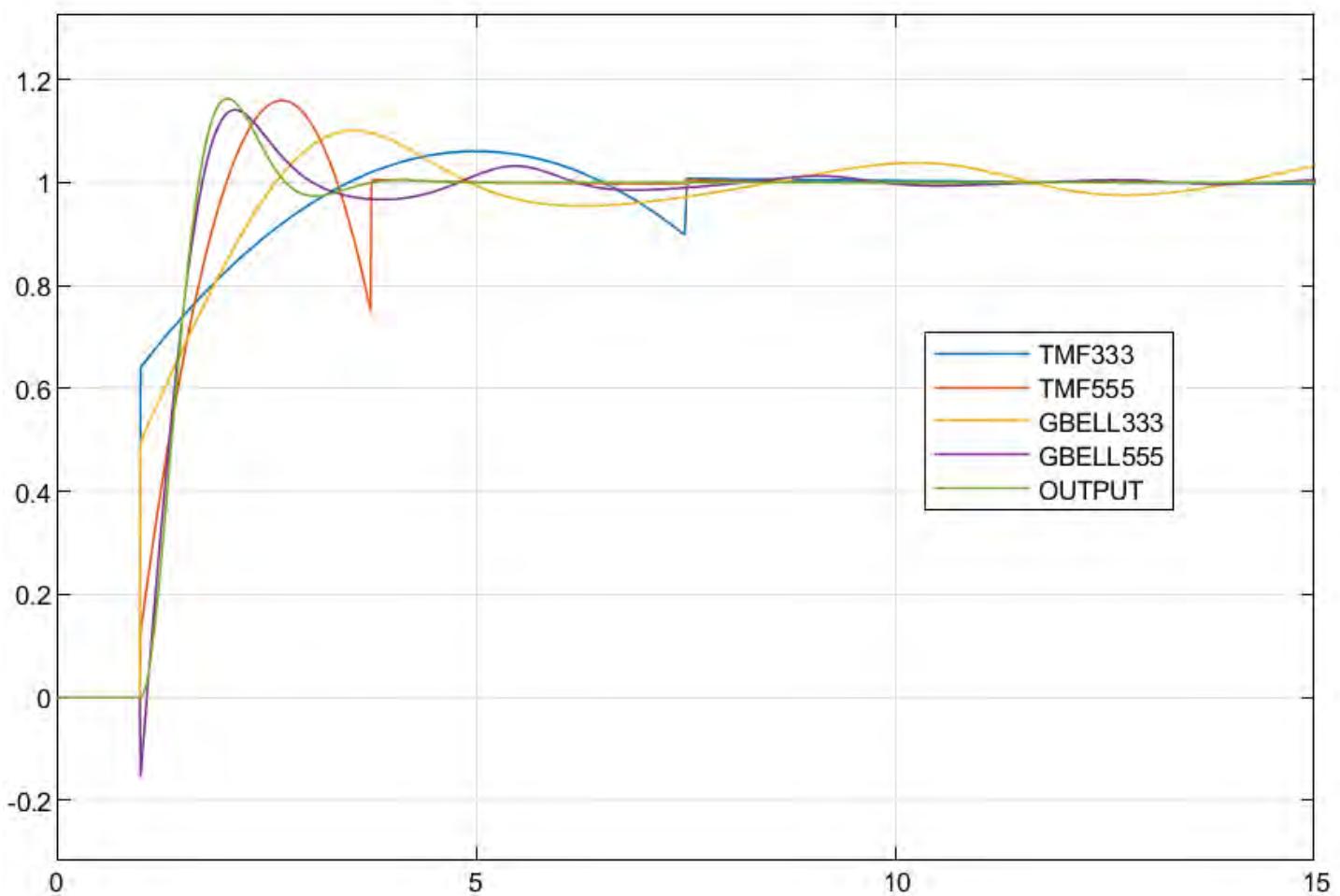
**Conclusion – Using average testing error values, GBELL 555 produces the best fit to training data followed by TMF 555**

## b. Comparison

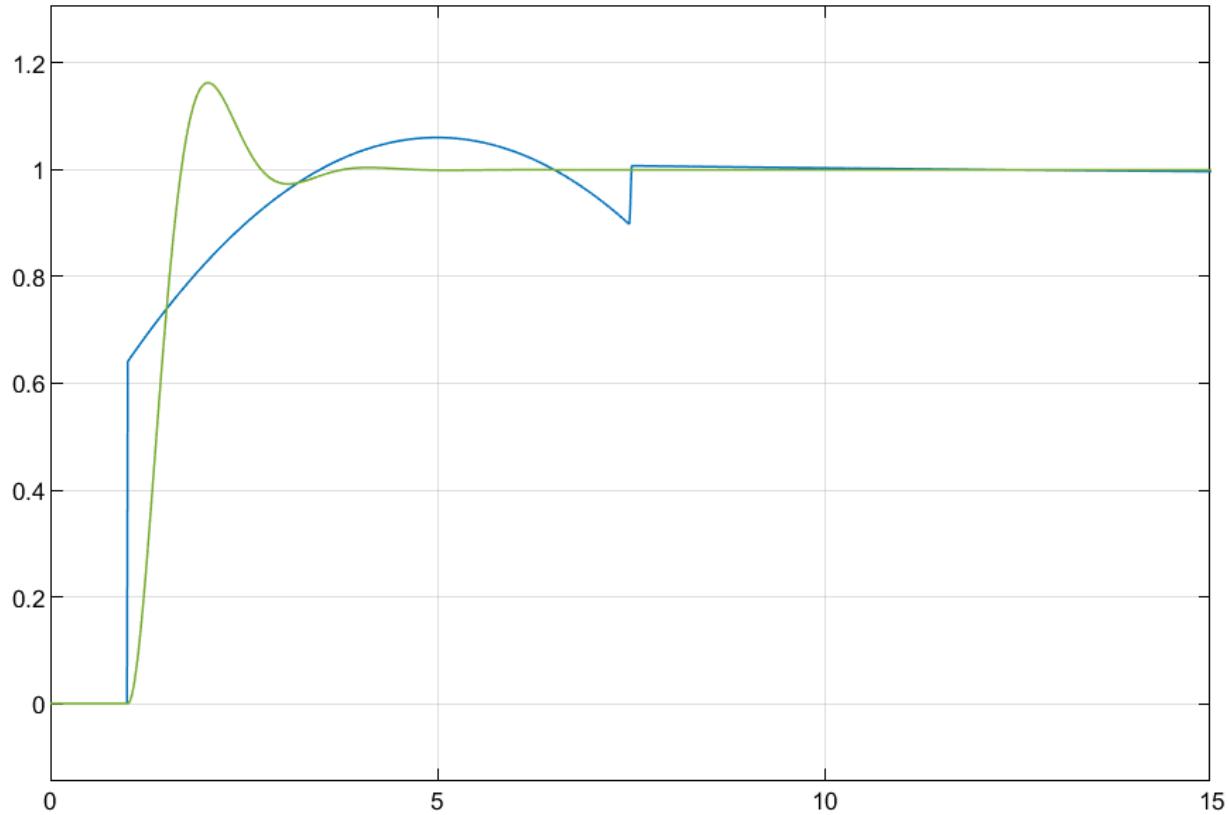


Simulink Diagram for comparing all the outputs

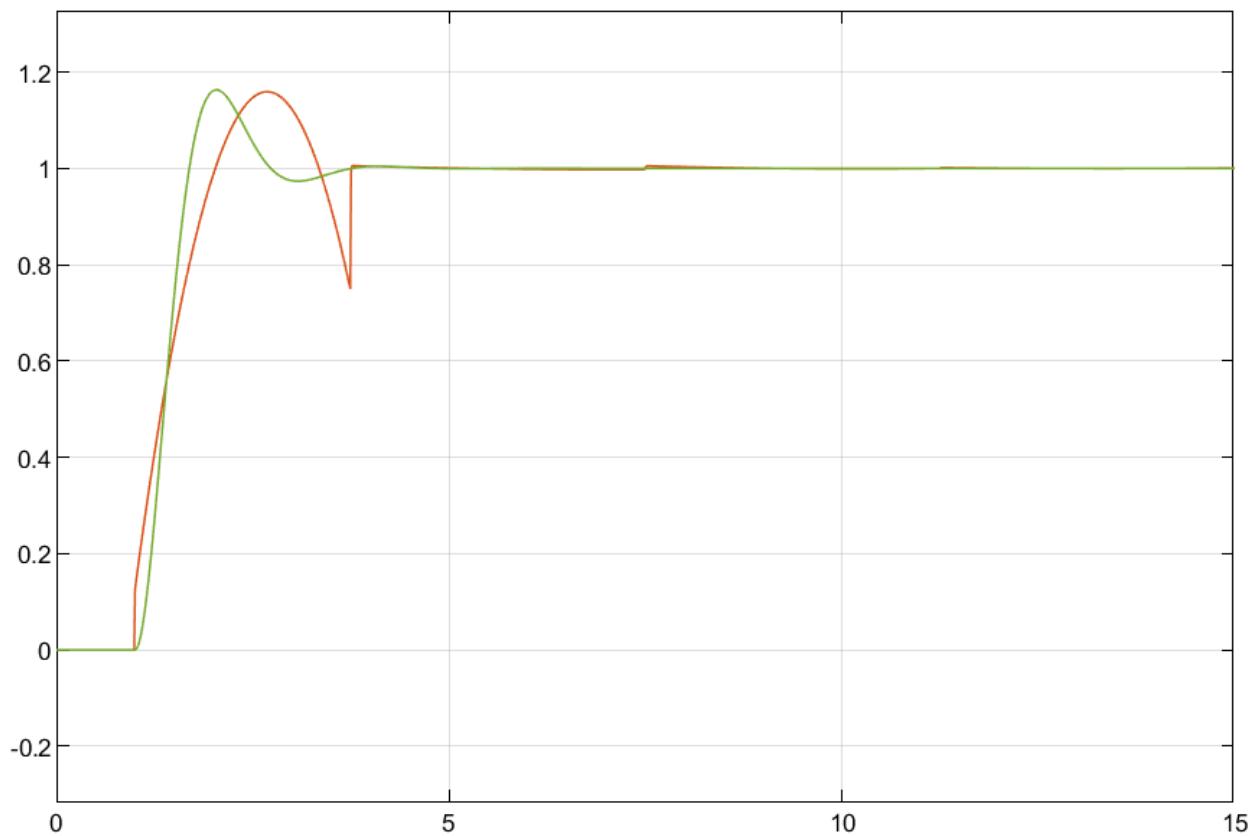
At Omega = 3.5



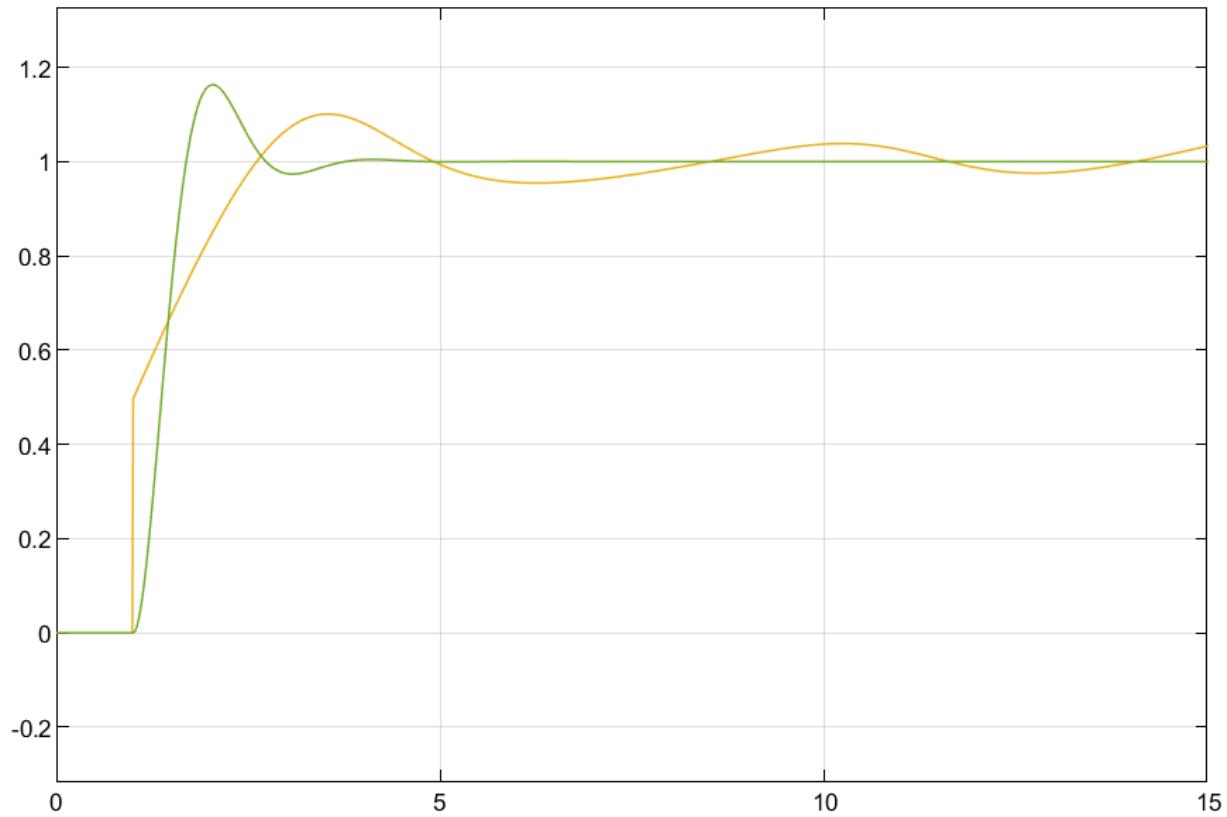
**TMF 333**



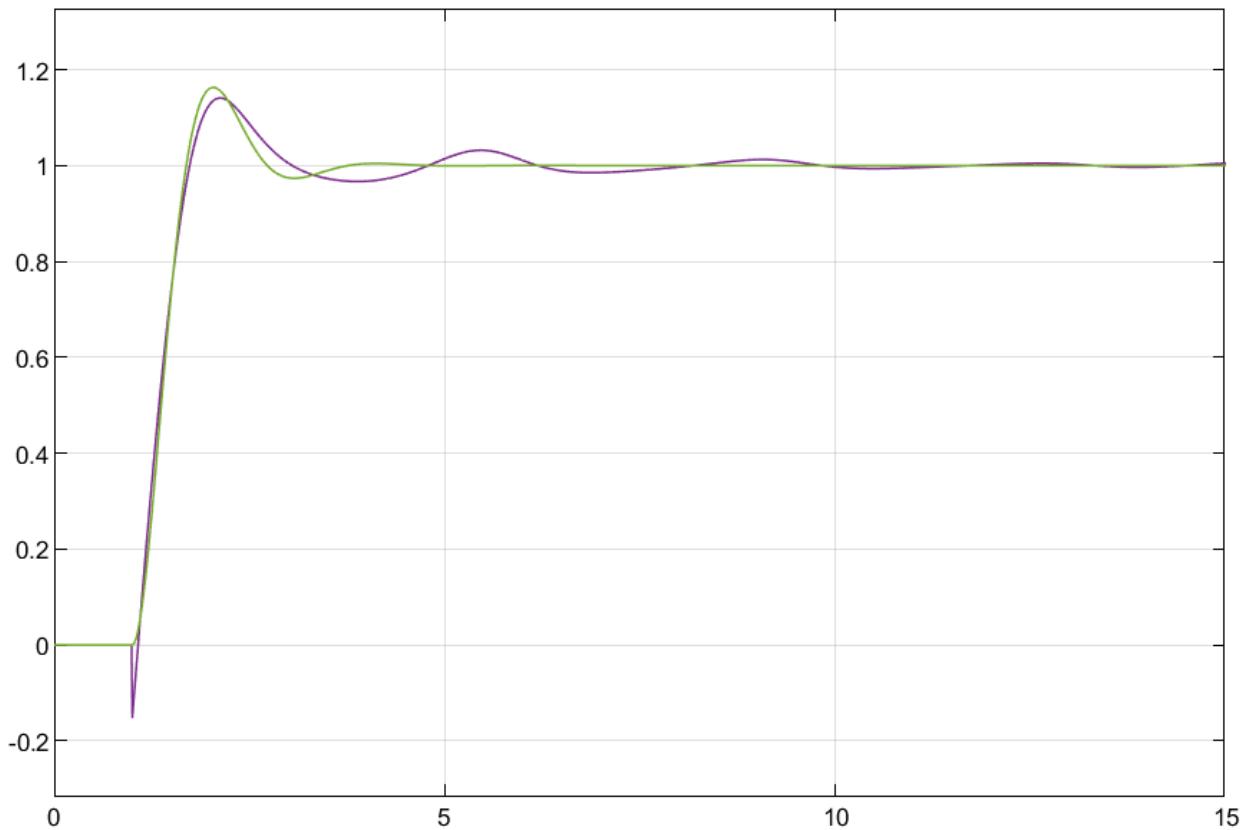
**TMF 555**



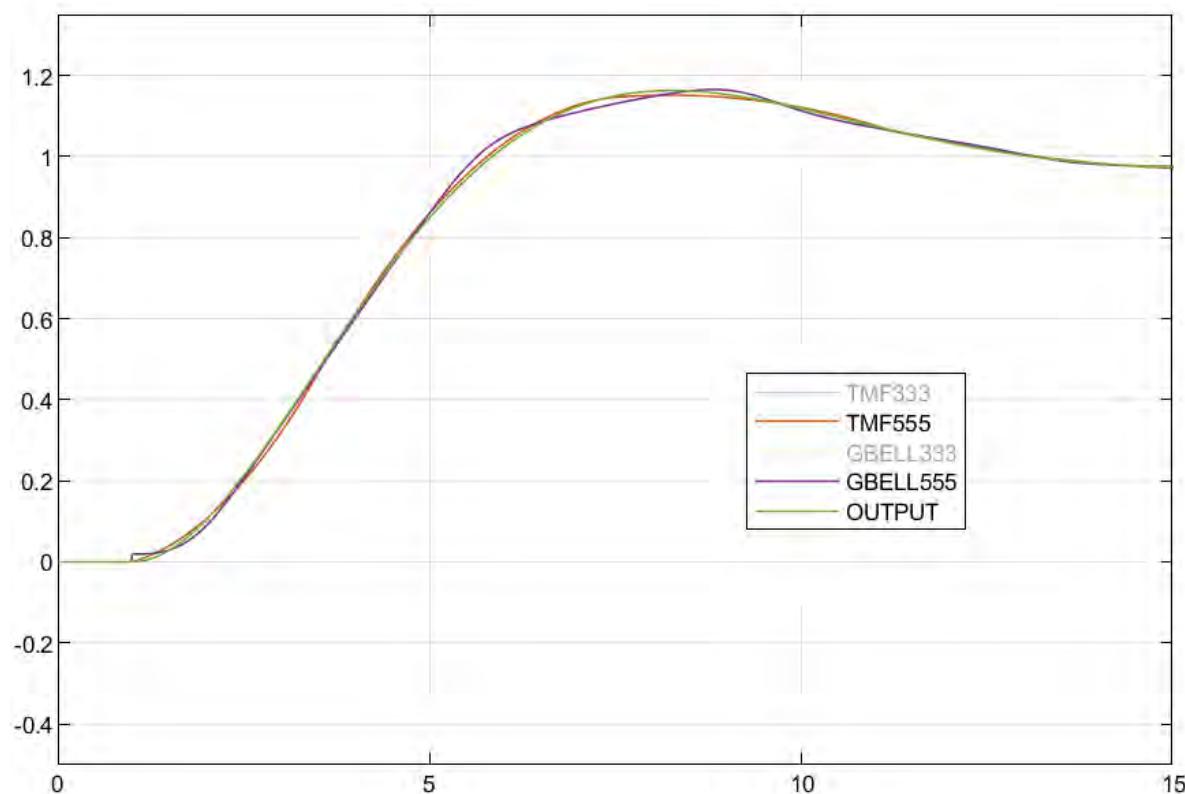
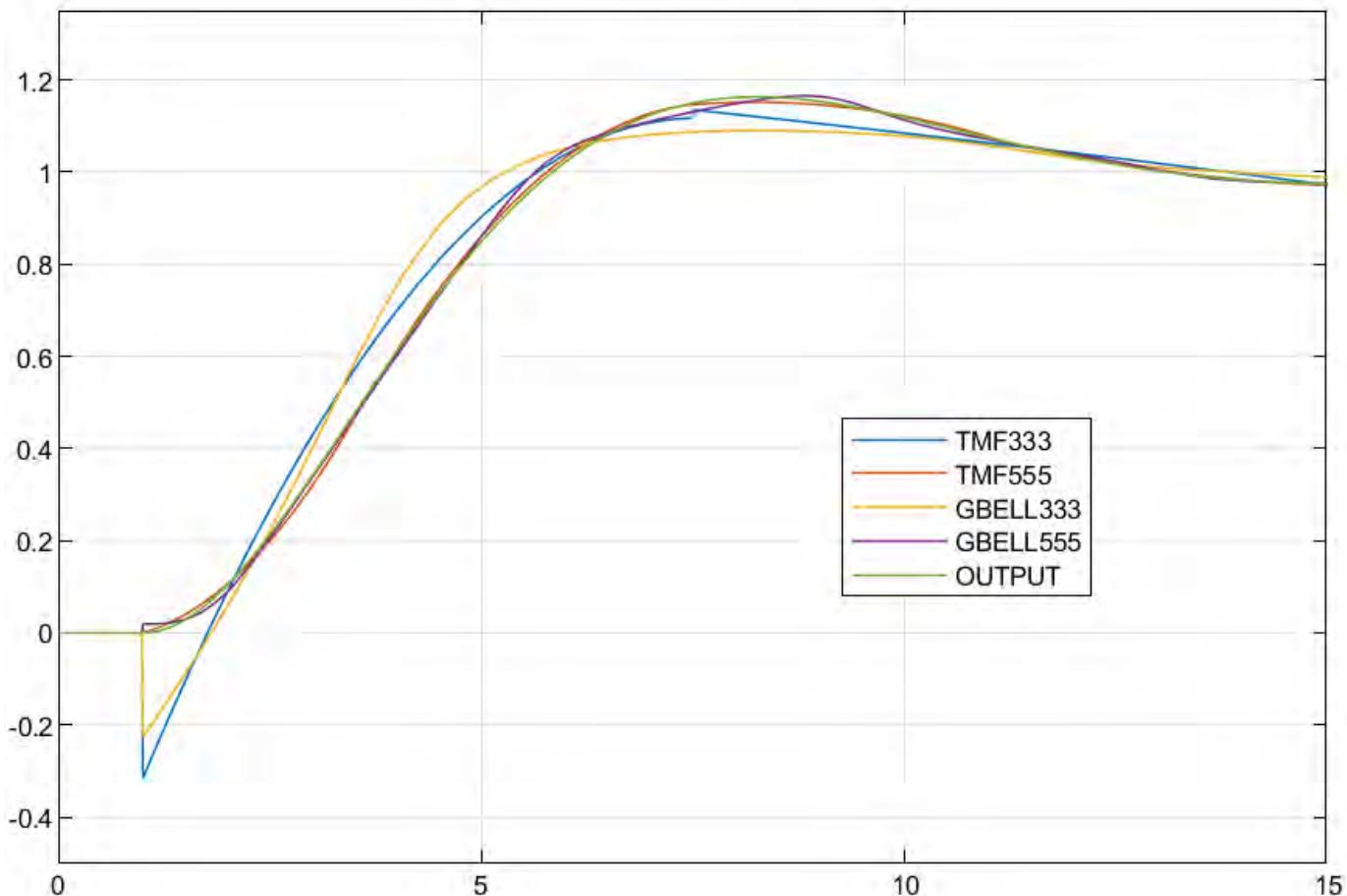
**GBELL 333**



**GBELL 555**



### At Omega = 0.5



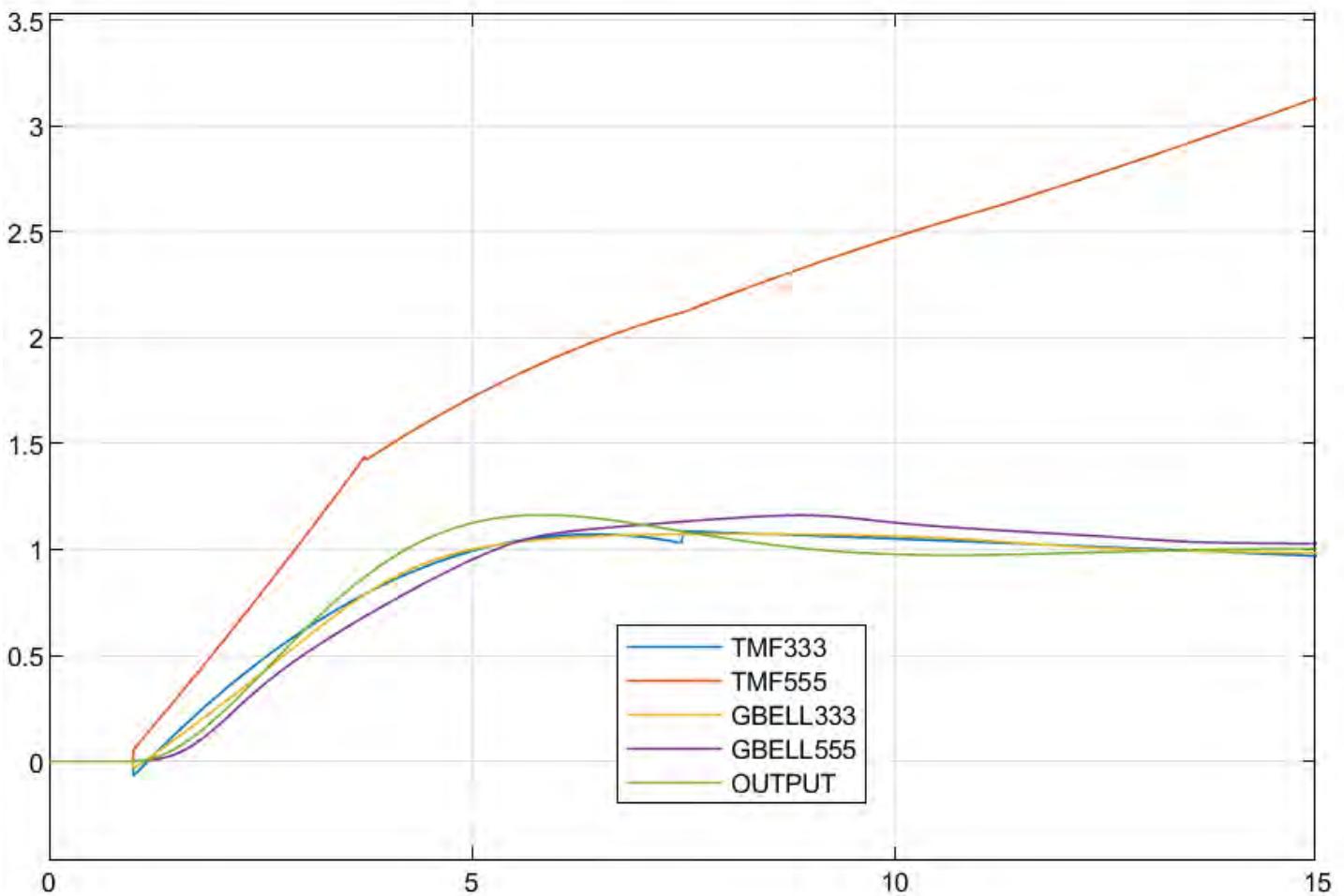
**Conclusion –** From the above plots it is observed that all fuzzy systems do a good job of matching the actual output at low omega values. As expected from the fit data in part a, the GBELL 555 and TMF 555 produce the best fits at omega = 0.5. On increasing the values of omega, the damping produces quicker transition to the steady state. This leads to divergence from the output but the GBELL 555 still produces a similar output to the actual output although with more oscillations. **It is seen that FIS with more membership functions is better at estimating the output.**

The **triangular membership function has sharp edges** due to the straight triangular edges used for defining the output. This is not seen in gaussian membership functions as they have smooth curves. The above values are for omega=3.5. As it can be seen, for lower omega values almost all fuzzy identification systems work well with the best ones being TMF555 and GBELL555 due to the higher number of functions.

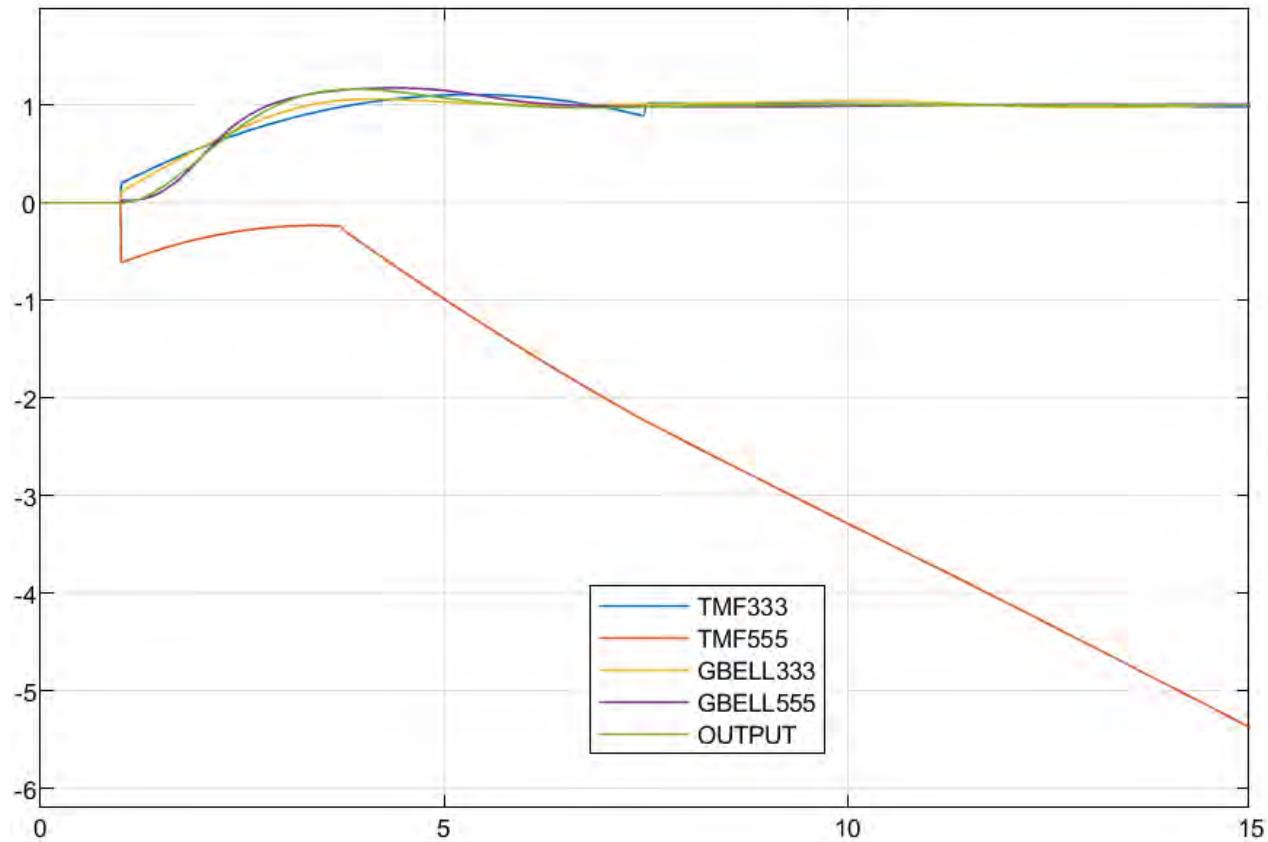
Other than that, it is also noted that while the GBELL 555 FIS produces the best fit, it gives a sharp dip before the input at higher omega values.

### c. For different omega values

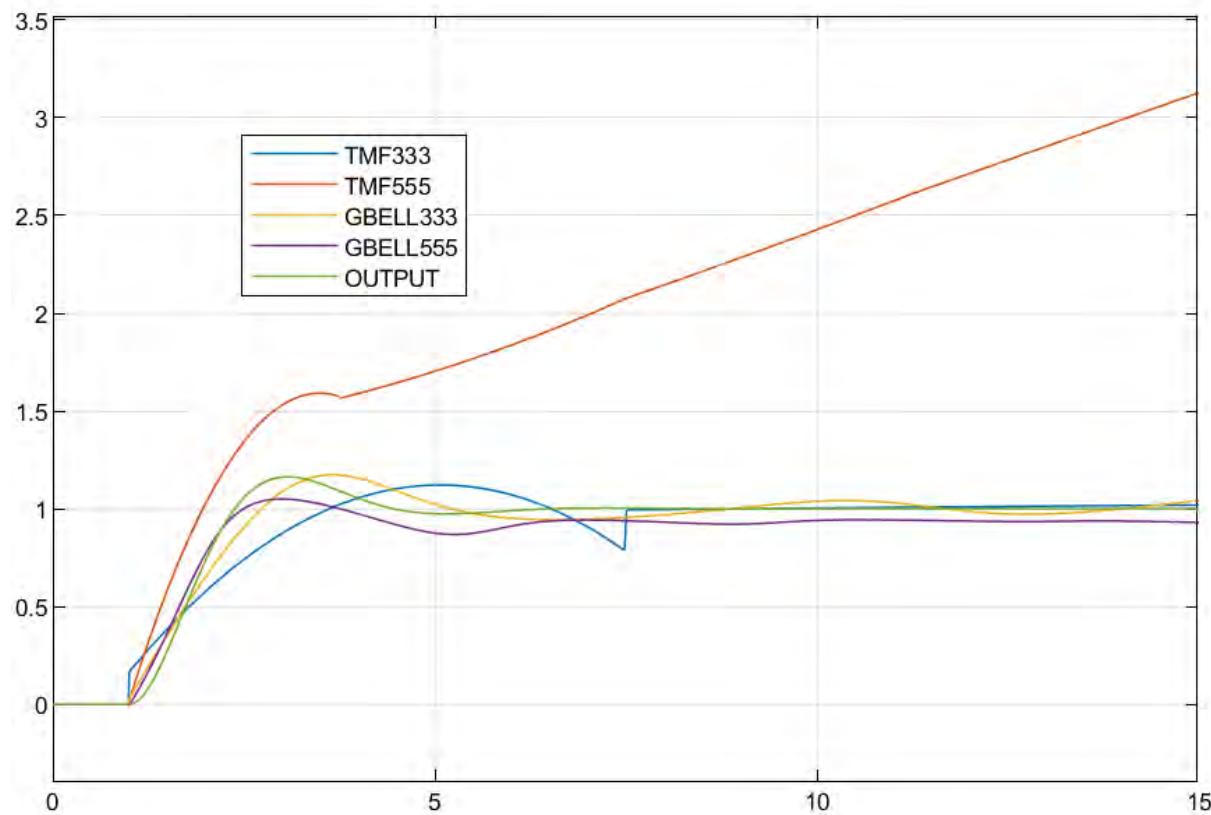
Omega 0.75



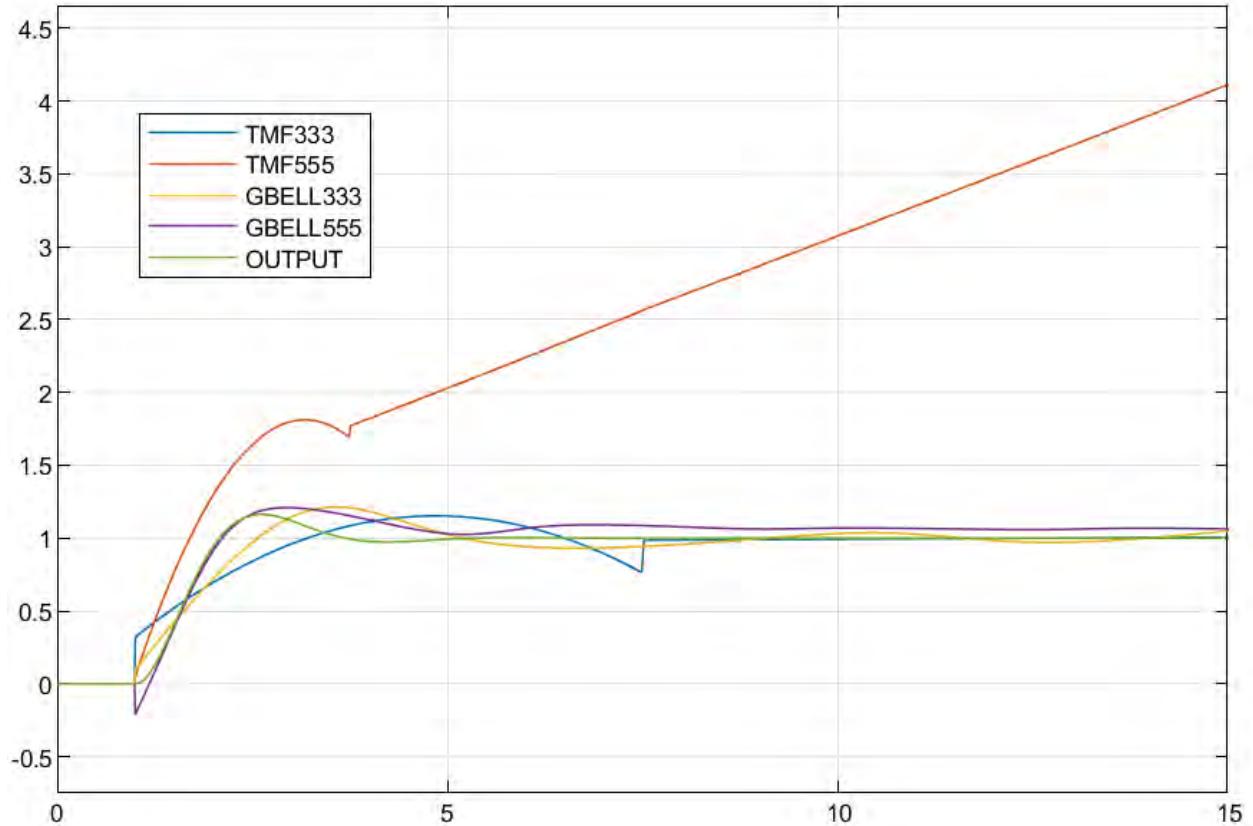
**Omega = 1.25**



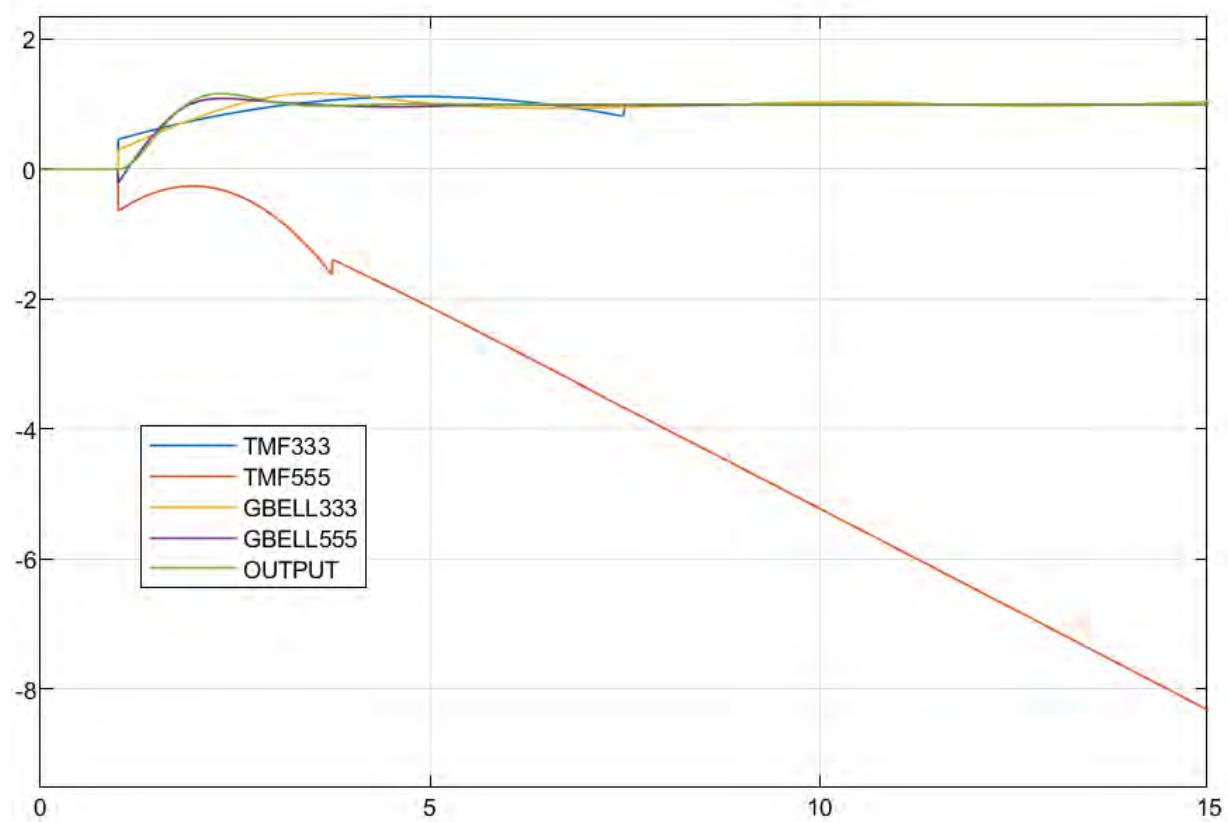
**Omega 1.75**



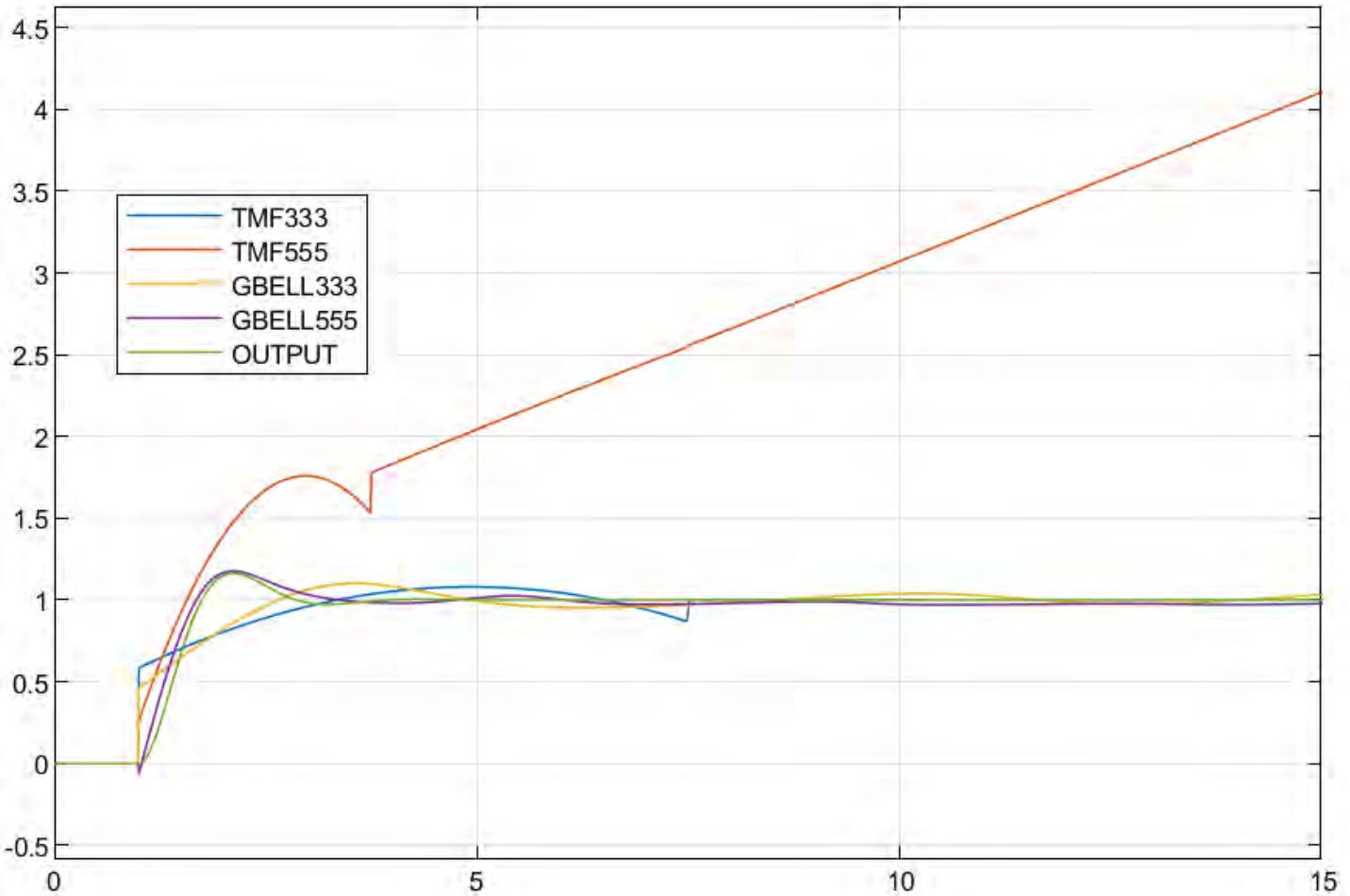
**Omega = 2.25**



**Omega = 2.75**



### Omega = 3.25

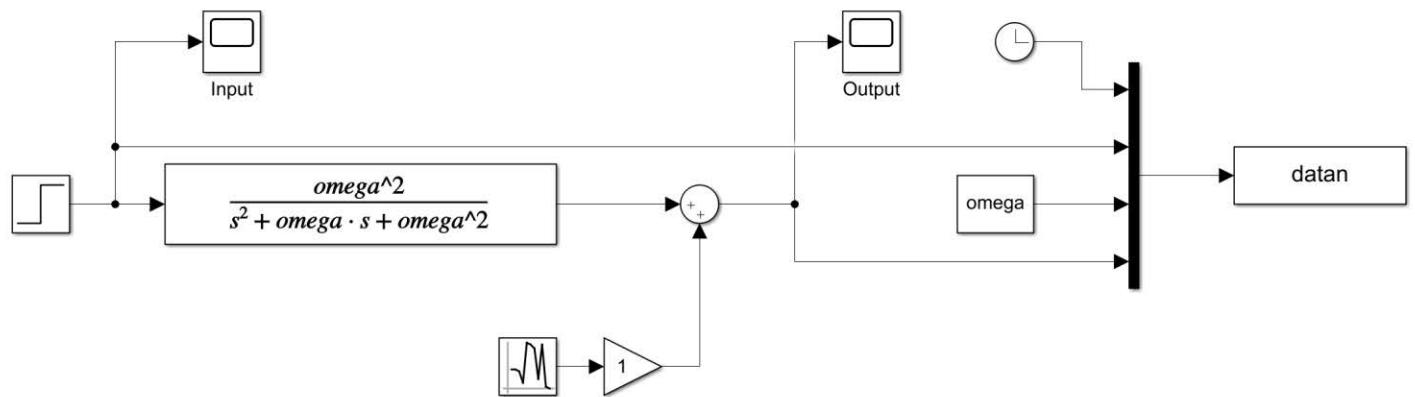
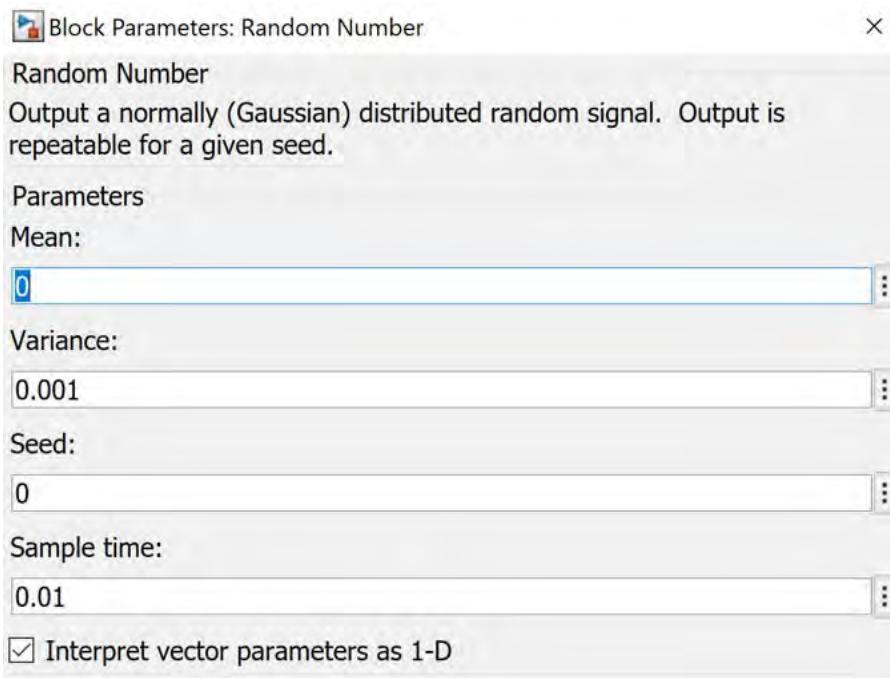


**Conclusion –** It can be observed that TMF555 is not able to provide output when values are interpolated. It might be because the triangular membership function tries to fit more information with 5 membership functions but can only use rigid triangles to do so. This leads to it overfitting incorrectly. **The best fit for all interpolated values in the transition state is provided by GBELL 555 FIS** but it often leaves a steady state error which may not be desirable. **In steady state, best fit is provided by TMF 333 FIS.** This may be because the function is simple and dependent on linear triangles so it only fits some data which is easy to interpolate.

- d. The **GBELL 555 FIS** is a good alternative if the steady state error can be removed by a different method while interpolating. If only operating on the values on which it has been trained, the **GBELL 555 and TMF 555** both do a good job of producing desired outputs. If the system stabilizes quickly and its performance during transition state can be neglected then **TMF 555** is the best choice. Overall, depending on the acceptable errors and desired output, both GBELL 555 and TMF 555 can work for this system.

## Problem – 2:

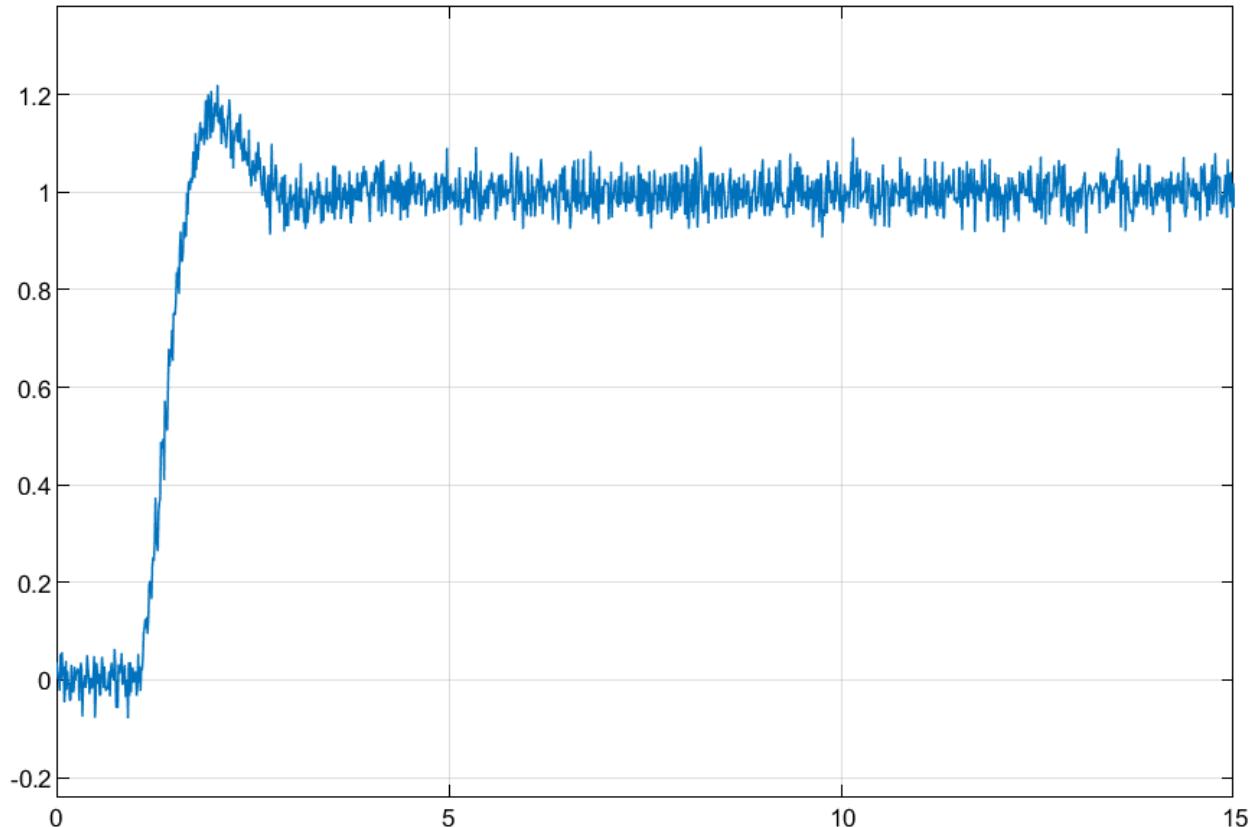
### a. Fitting



Simulink Diagram for generating noisy data

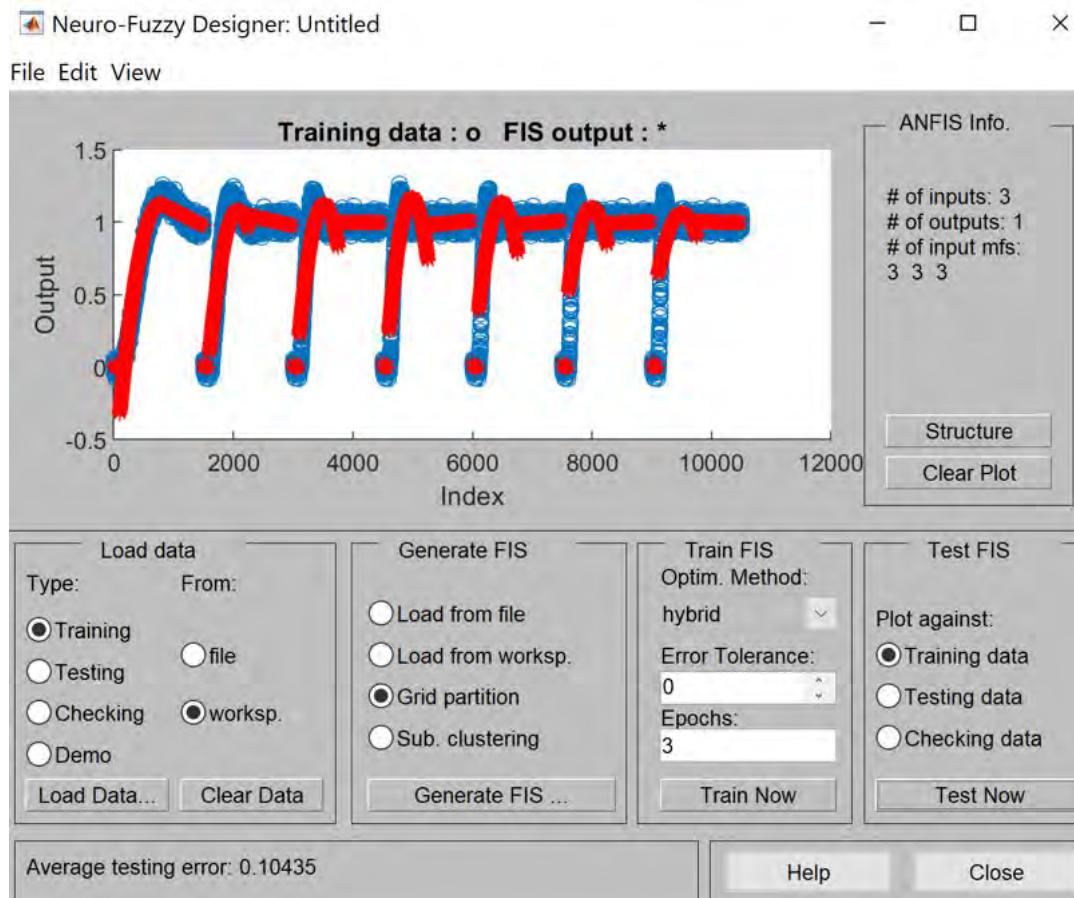
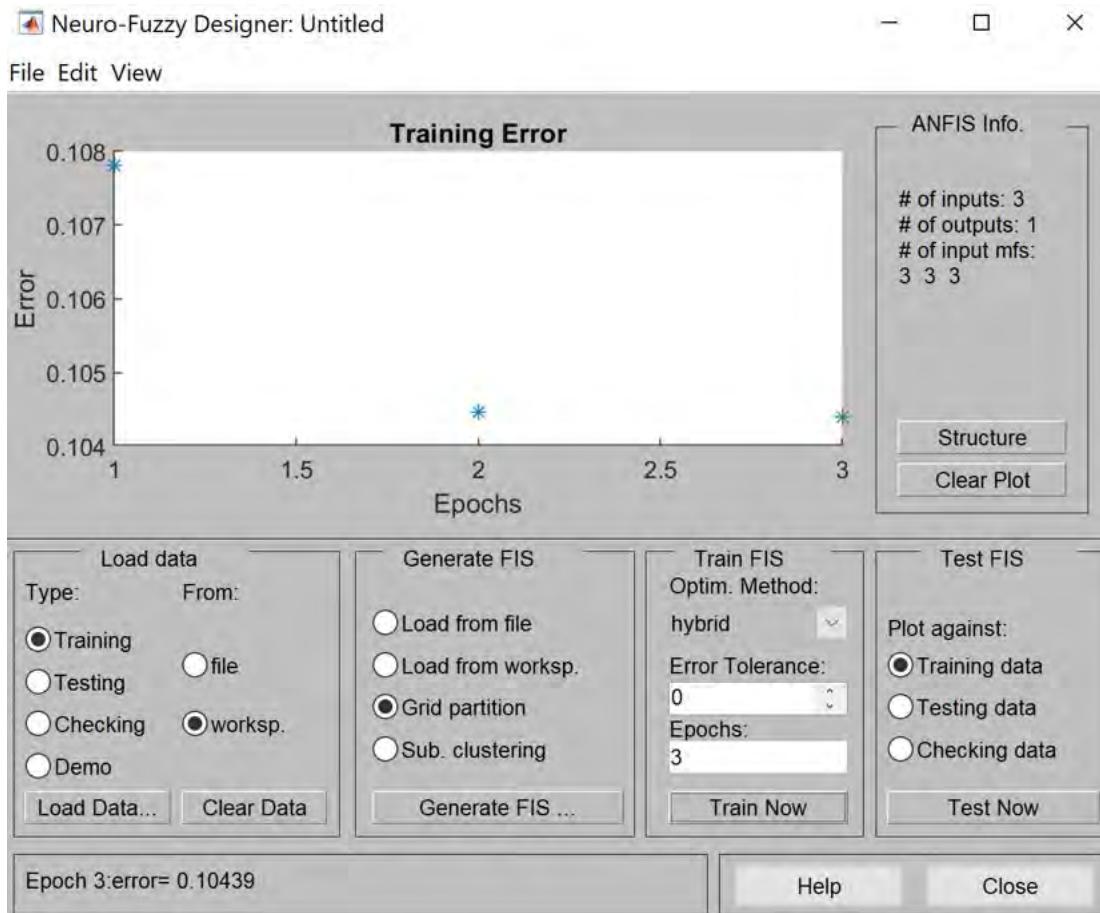
### MATLAB Commands to generate the training data

```
omega=0.5 %run the Simulink diagram each time after defining the omega value  
fc_trdatan=datan  
omega=1  
fc_trdatan=[fc_trdatan;datan]  
omega=1.5  
fc_trdatan=[fc_trdatan;datan]  
omega=2.0  
fc_trdatan=[fc_trdatan;datan]  
omega=2.5  
fc_trdatan=[fc_trdatan;datan]  
omega=3.0  
fc_trdatan=[fc_trdatan;datan]  
omega=3.5  
fc_trdatan=[fc_trdatan;datan]
```

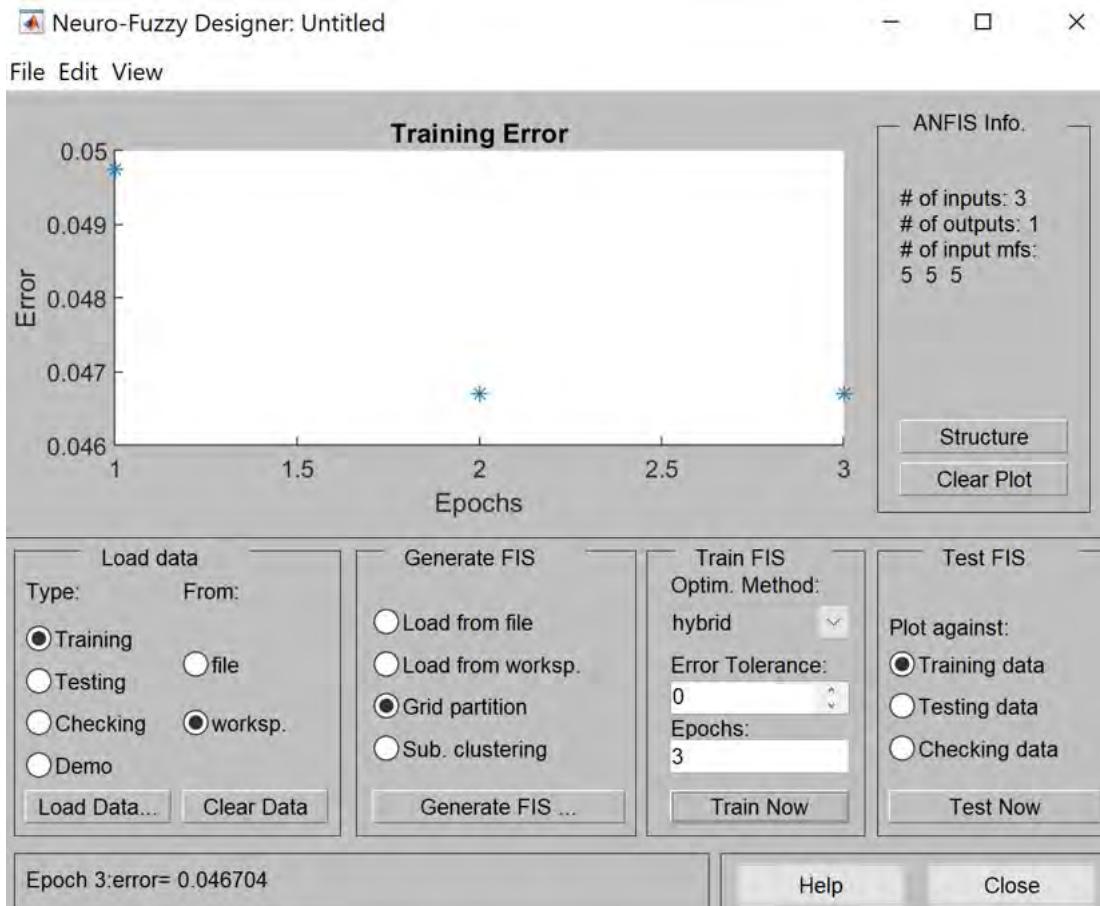


System output with noise for omega = 3.5

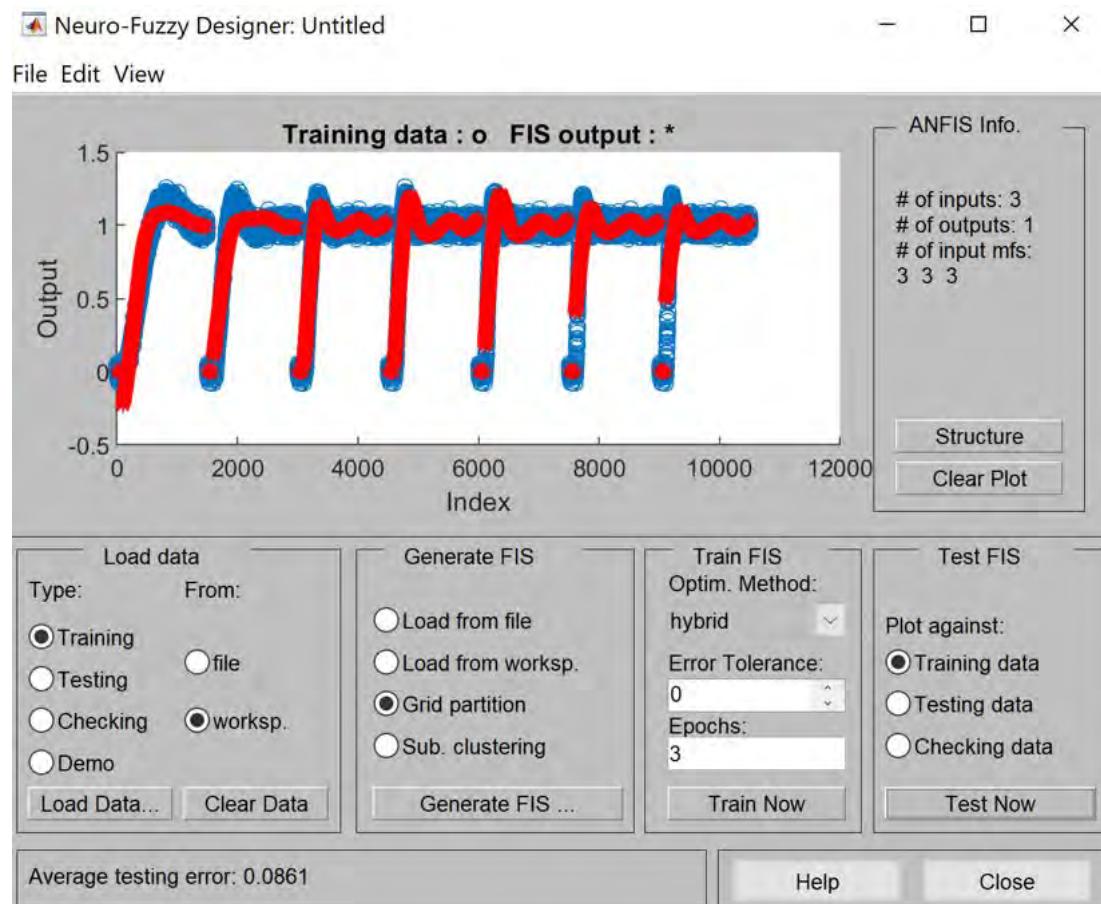
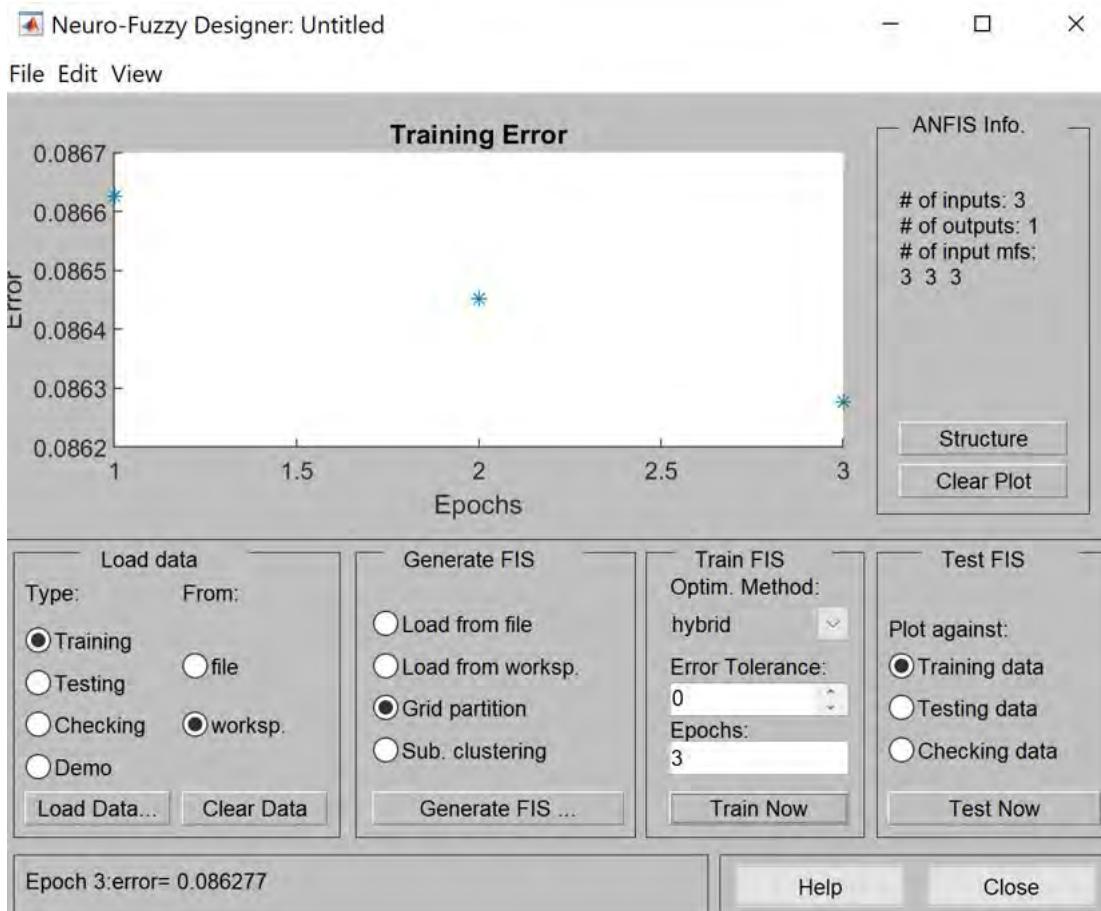
# TMF 333



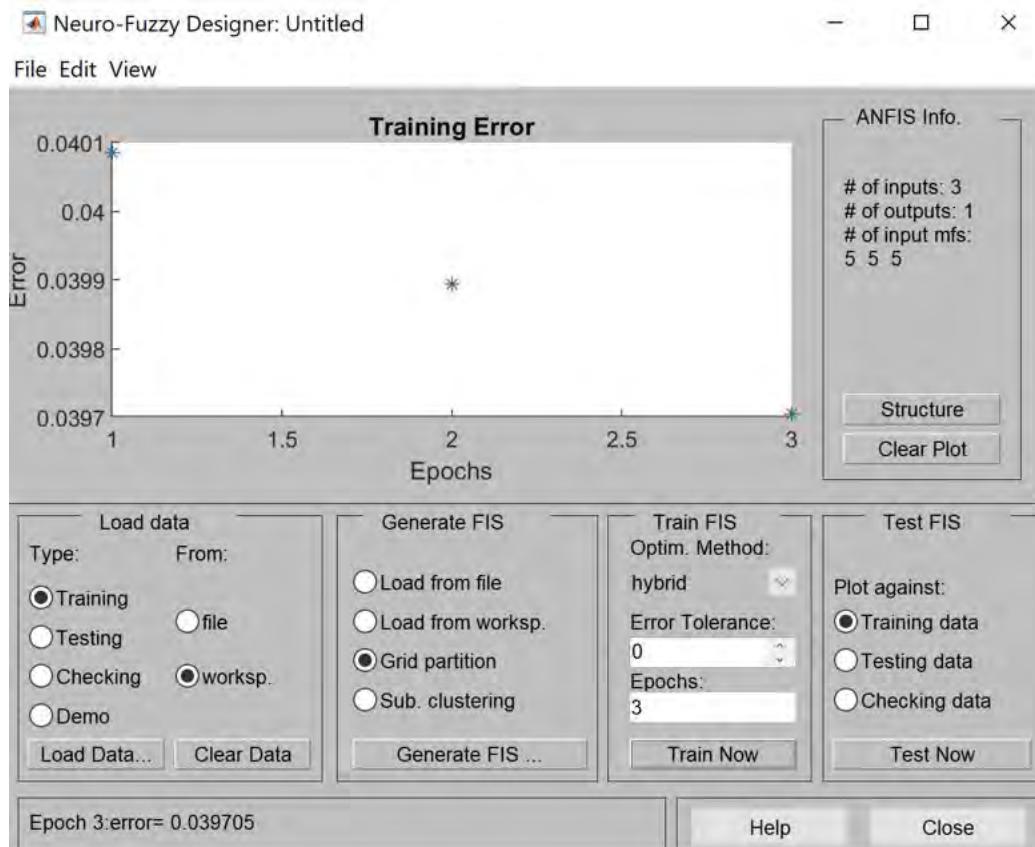
# TMF 555



# GBELL 333

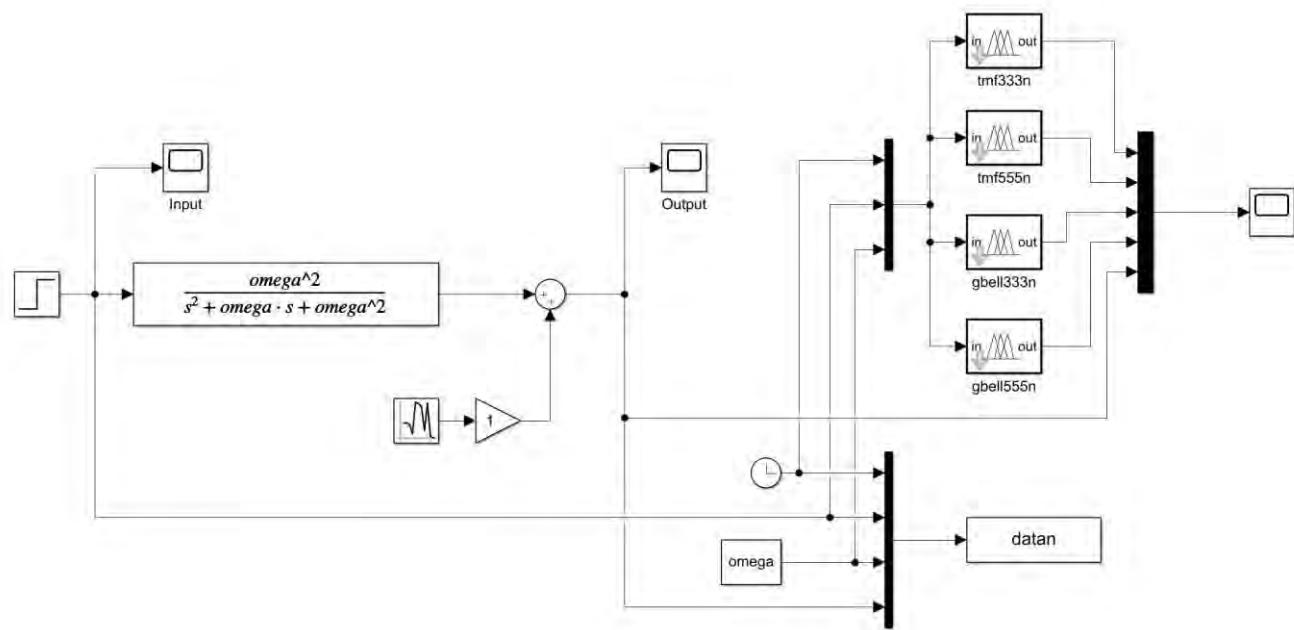


## GBELL 555



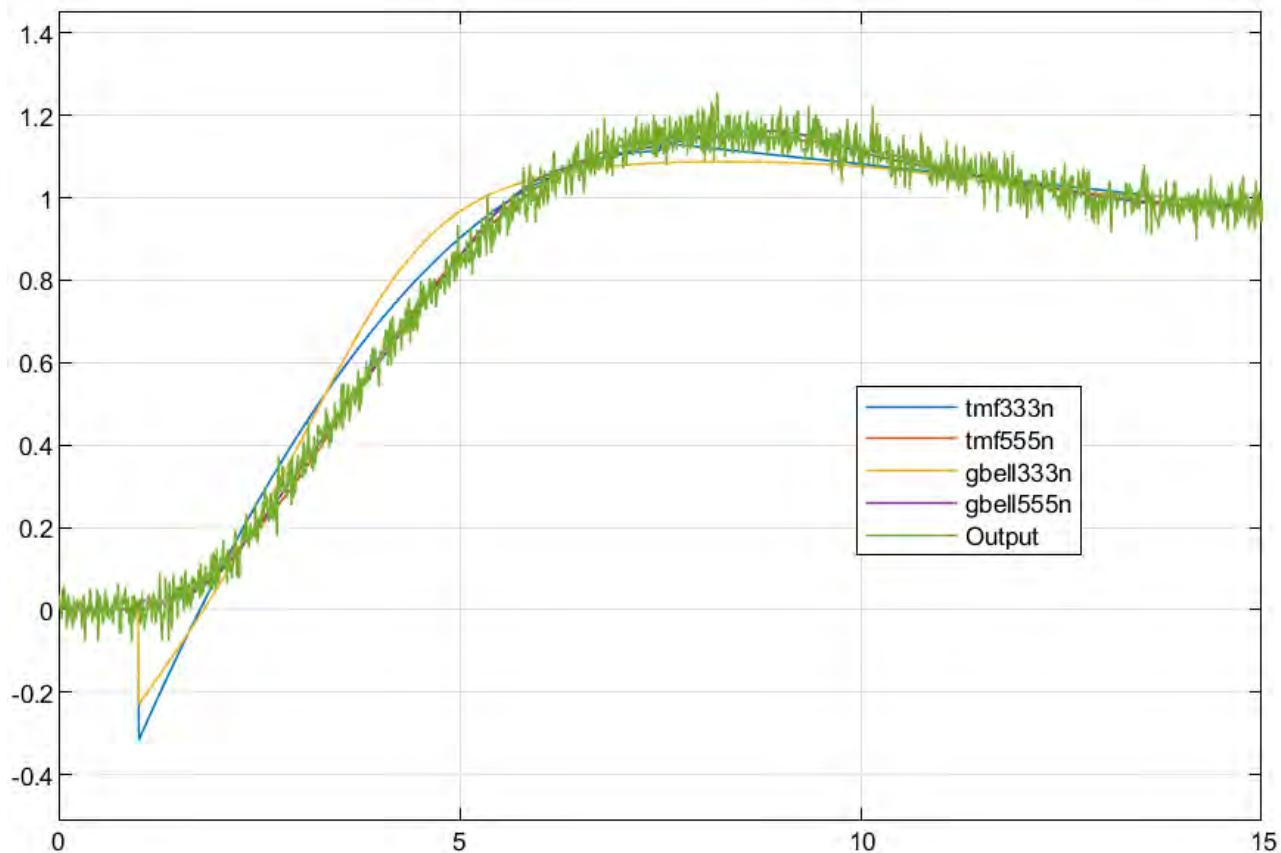
**Conclusion – Using average testing error values, GBELL 555 produces the best fit to training data followed by TMF 555**

## b. Comparison

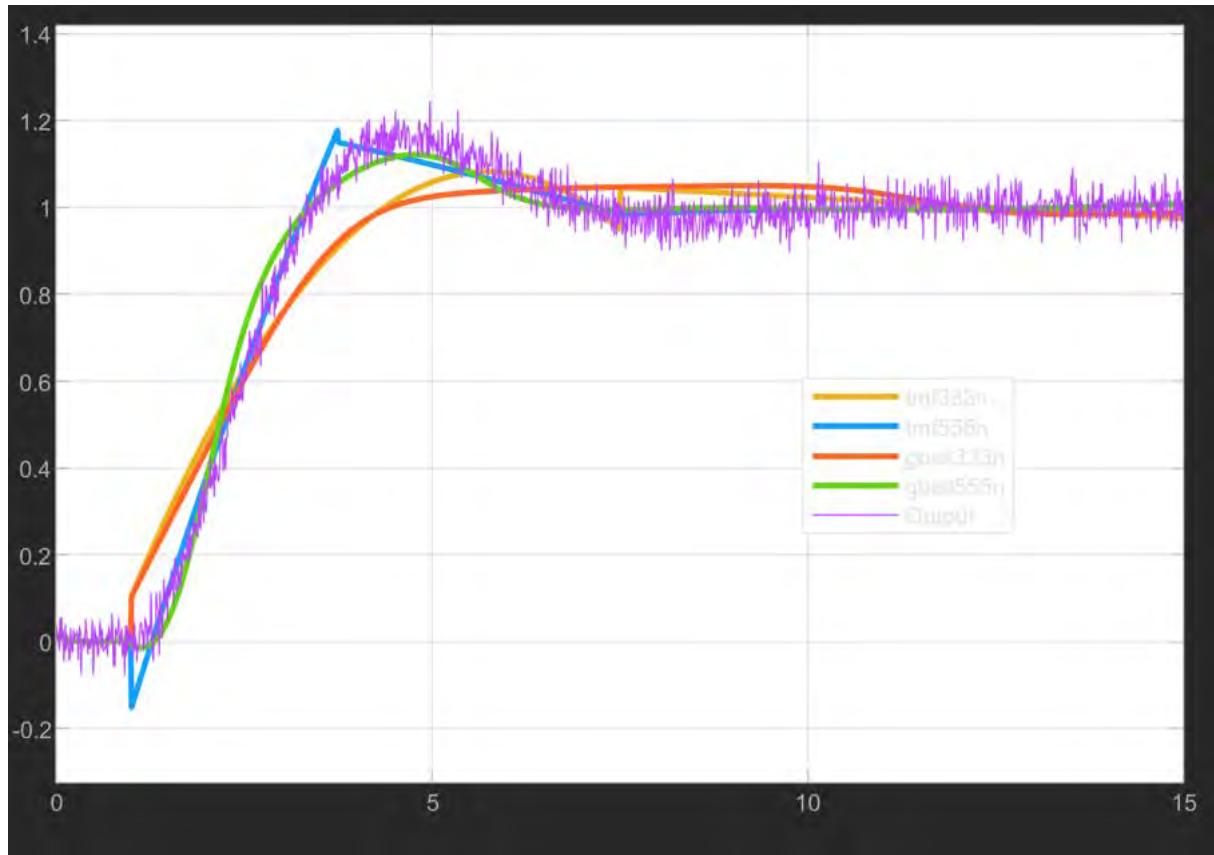


Simulink Diagram for comparing the outputs

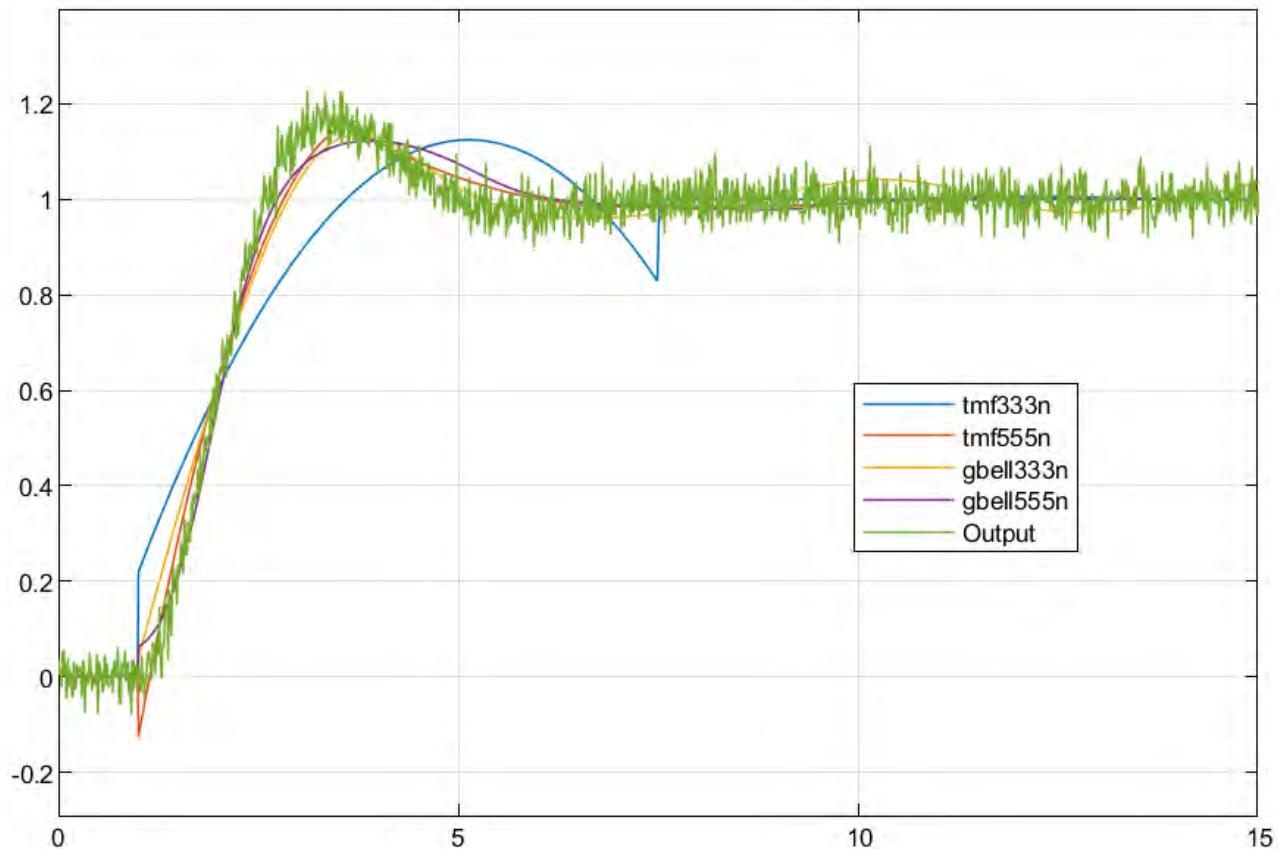
$\Omega = 0.5$



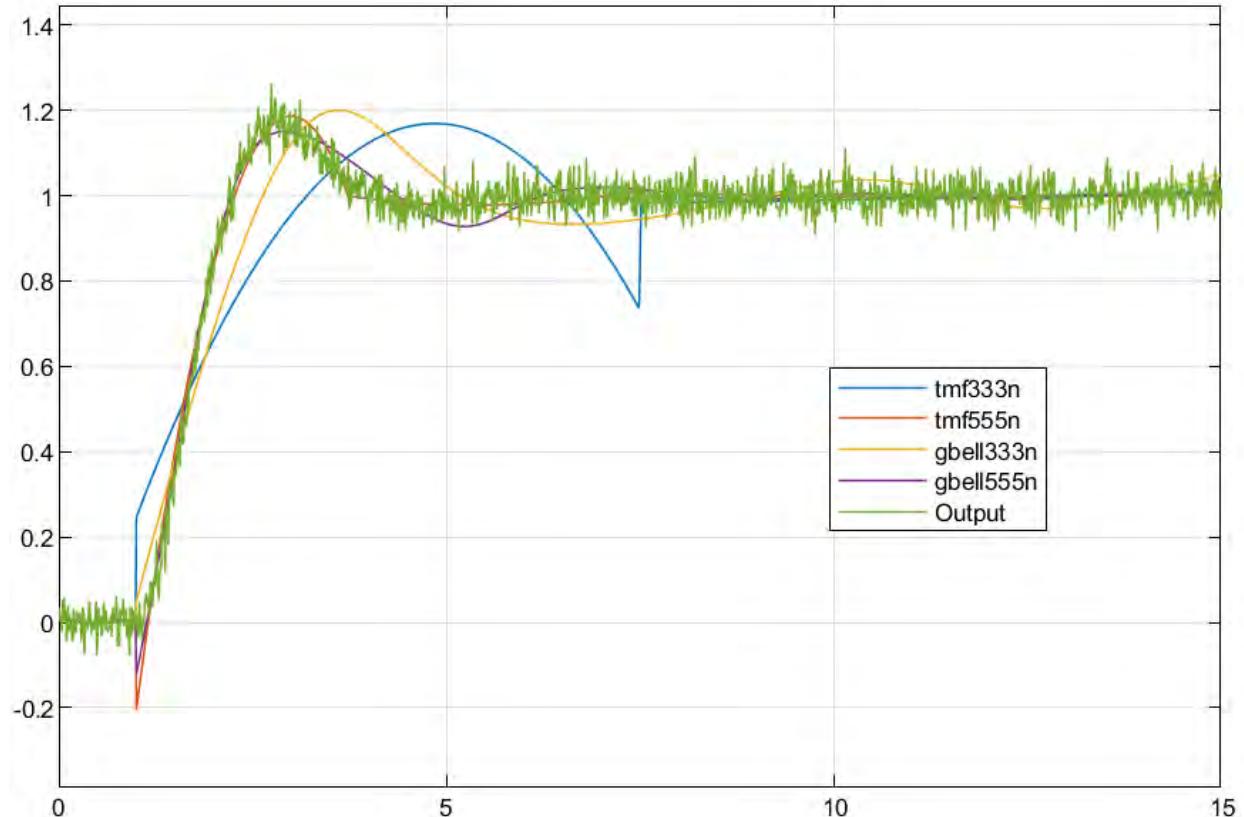
**Omega = 1**



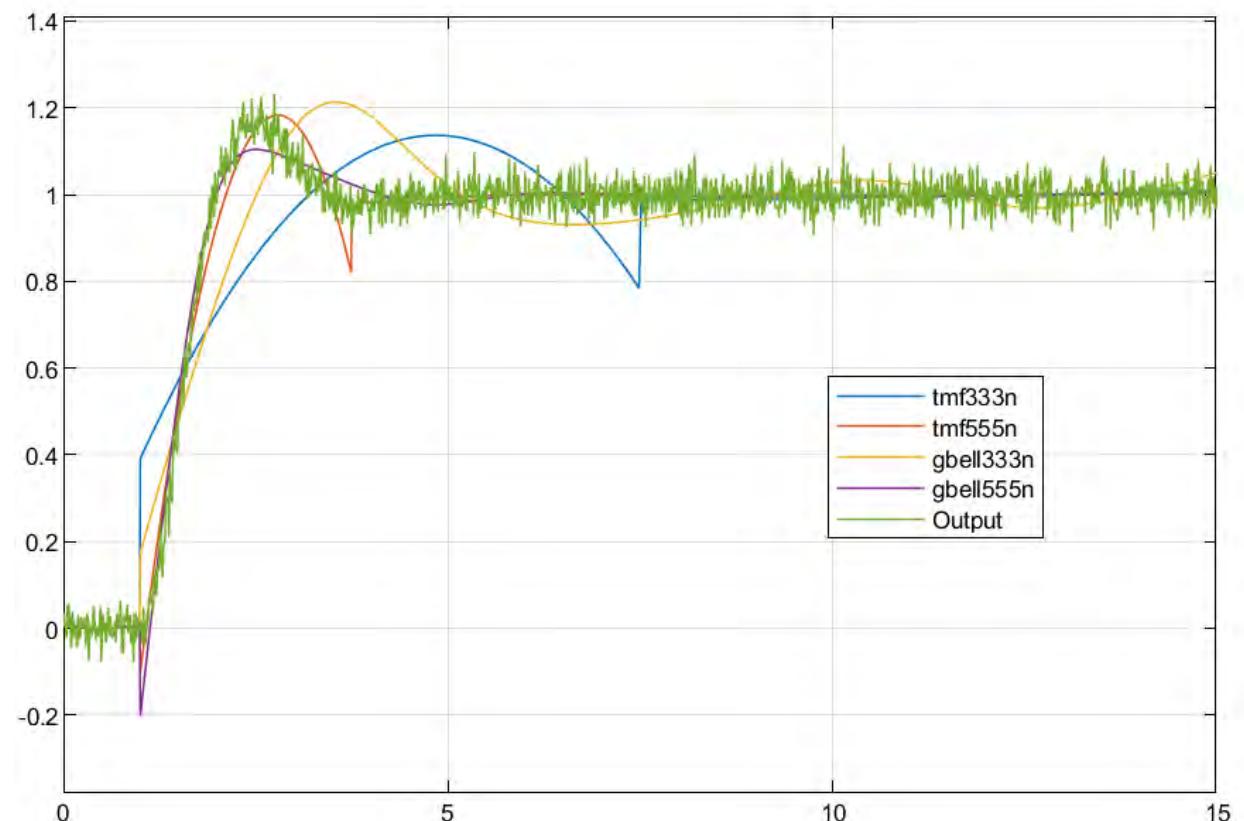
**Omega = 1.5**



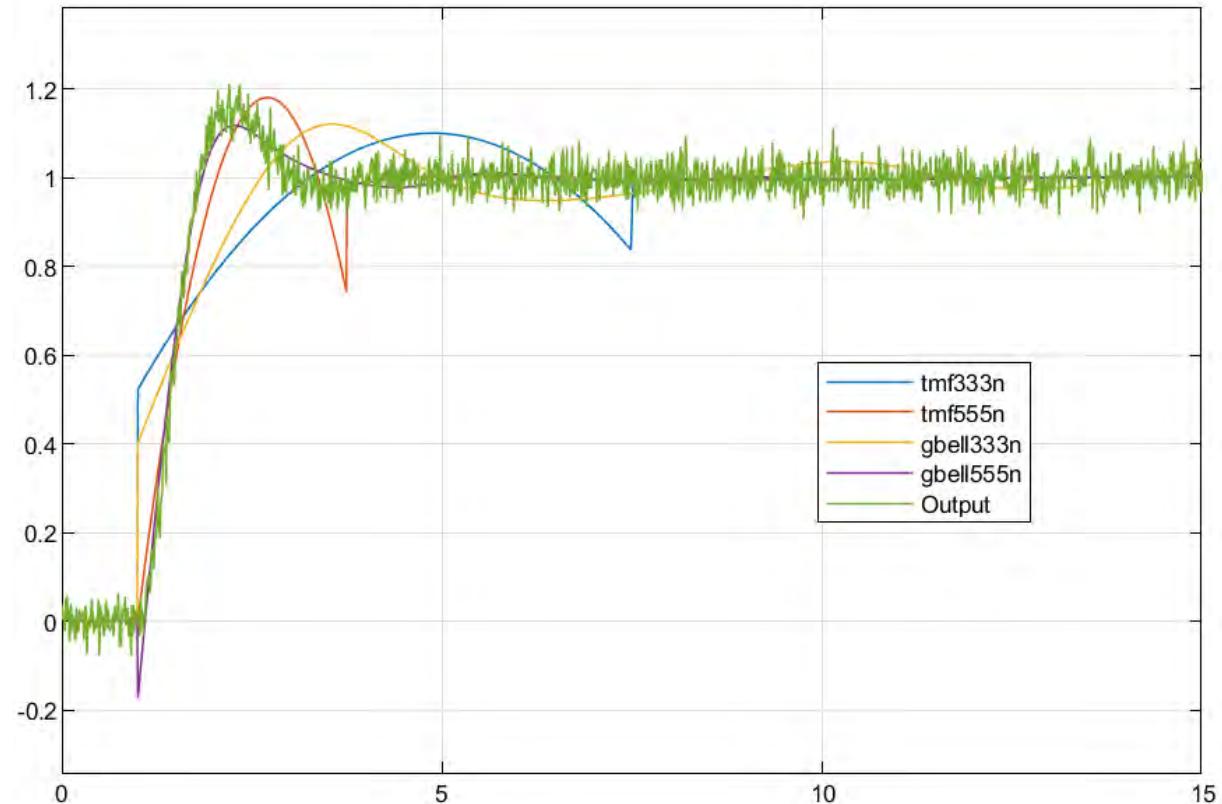
**Omega = 2.0**



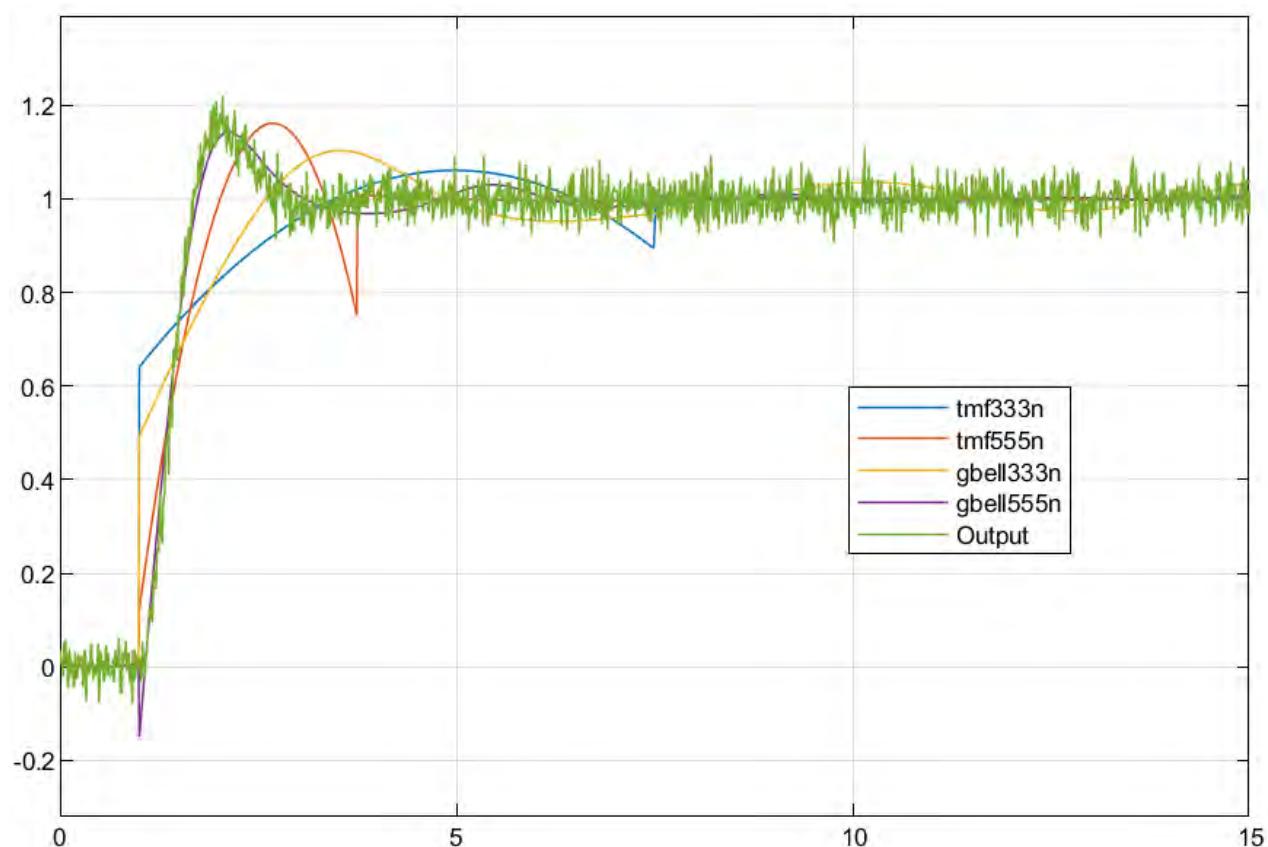
**Omega = 2.5**



**Omega = 3.0**

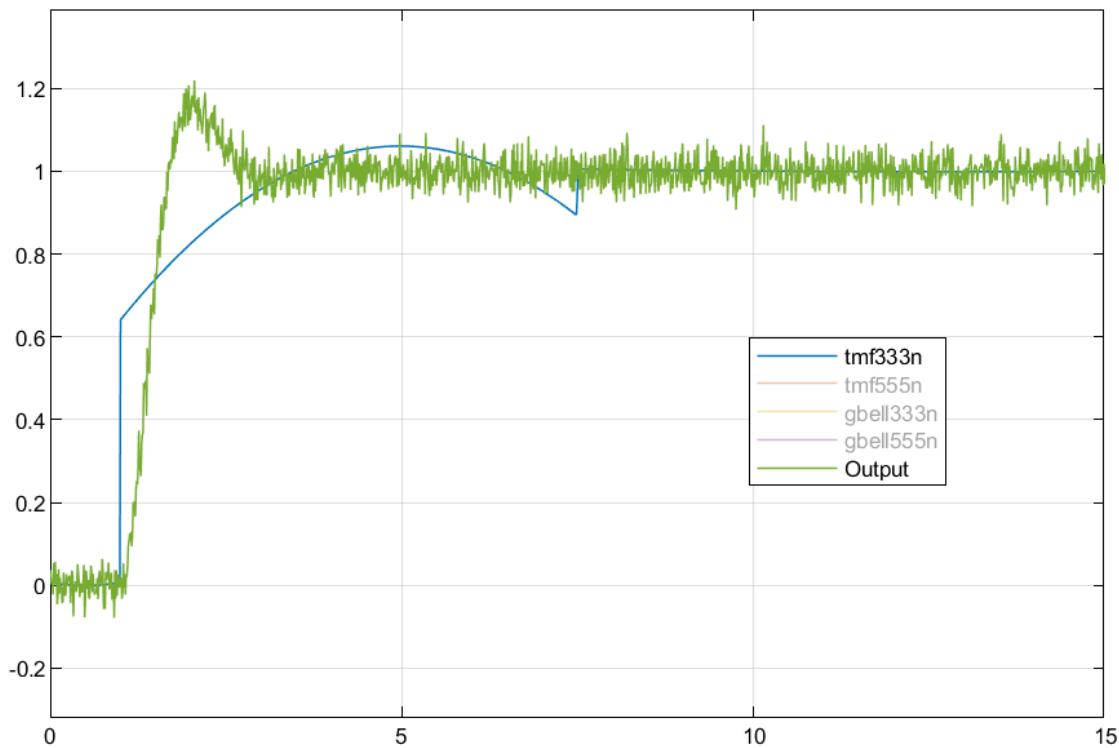


**Omega = 3.5**



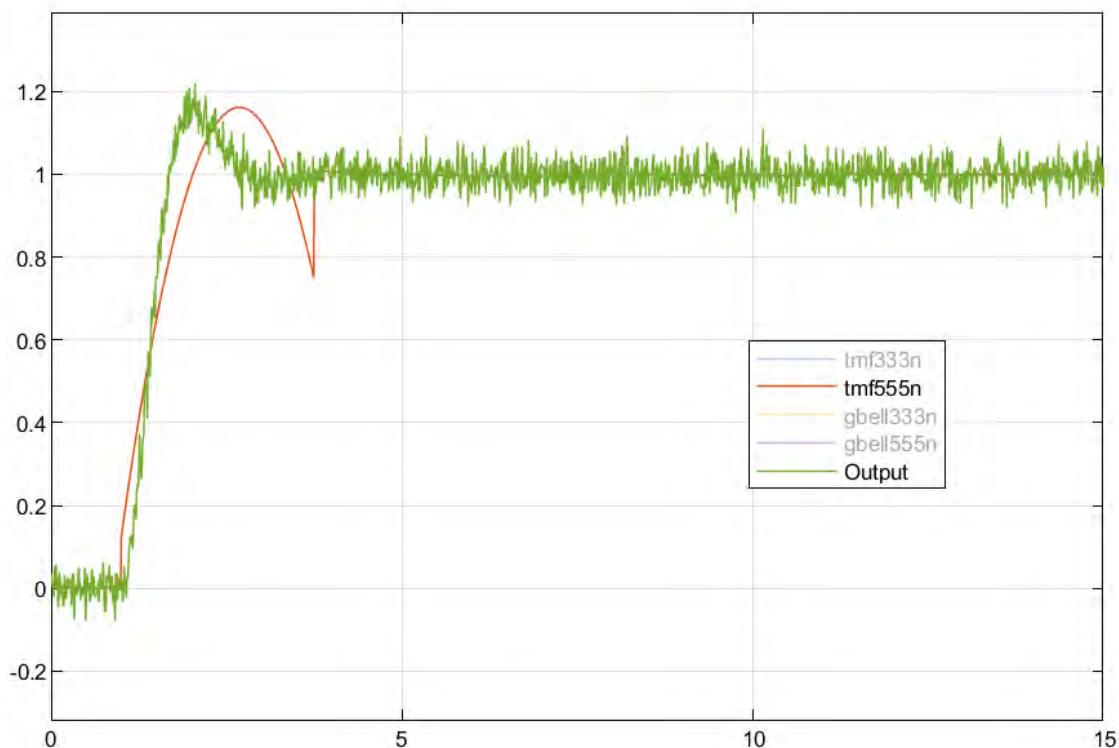
## One by One Comparison

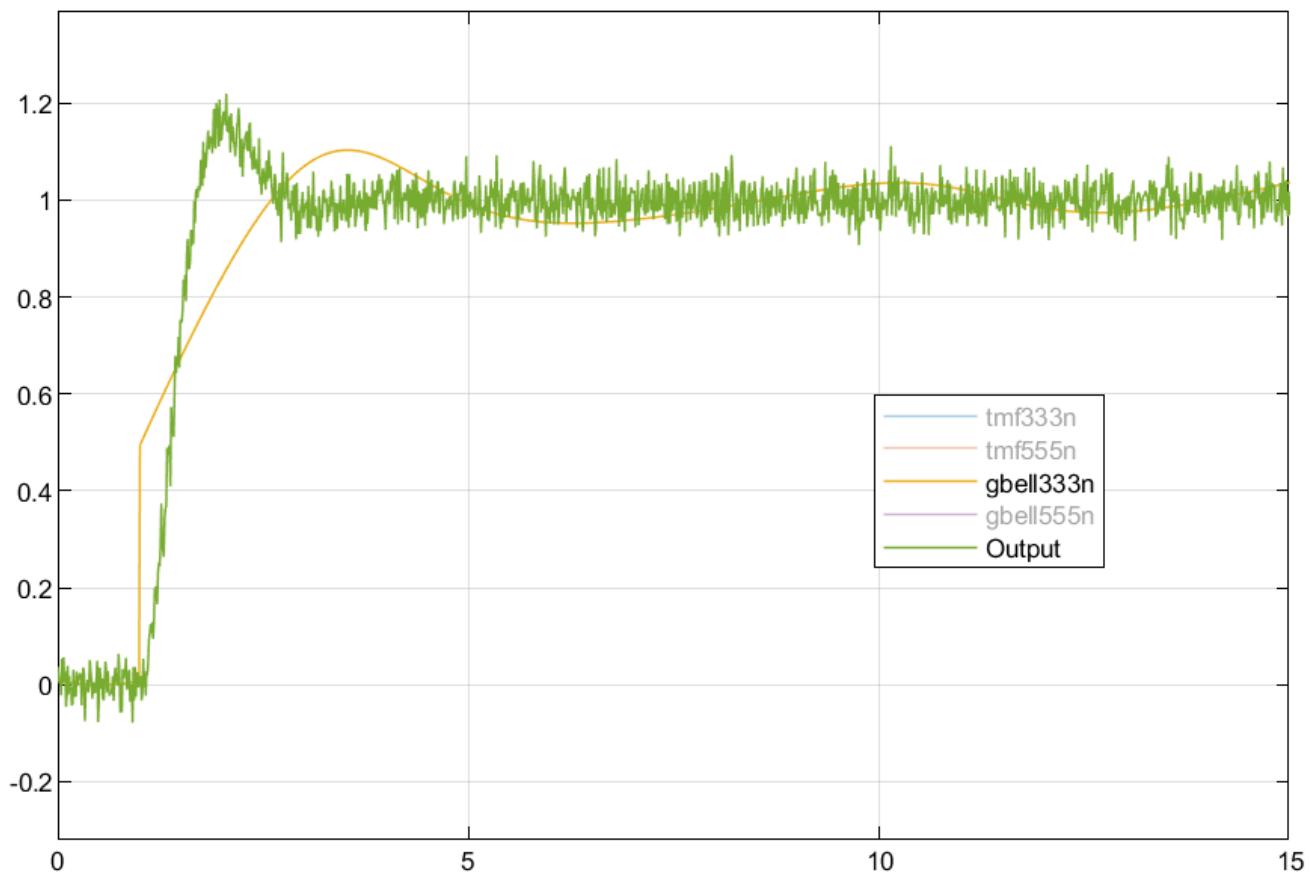
It is observed that the maximum divergence from output values occurs at  $\omega = 3.5$ . Thus, instead of comparing 28 different graphs to determine the best fit, only 4 graphs at  $\omega = 3.5$  are compared. This is also done for part b in the previous problem.



Plot above, TMF 333

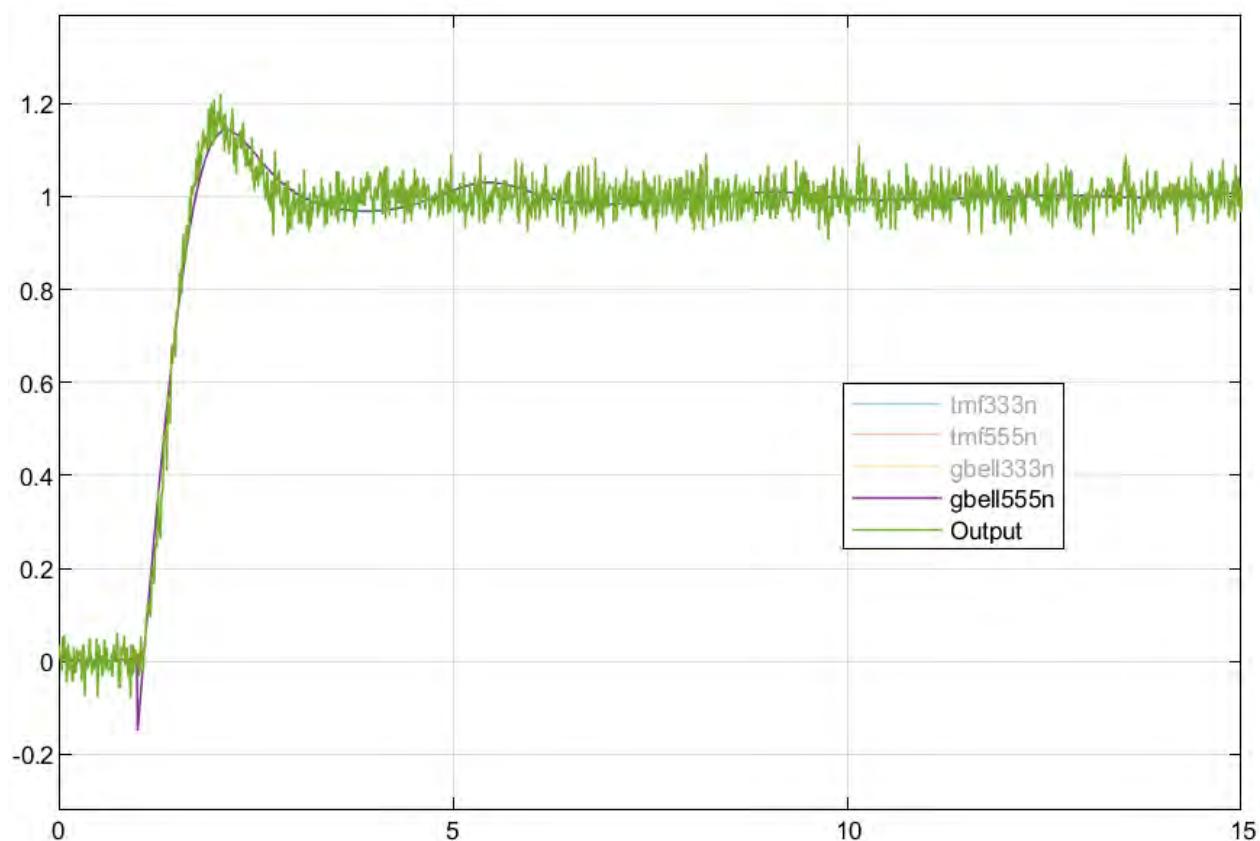
Plot below – TMF 555





Plot above – GBELL 333

Plot below – GBELL 555



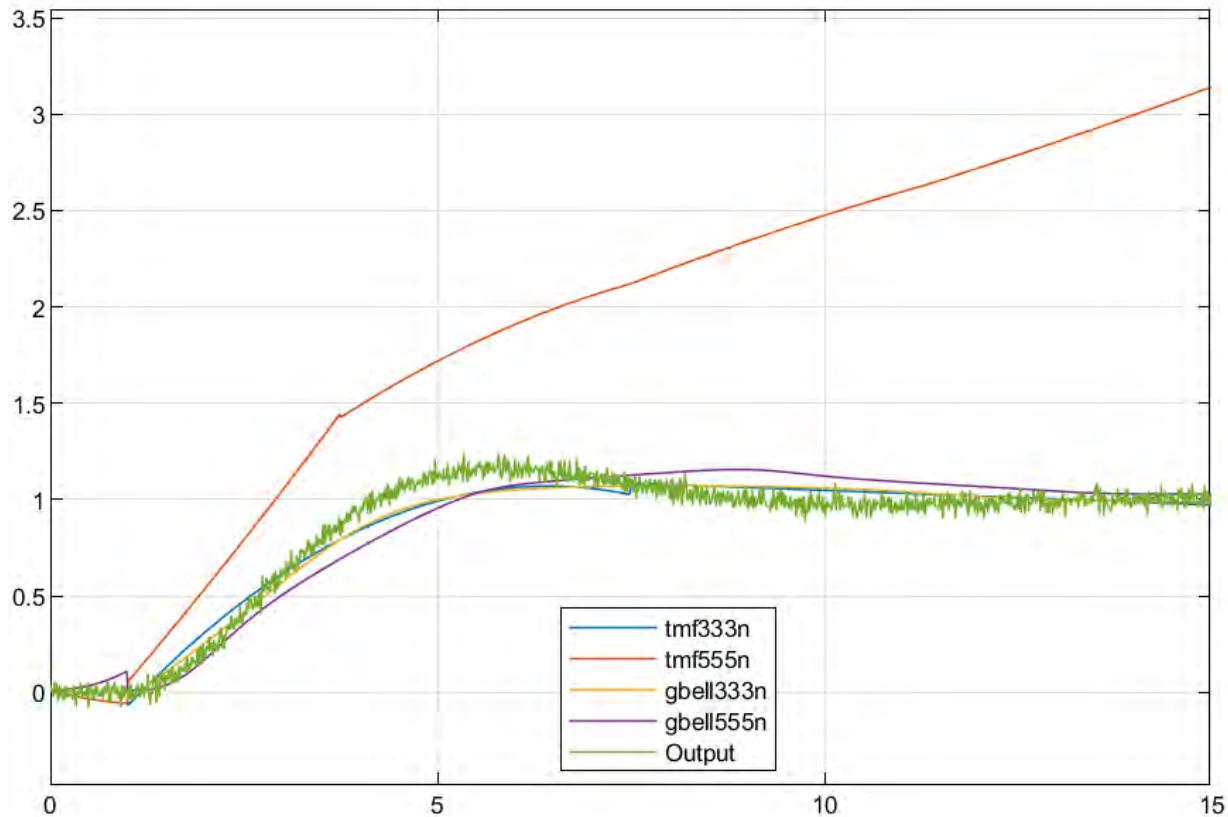
**Conclusion** – From the above plots it is observed that all fuzzy systems provide less accurate outputs in the presence of noise. As expected from the fit data in part a, **the GBELL 555 produces the best fits at all omega values**. Although the fits are close at omega =0.5, they quickly diverge from the actual output as omega values are increases. **It is seen that FIS with more membership functions is better at estimating the output.**

The **triangular membership function has sharp edges** due to the straight triangular edges used for defining the output. This is not seen in gaussian membership functions as they have smooth curves. As it can be seen, for omega=0.5 almost all fuzzy identification systems work well with the best ones being TMF555 and GBELL555 due to the higher number of functions.

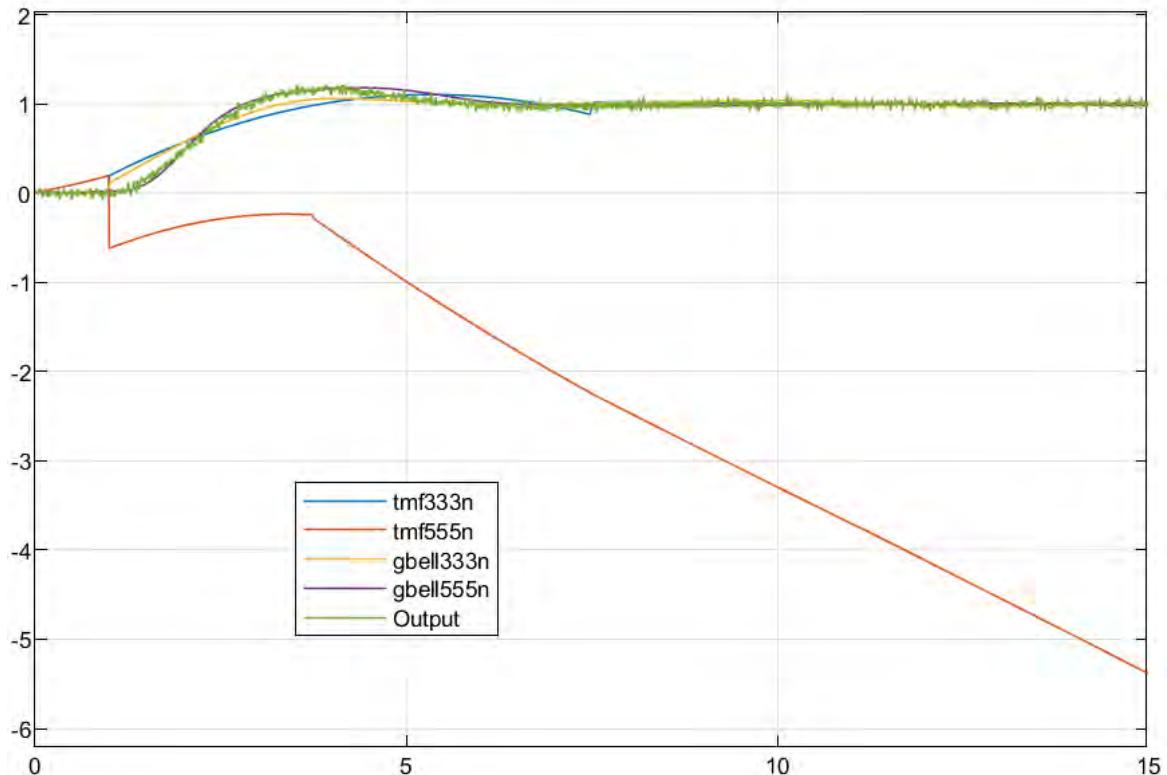
Other than that, it is also noted that while the GBELL 555 FIS produces the best fit, it gives a sharp dip before the input at higher omega values.

### c. Interpolating omega values

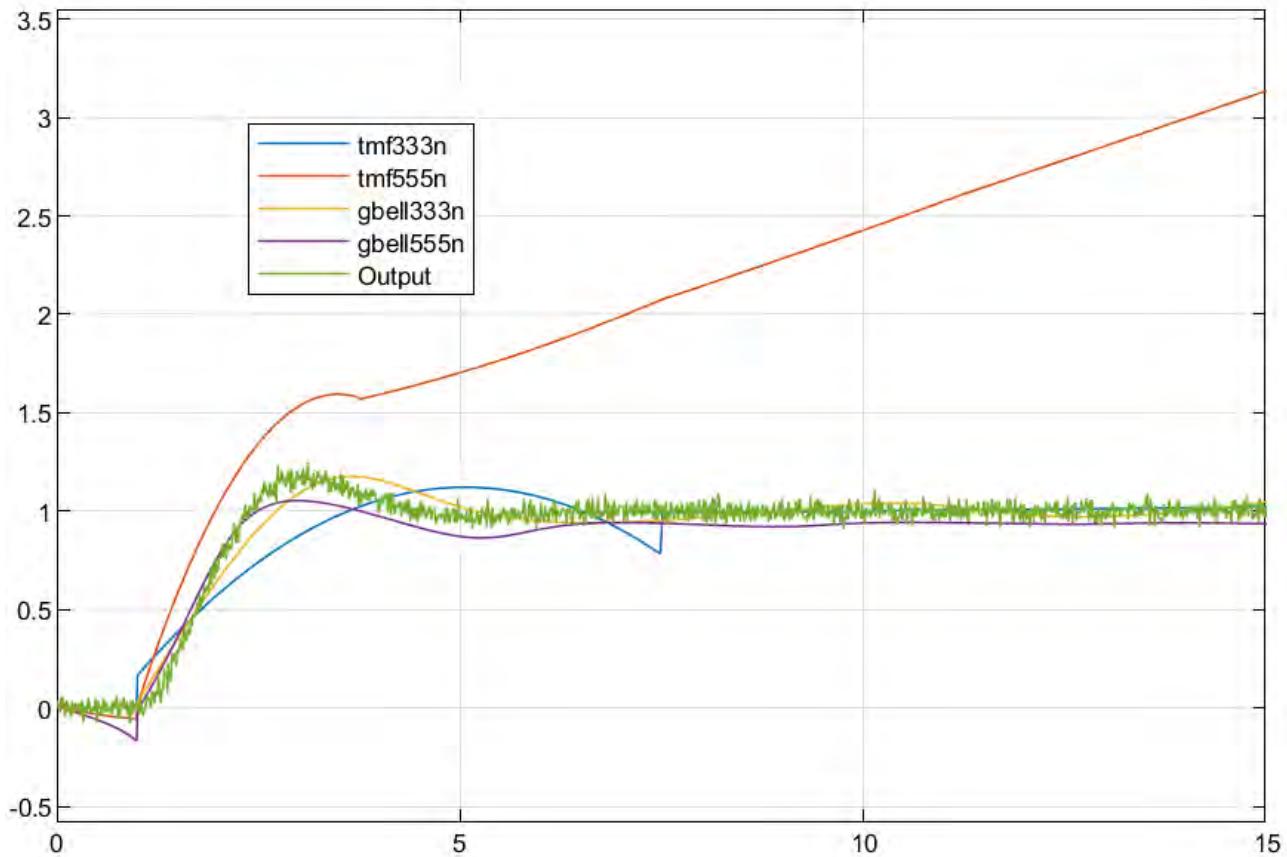
Omega = 0.75



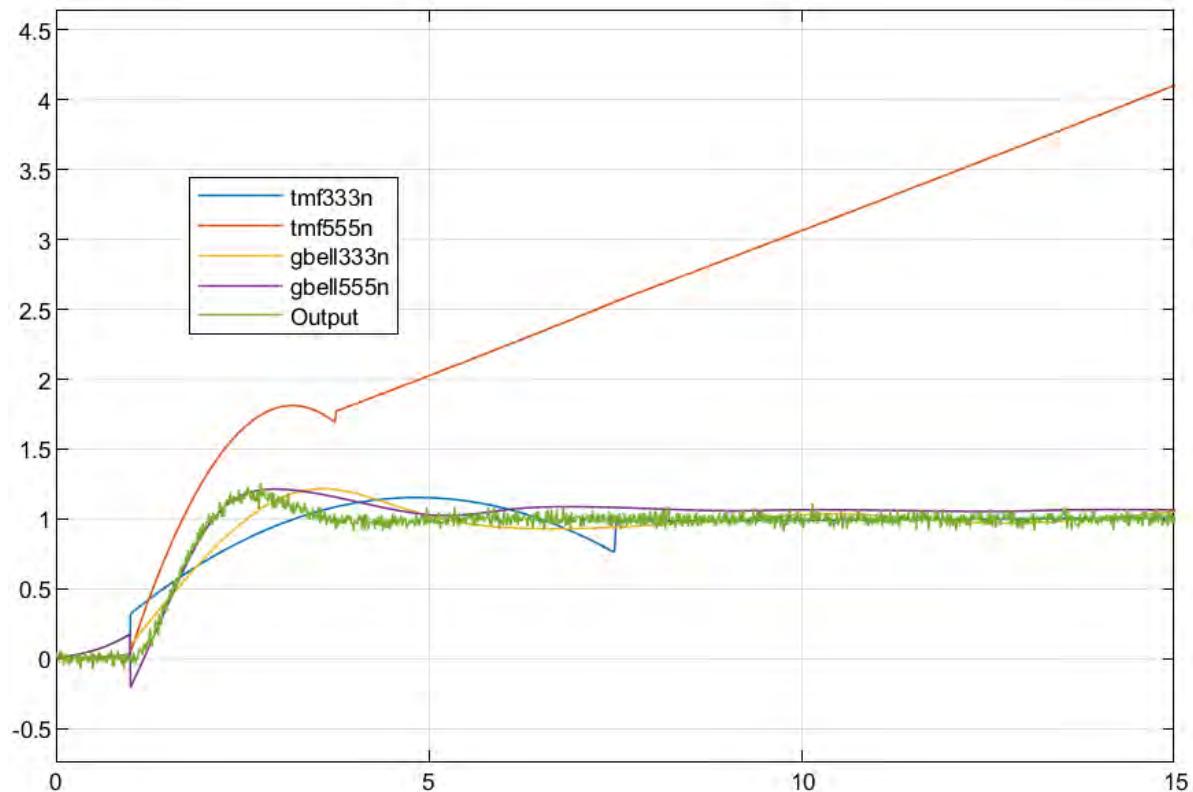
**Omega = 1.25**



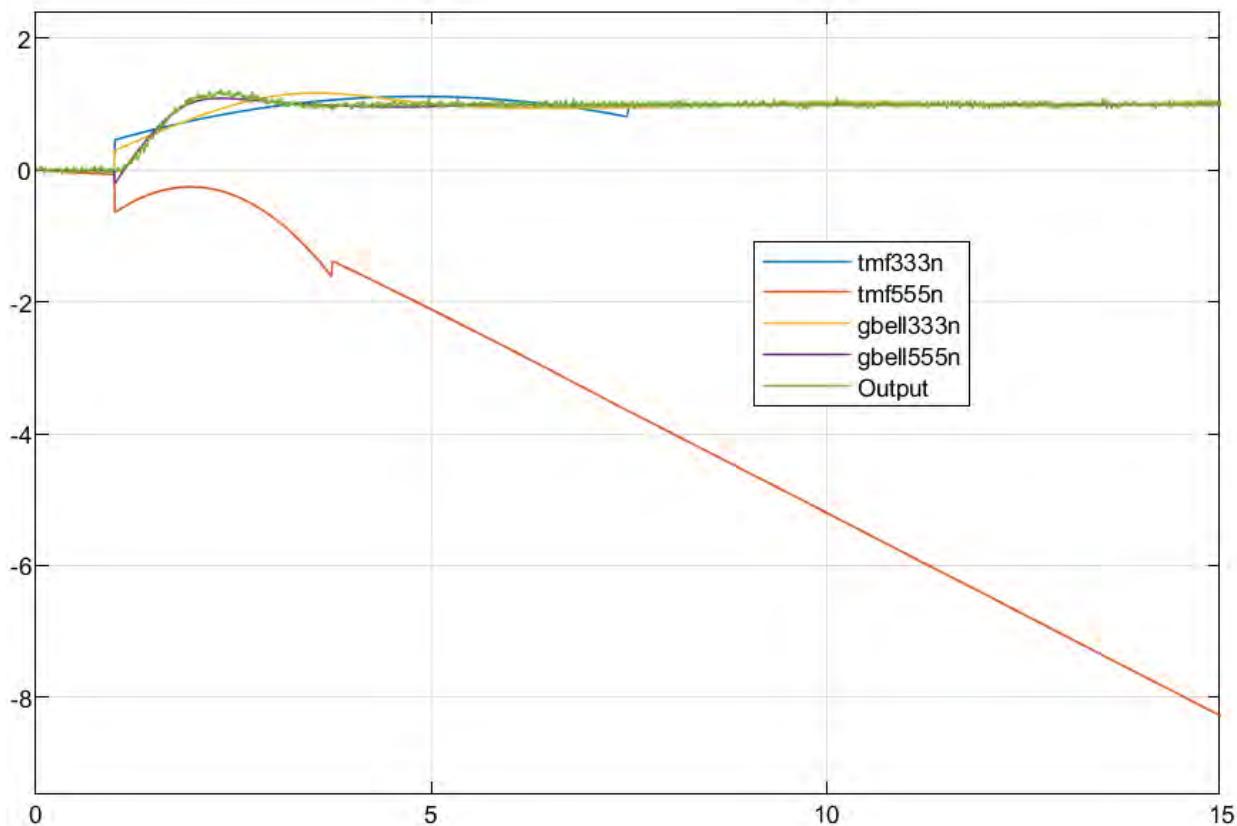
**Omega = 1.75**



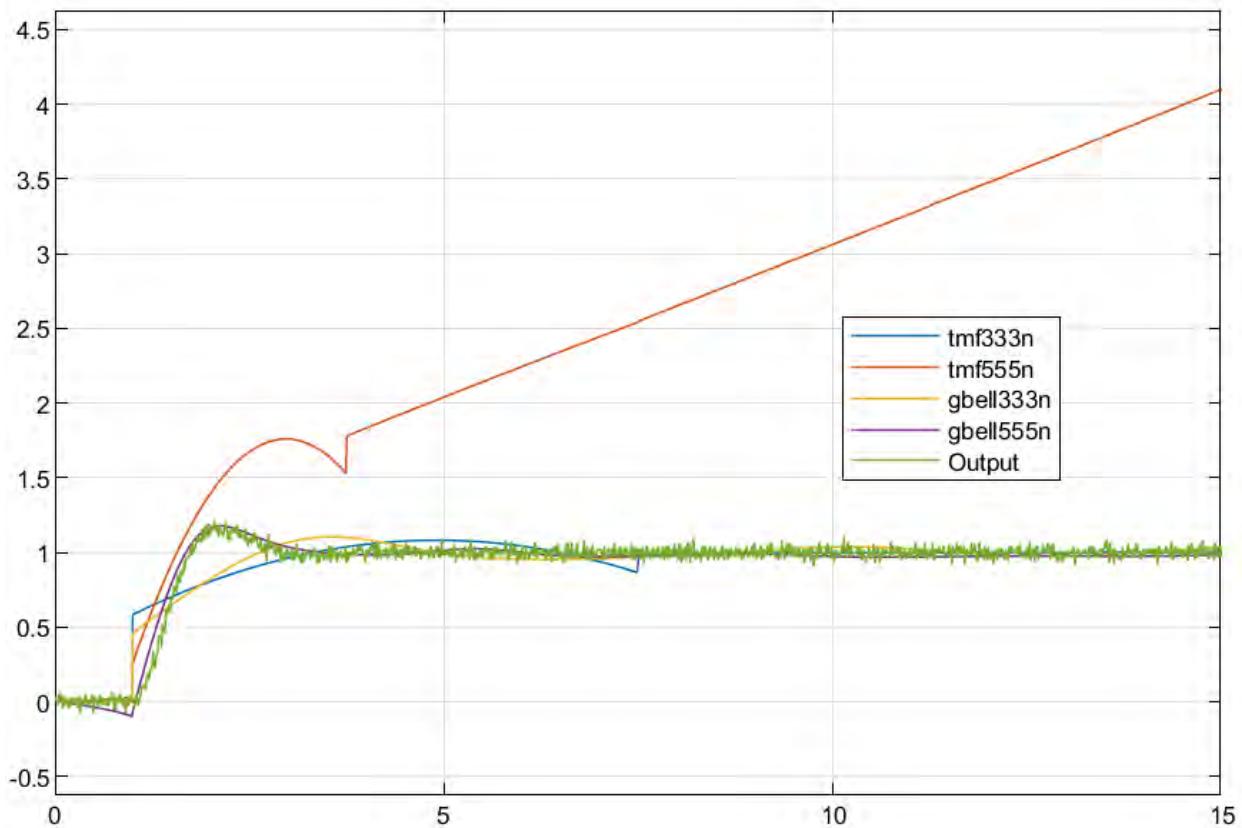
**Omega = 2.25**



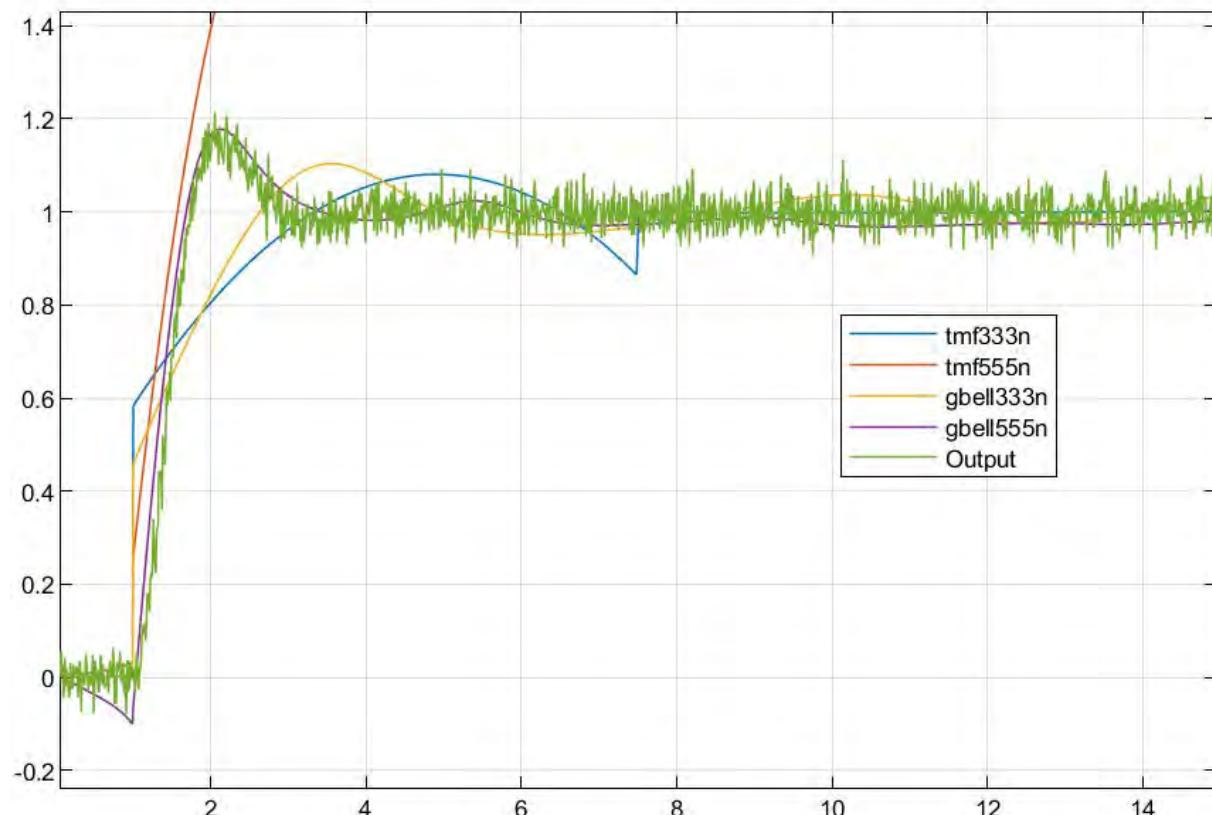
**Omega = 2.75**



**Omega = 3.25**



Improving resolution by ignoring tmf555 data,



**Conclusion** – None of the fuzzy systems are good at interpolation except for the Gaussian Bell with 5 membership functions. The TMF 555 behaves in a strange manner probably due to overfitting data through more information using 5 triangular membership functions which results in incorrect outputs when interpolating.

**The best fit for all interpolated values in the transition state is provided by GBELL 555 FIS** but it often leaves a steady state error which may not be desirable. **In steady state, best fit is provided by TMF 333 FIS.** This may be because the function is simple and dependent on linear triangles so it only fits some data which is easy to interpolate.

- d. The **GBELL 555 FIS** is a good alternative if the steady state error can be removed by a different method when omega values are interpolated. If the system stabilizes quickly and its performance during transition state can be neglected then **TMF 333** is the best choice. Overall, depending on the acceptable errors and desired output, both GBELL 555 and TMF 333 can work for this system in the presence of noise. **Overall, the presence of noise reduces the ability of the fuzzy system to model this data.** For comparison, the average testing error value for the best FIS i.e., GBELL 555 increases by 70 percent when fitting data in the presence of noise.

## EE5327, Fall 2020

### Project 2: Kalman Filter Bank for Failure Detection

Due December 15, 2020 (12 noon, not midnight)

**Please turn in this paper with your exam:**

Pledge of honor: "On my honor I have neither given nor received aid on this project except from the TA or class instructor."

Signature: \_\_\_\_\_

A process in a factory can be modeled by a simple second order ODE as follows:

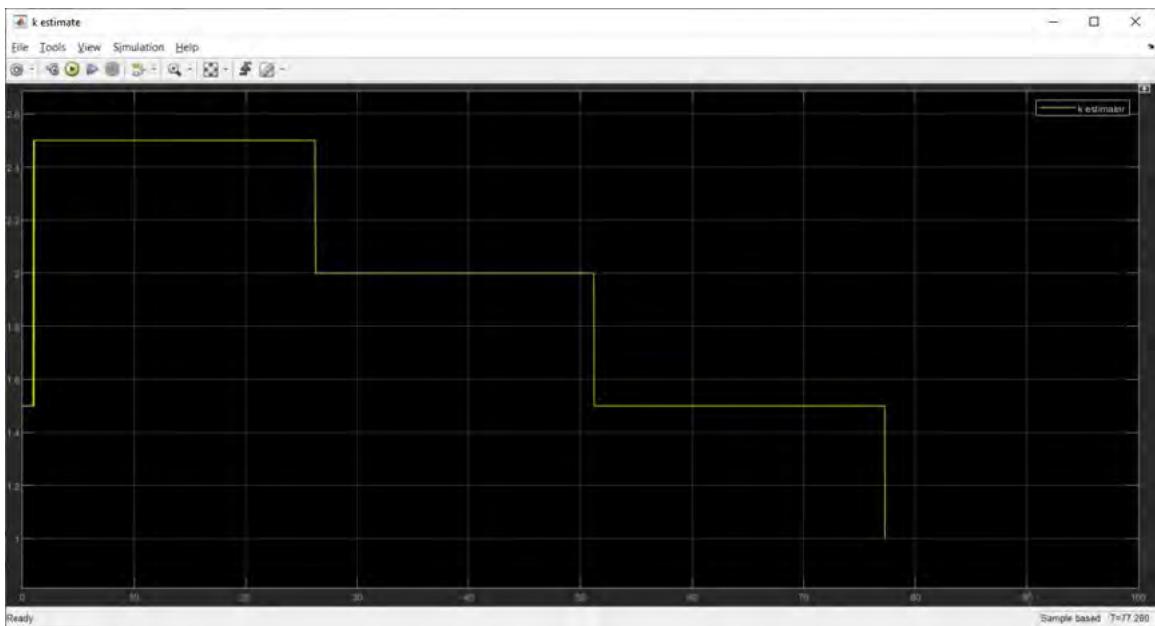
$$\begin{aligned}\ddot{x} + b\dot{x} + kx &= u \\ \dot{x}(0) &= 0 \\ x(0) &= 0 \\ b &= 0.5\end{aligned}$$

The process operates in response to a square wave input,  $u$ . The value of  $b$  is constant, but the value of  $k$  changes as failures in components occurs. Nominally,  $k = 2.5$ . When one failure occurs, the value of  $k$  becomes 2. When the second failure occurs, the value of  $k$  becomes 1.5. Similarly, with three failures,  $k$  becomes 1. The process can withstand three failures and continue operating, but a fourth failure would cause significant damage to the system, so it is imperative that the process be stopped upon the third failure. Your job is to develop a discrete-time, linear Kalman filter bank to predict the value of  $k$  based solely upon input ( $u$ ) and measured output values ( $z$ ) and stop the process when the third failure occurs.

Here are the particulars of the problem:

- 1) Implement the simple 2<sup>nd</sup> order system with a square wave of amplitude 1 and frequency of 0.05 Hz as the input,  $u$ , in Simulink with a fixed-step solver with a fixed-step time step of 0.01 seconds. Set the simulation to run 100 seconds. To test the system, have  $k$  start at 2.5 but changing to 2 at 23 seconds (first failure), changing to 1.5 at 49 seconds (second failure), and finally changing to 1 at 75 seconds (third failure).
- 2) To generate the measurements for the Kalman filters, add uncorrelated Gaussian noise (i.e. different seeds) to the states with zero mean and 0.001 variance.
- 3) While the simulation runs with a 0.01 fixed time-step, have the Kalman filters run updates (corrections) at 1 Hz instead of 100 Hz. You will have to have the corrector pass the predicted value as its output most of the time, only update the estimate once per second. Set up 4 filters, one set with  $k = 2.5$ , one set with  $k = 2$ , one set with  $k = 1.5$ , and the fourth set with  $k = 1$ . The choice of  $Q$  and  $R$  is up to you as the engineer, as is the initial value of covariance,  $P$ .
- 4) Determine which filter bank is most correct and use that information to indicate the current value of  $k$ . Set the simulation to stop when you determine that the third

- failure has occurred (i.e. when the estimated value of  $k = 1$ ). It seems appropriate to use either the difference between the predictions and the measurements or the square root of the covariance as a way to determine which filter matches the current system dynamics. However, the metric is up to you as the engineer. Note that the estimates will be noisy and you'll have to use a moving average or low pass filter to help determine which filter is most accurate. The choice is up to you.
- 5) Document your approach to this complete project, especially step 4), and show your model and simulation results, including reporting how much time it takes your system to identify when a failure happens. I'd expect to see state time histories, state estimate time histories, and square root of the covariance time histories plotted with the estimation error, all which document the filter performance. – *As a side note: since the filters are all starting up at the same time and since it takes a bit for them to settle, I suggest you add logic to prevent the simulation from stopping in the first 1 or 2 seconds, regardless of the estimate of  $k$  in that first 1 or 2 seconds.*



Note: The above figure is representative of what I'd expect the  $k$  estimate to look like. However, the above is not very fast at identifying the changes in  $k$ . I'd like to think that you could do better, but that's not a specific requirement for this project.

## EE5327, Fall 2020

### Project 2: Kalman Filter Bank for Failure Detection

Due December 15, 2020 (12 noon, not midnight)

**Please turn in this paper with your exam:**

Pledge of honor: "On my honor I have neither given nor received aid on this project except from the TA or class instructor."

Signature: Atul Shrotriya

A process in a factory can be modeled by a simple second order ODE as follows:

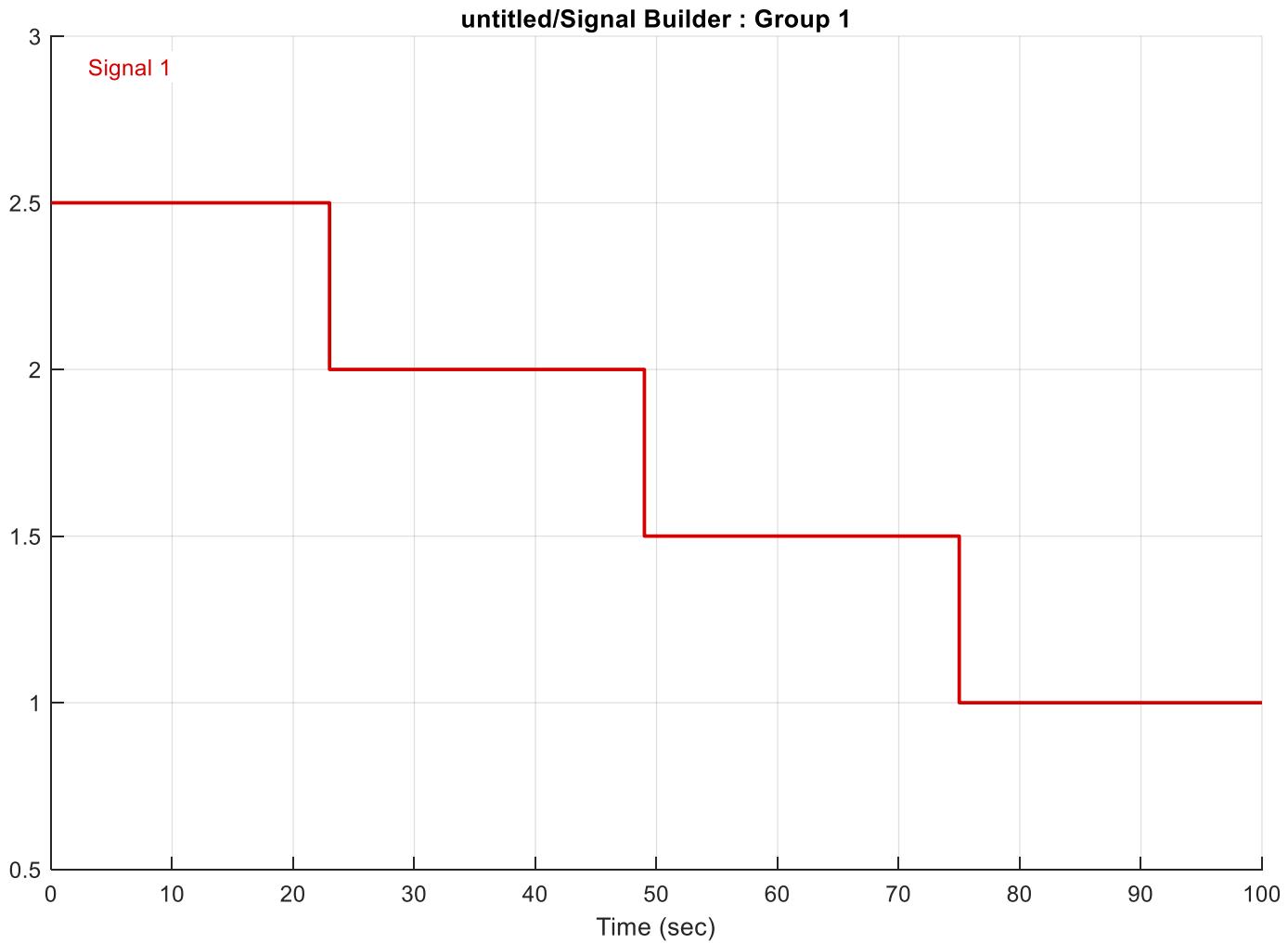
$$\begin{aligned}\ddot{x} + \frac{b}{m}\dot{x} + \frac{k}{m}x &= u \\ \dot{x}(0) &= 0 \\ x(0) &= 0 \\ \frac{b}{m} &= 0.5\end{aligned}$$

The process operates in response to a square wave input,  $u$ . The value of  $b$  is constant, but the value of  $k$  changes as failures in components occurs. Nominally,  $k = 2.5$ . When one failure occurs, the value of  $k$  becomes 2. When the second failure occurs, the value of  $k$  becomes 1.5. Similarly, with three failures,  $k$  becomes 1. The process can withstand three failures and continue operating, but a fourth failure would cause significant damage to the system, so it is imperative that the process be stopped upon the third failure. Your job is to develop a discrete-time, linear Kalman filter bank to predict the value of  $k$  based solely upon input ( $u$ ) and measured output values ( $z$ ) and stop the process when the third failure occurs.

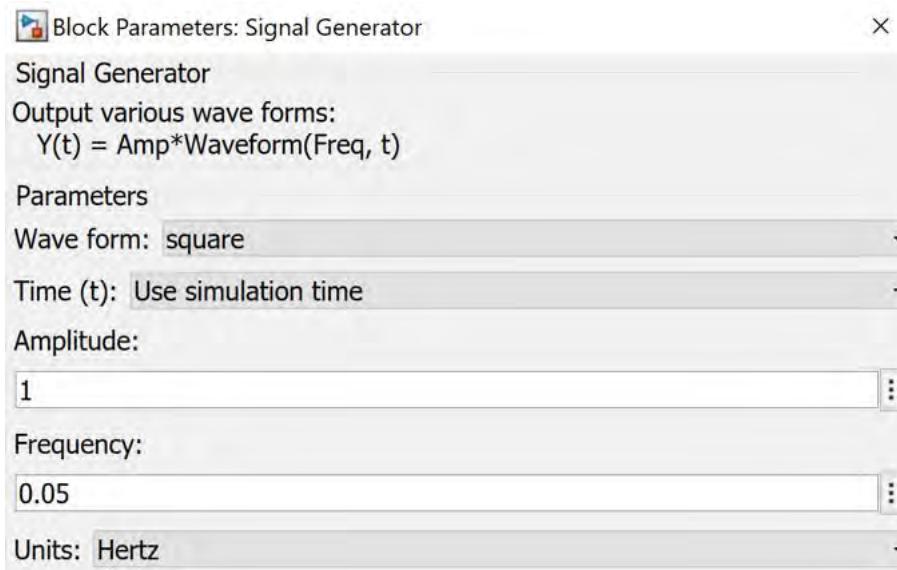
Here are the particulars of the problem:

- 1) Implement the simple 2<sup>nd</sup> order system with a square wave of amplitude 1 and frequency of 0.05 Hz as the input,  $u$ , in Simulink with a fixed-step solver with a fixed-step time step of 0.01 seconds. Set the simulation to run 100 seconds. To test the system, have  $k$  start at 2.5 but changing to 2 at 23 seconds (first failure), changing to 1.5 at 49 seconds (second failure), and finally changing to 1 at 75 seconds (third failure).
- 2) To generate the measurements for the Kalman filters, add uncorrelated Gaussian noise (i.e. different seeds) to the states with zero mean and 0.001 variance.
- 3) While the simulation runs with a 0.01 fixed time-step, have the Kalman filters run updates (corrections) at 1 Hz instead of 100 Hz. You will have to have the corrector pass the predicted value as its output most of the time, only update the estimate once per second. Set up 4 filters, one set with  $k = 2.5$ , one set with  $k = 2$ , one set with  $k = 1.5$ , and the fourth set with  $k = 1$ . The choice of  $Q$  and  $R$  is up to you as the engineer, as is the initial value of covariance,  $P$ .
- 4) Determine which filter bank is most correct and use that information to indicate the current value of  $k$ . Set the simulation to stop when you determine that the third

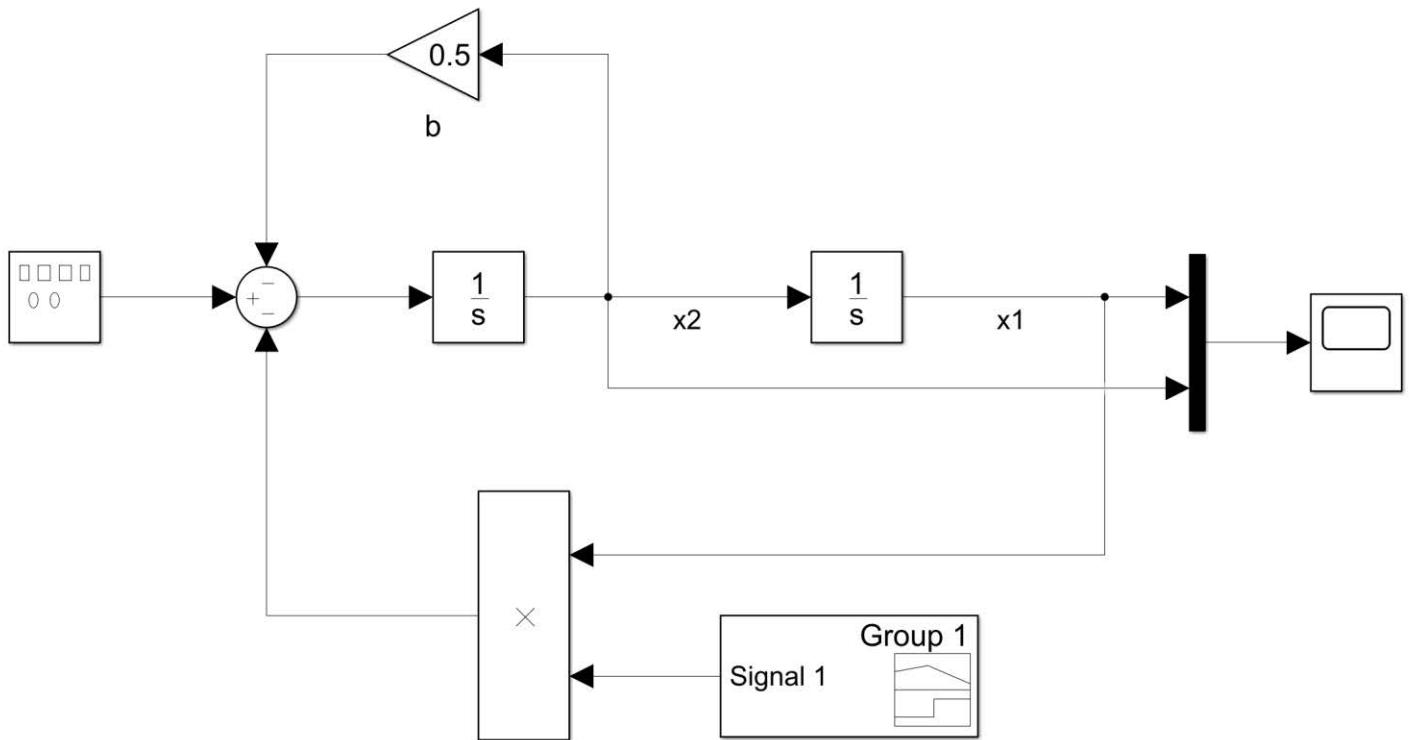
## 1. System Simulation



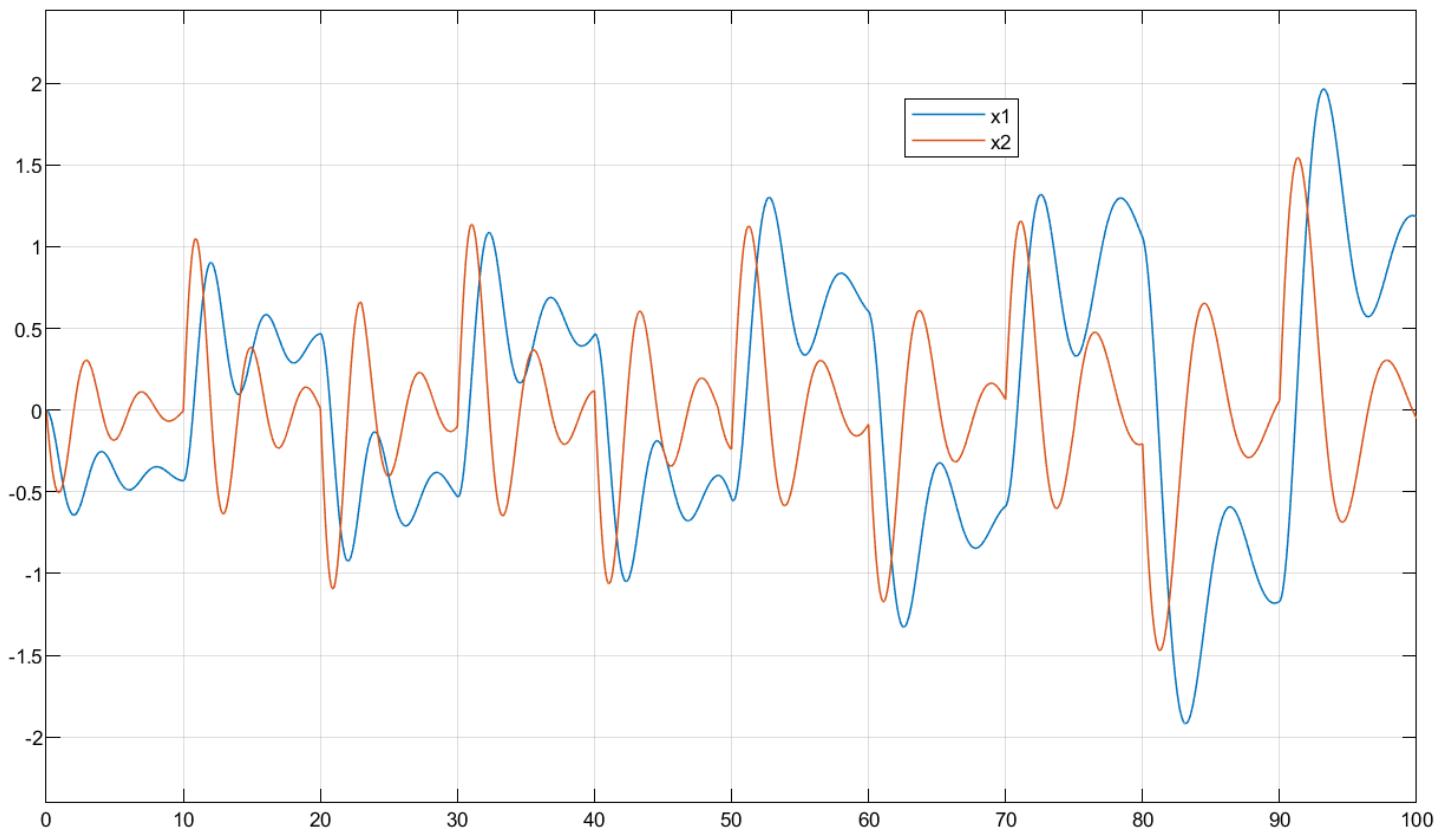
**Signal Builder for changing the values of k at specified times**



**Parameters for input signal**

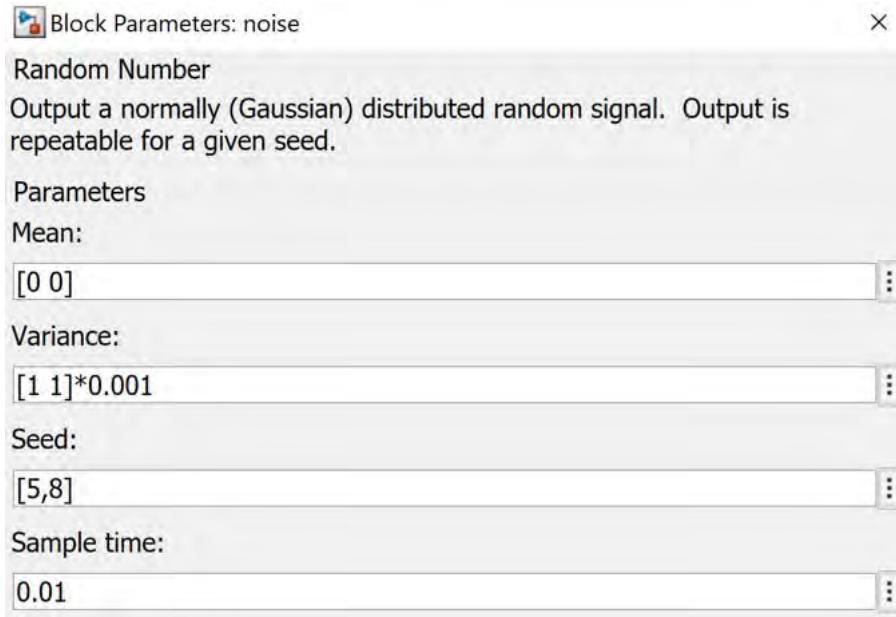


**Simulink Diagram for the system**

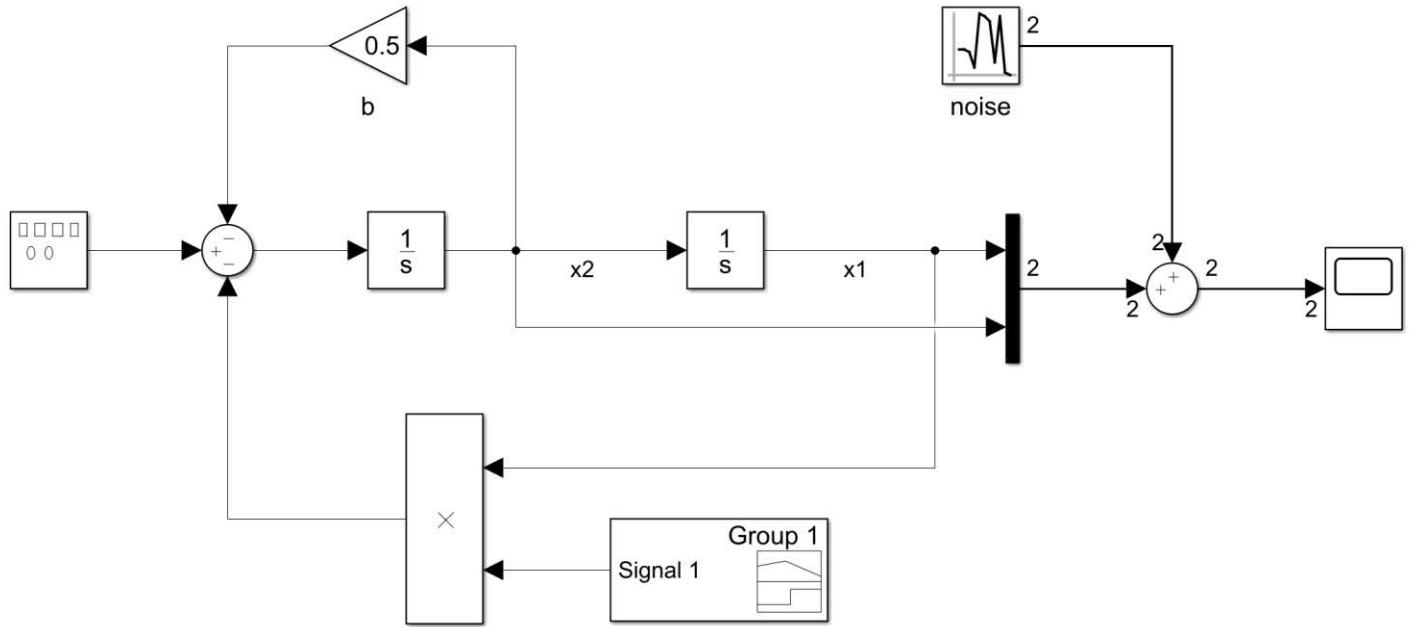


**Output of states**

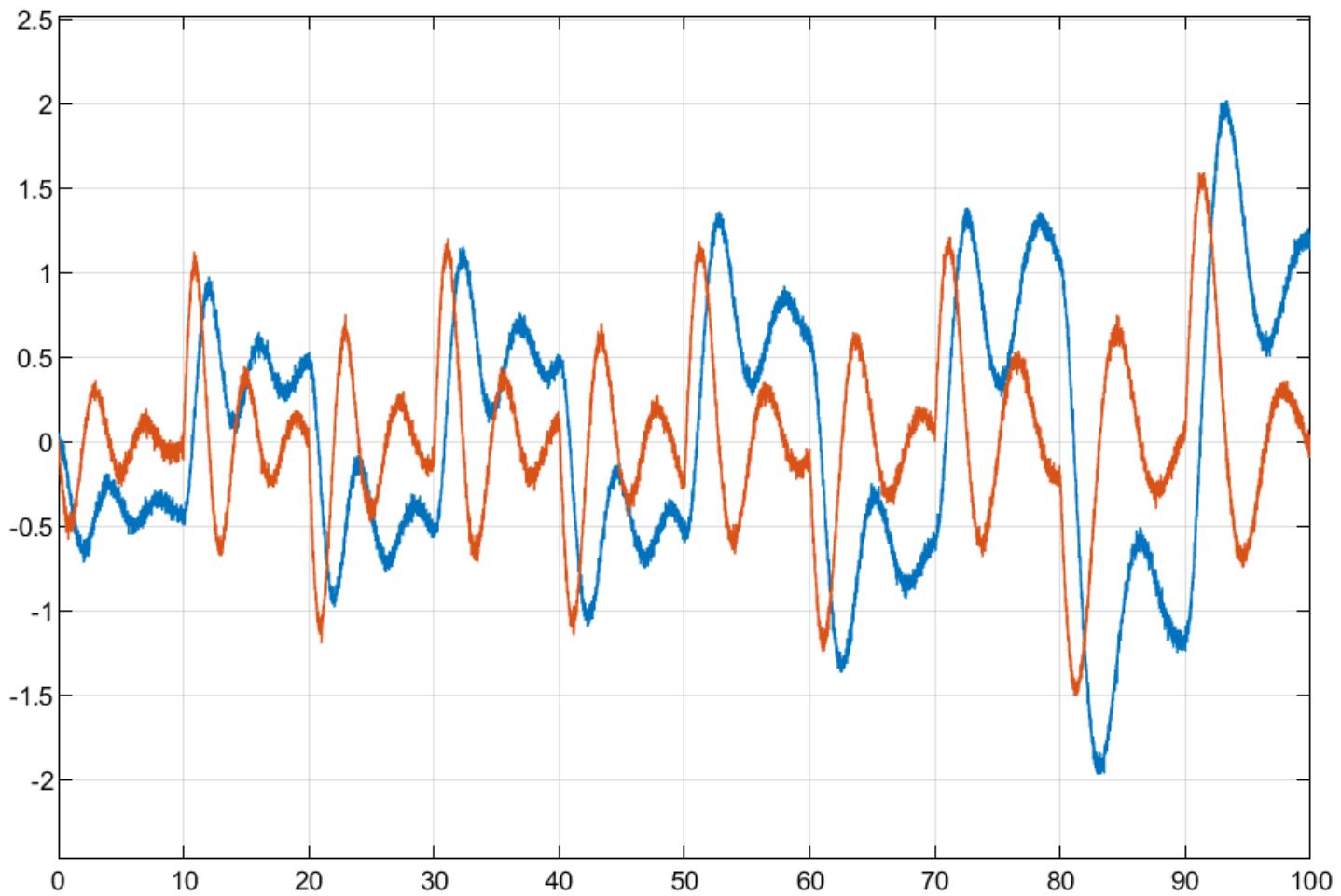
## 2. Adding noise to the system



**Noise Parameters**

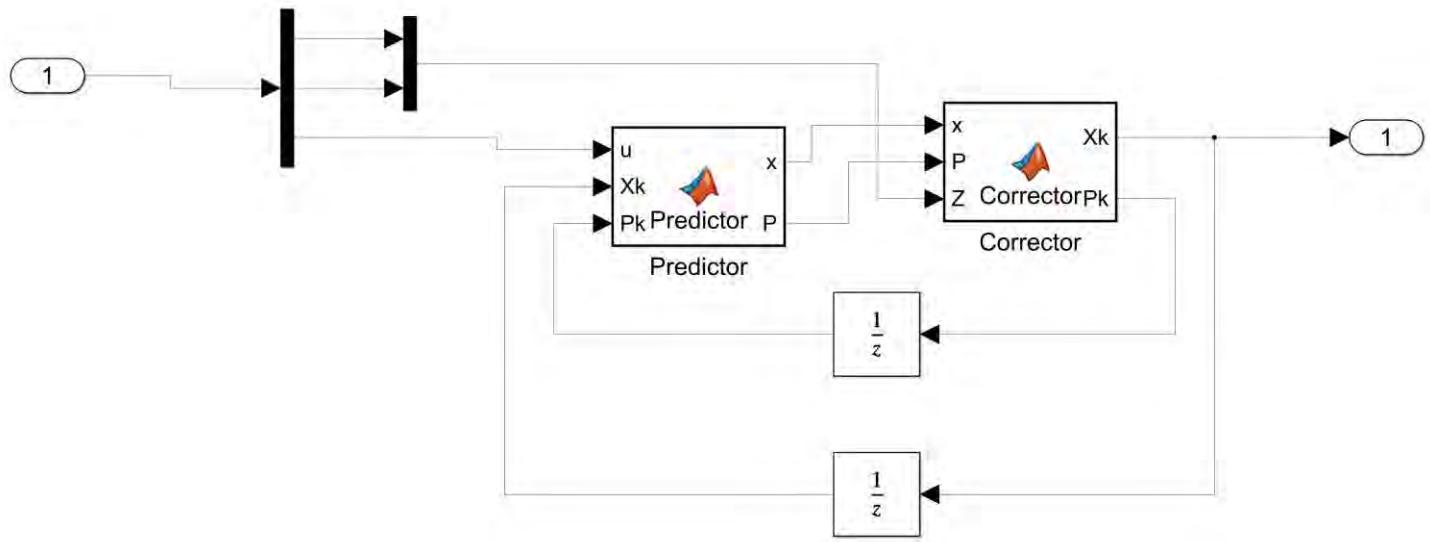


**Simulink Diagram for adding noise**



**System Output after addition of noise**

### 3. Implementing the Kalman filters



**Kalman Filter Subsystem Diagram**

**Note:** The code for all Kalman filters is the same, the only value which changes is of k in the predictor block which is 2.5, 2, 1.5 and 1 respectively.

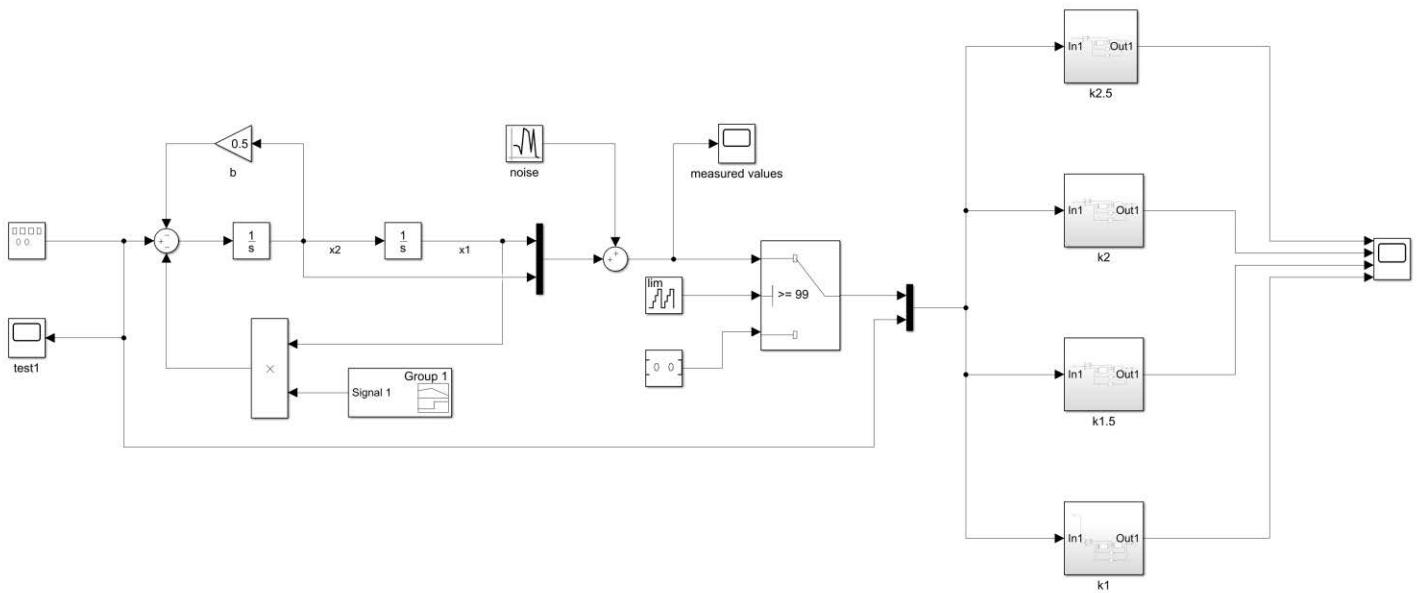
**Predictor Block MATLAB Code:**

```
function [x,P] = Predictor(u,Xk,Pk)
k=2.5; b=0.5;
A = [0 1;-k -b];
Ae=expm(A*0.01);
B= [0;1];
Bd = (inv(A))*(Ae - eye(2))*B;
Q=eye(2);
F = [-A Q; zeros(2,2) transpose(A)];
G = expm(F*0.01);
Qd=transpose(G(3:4,3:4))*G(1:2,3:4);
x=Ae*Xk + Bd*u;
P=(Ae*Pk*transpose(Ae)+Qd);
end
```

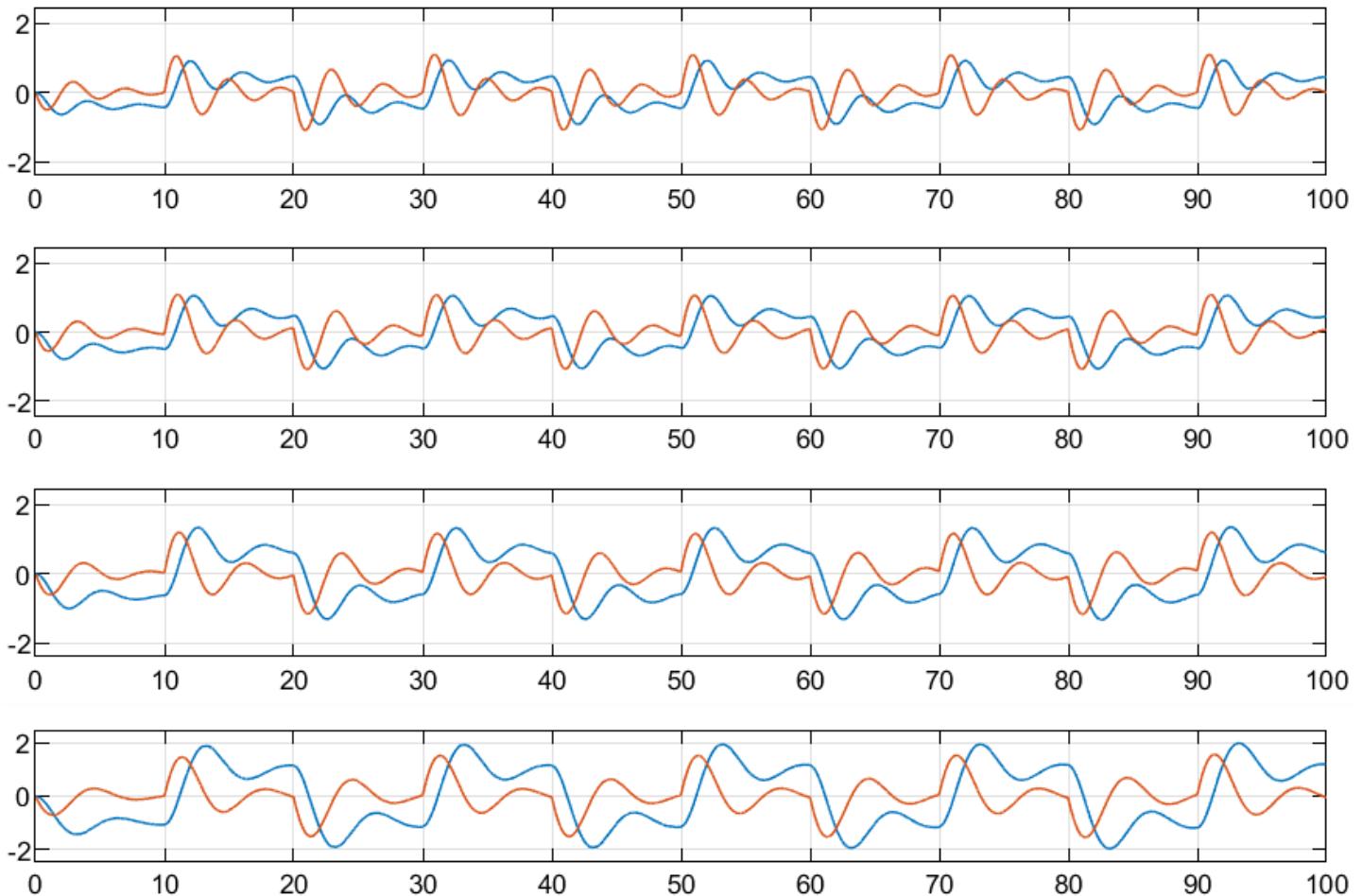
**Corrector Block MATLAB Code:**

```
function [Xk,Pk] = Corrector(x,P,Z)
H=eye(2); C=eye(2); R=eye(2)/0.01;
if Z(1)==0 && Z(2)==0
    Xk=x;
    Pk=P;
else
    K=P*transpose(H)*inv(H*P*transpose(H)+R);
    Pk=(eye(2)-K*H)*P*transpose(eye(2)-K*H)+K*R*transpose(K);
    Xk=x+K*(Z-H*x);
end
end
```

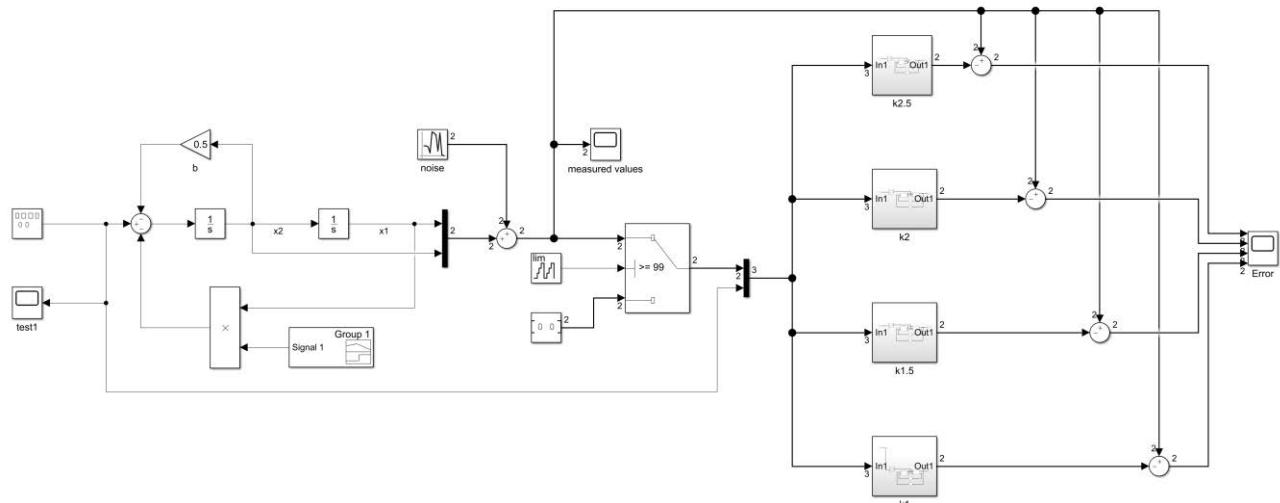
Four Kalman Filters are made by modifying the code as mentioned above. These are then put into a subsystem and their outputs are compared as shown in the following pages.



**Simulink Diagram for Comparing Kalman Filter Outputs**

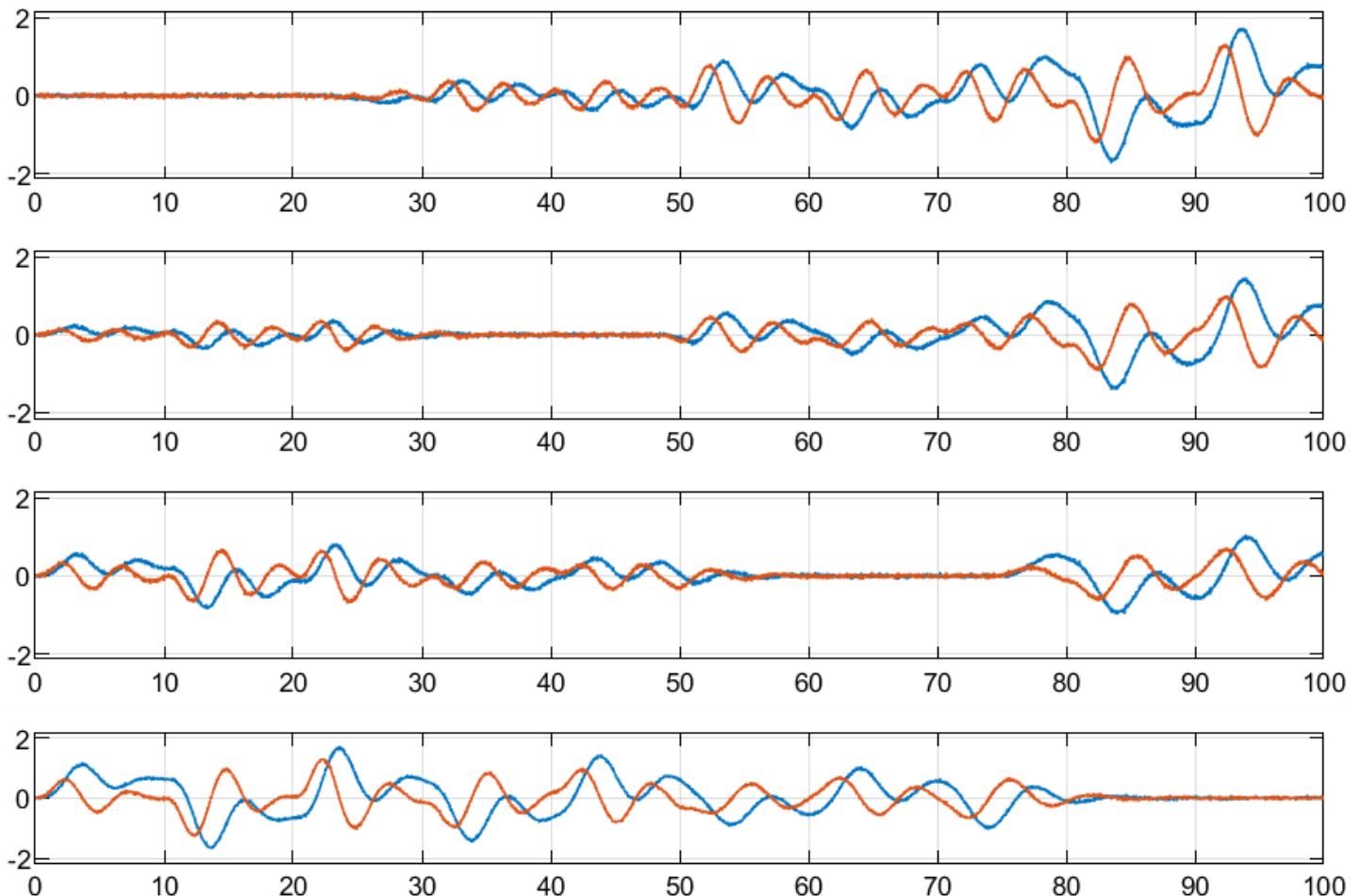


**Output Comparison of Kalman Filters**



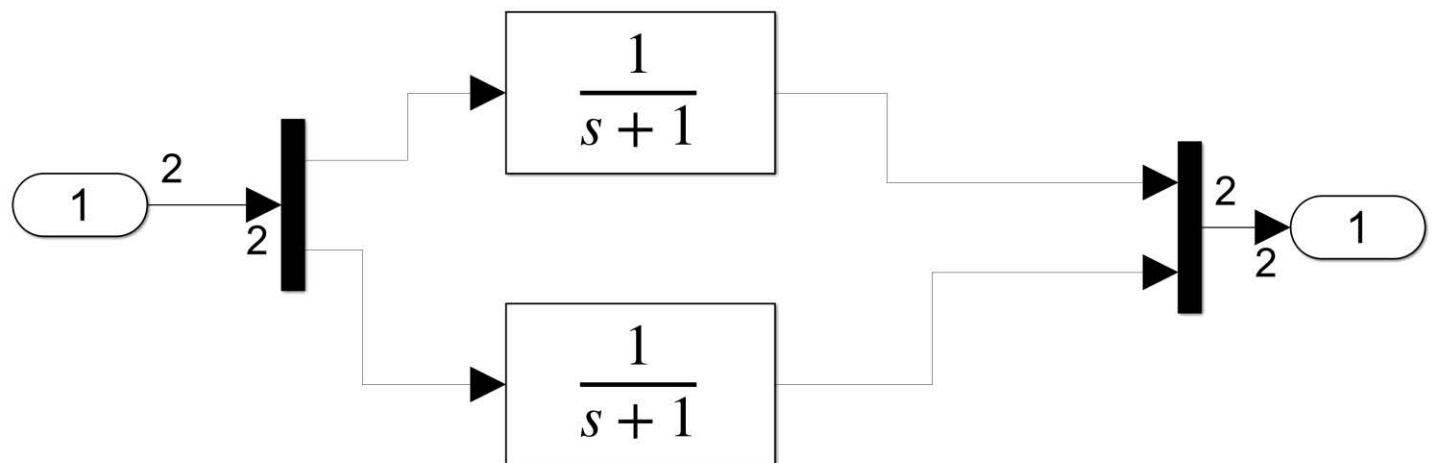
4.

**Simulink Diagram for Error Comparison**

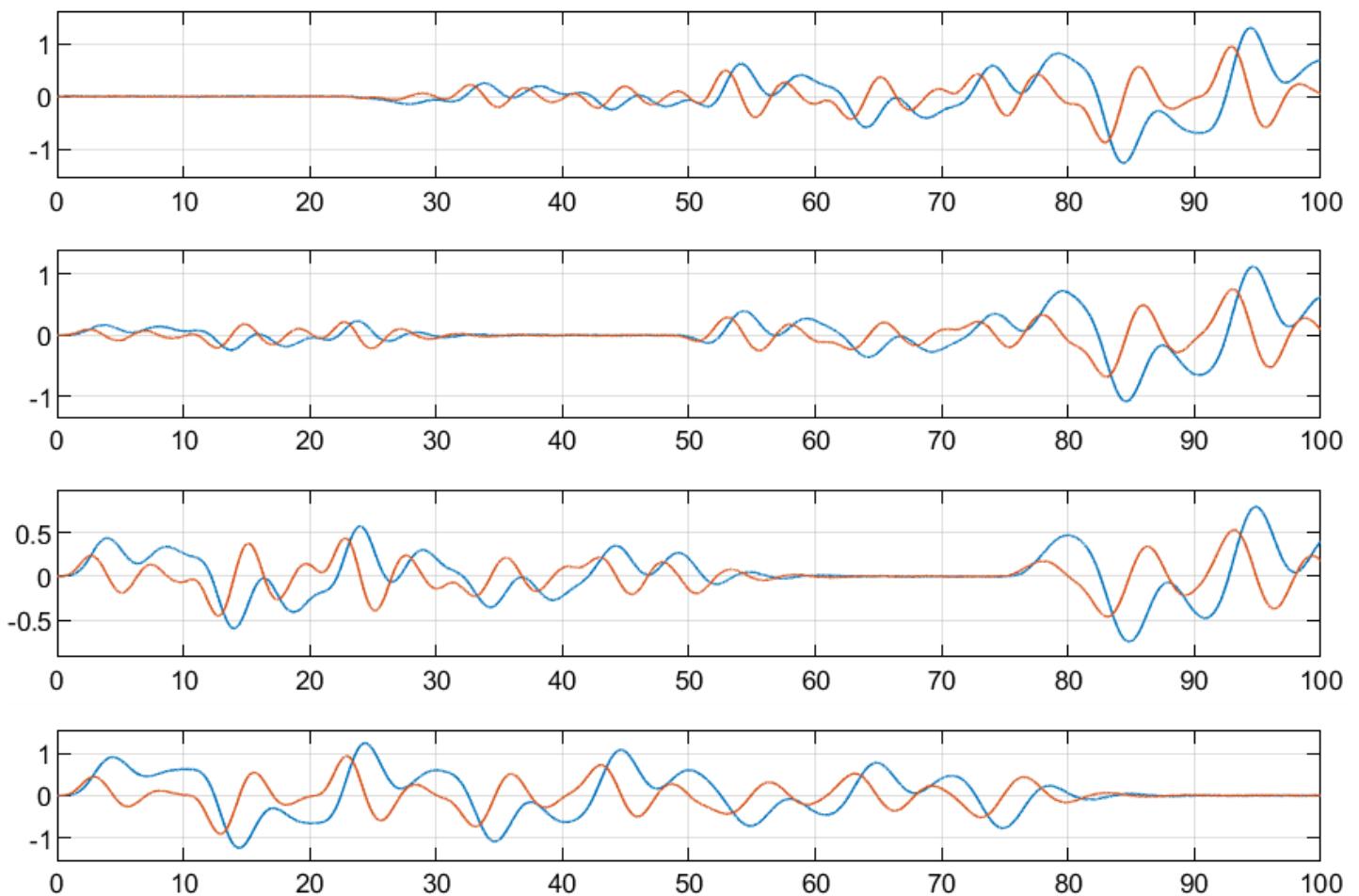


**Comparison of Errors**

As it can be seen from the above plots, the filters perform well and the errors are minimized when a filter has the same value of "k" as the system. This data can be used for selecting the best Kalman filter and to stop the system when the third failure occurs.



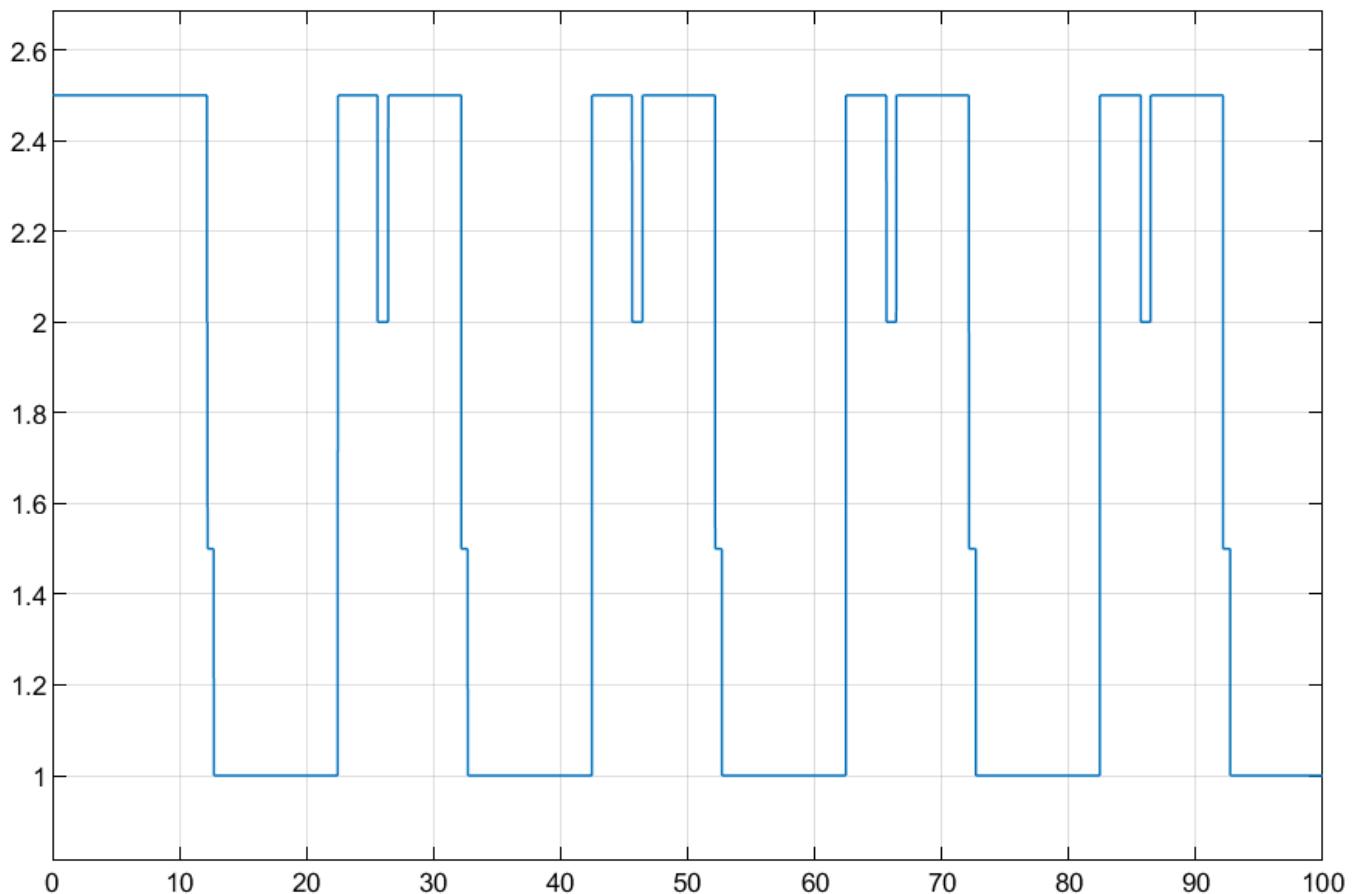
**Filter to smooth out the errors**



**Comparison of errors after filtering**

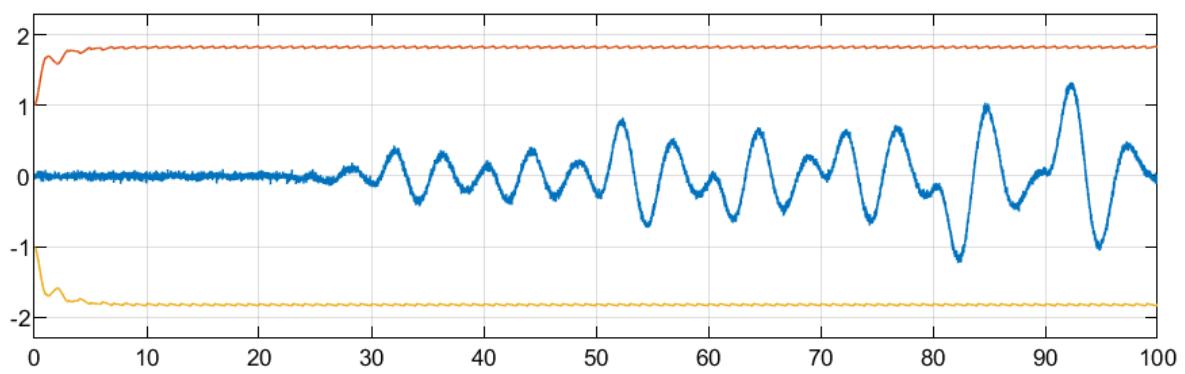
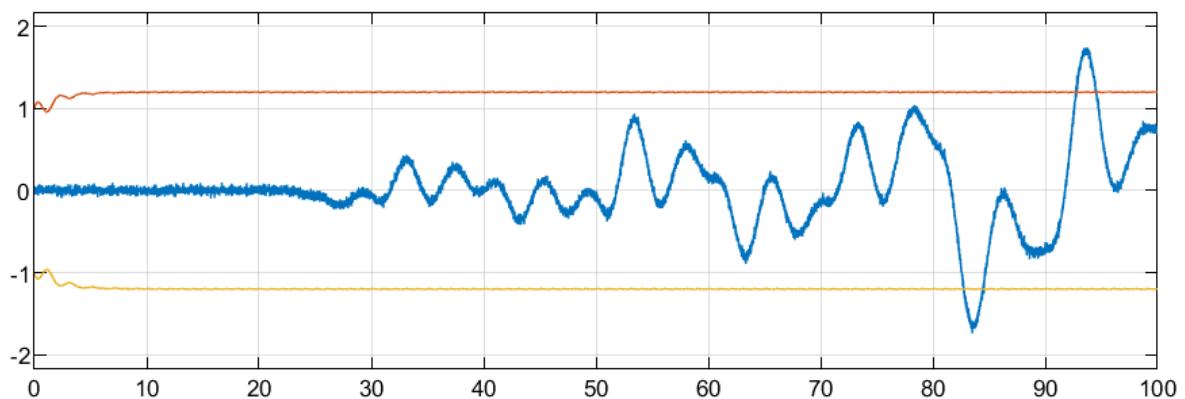
**MATLAB Function for selecting the right value of k:**

```
function K = selector(X1,X2,X3,X4)
[a,b]=min([X1(1),X2(1),X3(1),X4(1)]);
K=2.5;
switch b
    case 1
        K=2.5;
    case 2
        K=2;
    case 3
        K=1.5;
    case 4
        K=1;
end
```

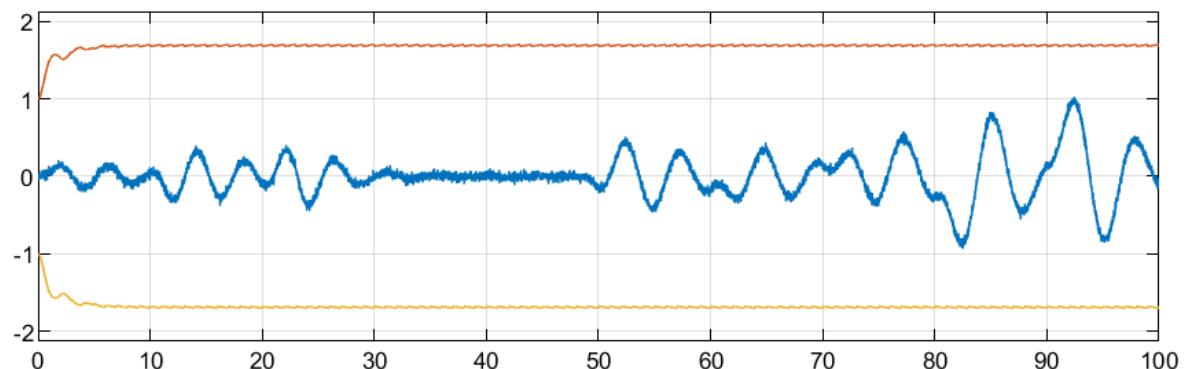
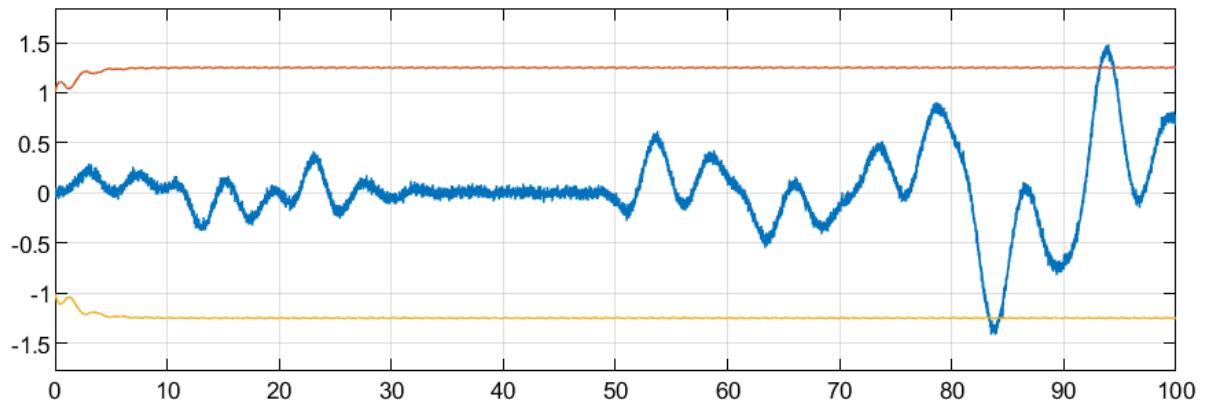


**Estimated output for K values**

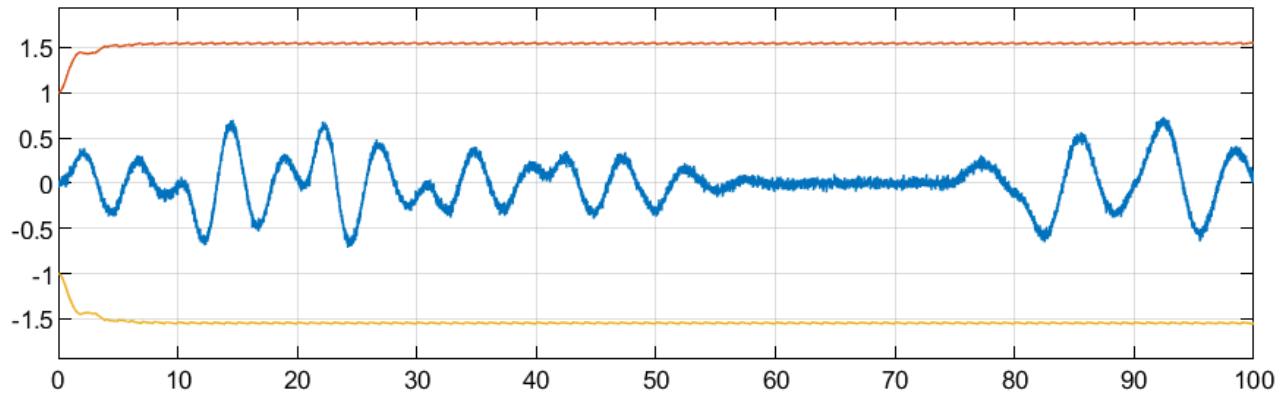
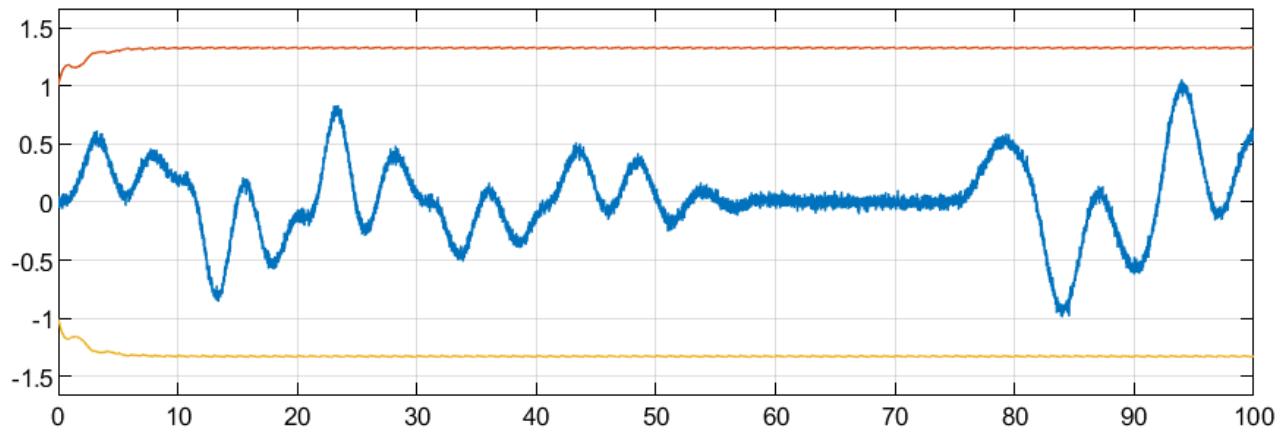
This doesn't work as well so another attempt is made using the covariance method.



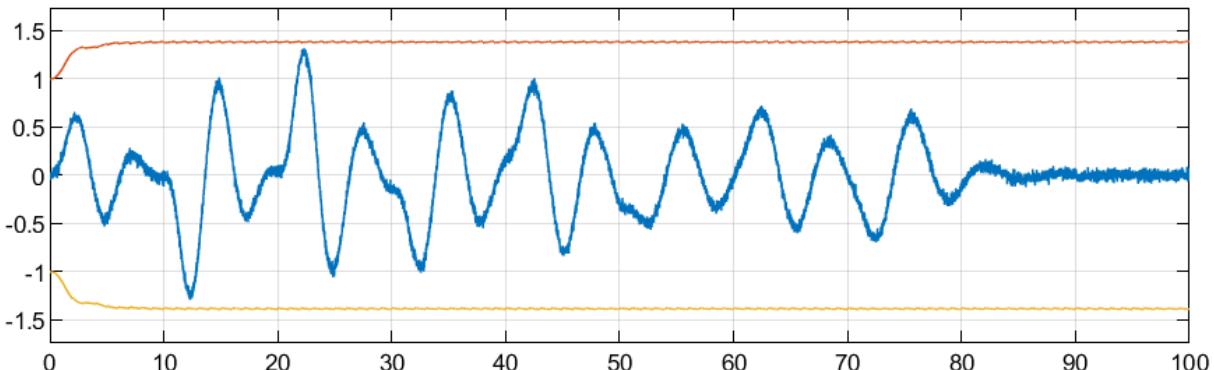
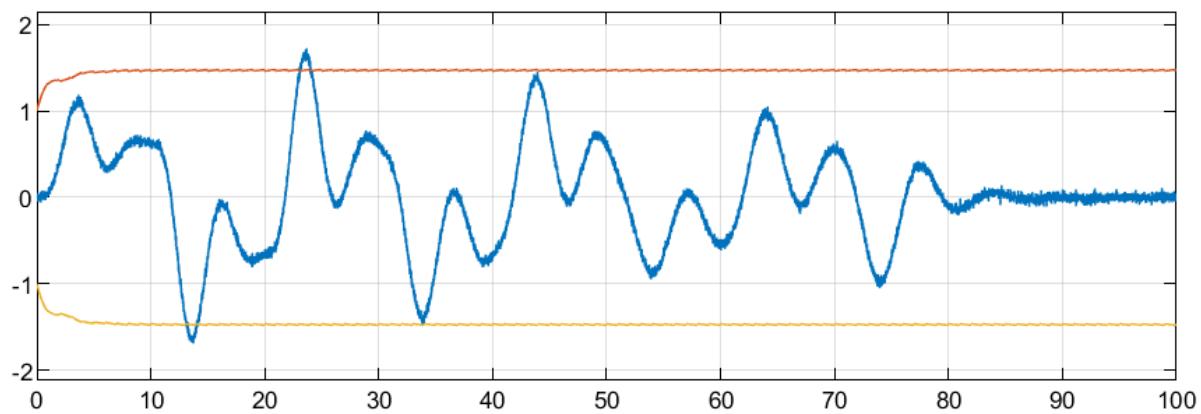
Covariance for  $k=2.5$



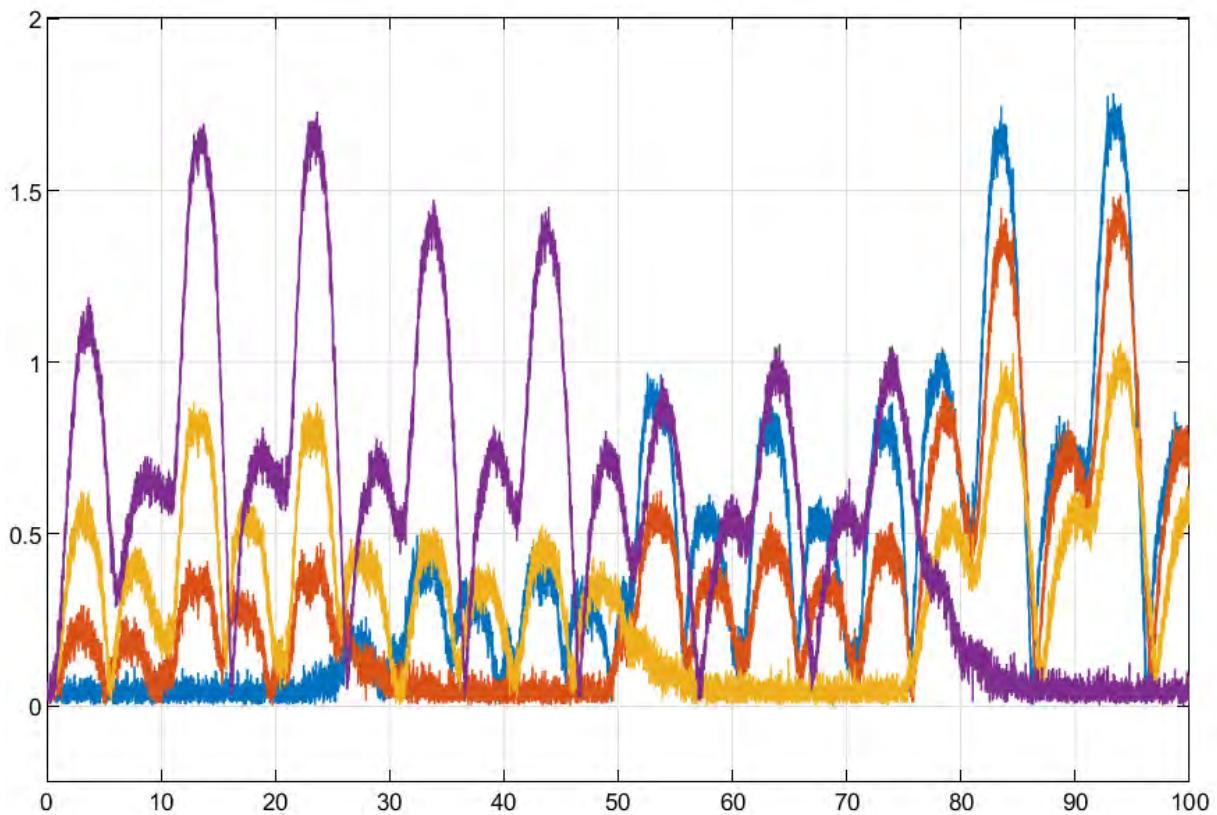
Covariance for  $k=2$



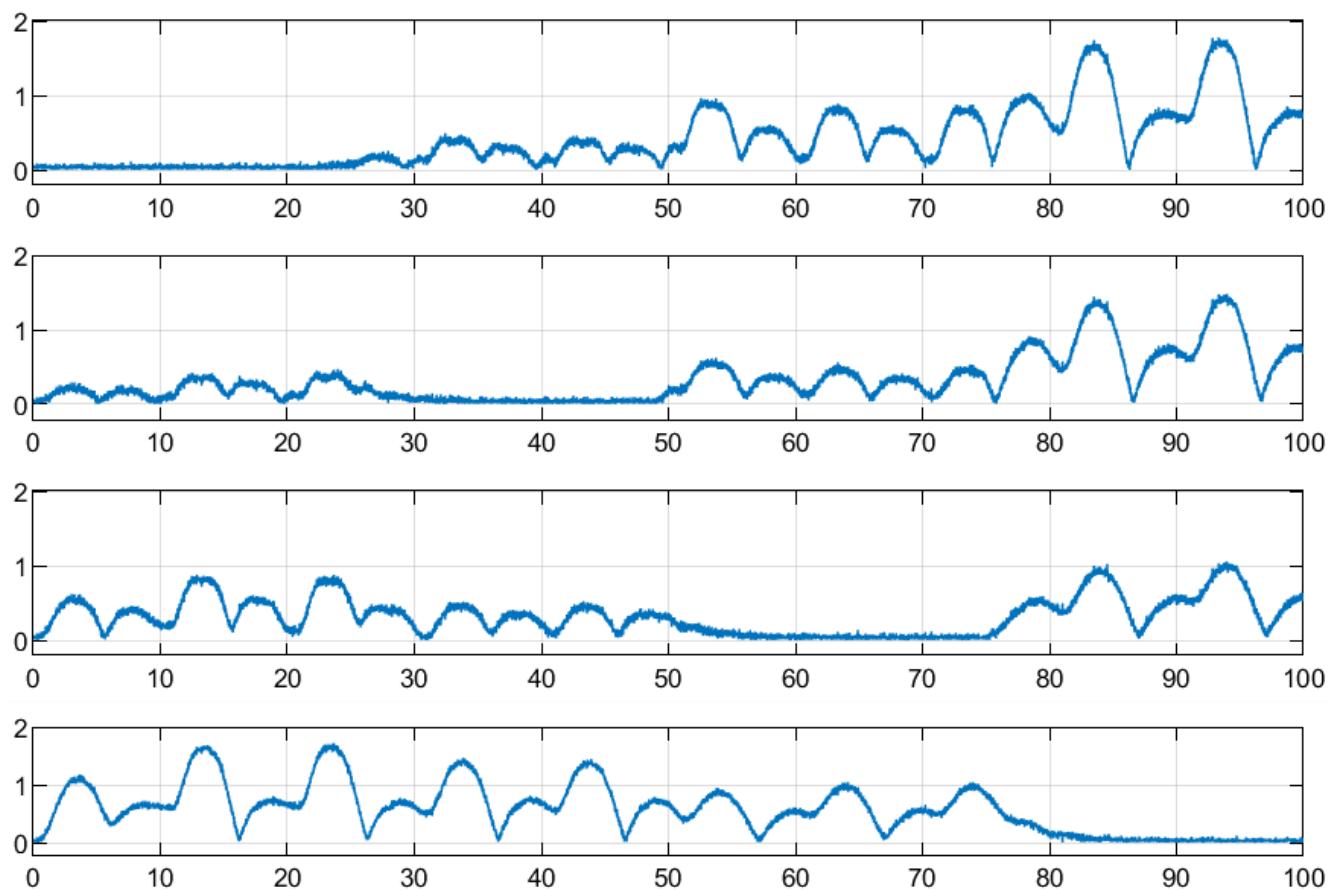
**Covariance for  $k=1.5$**



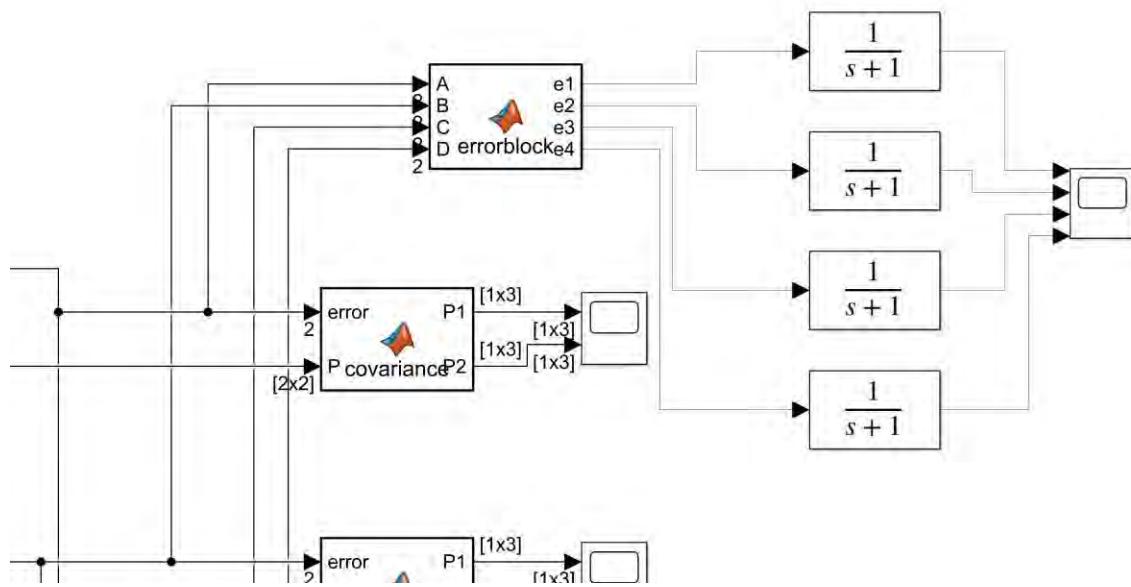
**Covariance for  $k=1$**



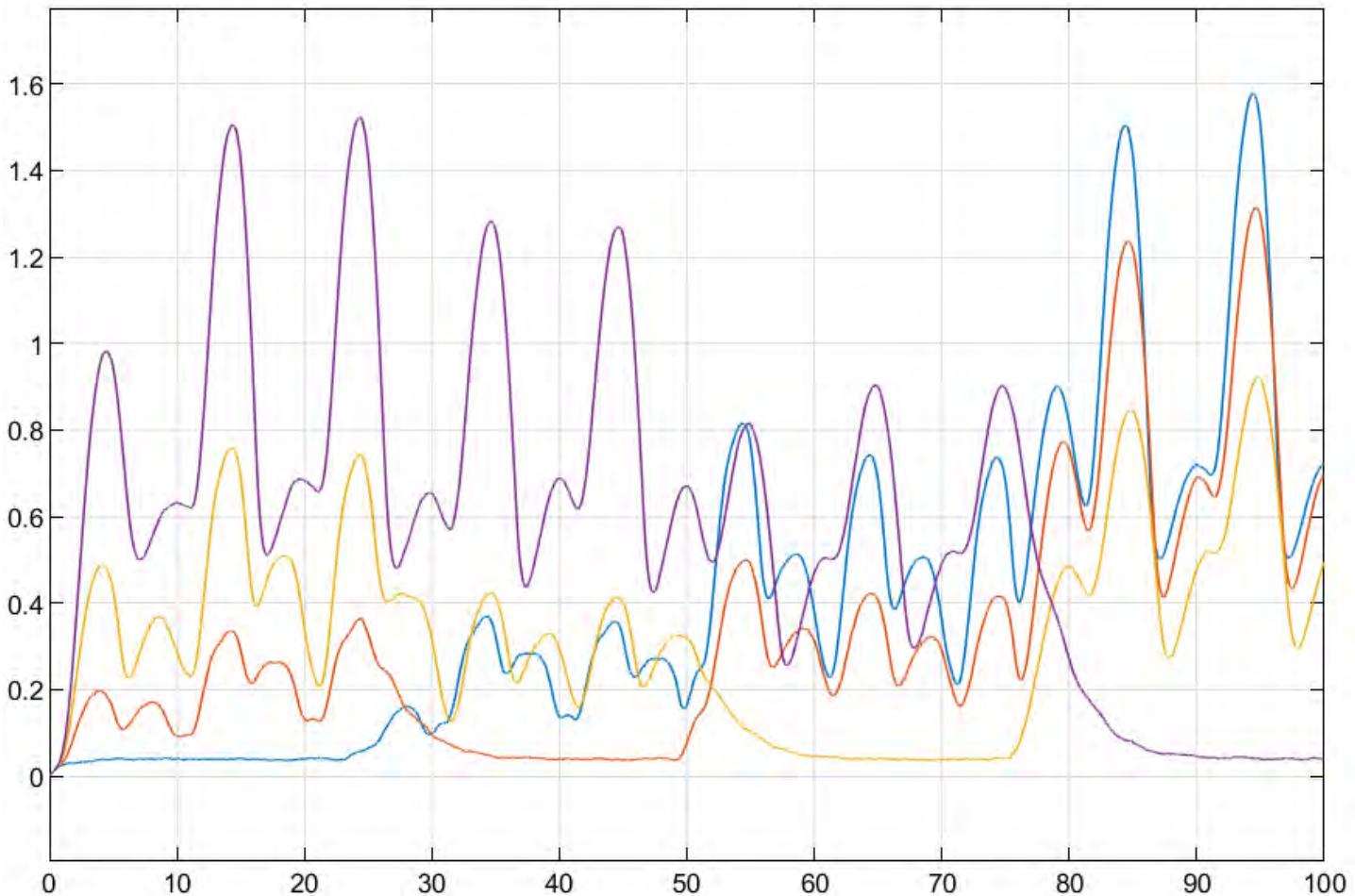
Error Block Outputs



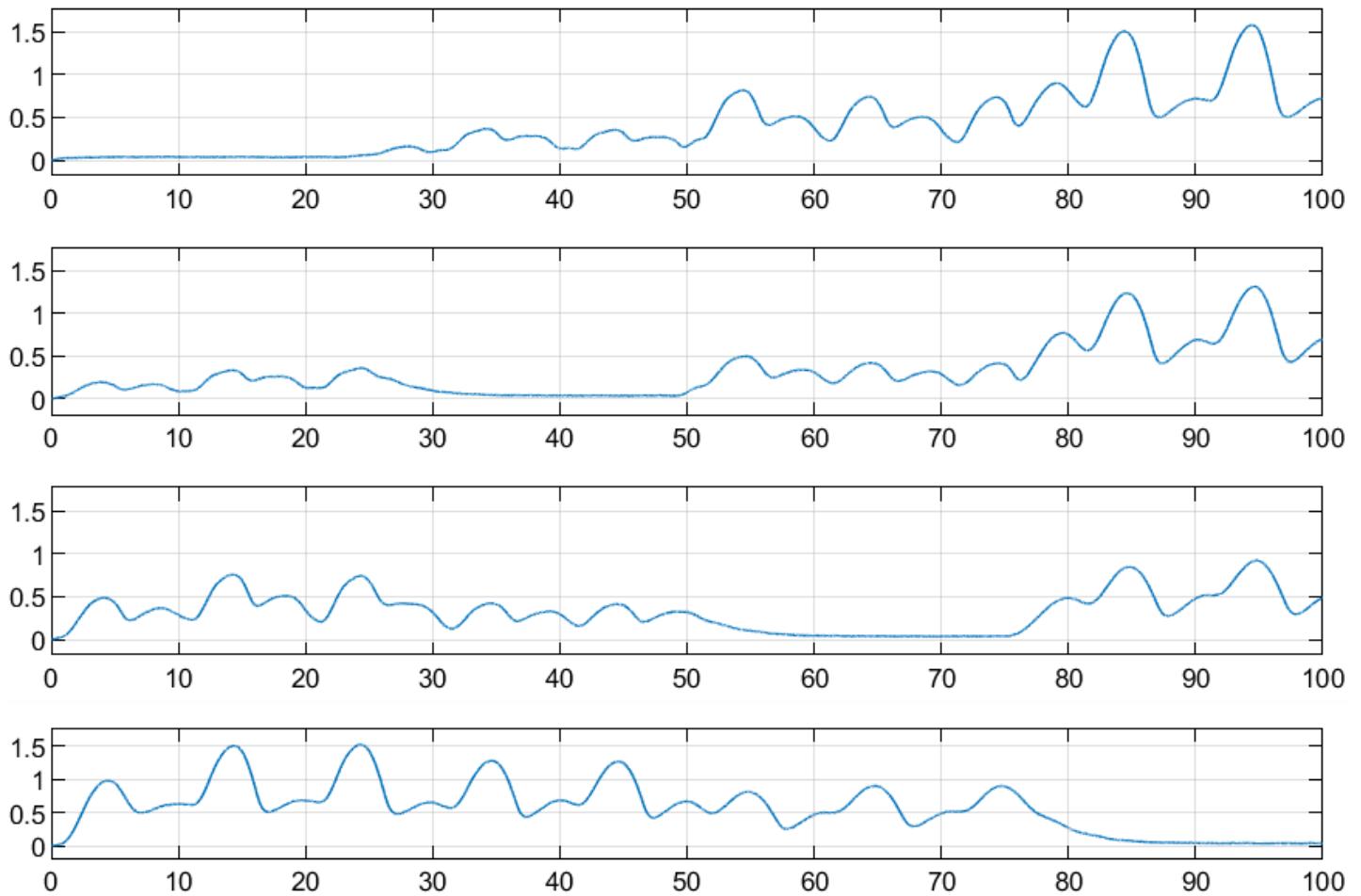
This output is noisy, so it might cause issues with selection of k. The errors have been filtered using a simple transfer function as shown ahead.



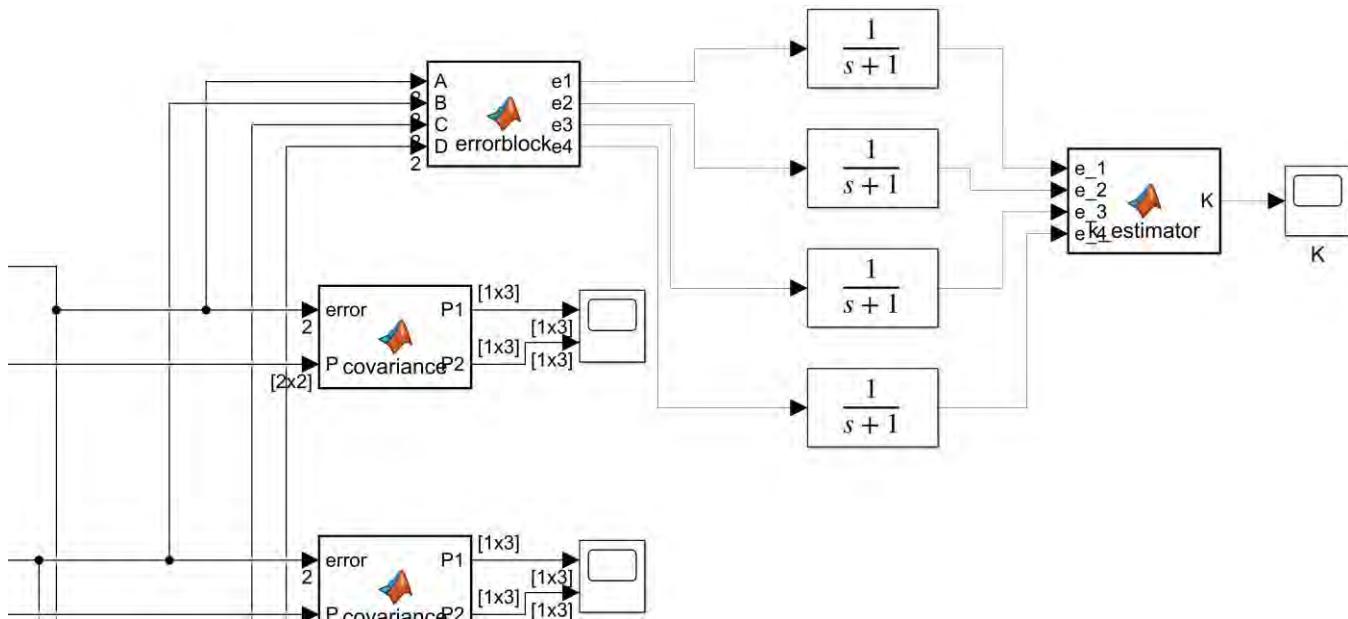
**Simulink Diagram for filtering and estimating k**



**Error output after filtering**



**Individual error outputs after filtering**



**Simulink Diagram for Selecting K**

**MATLAB code for covariance block**

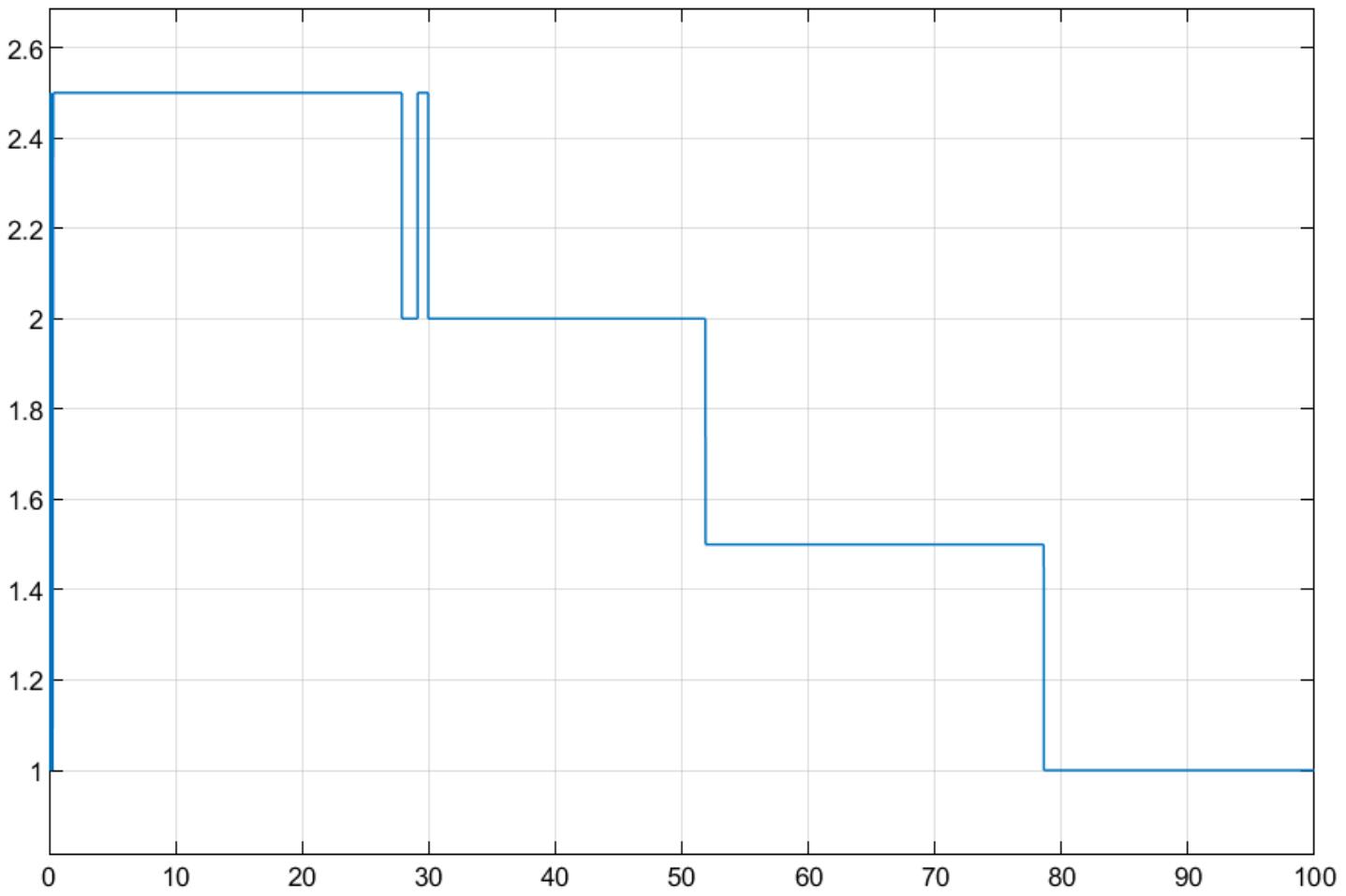
```
function [P1,P2]=covariance(error,P)
P1=[error(1) sqrt(P(1,1)) -sqrt(P(1,1))];
P2=[error(2) sqrt(P(2,2)) -sqrt(P(2,2))];
end
```

**MATLAB code for errorblock**

```
function [e1,e2,e3,e4] = errorblock(A,B,C,D)
e1=sqrt(A(1)^2+A(2)^2);
e2=sqrt(B(1)^2+B(2)^2);
e3=sqrt(C(1)^2+C(2)^2);
e4=sqrt(D(1)^2+D(2)^2);
end
```

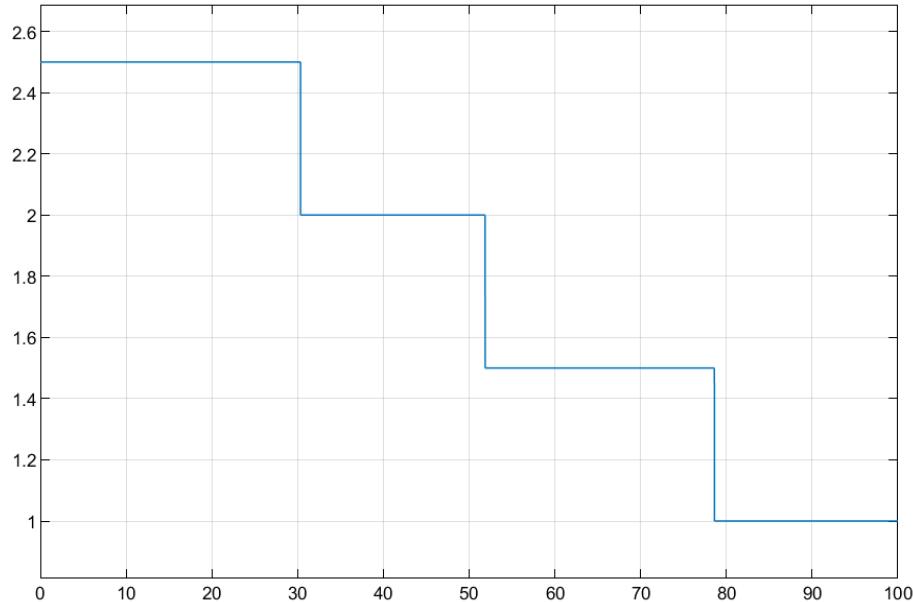
**MATLAB code for k\_estimator block**

```
function K = k_estimator(e_1,e_2,e_3,e_4)
[i,j]=min([e_1,e_2,e_3,e_4]);
K=2.5;
switch j
    case 1
        K=2.5;
    case 2
        K=2;
    case 3
        K=1.5;
    case 4
        K=1;
end
```



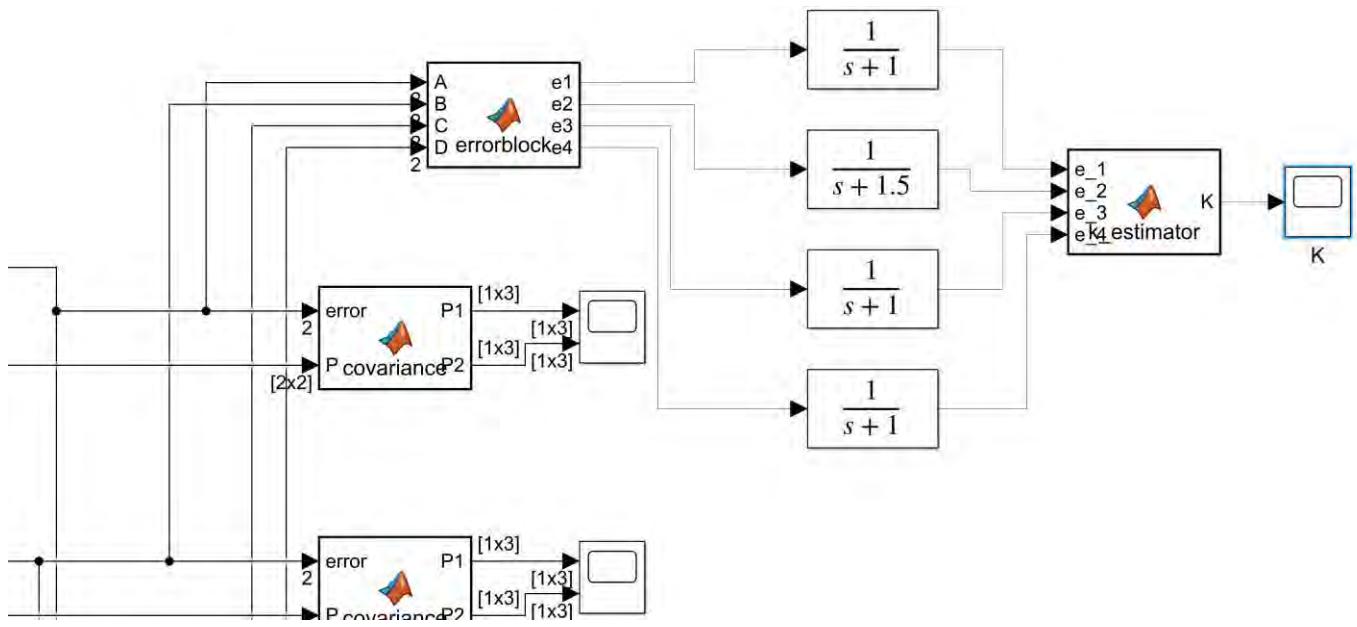
**Estimated plot for K value**

It can be seen that there is some issue at the start and at the point where k switches from 2.5 to 2. Manipulating the filter transfer functions may help.

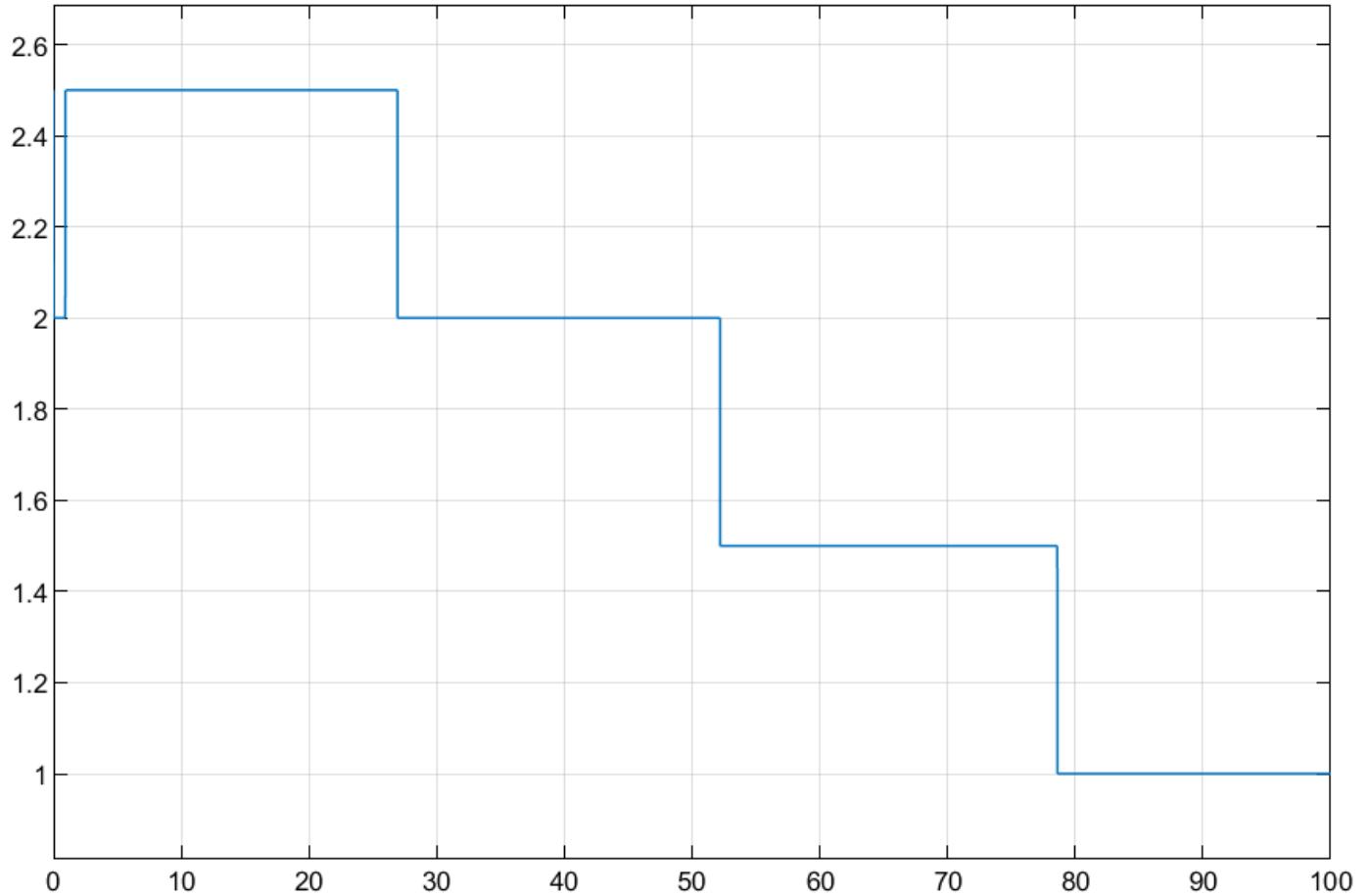


**Estimated plot for K value at  $1/s+1.25$**

Changing the value of first transfer function as  $1/s+1.25$  gives the following output. However, the system recognizes the change too late, changing it again to the following system provides a more accurate output.

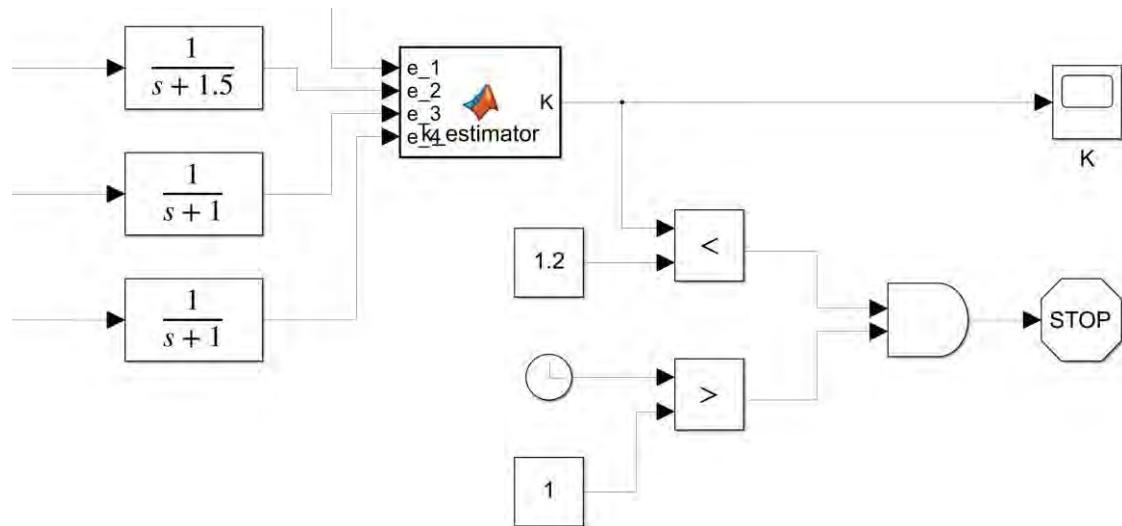


**Simulink Diagram for recognizing K values accurately**

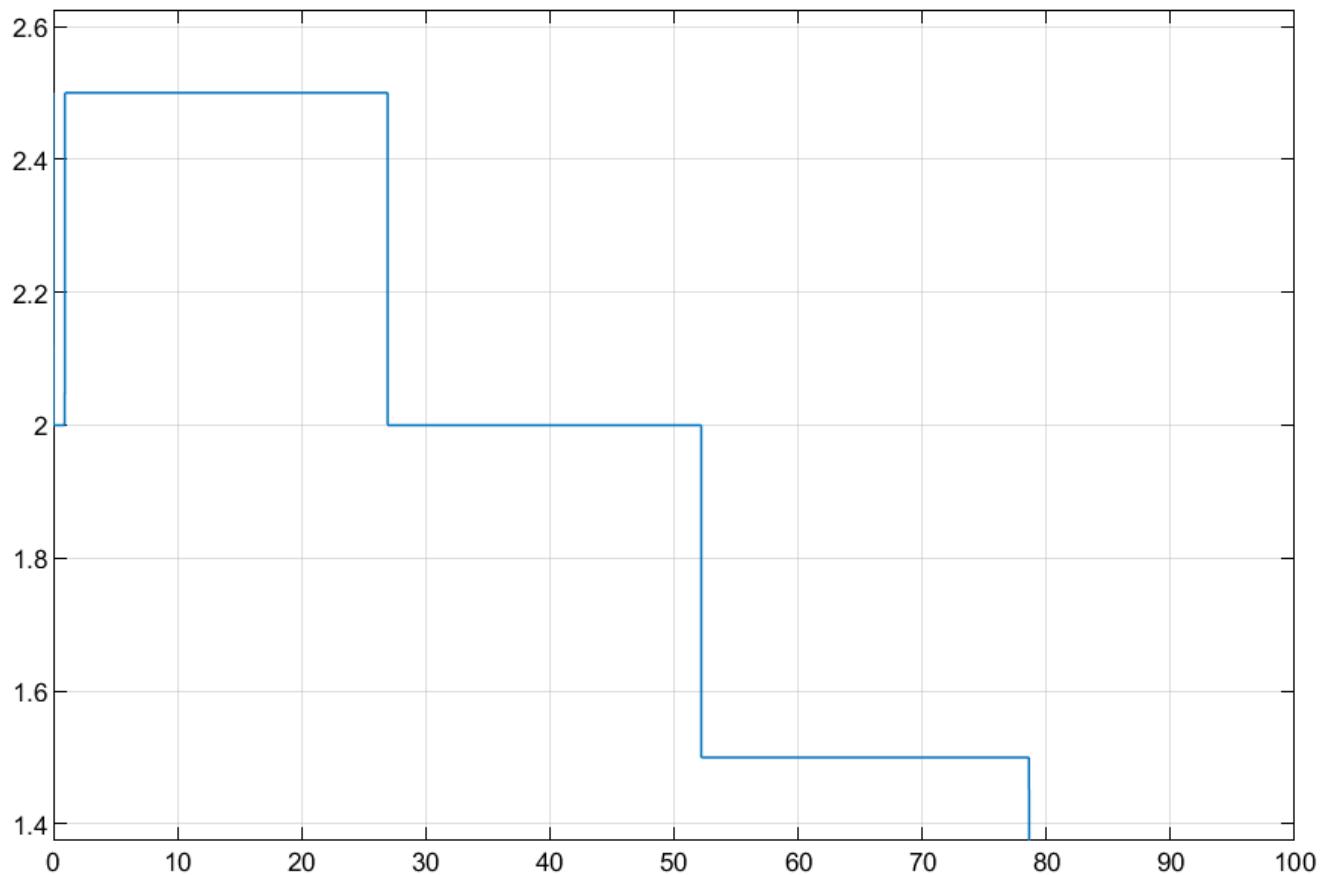


**Estimated plot for K values**

## 5. Implementing logic to stop the simulation after 3<sup>rd</sup> failure



Simulink Diagram



Estimated value of k

As it is observed, the simulation stops at 78.610 seconds i.e., almost 3.5 seconds after the last acceptable failure. Thus, the system works well and performs the task it was designed to do. There is a small initial error in the estimated value for k because the system is still adjusting to the inputs. The output response time and accuracy can further be improved by tuning the transfer functions further.

## EE 5323 Homework 1

### State Variable Systems, Computer Simulation

1. Simulate the van der Pol oscillator  $y'' + \alpha(y^2 - 1)y' + y = 0$  using MATLAB for various ICs. Plot  $y(t)$  vs.  $t$  and also the phase plane plot  $y'(t)$  vs.  $y(t)$ . Use  $y(0)=0.1$ ,  $y'(0)=0.1$ 
  - a. For  $\alpha=0.04$ .
  - b. For  $\alpha=0.85$ .
2. Do MATLAB simulation of the Lorenz Attractor chaotic system. Run for 150 sec. with all initial states equal to 0.4. Plot states versus time, and also make 3-D plot of  $x_1$ ,  $x_2$ ,  $x_3$  using PLOT3( $x_1, x_2, x_3$ ).

$$\dot{x}_1 = -\sigma(x_1 - x_2)$$

$$\dot{x}_2 = r x_1 - x_2 - x_1 x_3$$

$$\dot{x}_3 = -bx_3 + x_1 x_2$$

use  $\sigma=10$ ,  $r=28$ ,  $b=8/3$ .

3. Consider the Volterra predator-prey system

$$\dot{x}_1 = -x_1 + x_1 x_2$$

$$\dot{x}_2 = x_2 - x_1 x_2$$

Simulate the system using MATLAB for various initial conditions. Take ICs spaced in a uniform mesh in the box  $x1=[-2,2]$ ,  $x2=[-2,2]$ . Make one phase plane plot with all the trajectories on it. Plot phase plane on square  $[-5,5] \times [-5,5]$ .

**EE 5322 Intelligent Control**  
**Fall 2020**  
**Homework Pledge of Honor**

On all homeworks in this class - YOU MUST WORK ALONE.

*Any cheating or collusion will be severely punished.*

*It is very easy to compare your software code and determine if you worked together  
Or if you found code online written by someone else.  
It does not matter if you change the variable names.*

Please sign this form and include it as the first page of all of your submitted homeworks.

.....

Typed Name: ATUL SHROTRIYA

*Pledge of honor:*

"On my honor I have neither given nor received aid on this homework."

e-Signature: ATUL SHROTRIYA

## Problem – 1:

**MATLAB Code:** (Note: the initial all condition is changed to generate the second set of plots)

```
function hw1q1
%givens
all=0.85; y0=0.1; yd0=0.1;
tin=[0 100]; %time interval

[t,x]=ode23(@equations,tin,[0.1 0.1]);
y=x(:,1);

%plot1
figure(1)
plot(t,y,'r','LineWidth',2)
xlabel('time, s','FontWeight','bold')
ylabel('y(t)','FontWeight','bold')
title('Pol Oscillator','FontWeight','bold')

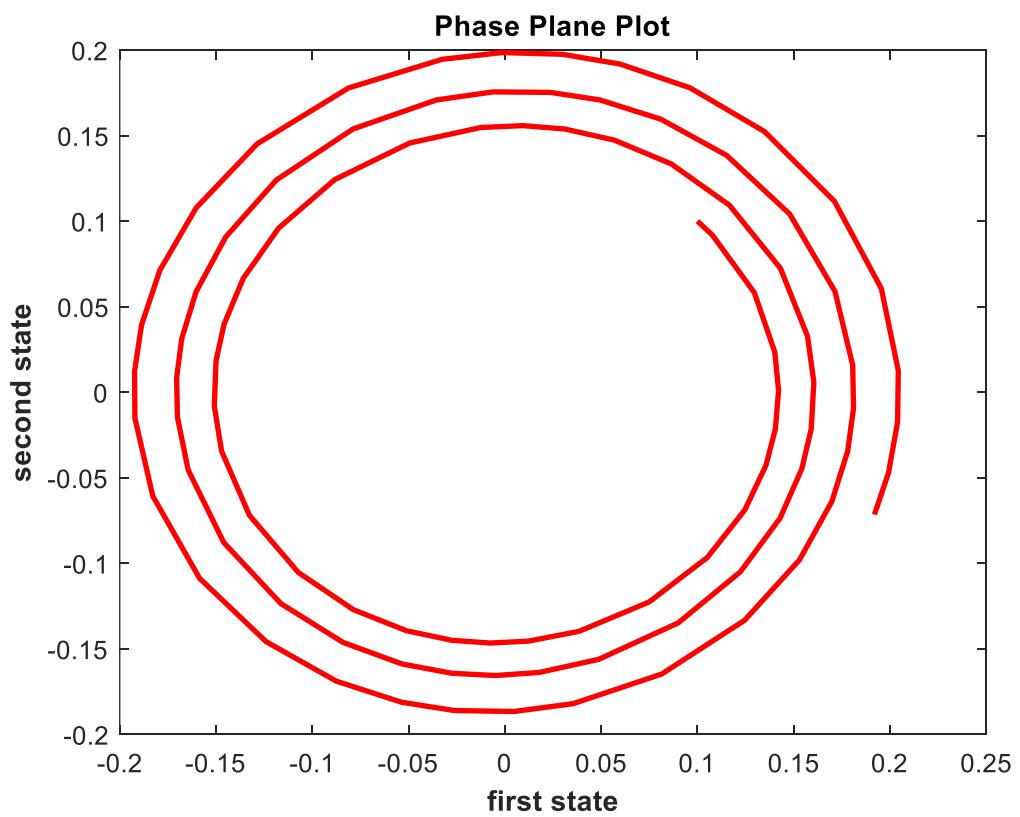
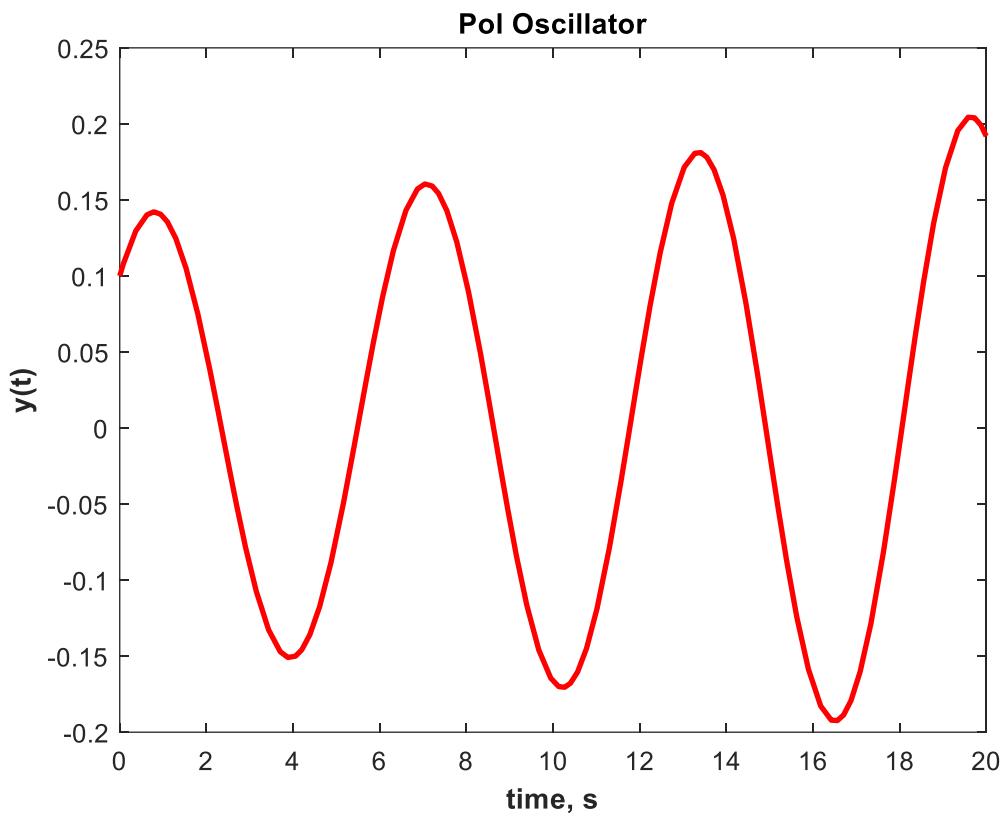
%plot2
figure(2)
plot(y,x(:,2),'r','LineWidth',2)
xlabel('first state','FontWeight','bold')
ylabel('second state','FontWeight','bold')
title('Phase Plane Plot','FontWeight','bold')

function dx=equations(t,x)
dx=zeros(2,1);
dx(1)=x(2);
dx(2)=-x(1)-(all*((x(1)^2)-1)*x(2));

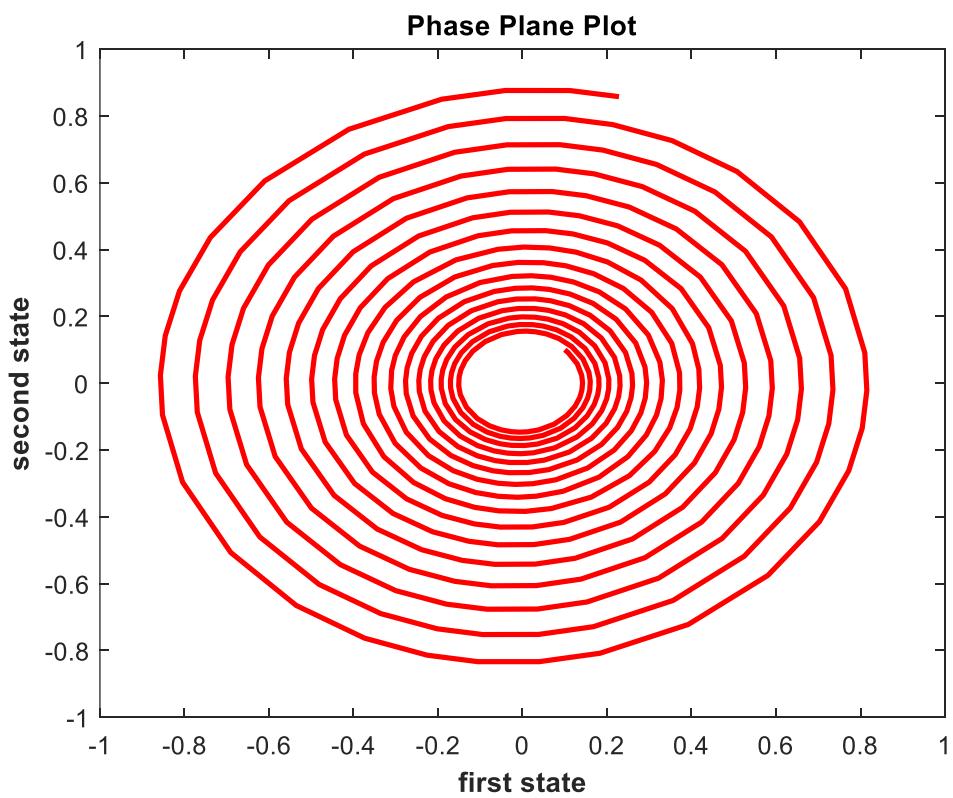
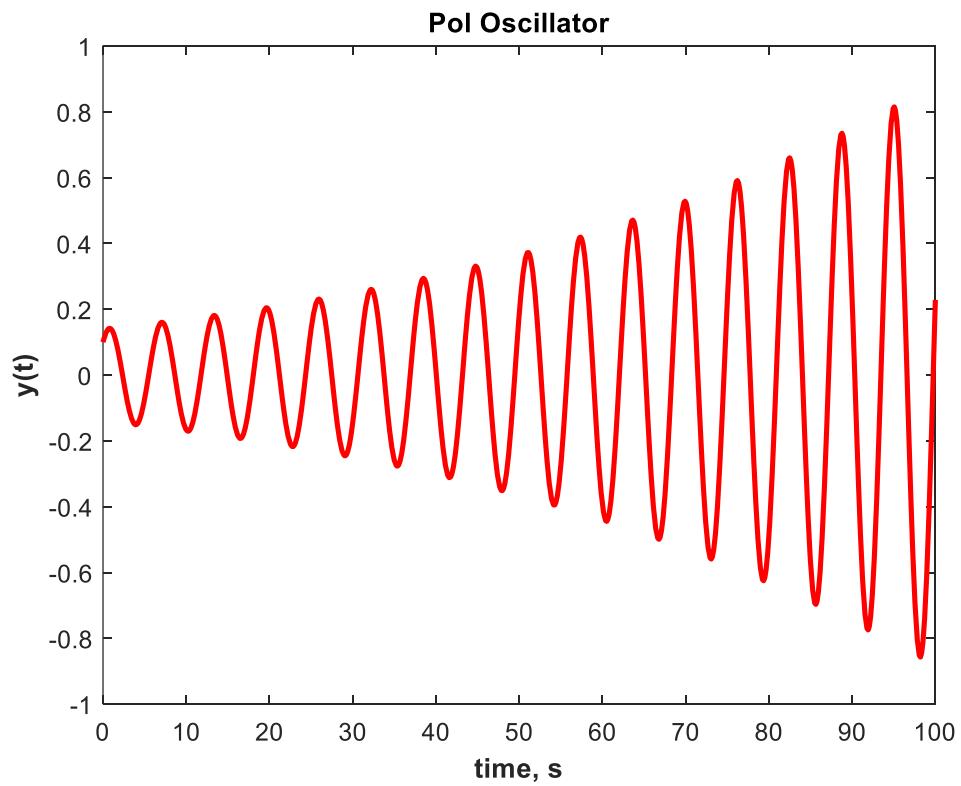
end
end
```

**Part - a:** (For alpha = 0.04)

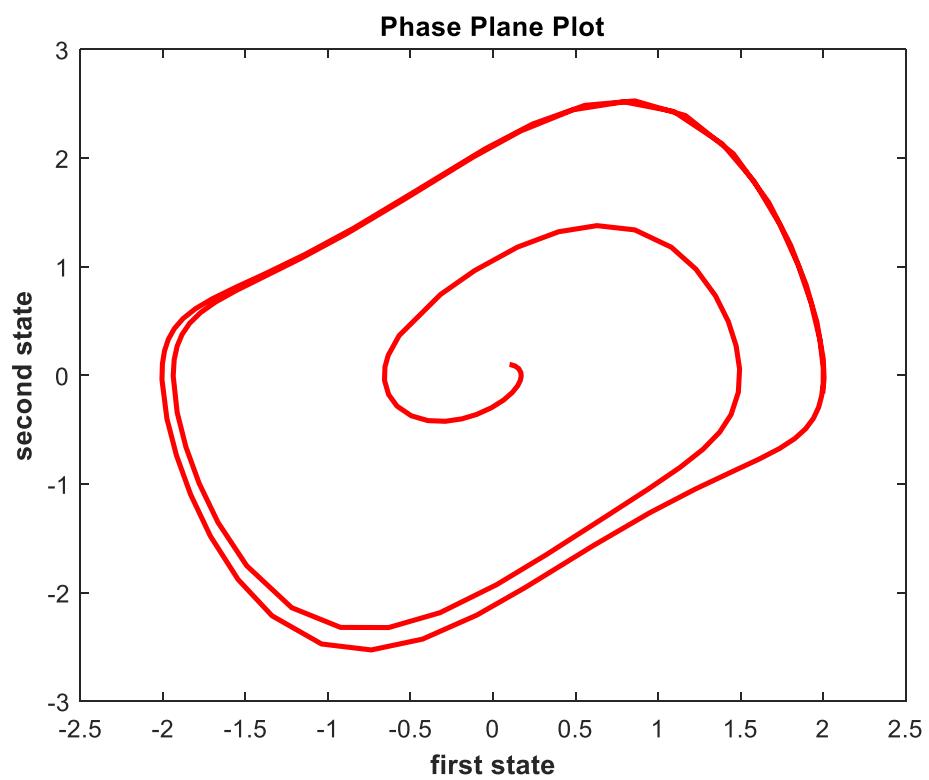
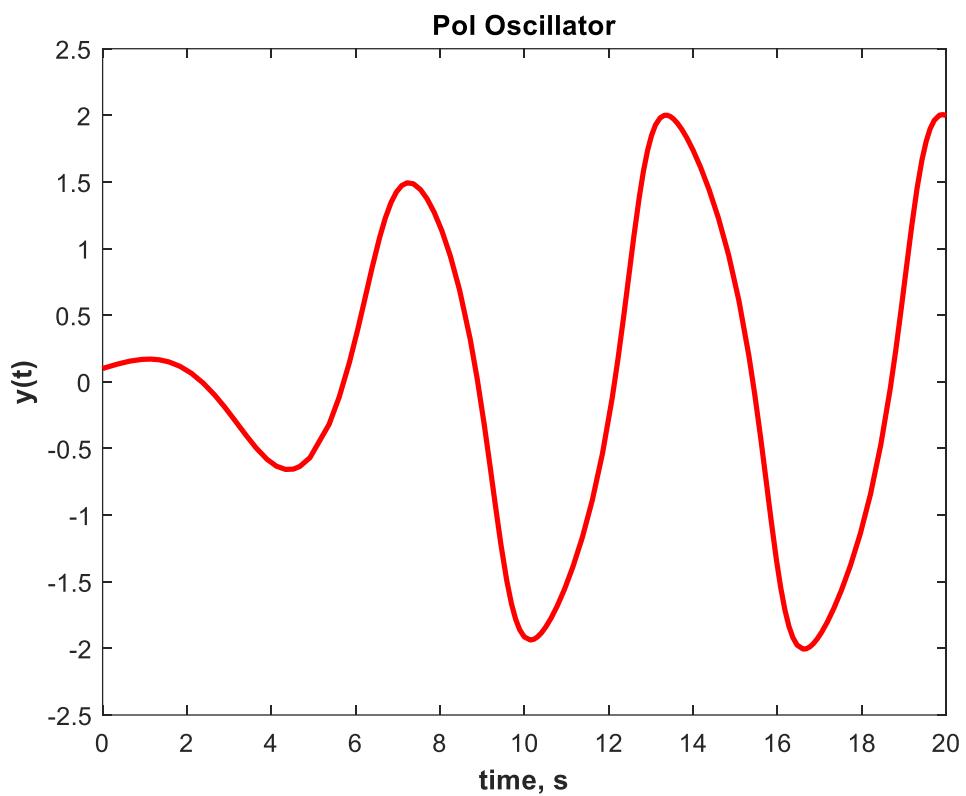
FINAL TIME – 20 SECONDS



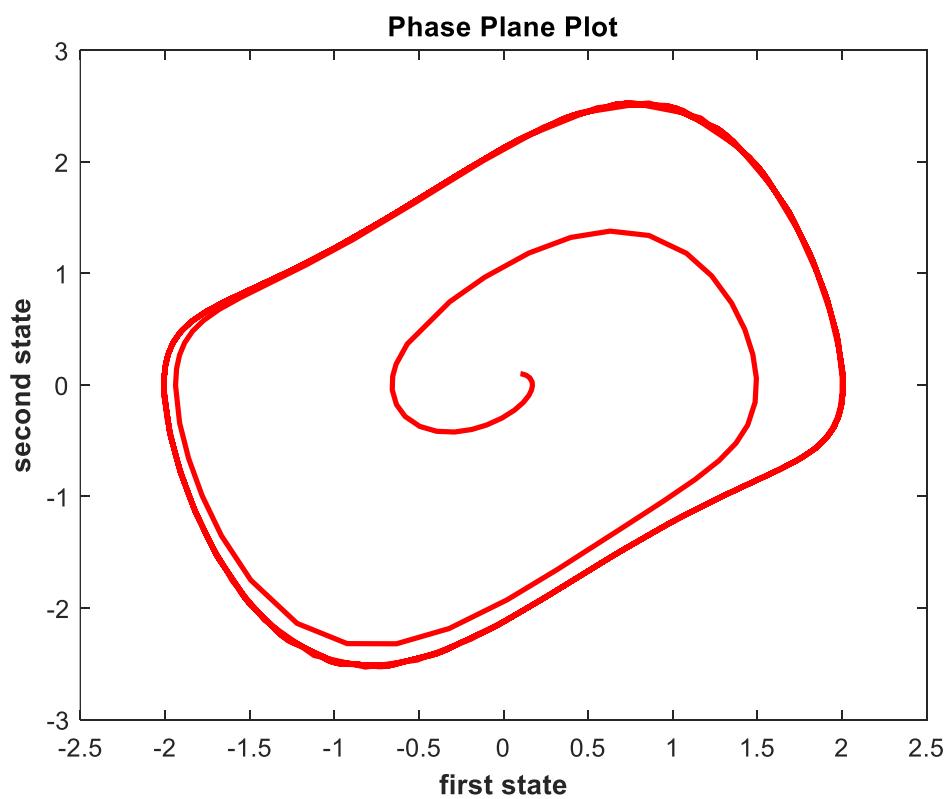
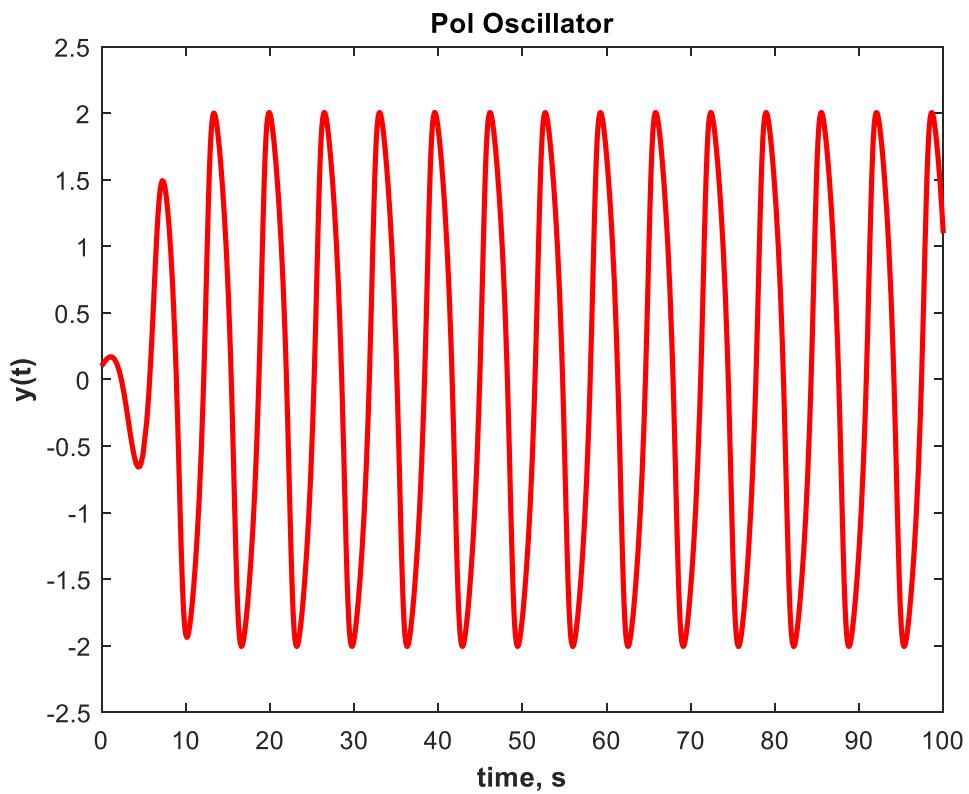
FINAL TIME - 100 SECONDS



**Part - b:** (For alpha = 0.85) (FINAL TIME - 20 SECONDS)



FINAL TIME - 100 SECONDS



## Problem – 2:

### MATLAB Code:

```
function hw1q2
%givens
sig=10; r=28; b=8/3;
tin=[0 150]; %time interval

[t,x]=ode23(@equations,tin,[0.4 0.4 0.4]);

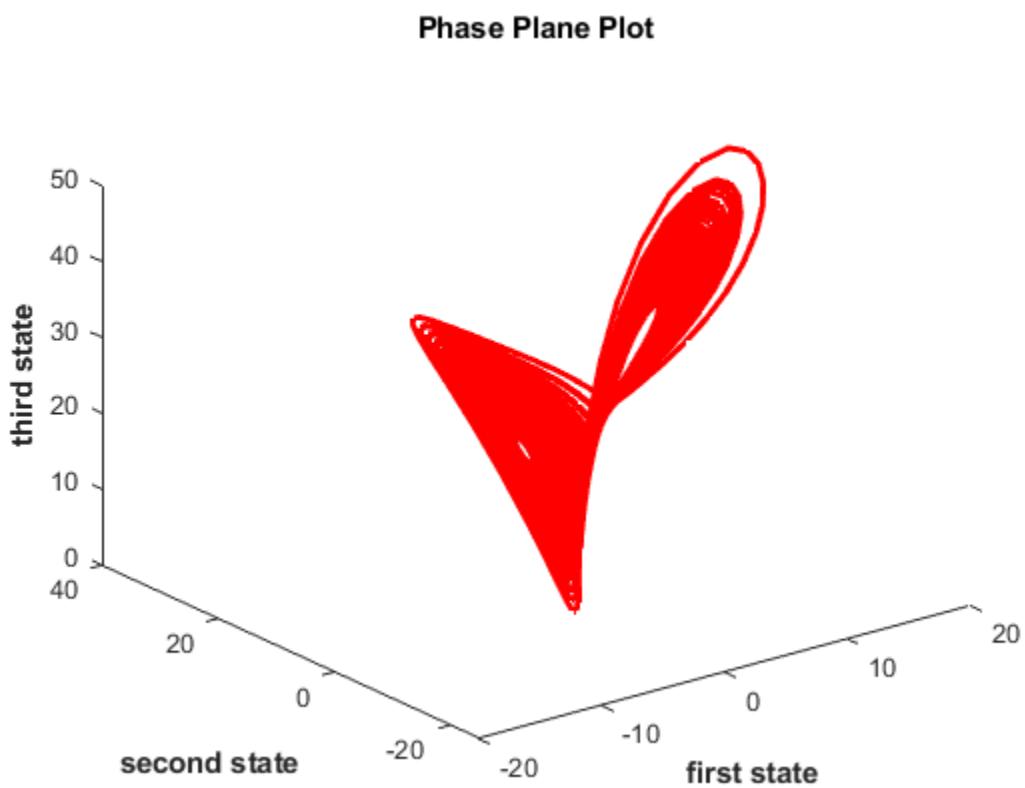
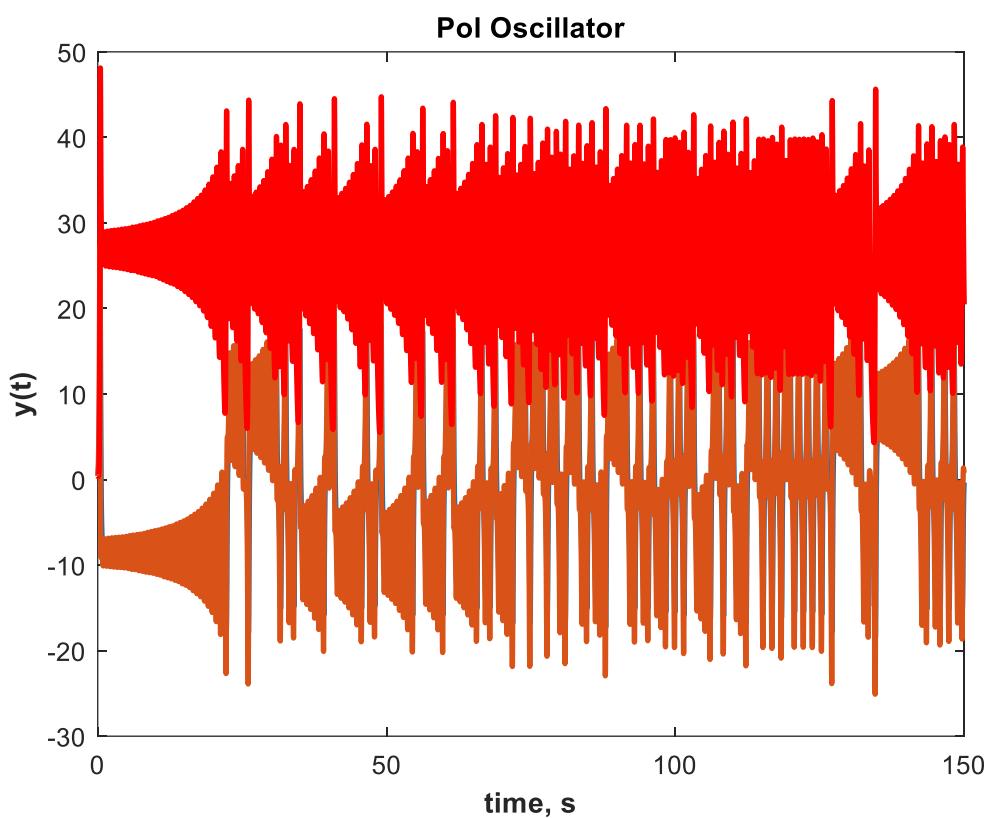
%outputs of interest
y1=x(:,1);
y2=x(:,2);
y3=x(:,3);
y=[y1 y2 y3];

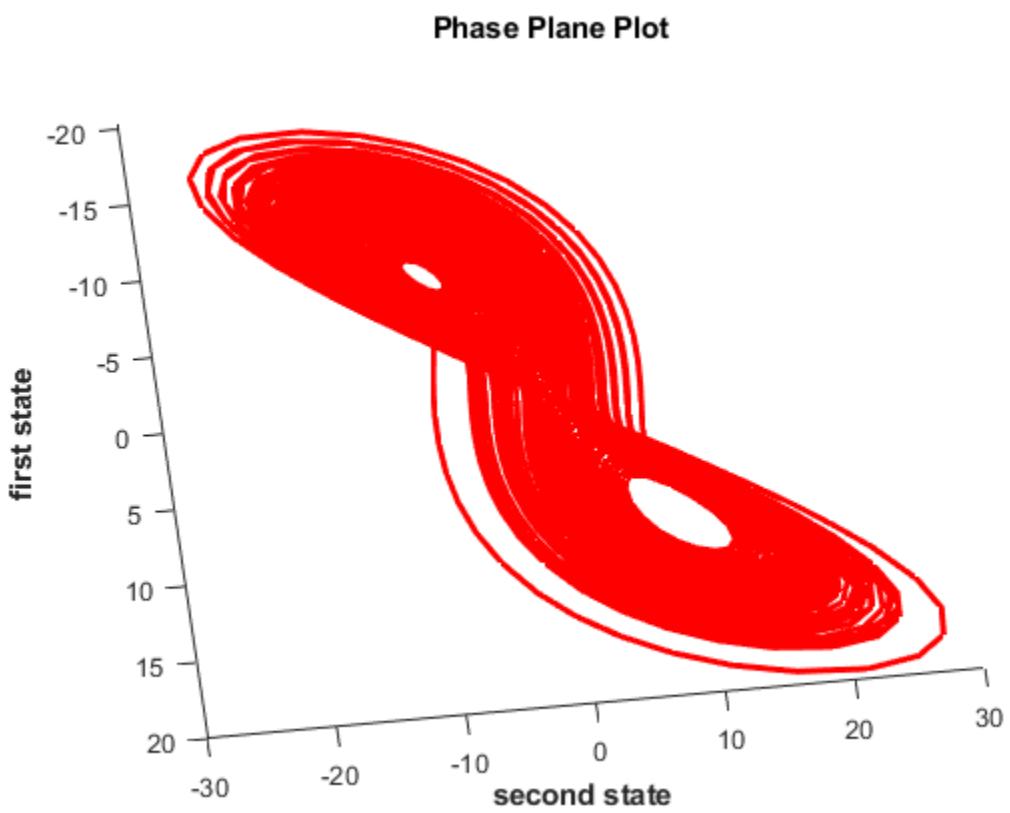
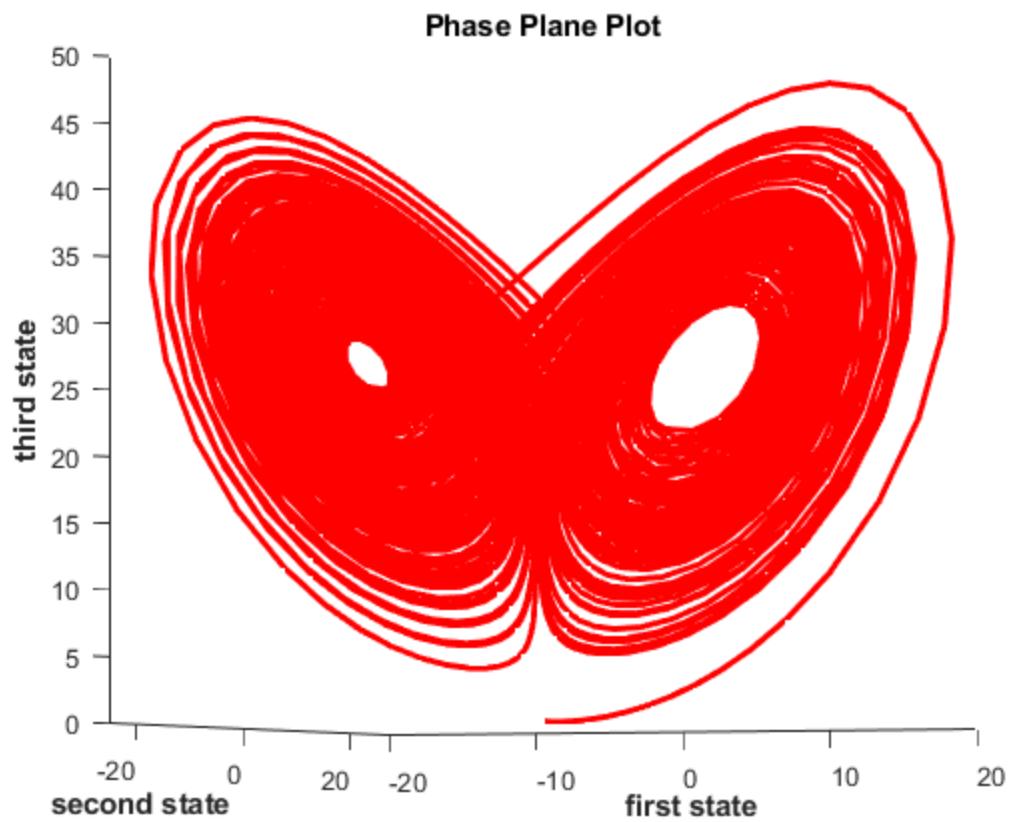
%plot1
figure(1)
plot(t,y1,t,y2,t,y3,'r','LineWidth',2)
xlabel('time, s','FontWeight','bold')
ylabel('y(t)','FontWeight','bold')
title('Pol Oscillator','FontWeight','bold')

%plot2
figure(2)
plot3(y1,y2,y3,'r','LineWidth',2)
xlabel('first state','FontWeight','bold')
ylabel('second state','FontWeight','bold')
zlabel('third state','FontWeight','bold')
title('Phase Plane Plot','FontWeight','bold')

function dx=equations(t,x)
dx=zeros(3,1);
dx(1)=-sig*(x(1)-x(2));
dx(2)=r*x(1)-x(2)-(x(1)*x(3));
dx(3)=-b*x(3)+(x(1)*x(2));

end
end
```



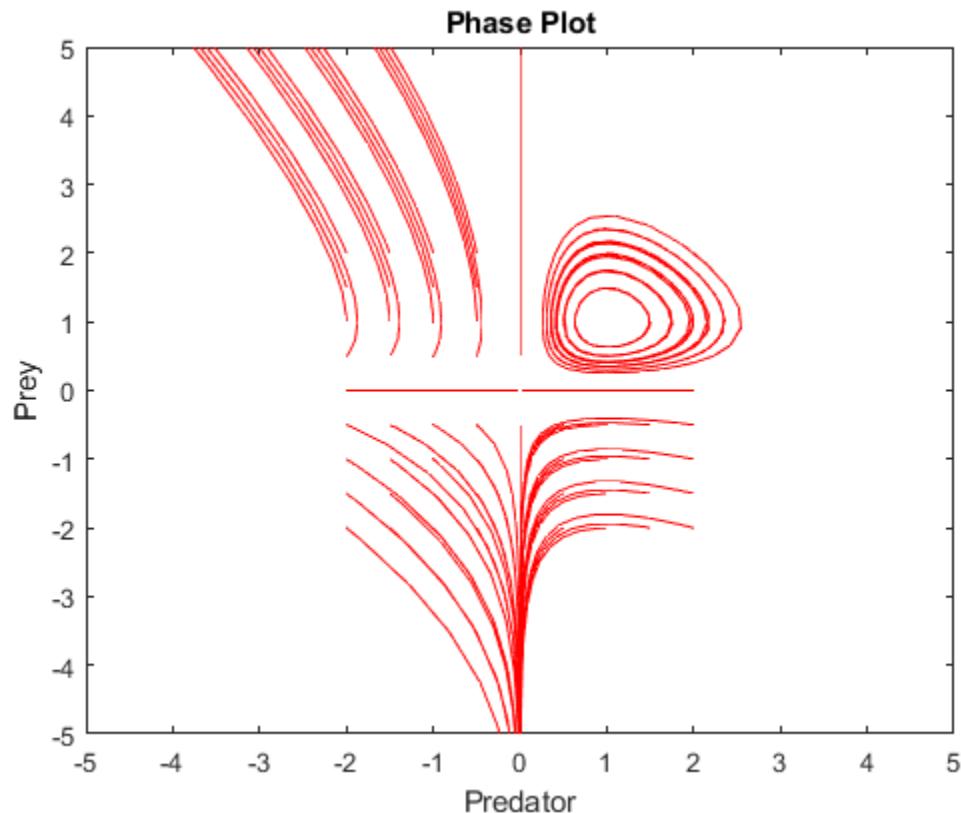


## Problem - 3:

### MATLAB Code:

```
function hw1q3
t0=0; tf=10; k=-2; m=-2;
figure
for k=-2:0.5:2
    for m=-2:0.5:2
        xin=[k;m];
        [t,x]=ode23(@eq, [t0 tf], xin);
        plot(x(:,1),x(:,2), 'r')
        axis([-5 5 -5 5]);
        hold on
    end
end
title('Phase Plot')
xlabel('Predator')
ylabel('Prey')

function dx=eq(t,x)
    dx=zeros(2,1);
    dx(1)=-x(1)+x(1)*x(2);
    dx(2)=x(2)-x(1)*x(2);
end
```



## References:

1. Lorenz, Edward Norton (1963). "Deterministic nonperiodic flow". *Journal of the Atmospheric Sciences*. 20 (2): 130–141 <https://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281963%29020%3C0130%3ADNF%3E2.0.CO%3B2>
  2. Resler, L. M. (2016) 'Edward N Lorenz's 1963 paper, "Deterministic nonperiodic flow", in *Journal of the Atmospheric Sciences*, Vol 20, pages 130–141: Its history and relevance to physical geography', *Progress in Physical Geography: Earth and Environment*, 40(1), pp. 175–180. doi: 10.1177/0309133315623099.
  3. MATLAB Support. Solve Predator-Prey Equations (<https://www.mathworks.com/help/matlab/math/numerical-integration-of-differential-equations.html>). Retrieved September 8, 2020.
-

# Rough Work

PROBLEM - 1:

$$\ddot{y} + \alpha(y^2 - 1)y' + y = 0$$

$$y = y_1, \quad y' = \dot{y}_1 = y_2 \quad \text{--- } ①$$

$$\ddot{y} = \dot{y}_2 = -[\alpha(y_1^2 - 1)y_2 + y_1] \quad \text{--- } ②$$

Equations used as given in the question  
for PROBLEM - 2 and PROBLEM - 3.

## **EE 5322 Homework 2**

### **Stock Market Time Series Analysis**

Download the file for this problem from the homework assignment home page.

The closing price for the NASDAQ tracking stock NVDA is given as an Excel file.

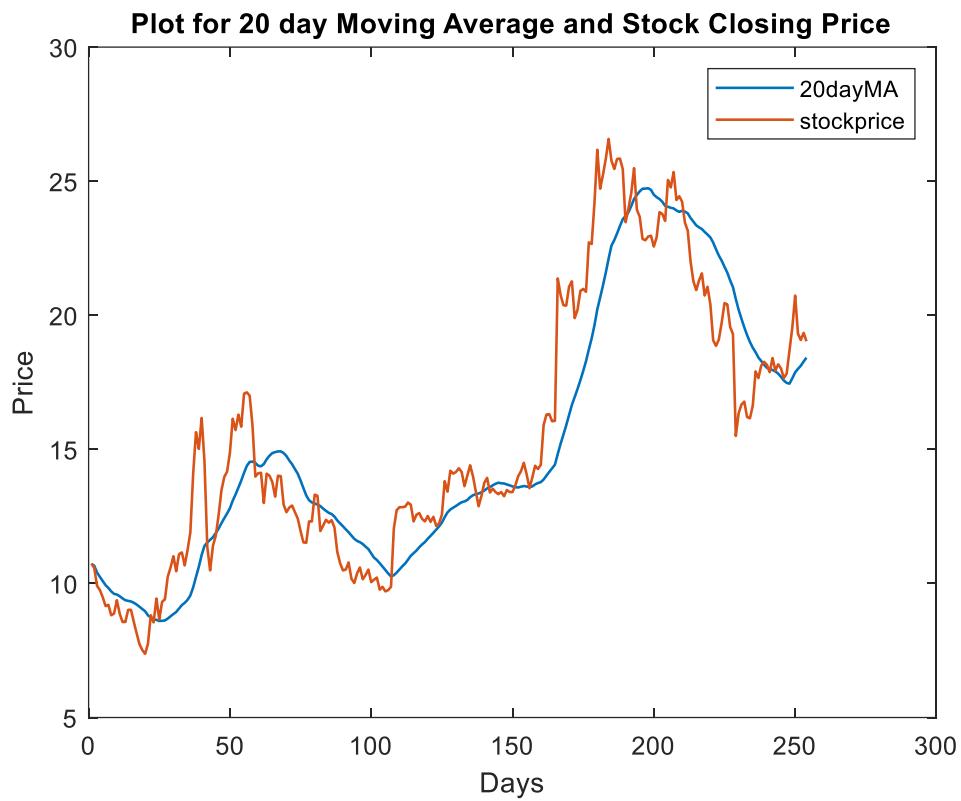
Note there are 254 trading days in the year. There are about 22 trading days in a month. Therefore, for trading on a monthly time scale, one considers a 20-day time window. This allows one to capture many motions of the stock while not spending too much in broker's fees by churning the stock. On-line trades now run about \$3 per transaction.

1. a. Compute the 20 day MA. Plot on the same figure as the stock closing price.  
b. Plot the stock minus the 20 day MA.  
c. Compute and plot the 20 day moving sample variance.  
d. On the same figure, plot the stock closing price, the 20 day MA, and  
the MA plus three times the 20 day standard deviation  
the MA minus three times the 20 day standard deviation.

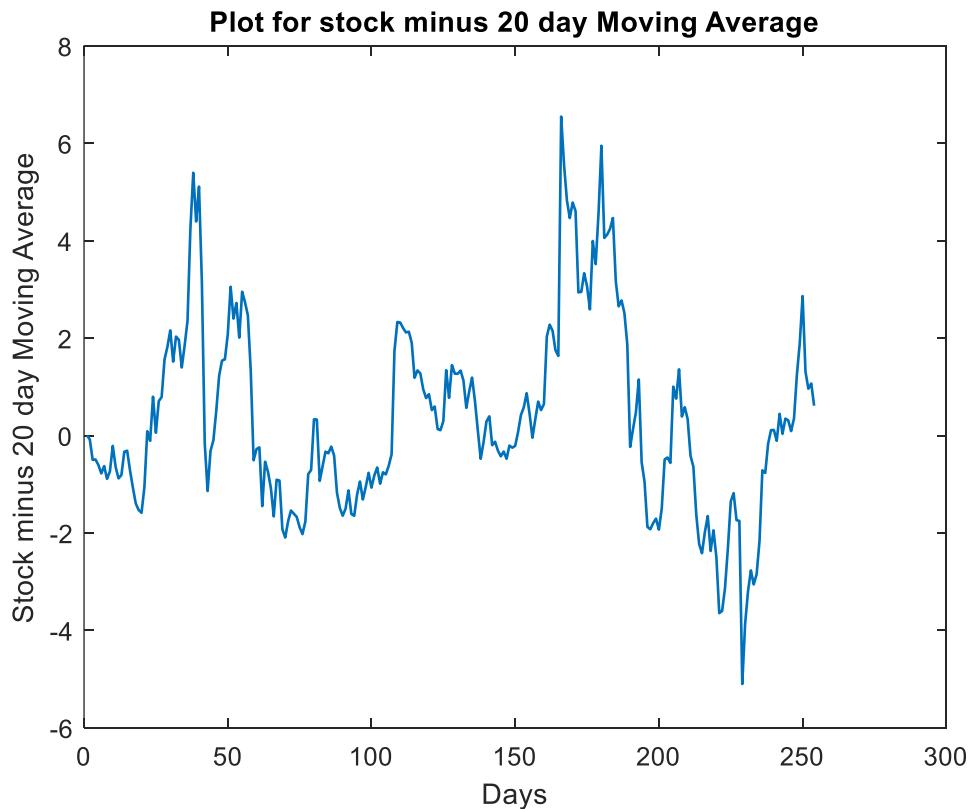
The last two lines are known as the Bollinger Bands, after John Bollinger.

2. a. Compute and plot the 20 day moving skew.  
b. Compute and plot the 20 day moving kurtosis.  
c. Can you use these statistics to find a leading indicator for movements in the stock?  
i.e. how can we predict using statistics when the stock is about to break its trend (change its pattern)?
3. a. Compute and plot the overall autocorrelation.  
b. Compute and plot the overall autocovariance.
4. Any news about predicting movements in this stock?  
Is it time to buy this stock now?

### Problem - 1

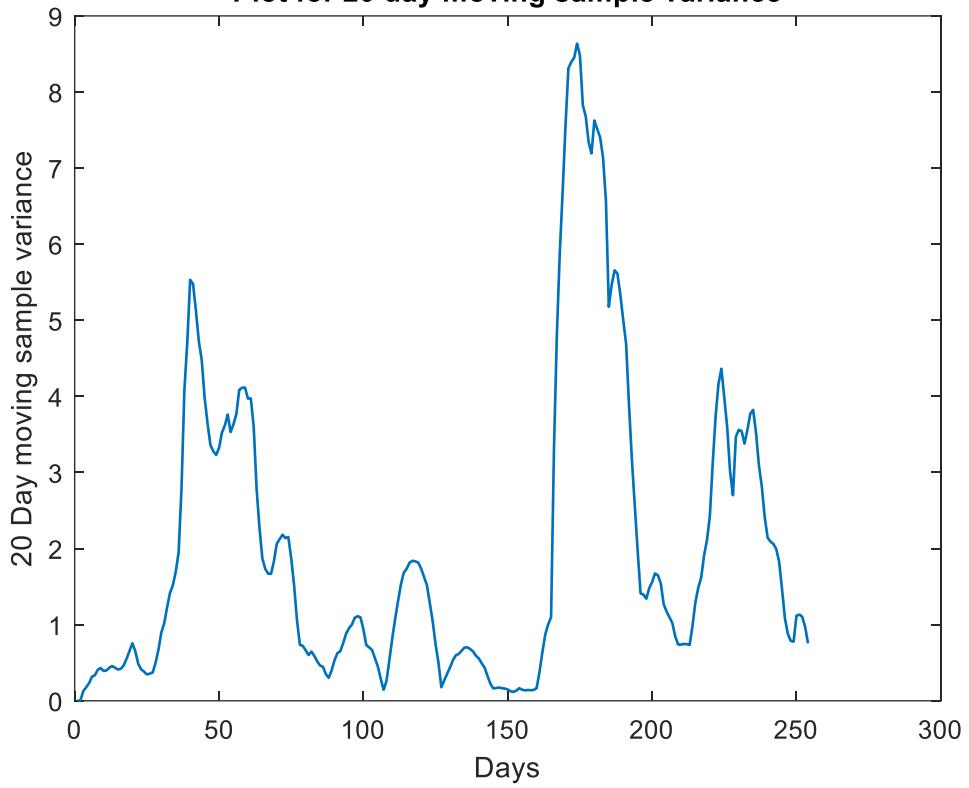


a.



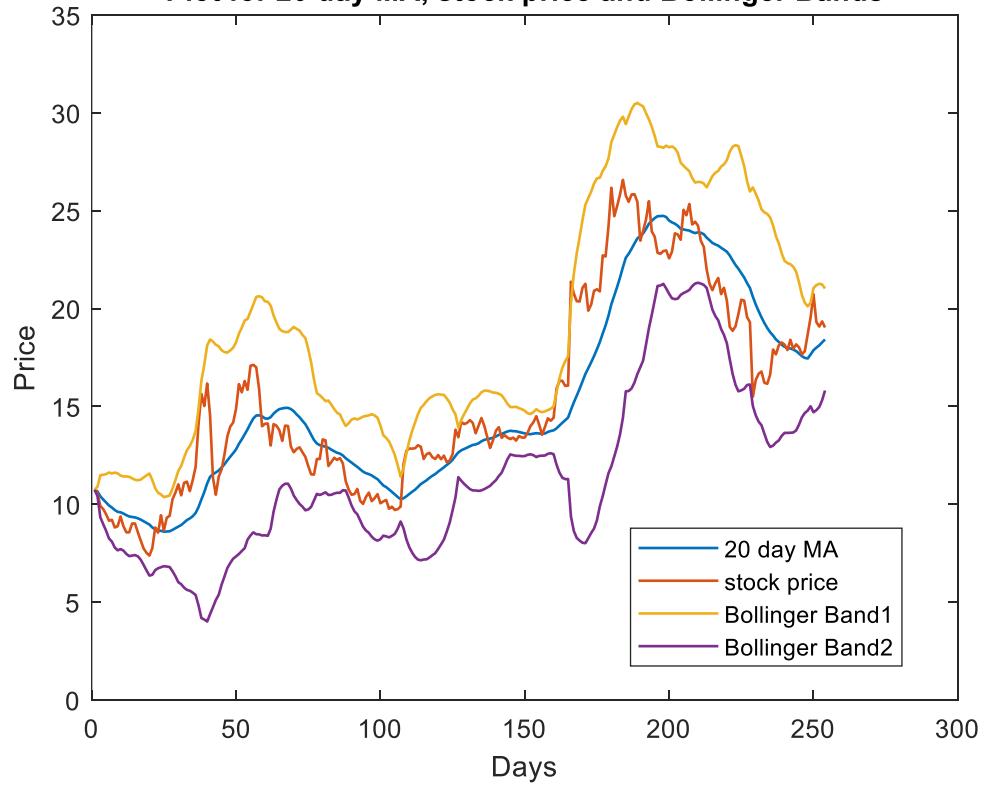
b.

**Plot for 20 day moving sample variance**



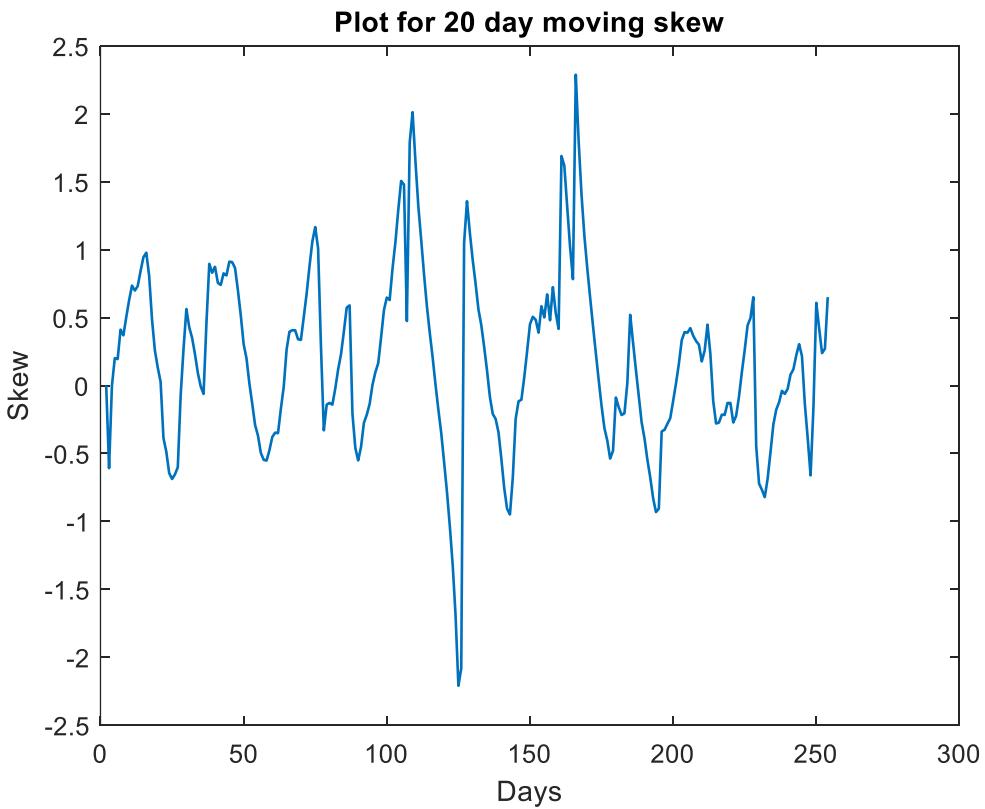
c.

**Plot for 20 day MA, stock price and Bollinger Bands**

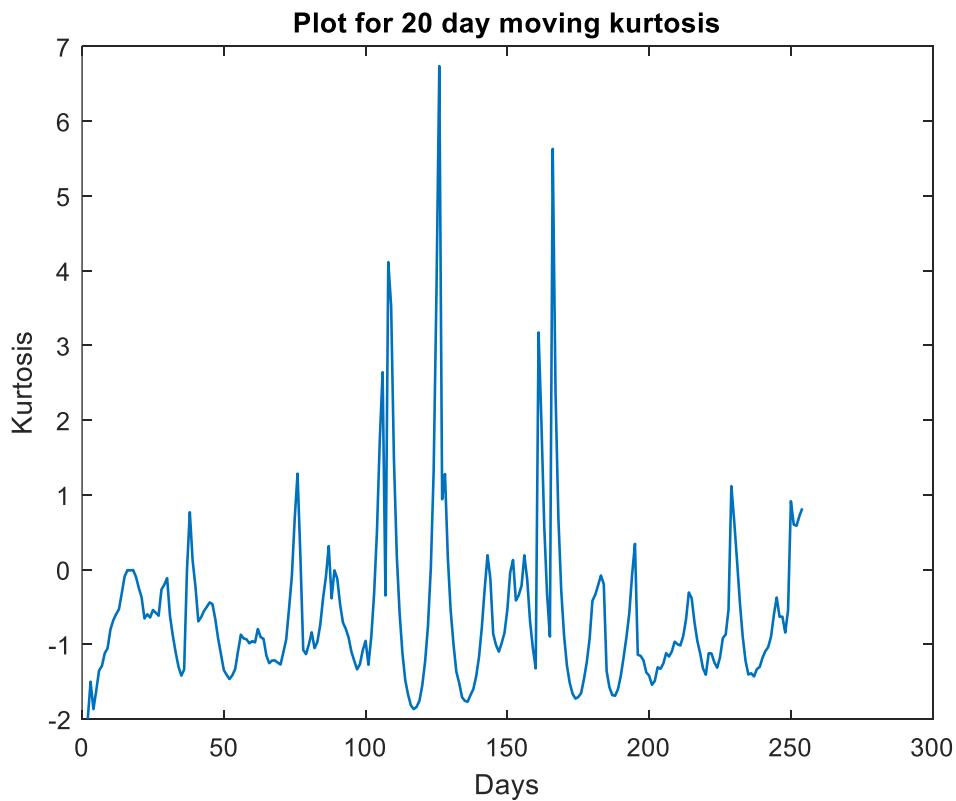


d.

## Problem - 2



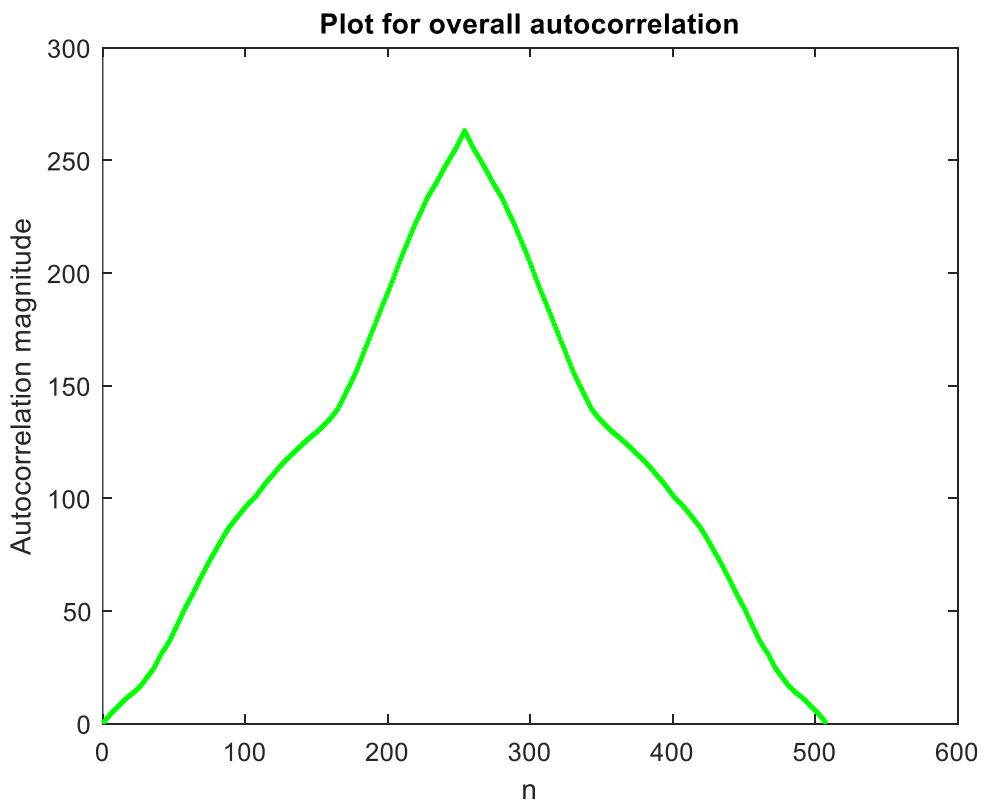
a.



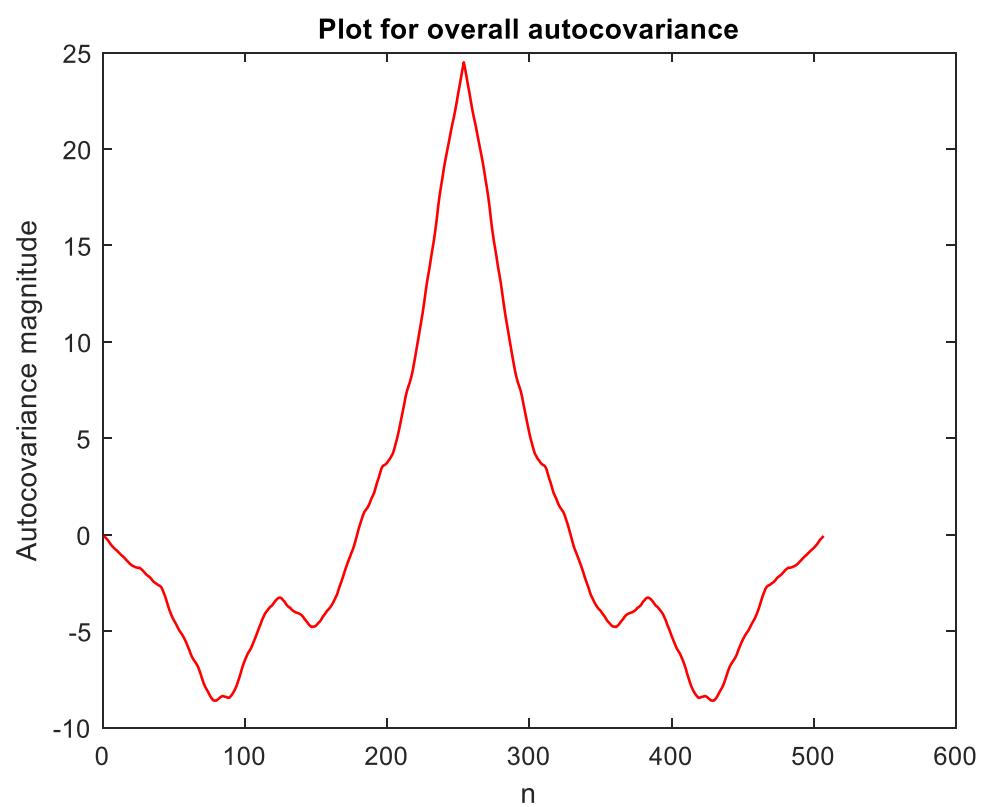
b.

c. Yes, we can use these statistics to indicate the changing patterns or break in trends of the stock. The stock minus moving average curve has a moving average of 0. When the stock goes below that mean, it's an indicator to buy as the prices will likely rise above the mean at some point whereas when it goes above the mean, it's an indicator to sell as the prices will fall at some point (time to buy more stocks). This is also an indicator of the rapidness of stock change as larger magnitudes in either direction indicate a more significant fluctuation. However, it's delayed by 20 day delay so some extrapolation is required. Bollinger Bands (MA +3\*standard deviation and MA -3\*standard deviation) are helpful to assess that when the stock price goes up and crosses the MA line, it normally jumps to the upper Bollinger Band and vice versa. When the Bollinger Bands come close to each other, it indicates that the stock price is about to change rapidly. Skew is an indicator of stock price volatility and can be used to assess risk before investment. A stock with high skew such as the Bitcoin (BTC) is highly volatile and might not be suitable for some types of stock market trading. Kurtosis tells about the uncertainty of the stock price where larger peaks indicate more rapid changes. Accounting for skew and kurtosis together allows us to estimate the direction of the next stock change.

**Problem - 3**



a.



b.

## **Problem - 4**

Based on the given data, it seems like the stock price is falling which makes it a bad time to buy the stock. This is inferred from observing that the stock price has already touched the upper Bollinger Band after crossing the moving average and is coming back towards the moving average and is likely to cross the moving average again on its way down. The stock minus the 20 day moving average is slightly above 0 so its values are not of significant help although a person only depending on this measure may want to close the deal immediately as it looks it's heading below the 0 line.

However, this data is lagging by 20 days and the overall autocorrelation and overall autocovariance indicate that the relation in stock prices is very low. So let's only focus on the skew and kurtosis. The kurtosis shows that this is a small peak indicating low level of fluctuation and thus a lower level of risk. The skew plot confirms this as it shows a lower peak in the positive direction. Since the skew is in the positive direction and rising, it may go higher but it could also touch a temporary ceiling/resistance and drop immediately. The temporary ceiling/resistance is calculated based on previous peaks and the trading window. Based on the ceiling metric and available data, if you are a quarterly trader then it may not be a good time to buy the stock. However, if you are a trader who can comfortably invest money for more than that duration then it's a good time to buy which can also be supported by just looking at the 20 day moving average.

Also, this data is not qualitative data. Qualitative data in terms of stock trading is the knowledge of the organization's plans and processes. The company may have reported a loss in the last fiscal quarter or laid off many workers recently this will reasonably prompt many traders to sell their stocks. However, a person doing a deeper analysis may realize that this may be an impact of their recent expenditure on assets or even a shift towards better technology. Knowing about the organization's long term strategy is highly effective for making profits when trading in the stock market. Therefore qualitative analysis is also important. I'm not saying this, Warren Buffet did.

## MATLAB function for computation and plots

```
function hw2
filename='Hwlndva.xls';
T=readtable(filename);
days=T{1:254,1};
price=T{1:254,2};
t=1:1:507;
cors=zeros(507,1);
global mean
mean=(sum(price(1:254,1)))/254;
global calls

%computing moving average, stock minus the moving average
%moving sample variance and Bollinger Bands
for i=1:1:254
    calls=zeros(254,1);
    if i<20
        ma(i,1)=(sum(price(1:i,1)))/i;
        rang(i,1)=(price(i,1)-ma(i,1));
        for n=1:i
            calls(i,1)=calls(i,1)+((price(n,1)-ma(i,1))^2);
        end
        var(i,1)=calls(i,1)/i;
    else
        ma(i,1)=(sum(price(i-19:i,1))/20;
        rang(i,1)=(price(i,1)-ma(i,1));
        for n=(i-19):i
            calls(i,1)=calls(i,1)+((price(n,1)-ma(i,1))^2);
        end
        var(i,1)=calls(i,1)/20;
    end
    sig(i,1)=sqrt(var(i,1));
    bolup(i,1)=ma(i,1)+(3*sig(i,1));
    boldn(i,1)=ma(i,1)-(3*sig(i,1));
end

%computing the 20 day moving skew and kurtosis
for k=1:1:254
    calls=zeros(254,1);
    if k<20
        for n=1:k
            calls(k,1)=calls(k,1)+((price(n,1)-ma(k,1))^3);
        end
        skew(k,1)=calls(k,1)/(k*((sig(k,1))^3));
        calls=zeros(254,1);
        for n=1:k
            calls(k,1)=calls(k,1)+((price(n,1)-ma(k,1))^4);
        end
        kurt(k,1)=(calls(k,1)/(k*((sig(k,1))^4))-3;
    else
        for n=(k-19):k
            calls(k,1)=calls(k,1)+((price(n,1)-ma(k,1))^3);
        end
        skew(k,1)=calls(k,1)/(20*((sig(k,1))^3));
        calls=zeros(254,1);
```

```

        for n=(k-19):k
            calls(k,1)=calls(k,1)+((price(n,1)-ma(k,1))^4);
        end
        kurt(k,1)=(calls(k,1)/(20*((sig(k,1))^4)))-3;
    end
end

%computing the overall correlation and autocovariance
for n=-253:1:253
    init=0;
    call=0;
    for k=1:1:254
        if k+n>0 && k+n<=254
            init=init+price(k,1)*price(k+n,1);
            call=call+(price(k,1)-mean).* (price(k+n,1)-mean);
        end
    cors(n+254,1)=init/254;
    covs(n+254,1)=call/254;
    end
end

%plot for stock price and 20 day moving average
figure(1)
plot(days(:,1),ma(:,1),days(:,1),price(:,1))

%plot for stock minus the 20 day moving average
figure(2)
plot(days(:,1),rang(:,1))

%plot for 20 day moving sample variance
figure(3)
plot(days(:,1),var(:,1))

%plot for stock closing price, 20 day moving average and Bollinger Bands
figure(4)
plot(days(:,1),ma(:,1),days(:,1),price(:,1),days(:,1),bolup(:,1),days(:,1),boldn(:,1))

%plot for 20 day moving skew
figure(5)
plot(days(:,1),skew(:,1))

%plot for 20 day moving kurtosis
figure(6)
plot(days(:,1),kurt(:,1))

%plot for overall autocorrelation
figure(7)
plot(1:length(cors(:,1)),cors,'g.')

%plot for overall autocovariance
figure(8)
plot(1:length(covs),covs(:,1),'r')

end

```

## EE 5322 Homework 3

### DFT Analysis

Download the files for this problem from the homework assignment home page.

#### 1. Digital Speech Processing

(The frequencies here are about 1/10 the actual values for ease of processing using MATLAB.)

In speech, the vowels are characterized by three main frequencies known as formants. The first two formants for each vowel in English are as follows:

vowel	Formant 1 (Hz)	Formant 2 (Hz)
A	70	110
E	50	180
I	40	200
O	60	80
U	30	80

The data for this homework contains an 8 sec speech signal that contains some vowels. The sampling period is 1 msec = 0.001 sec. Chop the signal into eight bins of length 1 sec. In each bin, do the FFT (using  $N=$  a power of two).

Determine which vowels occur and when. Finally, plot the DFT vs. time as a 3-D plot.

#### 2. Machinery Monitoring

An induction motor drive has a base rotation frequency of  $f_0 = 50$  Hz, a frequency of  $3f_0$  due to a three-bladed fan, and a component at  $4f_0$  due to a 4:1 gearbox. When a certain pinion gear wears badly enough, a prominent frequency component of 277 Hz appears. Soon after that, the amplitude of the frequency component at  $4f_0$  significantly increases due to the failure of a gear tooth.

In the 6 sec data file, the sampling period is 1 msec. Find out when the two anomaly failure events occur. Plot the DFT vs. time as a 3-D plot. Use moving average window for the DFT of length  $\frac{1}{2}$  sec. Use  $N=$  a power of two.

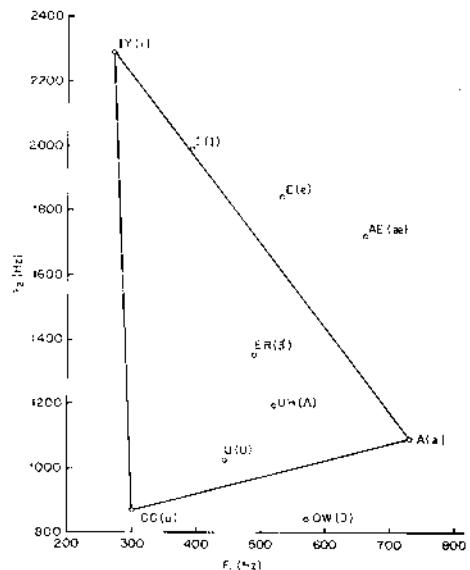


Fig. 3.5 The vowel triangle.

from Rabiner and Schafer, 1978

**ATUL SHROTRIYA**

**UTA ID - 1001812437**

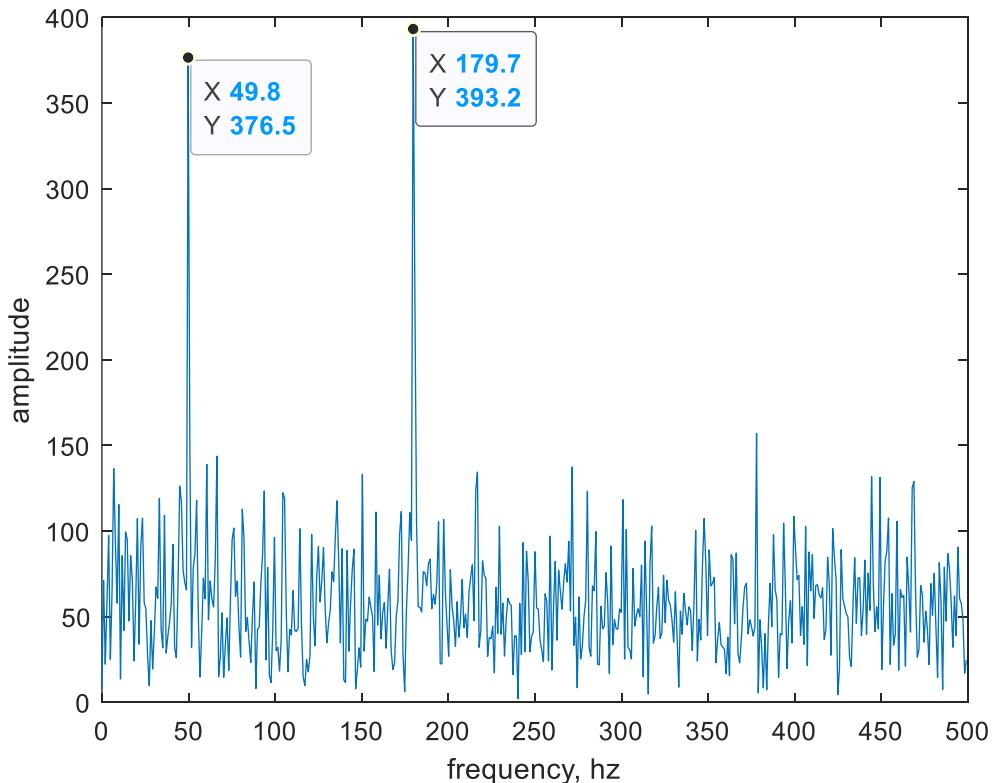
**Homework - 3**

**Intelligent Control Systems**

## Problem - 1:

The signal is first divided into 8 separate bins as asked. This is so that the total signal of 8 seconds can be divided into 1 second each. Following this, Discrete Fourier Transform (DFT) is done on each bin using N=1024.

1<sup>st</sup> second:

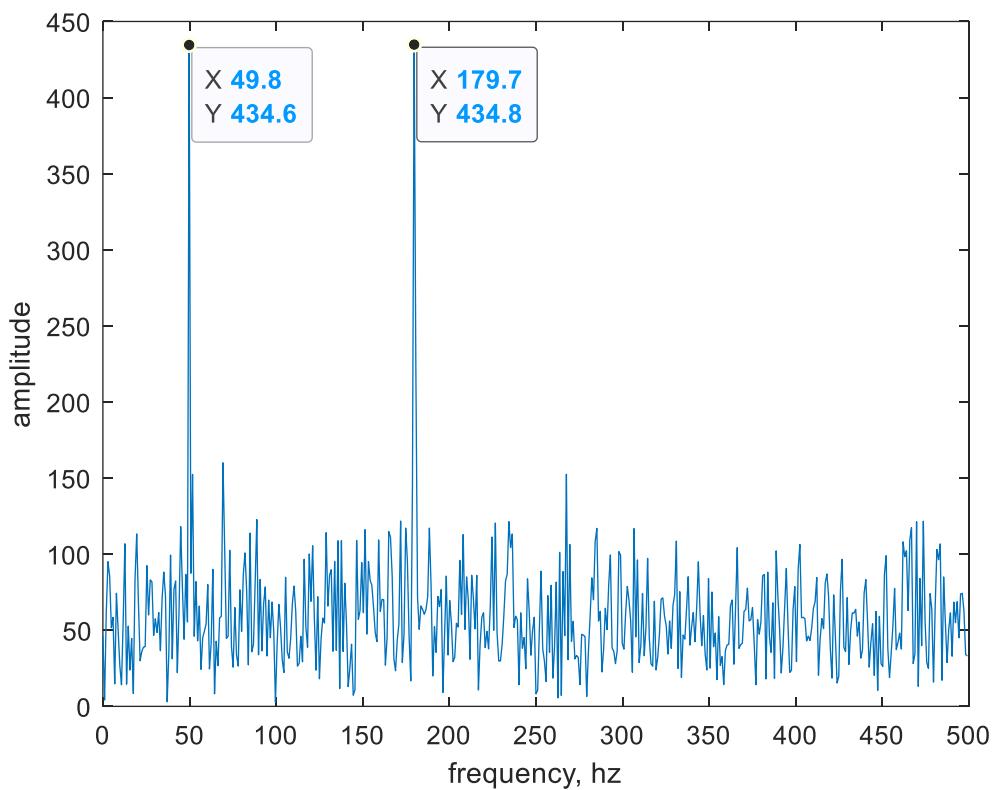


There's no need to magnify the signal. It can be seen that the first formant is near 50 Hz and second formant is near 180 Hz. Therefore this is an 'E'

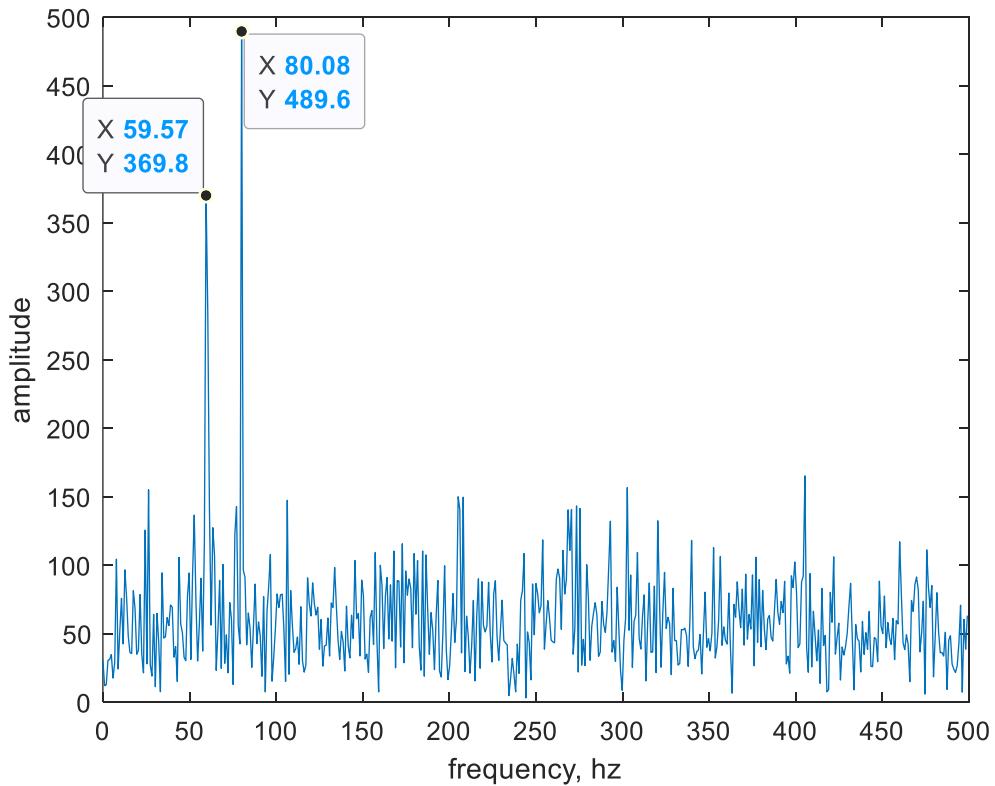
I could have magnified it but it seems a bit redundant to me and I'm in a slight hurry to magnify each plot. Furthermore, it would be cumbersome for you to check so many plots. Also, I have not included explanation for each of the next vowels because it's readily apparent by looking at the provided table.

For calculation of DFT and PSD in the final parts, the time window used is of 1000 samples. And N is taken as 1024 ( $2^{11}$ )

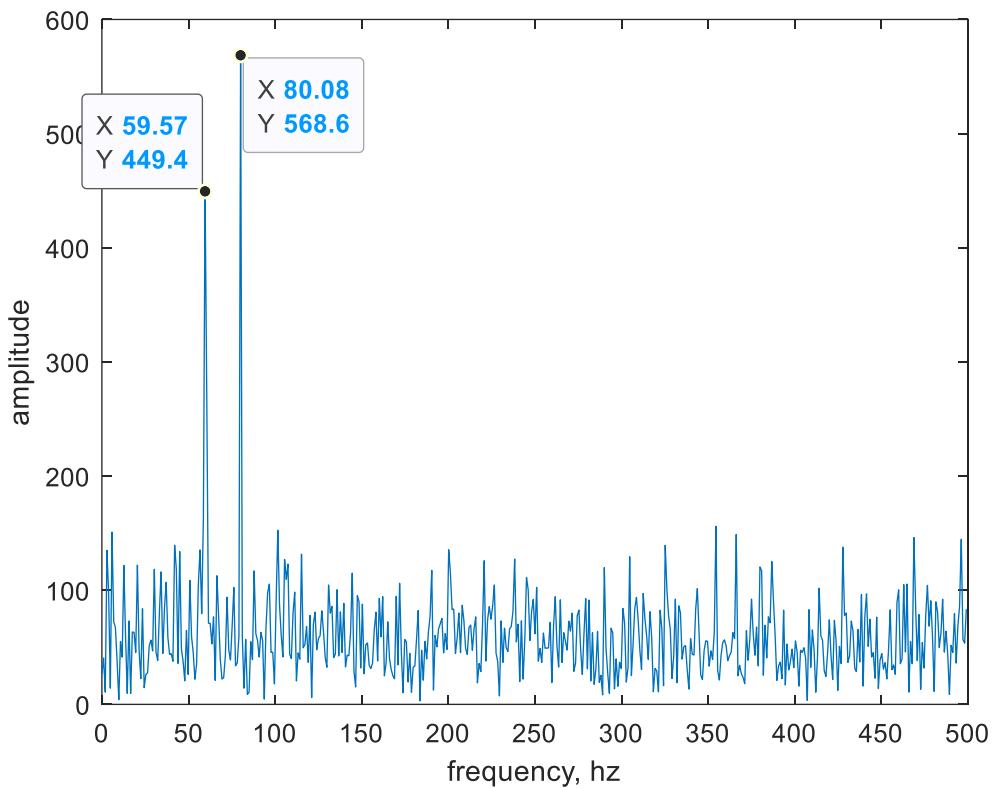
**2<sup>nd</sup> second - This is also an ‘E’**



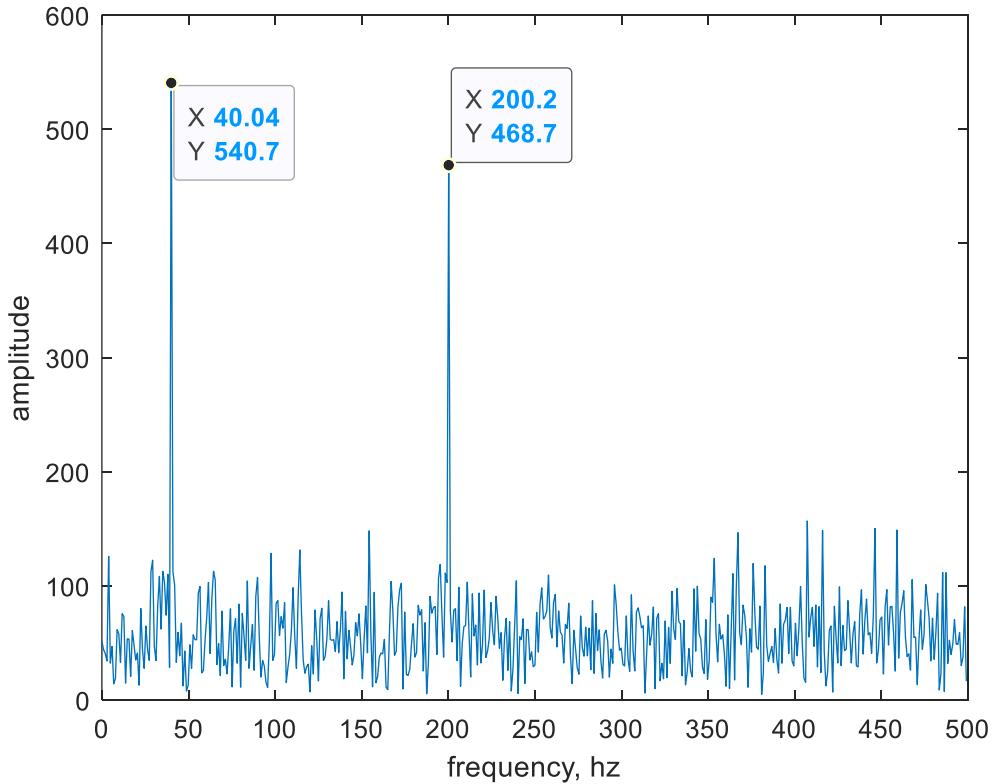
**3<sup>rd</sup> second - This is an ‘O’ (first formant 60 hz and second formant 80 hz)**



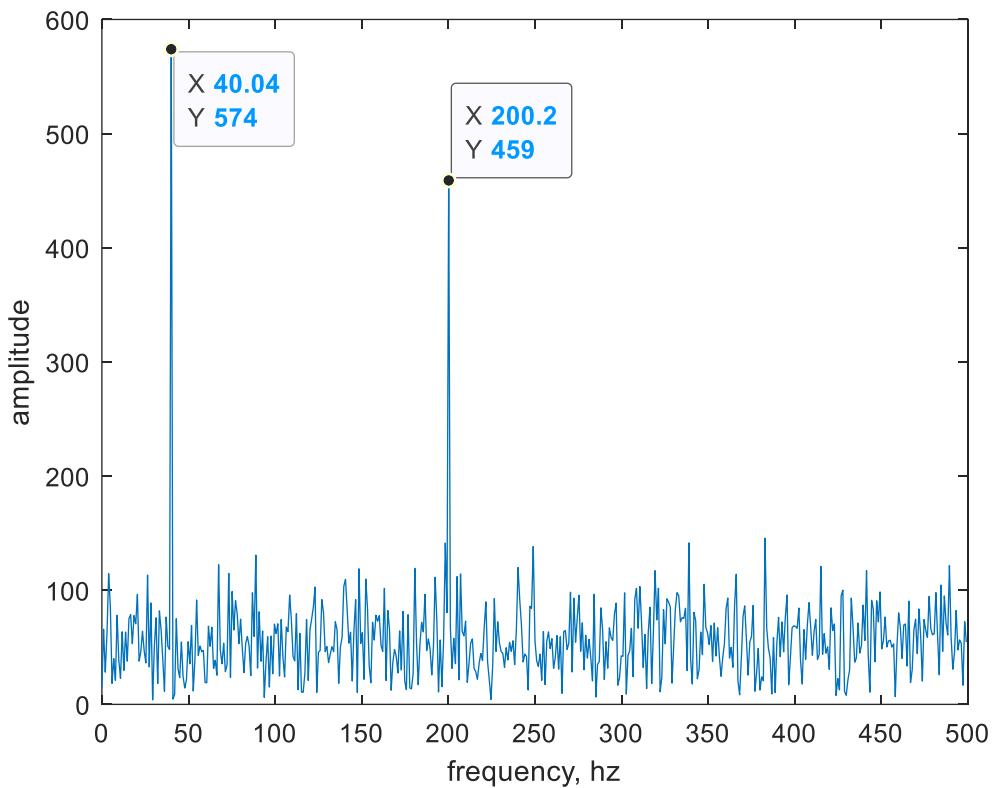
**4<sup>th</sup> second - This is also an 'O'**



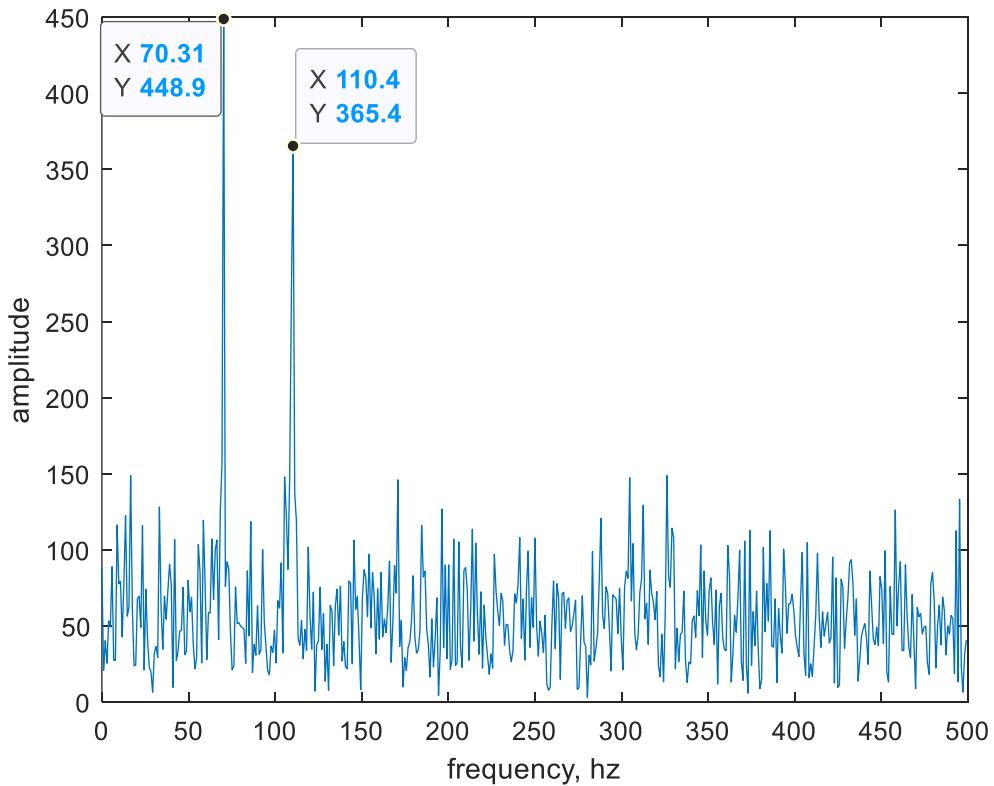
**5<sup>th</sup> second - This is an 'I' (first formant 40 Hz and second formant 200 Hz)**



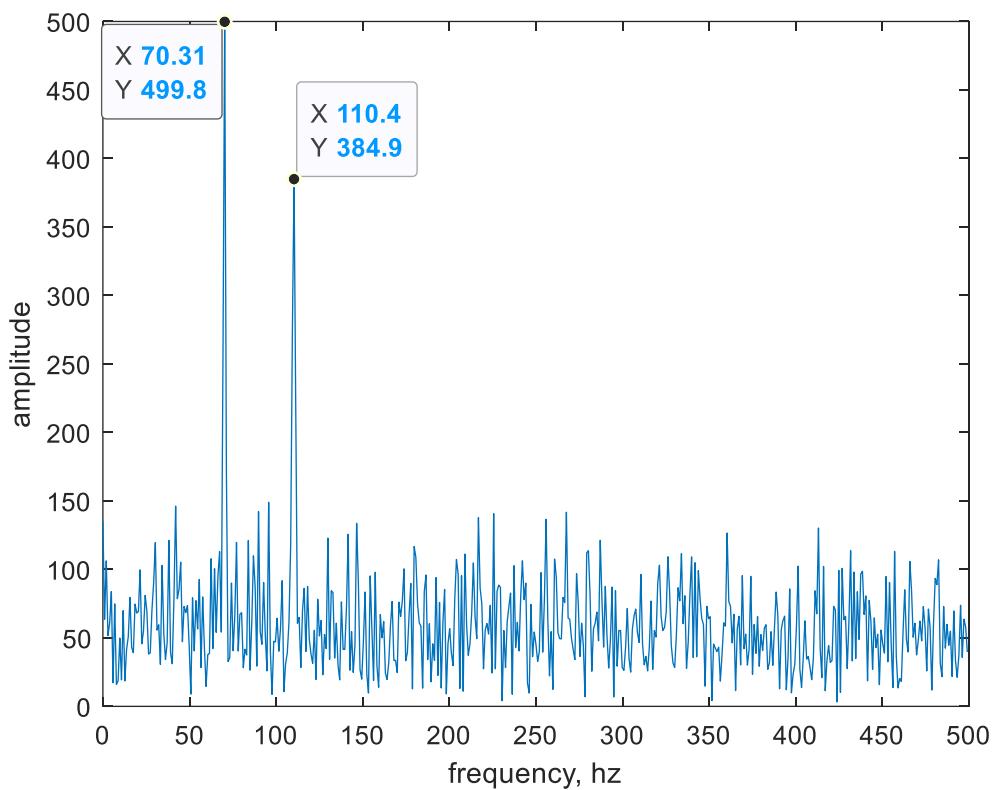
**6<sup>th</sup> second - This is also an 'I'**



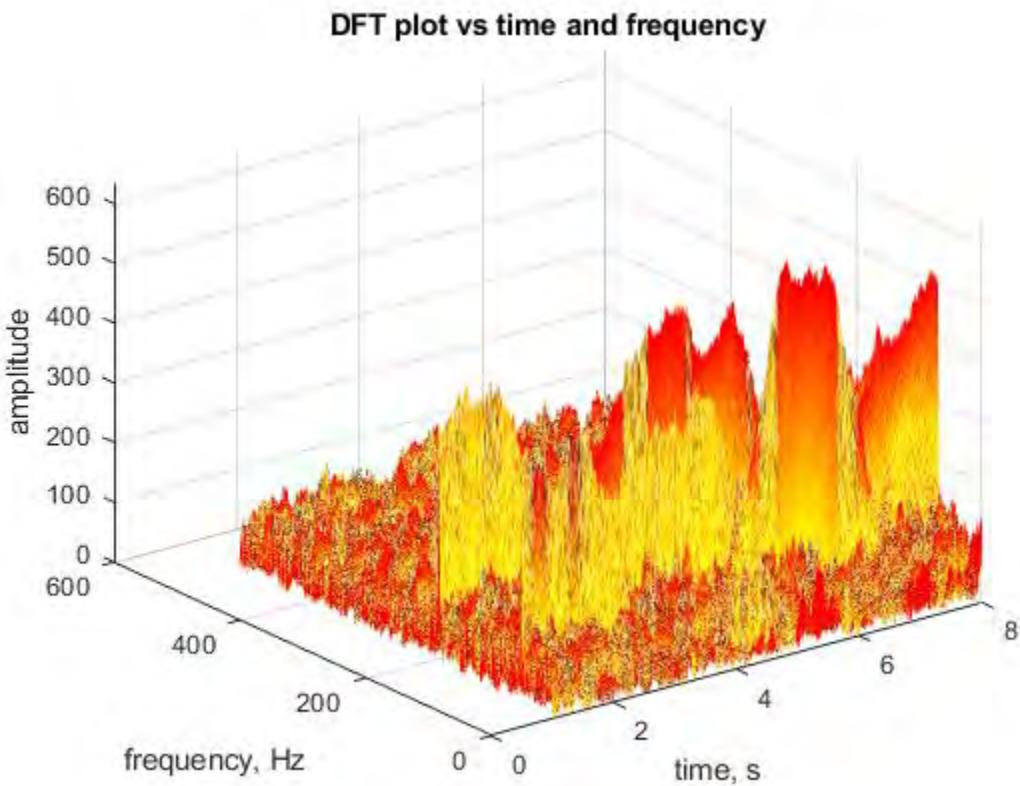
**7<sup>th</sup> second - This is an 'A' (first formant is 70 Hz and second formant is 110 Hz)**



8<sup>th</sup> second - This is also an 'A'

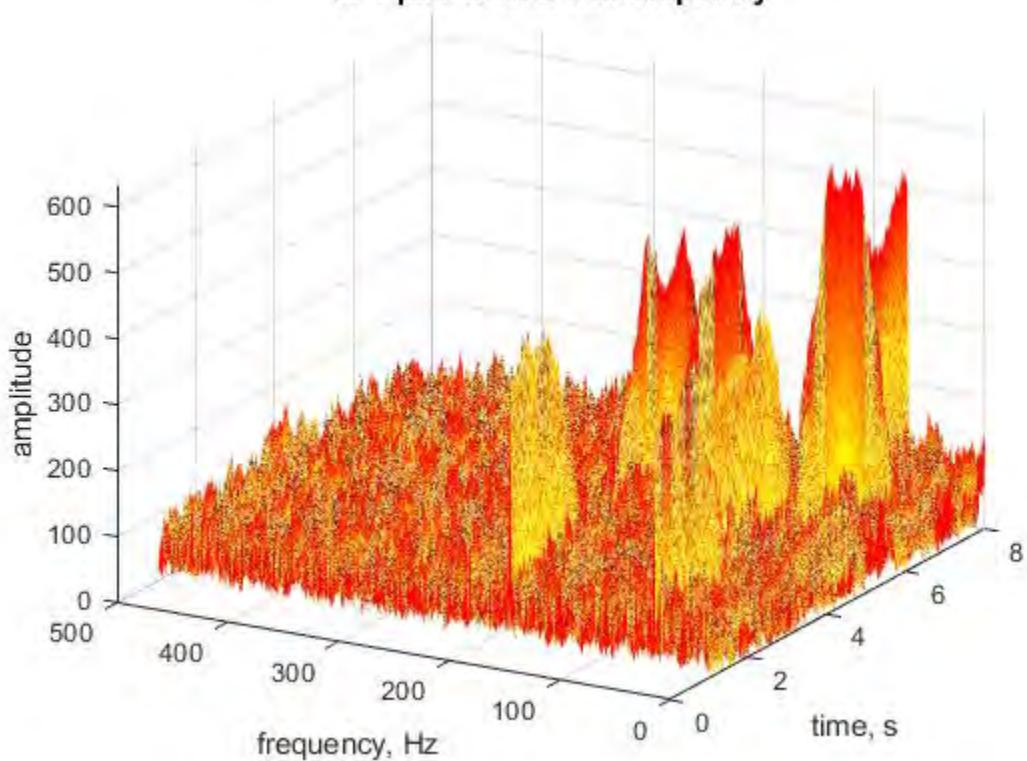


Therefore, the sequence is 'E,E,O,O,I,I,A,A'

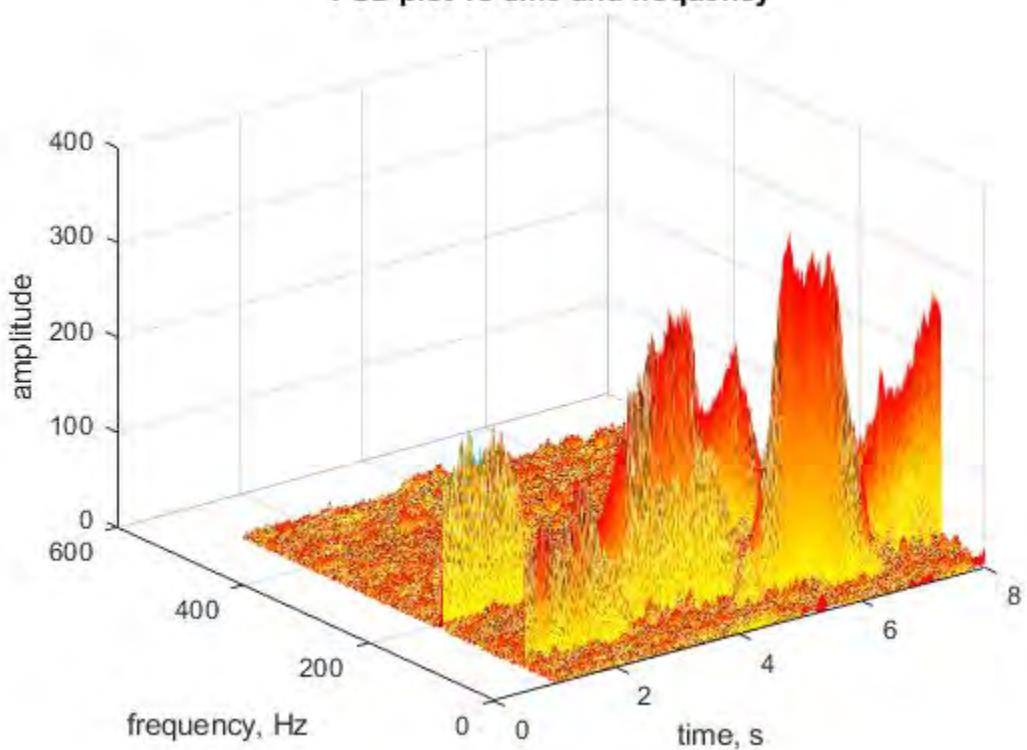


DFT Plot slightly rotated for a different view

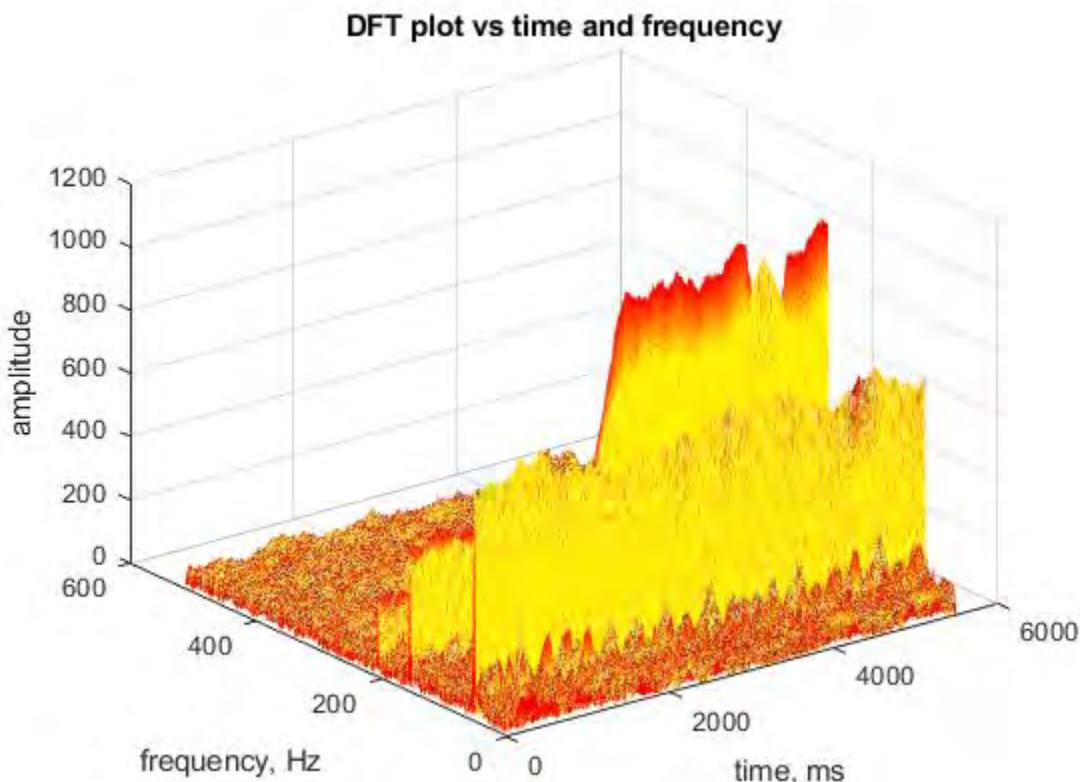
DFT plot vs time and frequency



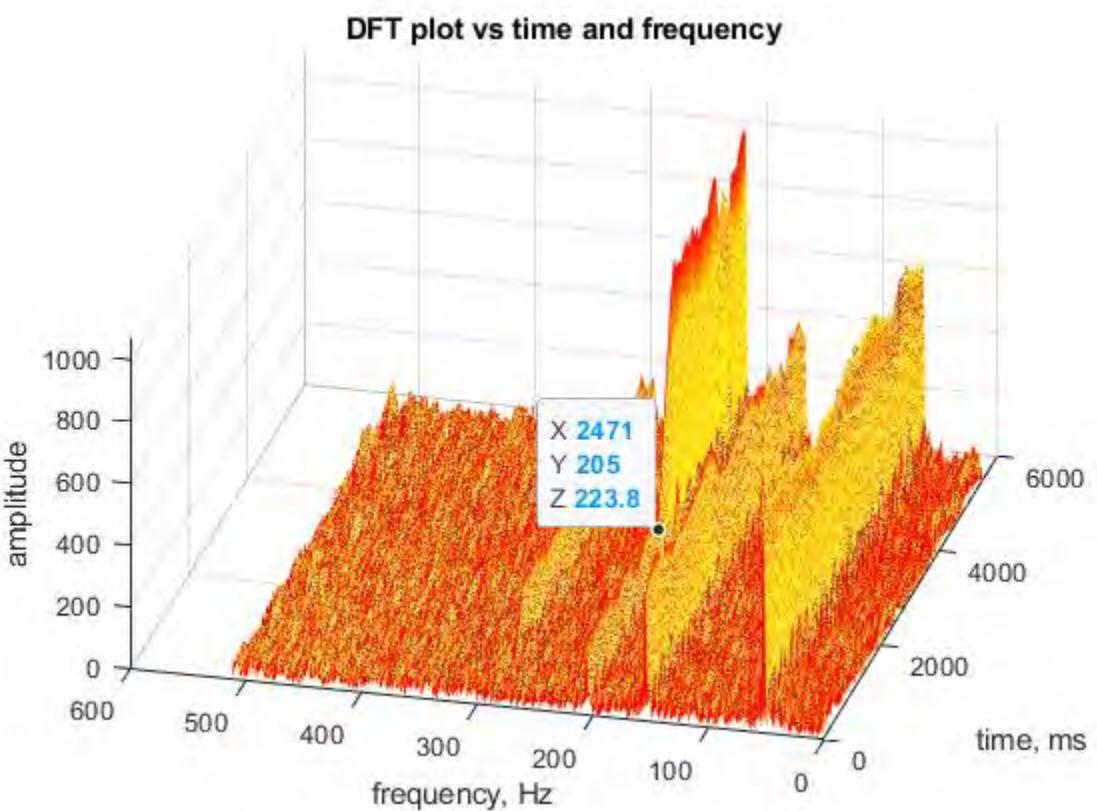
PSD plot vs time and frequency



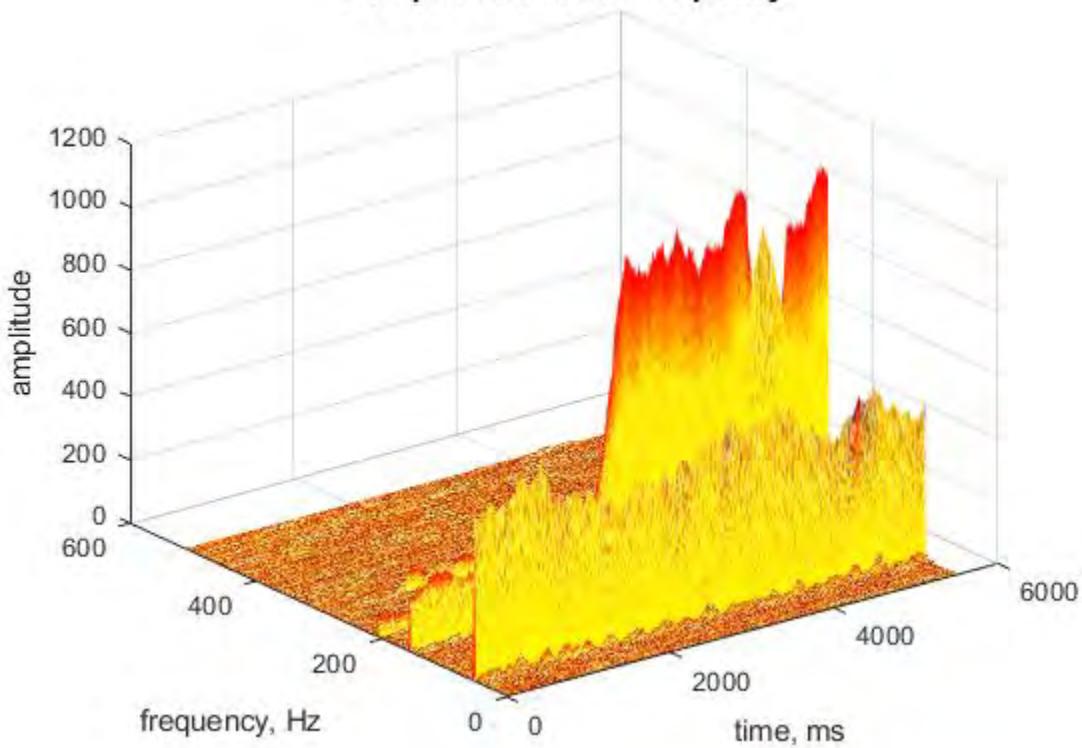
## Problem - 2:



Rotated plot for better assessment

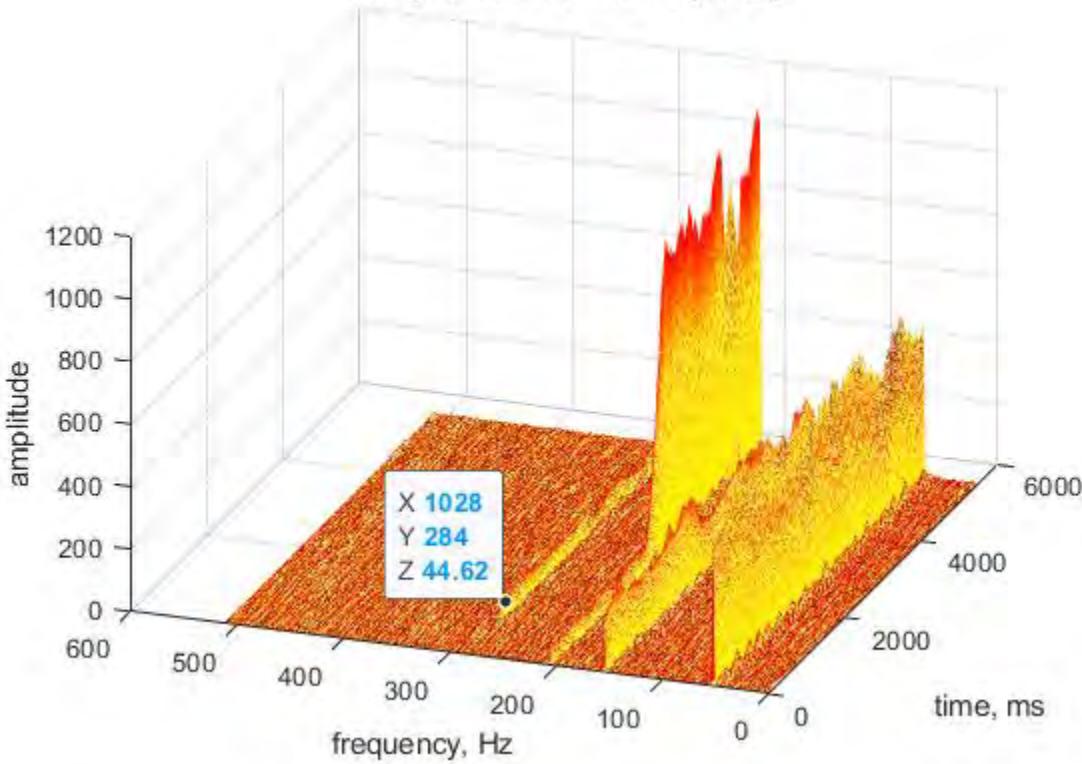


**PSD plot vs time and frequency**

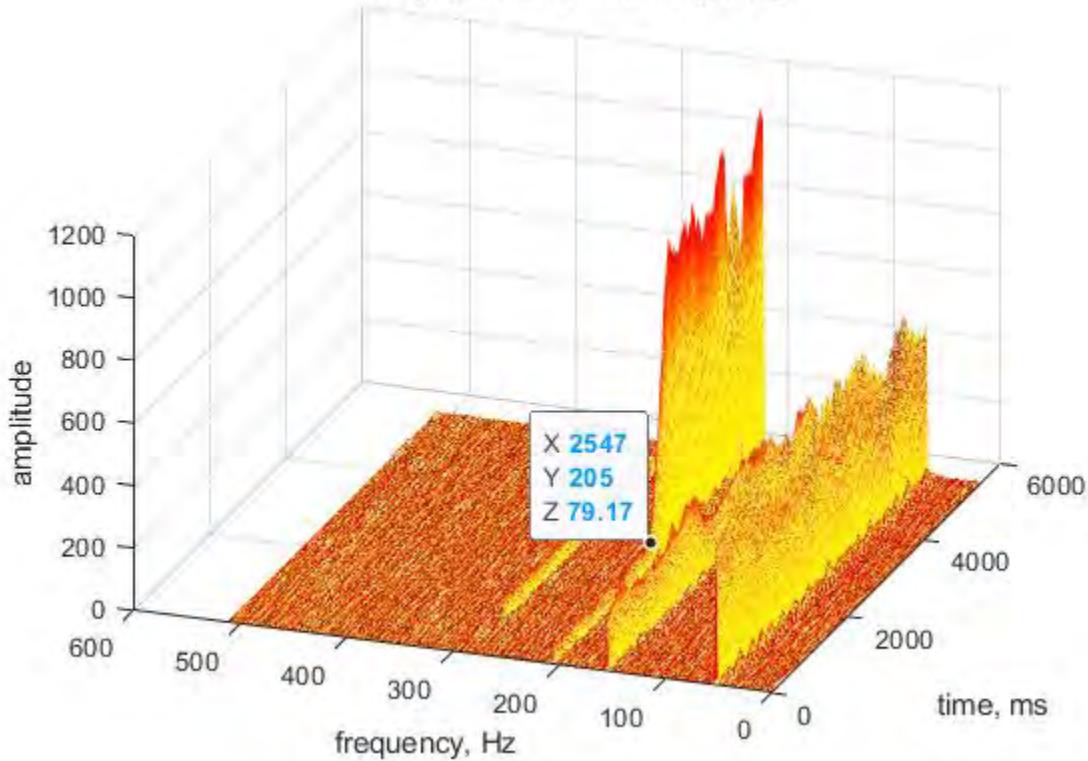


Rotated PSD plot for a better view

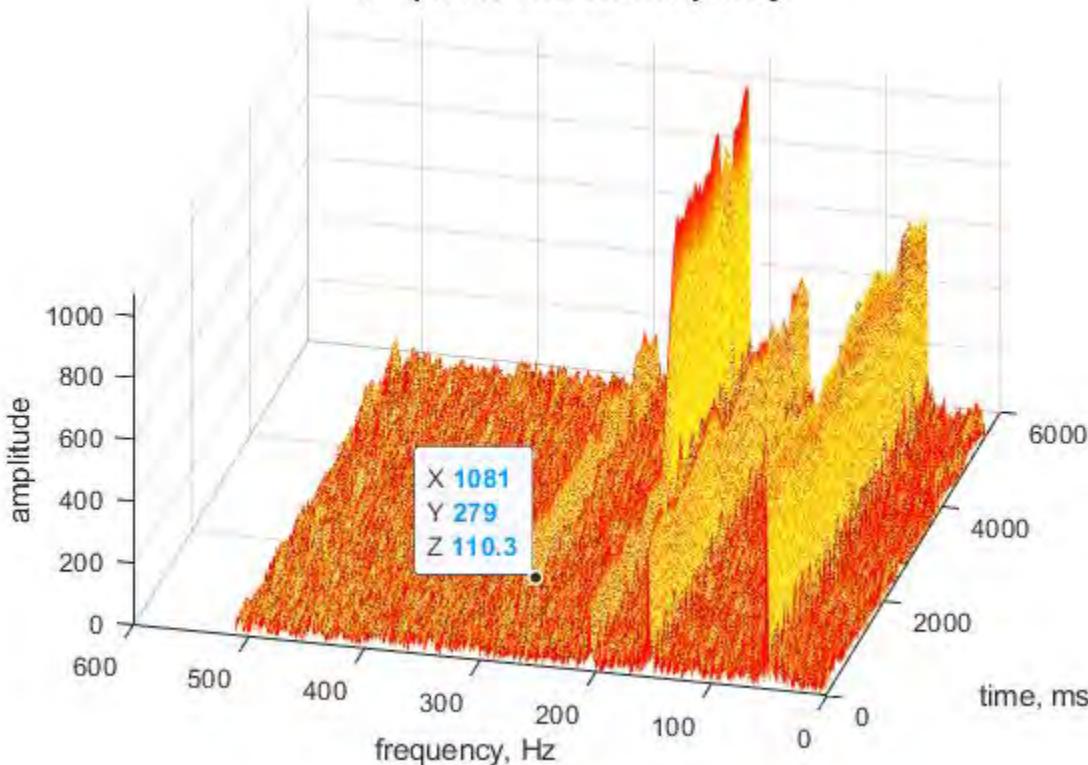
**PSD plot vs time and frequency**



**PSD plot vs time and frequency**



**DFT plot vs time and frequency**



It is clear from above plots that after about 1 second, there is a failure of the gear tooth causing a significant vibration amplitude. The frequency at which this starts is approximately 277 Hz as given in the problem statement.

The base rotation frequency of the motor can be seen clearly at 50 Hz, the fan at 150 Hz and the gearbox at 200 Hz. After approximately 2.5 seconds, there is a sudden peak in the gearbox frequency caused due to the tooth failure. This is observed in both DFT and PSD plots.

For calculation of DFT and PSD, the time window used is of 500 samples as moving average is taken over a length of 0.5 seconds. And N is taken as 1024 ( $2^{11}$ )

The MATLAB function files to generate all plots are given below.

Note: I am more familiar with hot(60) colormap for reading graphs so I have used that. Since there was no restriction or suggestion on this, I noticed much later that my plots were of a different color than those in the solution. I hope that doesn't cause an issue and no points are deducted for this.

## MATLAB Code:

### For Problem - 1:

```
function hw3q1
filename='HW2speech signal.xls';
T=readtable(filename);
waveform=T{2:8001,3};
sam=1000; n=1024; freq=1000*(0:511)/1024;
init=1; init2=1;
while init<9
    st=sam*(init-1)+1;
    en=sam*init;
    bin=waveform(st:en);
    qdft=fft(bin,n);
    figure
    plot(freq,abs(qdft(1:n/2)), '.');
    xlabel('frequency, hz')
    ylabel('amplitude')
    init=init+1;
end
win=8000-sam+1;
A=zeros(1024,win);
PSD=zeros(1024,win);
while init2<win+1
    mwin=waveform(init2:init2+sam-1);
    A(:,init2)=fft(mwin,n);
    PSD(:,init2)=A(:,init2).*conj(A(:,init2))/n;
    init2=init2+1;
end
figure
surf(1:0.001:8,freq,abs(A(1:n/2,:)));
colormap(hot(60))
shading interp
figure
surf(1:0.001:8,freq,abs(PSD(1:n/2,:)));
colormap(hot(60))
shading interp
end
```

## For Problem - 2:

```
function hw3q2
filename='Hw2machine vibration data.xls';
T=readtable(filename);
waveform=T{3:6003,3};
t=0:0.001:6;
n=1024;
mwin=500;
init=1;
nwin=length(t)-mwin;
A=zeros(n,nwin);
PSD=zeros(n,nwin);
while init<nwin+1
    movin=waveform(init:init+mwin-1);
    A(:,init)=fft(movin,n);
    PSD(:,init)=A(:,init).*conj(A(:,init))/n;
    init=init+1;
end
figure
surfl(abs(A(1:n/2,:)));
colormap(hot(60))
shading interp
figure
surfl(PSD(1:n/2,:));
colormap(hot(60))
shading interp
end
```

## References:

1. Analytic Methods in Engineering – Dr. Seiichi Nomura (ME 5331)
2. Dynamic Systems Modeling – Dr. David A Hullender (ME – 5305)
3. Fast Fourier Transform – MATLAB Documentation (retrieved from:  
<https://www.mathworks.com/help/matlab/ref/fft.html>)
4. Dynamic Systems Modeling and Simulation (18<sup>th</sup> Edition) – Dr. David A Hullender

**EE 5322 Exam 1**  
**Discrete Time Simulation, RLS**

**1. Discrete-Time System.**

A discrete time system is given by

$$x_{k+1} = Ax_k + Bu_k, \quad x_0$$

Write a MATLAB m file to simulate the system, i.e. to compute  $x_k$  for a given input  $u_k$ , initial condition  $x_0$ , and range of the time index  $k=1,2,\dots,N$ .

- a. Simulate the system

$$x_{k+1} = \begin{bmatrix} 0 & 1 \\ -0.9 & 1.85 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k$$

for  $u_k$  equal to the unit step and  $x_0=0$ . Plot  $x_k$  vs. k for 100 time samples.

- b. Simulate the same system but now add process noise so that

$$x_{k+1} = Ax_k + Bu_k + w_k.$$

Take the noise  $w_k$  as Gaussian with mean of zero and each component having covariance of 0.1, that is  $Q=0.1I$ . Use MATLAB function randn. Plot  $x_k$  vs. k for 100 time samples.

**2. RLS System Identification**

Download the file for this problem from the homework assignment home page.

The input  $u_k$  and output  $y_k$  of a discrete time system are given in the data file. The system is of second order with a delay of d=2.

- a. Write a RLS program to identify the system transfer function. Take measurement noise covariance constant at 0.1.  
b. Plot the output  $y_k$  in the excel file and the output of your identified system given the input  $u_k$ . They should be the same.

## **EE 5322 Intelligent Control- Exam 1**

**Fall 2020**

1. This is a take home exam. YOU MUST WORK ALONE.  
Any cheating or collusion will be severely punished.
2. To obtain full credit, show all your work. No partial credit will be given without the supporting work.
3. Please sign this form and include it as the first page of your submitted exam.

.....  
**Typed Name: Atul Shrotriya**

Pledge of honor:

"On my honor I have neither given nor received aid on this examination."

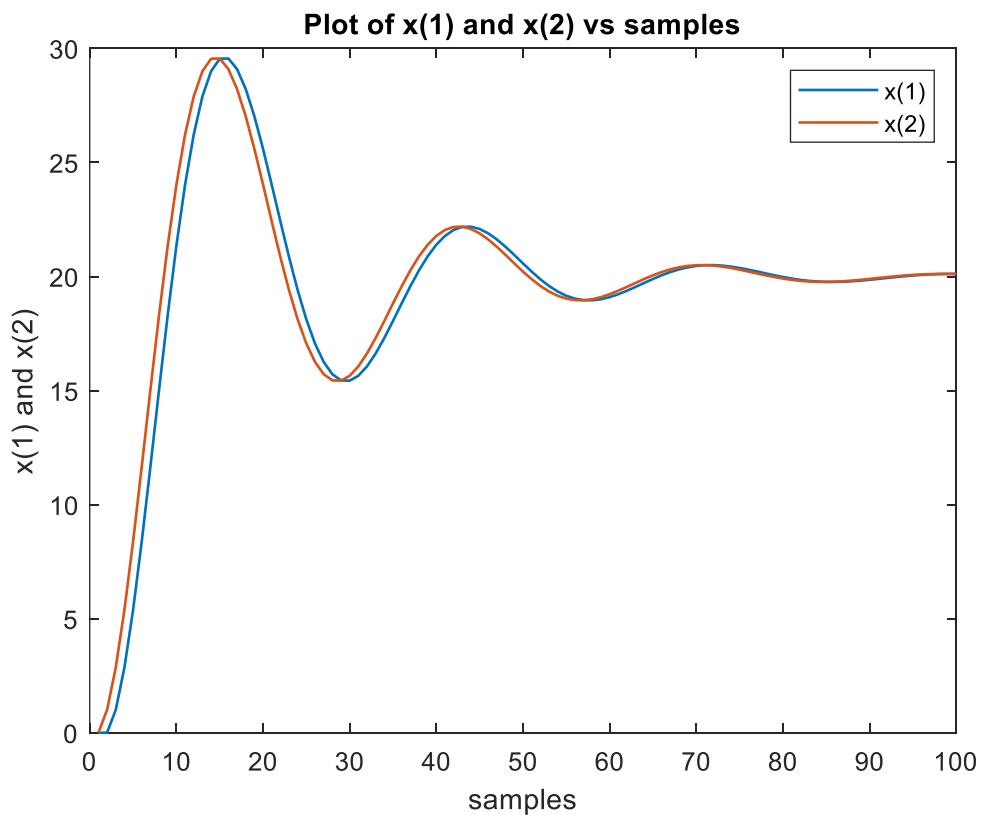
e-Signature: **Atul Shrotriya**

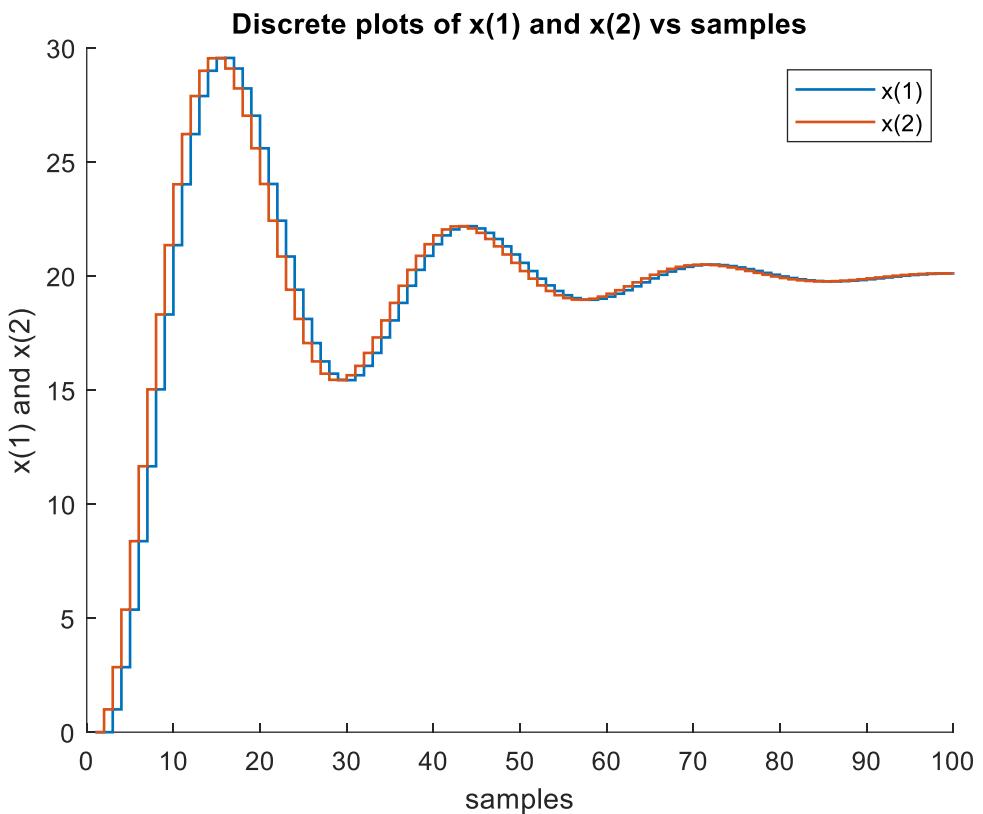
## Problem - 1:

### a. MATLAB Code

```
function exam1q1a
x(1,2)=0;
u=1;
for k=1:1:100
    x(k+1,1)=x(k,2);
    x(k+1,2)=-0.9*x(k,1)+1.85*x(k,2)+u;
end
t=1:1:100;
figure(1)
plot(t,x(1:100,1),t,x(1:100,2))
figure(2)
hold on
stairs(t,x(1:100,1))
stairs(t,x(1:100,2))
end
```

## Plots





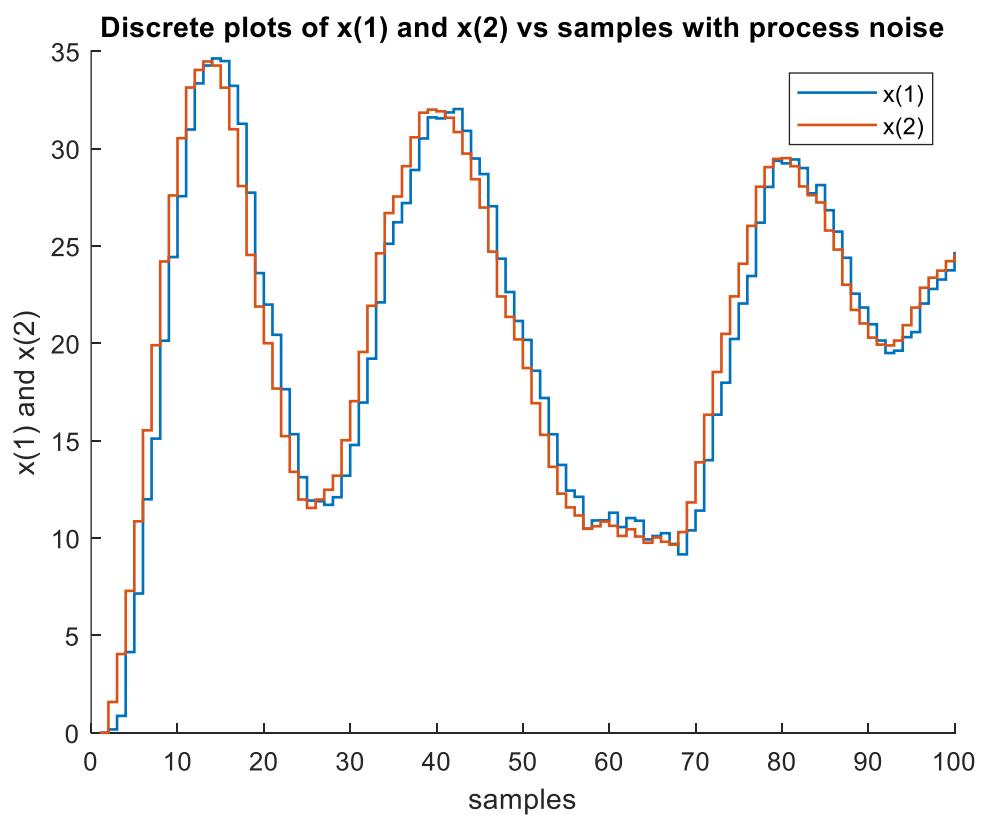
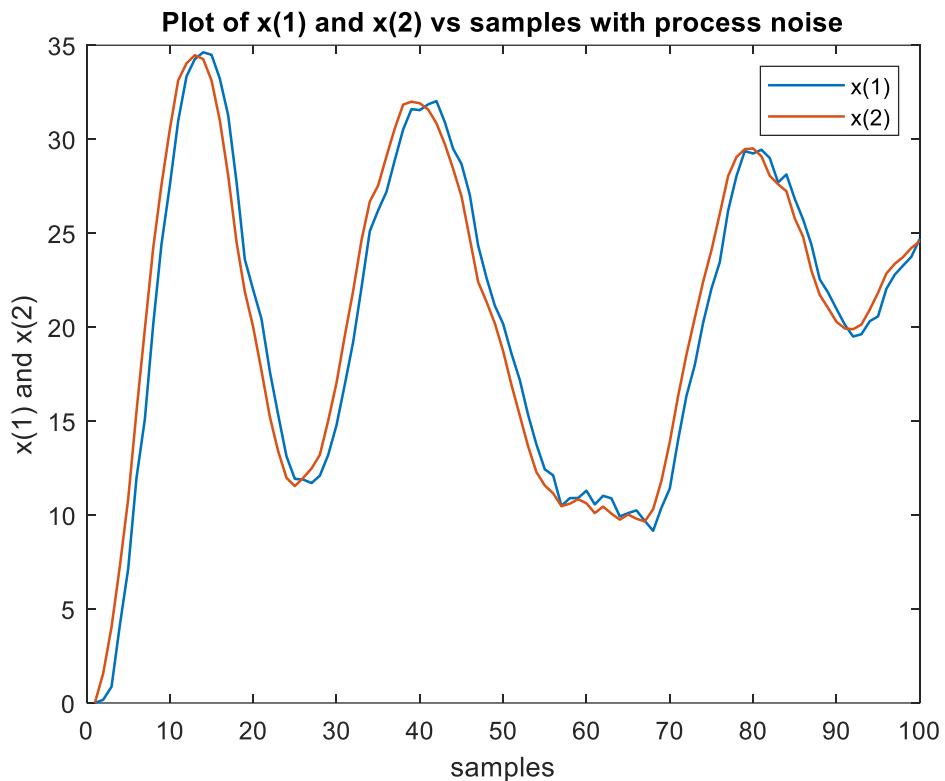
### b. MATLAB code

```

function exam1q1b
x(1,2)=0;
u=1;
for k=1:1:100
    mu = [0 0];
    sigma = [0.1 0; 0 0.1];
    R = chol(sigma);
    z = repmat(mu,1,1) + randn(1,2)*R;
    x(k+1,1)=x(k,2)+z(1,1);
    x(k+1,2)=-0.9*x(k,1)+1.85*x(k,2)+u+z(1,2);
end
t=1:1:100;
figure(1)
plot(t,x(1:100,1),t,x(1:100,2))
figure(2)
hold on
stairs(t,x(1:100,1))
stairs(t,x(1:100,2))
end

```

## Plots



## Problem - 2:

### a. MATLAB code

```
function exam1q2a
%fixed and initial values
cov=0.1; Pk1=10000*eye(3); thetak1=zeros(3,1);

%retrieving data
filename='Hw3RLS ID data.xls';
T=readtable(filename);
uin=str2double(T{3:603,3});
yout=str2double(T{3:603,5});

%estimating system parameters
for k=3:601
    hk1=[-yout(k-1);-yout(k-2);uin(k-2)];
    Pk1=Pk1-Pk1*hk1*(inv(transpose(hk1)*Pk1*hk1+cov))*(transpose(hk1))*Pk1;
    thetak1=thetak1+Pk1*(hk1/cov)*(yout(k)-(transpose(hk1))*thetak1);
end
Sys=thetak1
end
```

### Output

```
Sys =
-1.9000
 0.9500
 0.2000
```

### b. MATLAB Code

```
function exam1q2b
%fixed and initial values
cov=0.1; Pk1=1e6*eye(3); thetak1=zeros(3,1); Yhat=zeros(601,1); xp=zeros(2,1);

%retrieving data
filename='Hw3RLS ID data.xls';
T=readtable(filename);
uin=str2double(T{3:603,3});
yout=str2double(T{3:603,5});

%estimating system parameters
for k=3:601
    hk1=[-yout(k-1);-yout(k-2);uin(k-2)];
    Pk1=Pk1-Pk1*hk1*(inv(transpose(hk1)*Pk1*hk1+cov))*(transpose(hk1))*Pk1;
    thetak1=thetak1+Pk1*(hk1/cov)*(yout(k)-(transpose(hk1))*thetak1);
end

%output from estimated parameters
for k=1:601
```

```

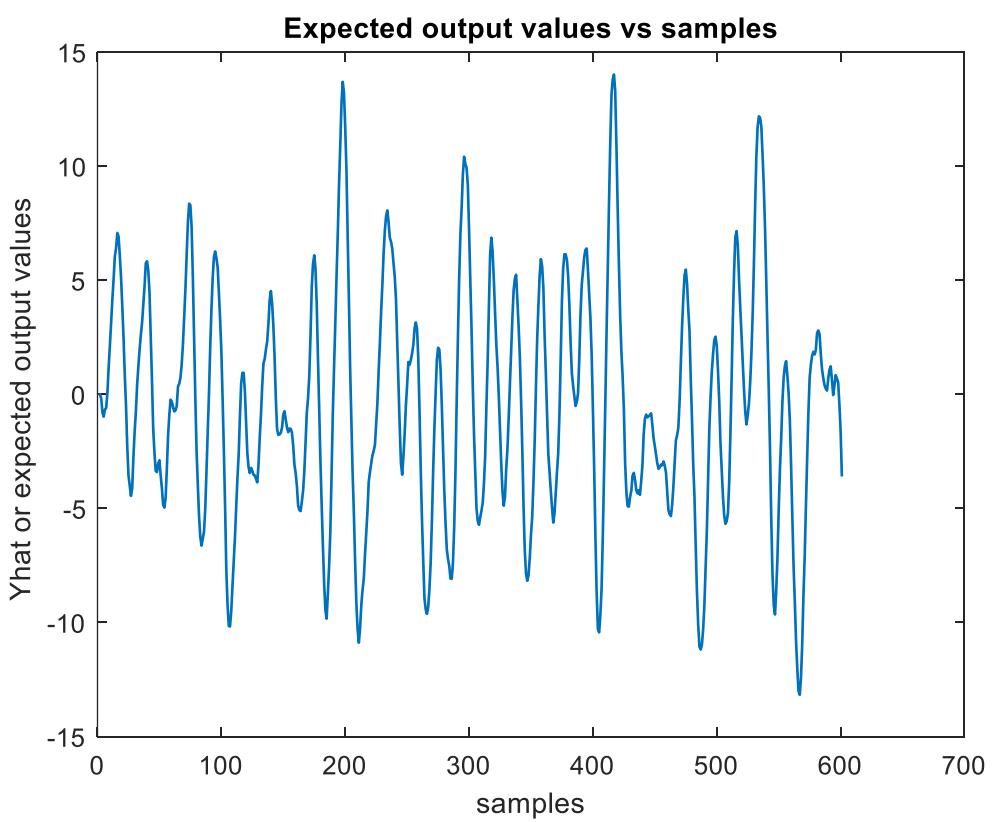
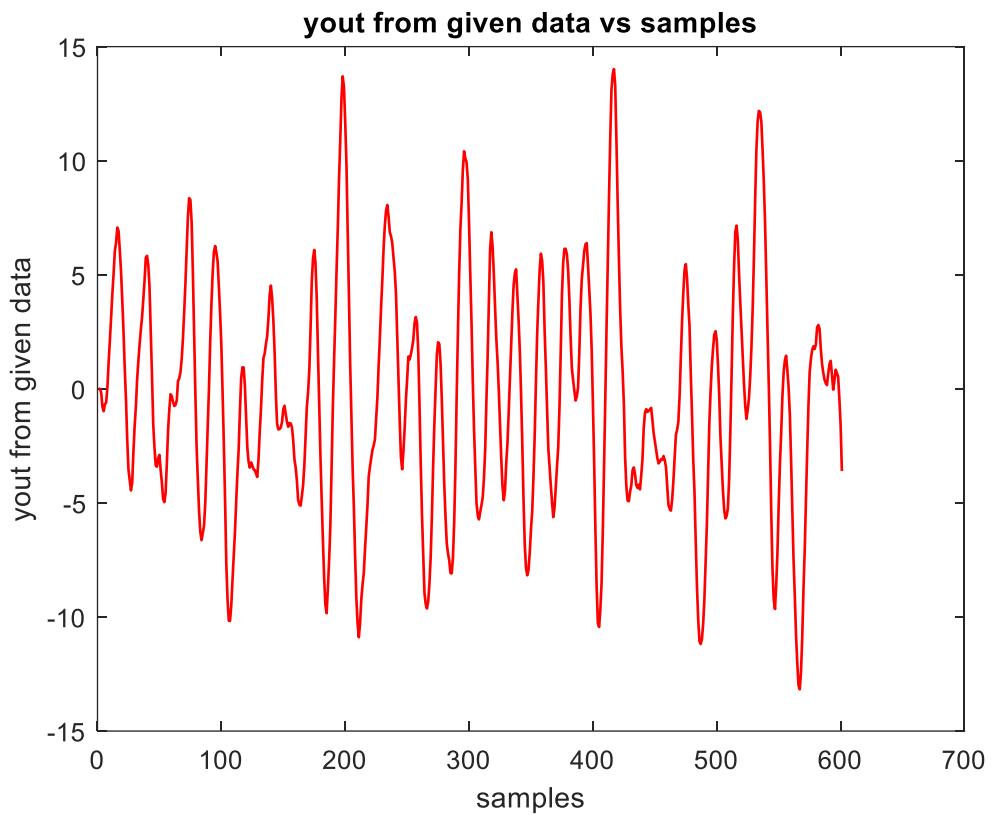
Yhat(k)=[thetak1(3) 0]*xp;
xp=[0 1;-thetak1(2) -thetak1(1)]*xp+[0;1]*uin(k);
end

%comparison plots
figure(1)
plot(1:601,yout,'r')
figure(2)
plot(1:601,Yhat,'')
figure(3)
subplot(1,2,1)
plot(1:601,yout,'r')
subplot(1,2,2)
plot(1:601,Yhat,'')
figure(4)
subplot(2,1,1)
plot(1:601,yout,'r')
subplot(2,1,2)
plot(1:601,Yhat,'')
figure(5)
hold on
plot(1:601,yout,'r','linewidth',1)
plot(1:601,Yhat,'k:','linewidth',2)
end

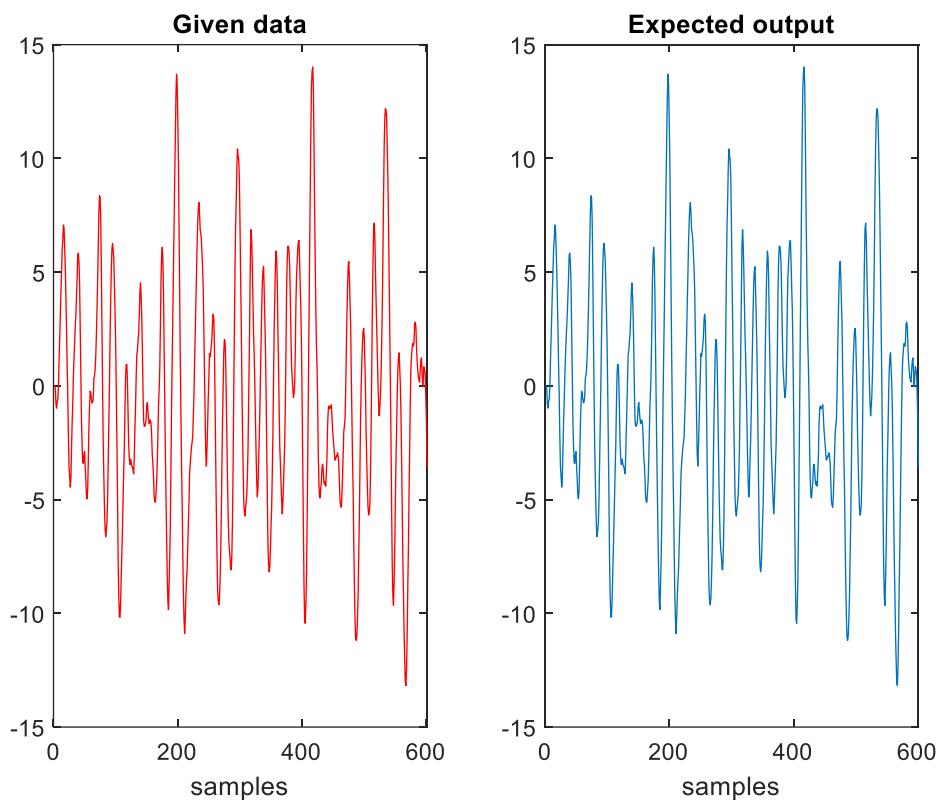
```

**Note:** I have drawn so many comparison plots to ensure that I didn't miss out on anything and to show better that both the plots are same. The plots are given on the next page.

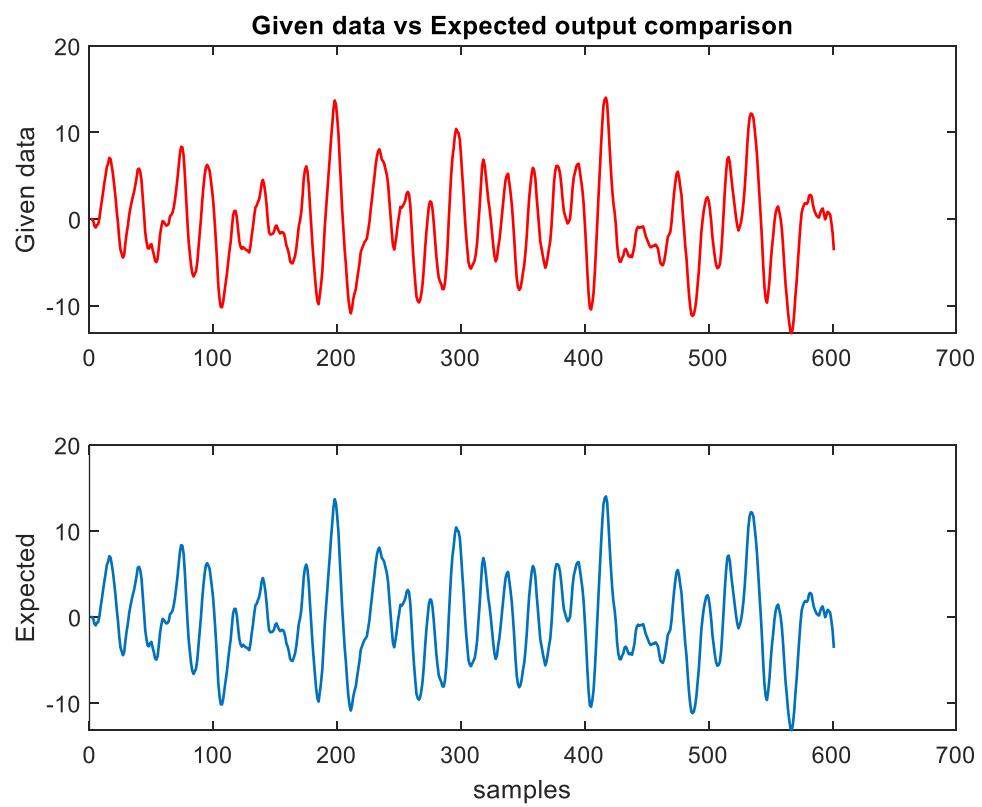
## plots



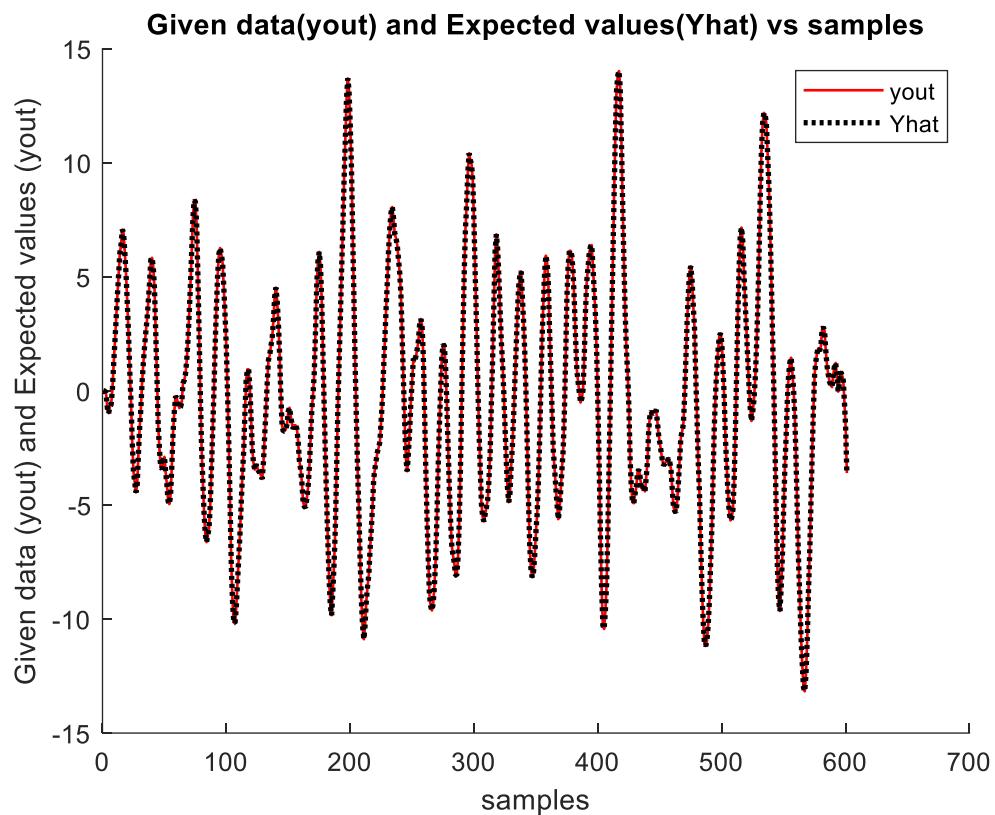
### Side by Side Comparison



### Top Bottom Comparison



### Comparison on the same axes



## EE 5322 Homework 4

### Mobile Robot Control & Potential Fields

- Potential Field.** Use MATLAB to make a 3-D plot of the potential fields described below. You will need to use plot commands and maybe the mesh function. The work area is a square from (0,0) to (13,13) in the (x,y) plane. The goal is at (10,10). There are obstacles at (3,4) and (8,5). Use a repulsive potential of  $K_i / r_i$  for each obstacle, with  $r_i$  the vector to the  $i$ -th obstacle. For the target use an attractive potential of  $K_T r_T$ , with  $r_T$  the vector to the target. Adjust the gains to get a decent plot. Plot the sum of the three potential fields in 3-D on the x,y-plane square from (0,0) to (13,13).
- Potential Field Navigation.** For the same scenario as in Problem 1, a mobile robot starts at (0,0). The front wheel steered mobile robot has dynamics

$$\dot{x} = v_T \cos \phi \cos \theta$$

$$\dot{y} = v_T \cos \phi \sin \theta$$

$$\dot{\theta} = \frac{v_T}{L} \sin \phi$$

with  $(x,y)$  the position,  $\theta$  the heading angle,  $v_T$  the wheel speed,  $L$  the wheel base, and  $\phi$  the steering angle. Set  $L=2$ .

- a. Compute forces due to each obstacle and goal. Compute total force on the vehicle at point  $(x,y)$ .
- b. Design a feedback control system for force-field control. Draw your control system.
- c. Use MATLAB to simulate the nonlinear dynamics assuming a constant velocity  $v_T$  and a steerable front wheel. The wheel should be steered so that the vehicle always goes downhill in the force field plot. Plot the resulting trajectory in the  $(x,y)$  plane. Use a square from (0,0) to (13,13).
- Not Required for the Exam. You get Extra Credit if you do this problem-Swarm/Platoon/Formations.** Do what you want to for this problem. The intent is to focus on some sort of swarm or platoon or formation behavior, not the full dynamics. Therefore, take 5 vehicles each with the simple point mass (Newton's law) dynamics

$$\ddot{x} = F_x / m$$

$$\ddot{y} = F_y / m$$

with  $(x,y)$  the position of the vehicle and  $F_x, F_y$  the forces in the  $x$  and  $y$  direction respectively. The forces might be the sums of attractive forces to goals, repulsive forces from obstacles, and repulsive forces between the agents.

Make some sort of interesting plots or movies showing the leader going to a desired goal or moving along a prescribed trajectory and the followers staying close to him, or in a prescribed formation. Obstacle avoidance by a platoon or swarm is interesting.

## **EE 5322 Intelligent Control Fall 2020**

Homework Pledge of Honor On all homeworks in this class - YOU MUST WORK ALONE. Any cheating or collusion will be severely punished. It is very easy to compare your software code and determine if you worked together Or if you found code online written by someone else. It does not matter if you change the variable names. Please sign this form and include it as the first page of all of your submitted homeworks.

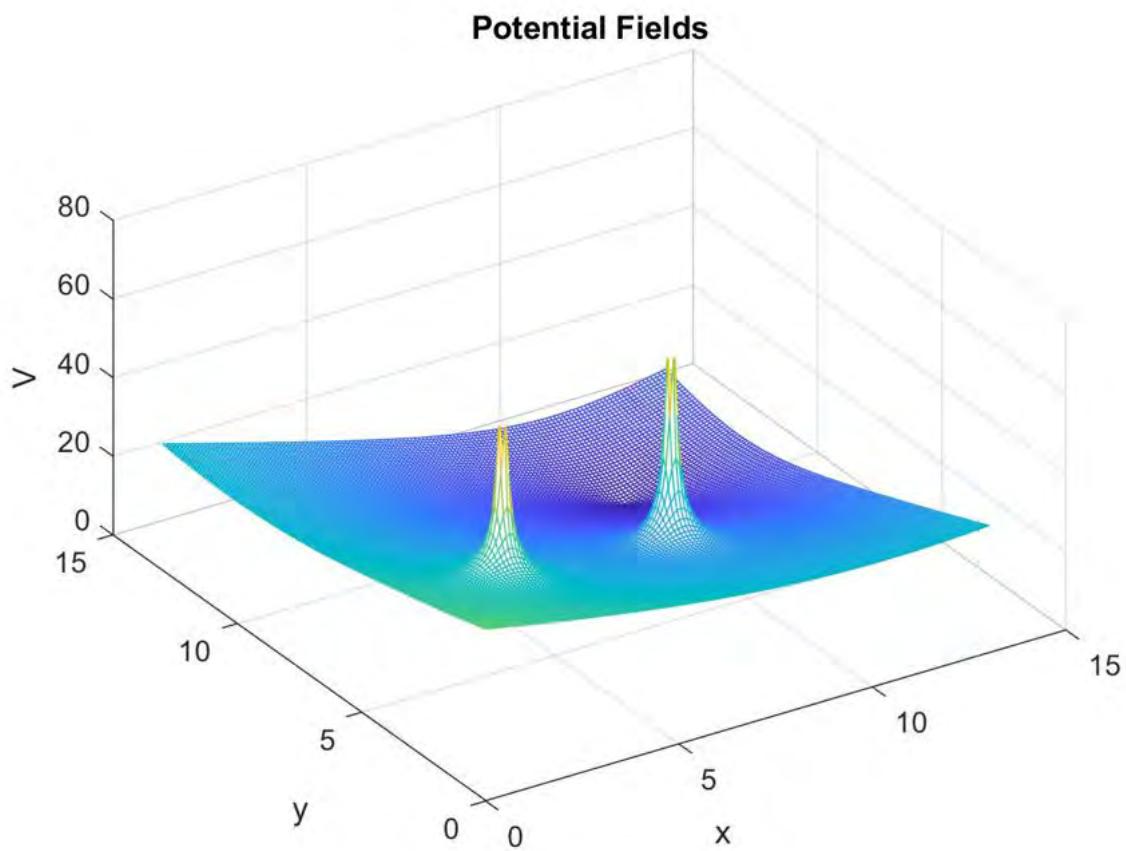
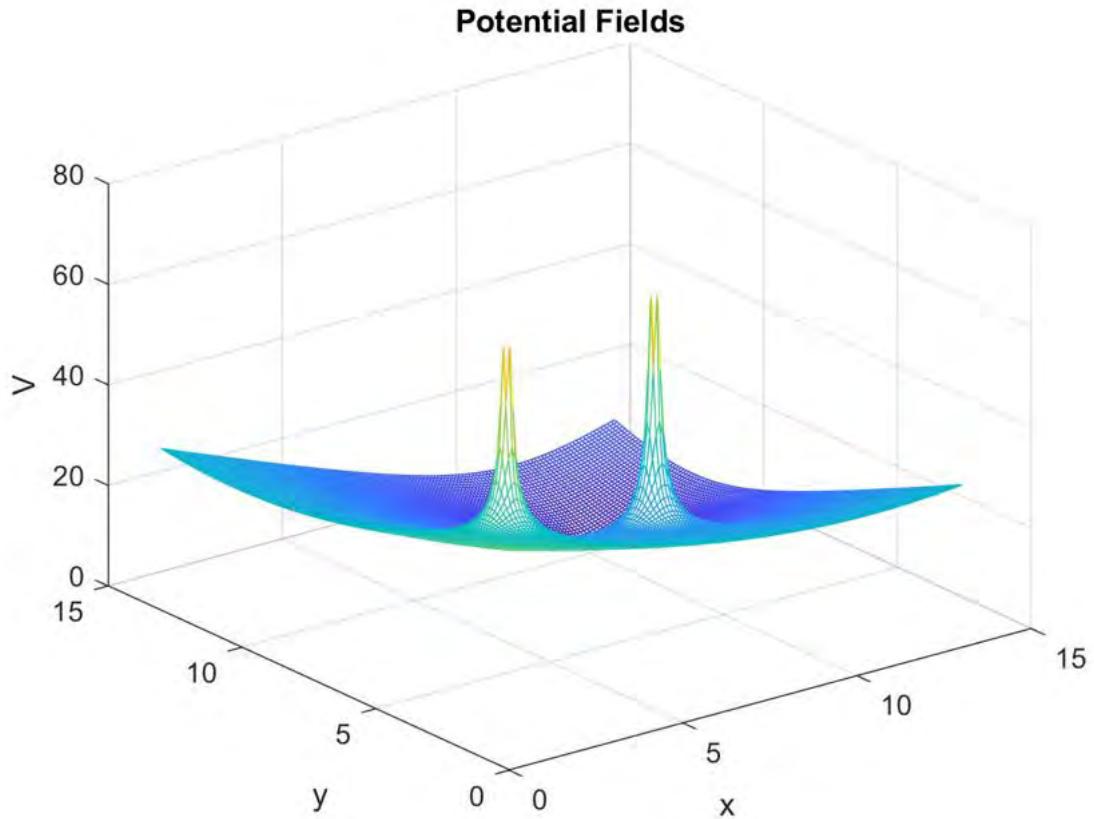
.....

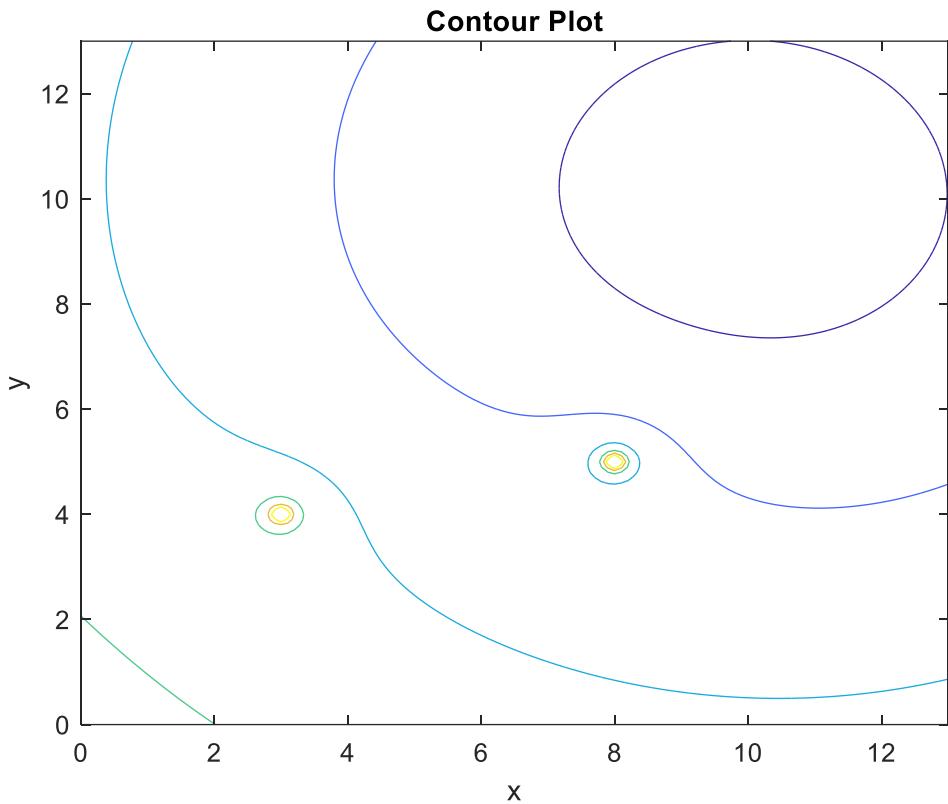
Typed Name: Atul Shrotriya \_\_\_\_\_

Pledge of honor: "On my honor I have neither given nor received aid on this homework."

e-Signature: Atul Shrotriya \_\_\_\_\_

## Problem - 1:





### MATLAB Code

```

function hw4q1
%work area
wa=[0:0.1:13;0:0.1:13];
%locations of obstacles and target
T=[10,10];
obs=[3,4;8,5];
n=length(wa(1,:));
%gains
kT=3;
ko=[4,5];

for p=1:n
    for q=1:n
        VT(q,p)=kT*sqrt(((wa(1,p)-T(1,1))^2)+(wa(2,q)-T(1,2))^2);
        Vo1(q,p)=ko(1,1)/sqrt(((wa(1,p)-obs(1,1))^2)+(wa(2,q)-obs(1,2))^2);
        Vo2(q,p)=ko(1,2)/sqrt(((wa(1,p)-obs(2,1))^2)+(wa(2,q)-obs(2,2))^2);
    end
end

```

```
V=VT+Vo1+Vo2;  
end  
end  
figure  
mesh(wa(1,:),wa(2,:),V)  
  
figure  
contour(wa(1,:),wa(2,:),V)  
end
```

## Problem - 2:

- a. Note: All the forces can be broken into x and y components using the matrices so I didn't write the formulae again and again to avoid redundancy.

### Force due to obstacle 1

$$\mathbf{F}_1 = -\frac{1}{r_1^2} \begin{bmatrix} 3 & - \\ 4 & - \\ -1 & \end{bmatrix}$$

where  $r_1$  = gain for obstacle 1

$$r_1 = \sqrt{(3 - x)^2 + (4 - y)^2}$$

### Force due to obstacle 2

$$\mathbf{F}_2 = -\frac{1}{r_2^2} \begin{bmatrix} 8 & - \\ 5 & - \\ -2 & \end{bmatrix}$$

where  $r_2$  = gain for obstacle 2

$$r_2 = \sqrt{(8 - x)^2 + (5 - y)^2}$$

### Force due to goal

$$= \begin{bmatrix} 10 & - \\ - & \end{bmatrix}$$

where  $r$  = gain for goal

$$= \sqrt{(10 - x)^2 + (10 - y)^2}$$

### Total Force

$$\begin{aligned} &= \mathbf{F}_1 + \mathbf{F}_2 + \\ &= -\frac{1}{r_1^2} \begin{bmatrix} 3 & - \\ 4 & - \\ -1 & \end{bmatrix} - \frac{1}{r_2^2} \begin{bmatrix} 8 & - \\ 5 & - \\ -2 & \end{bmatrix} + \begin{bmatrix} 10 & - \\ - & \end{bmatrix} \end{aligned}$$

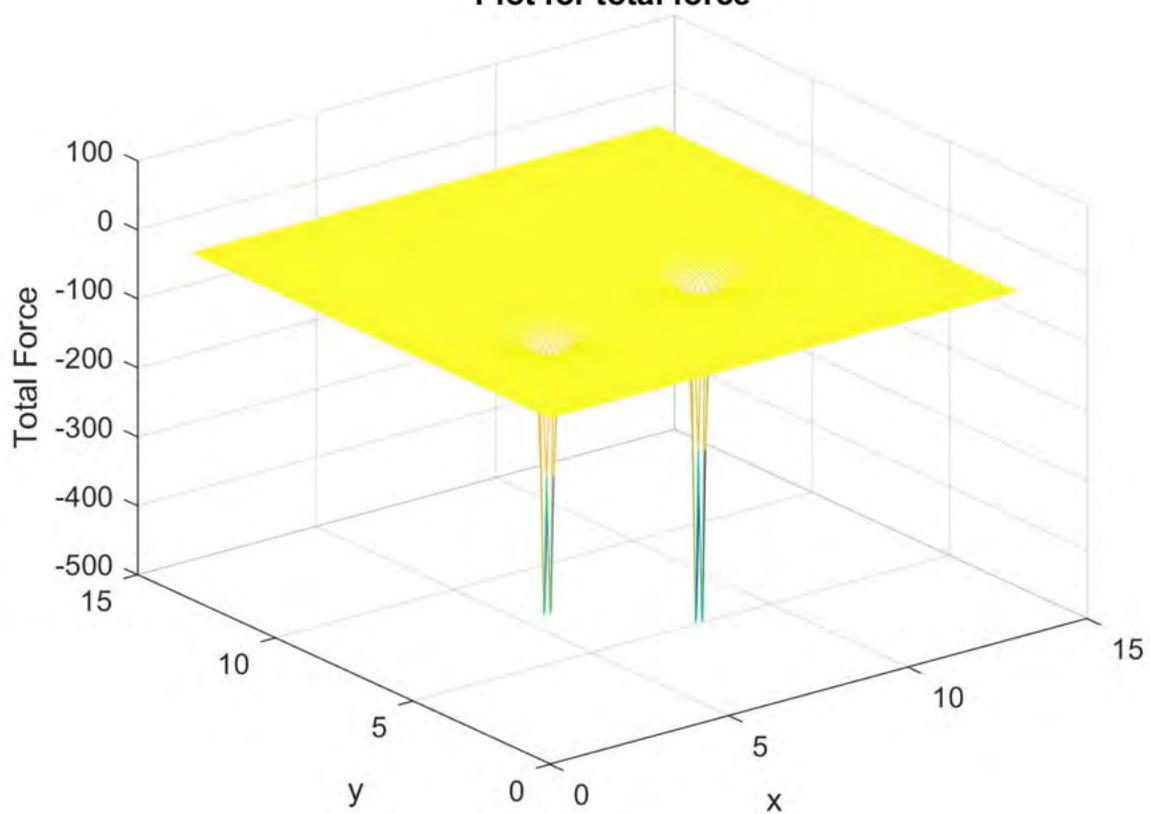
This can be simplified and written without the individual x and y components as

$$= -\frac{1}{r_1^2} - \frac{2}{r_2^2} +$$

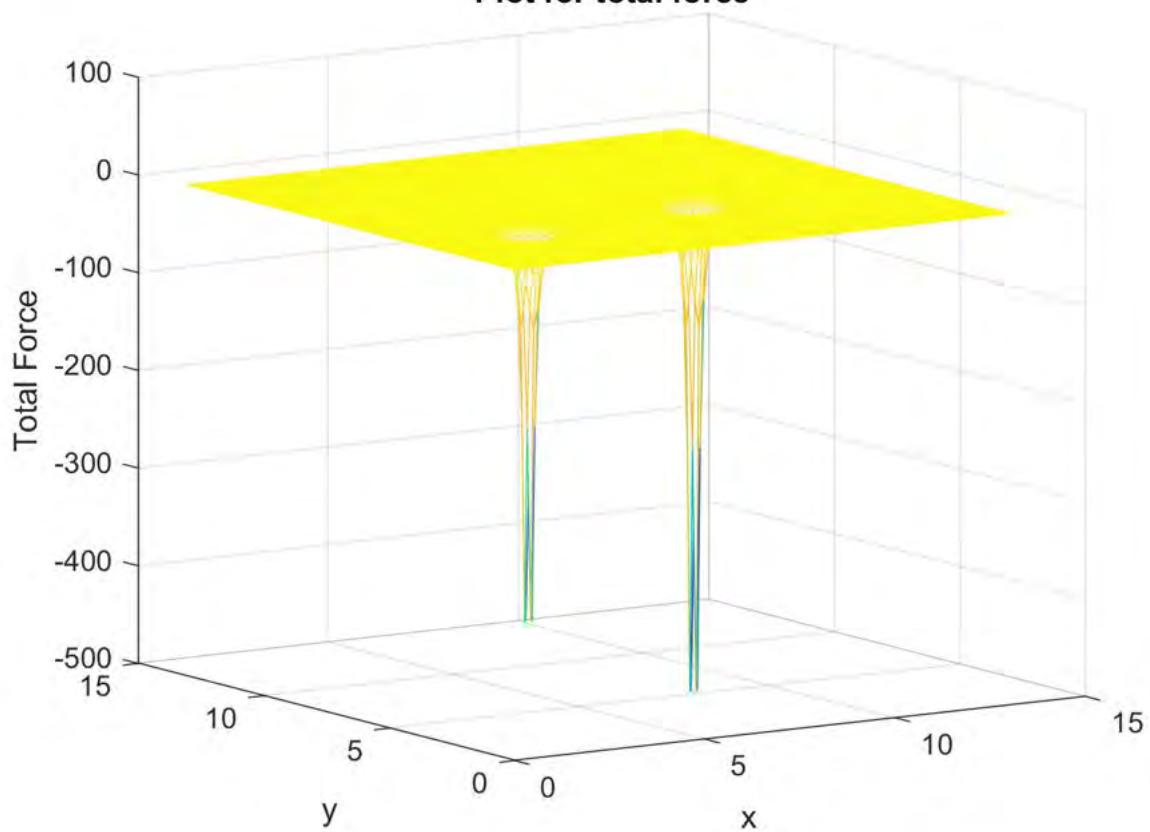
It shows that attractive force is linear whereas the repulsive forces follow the inverse square law.

I know we weren't required to plot anything but I did it anyway and the plots and MATLAB code are given below.

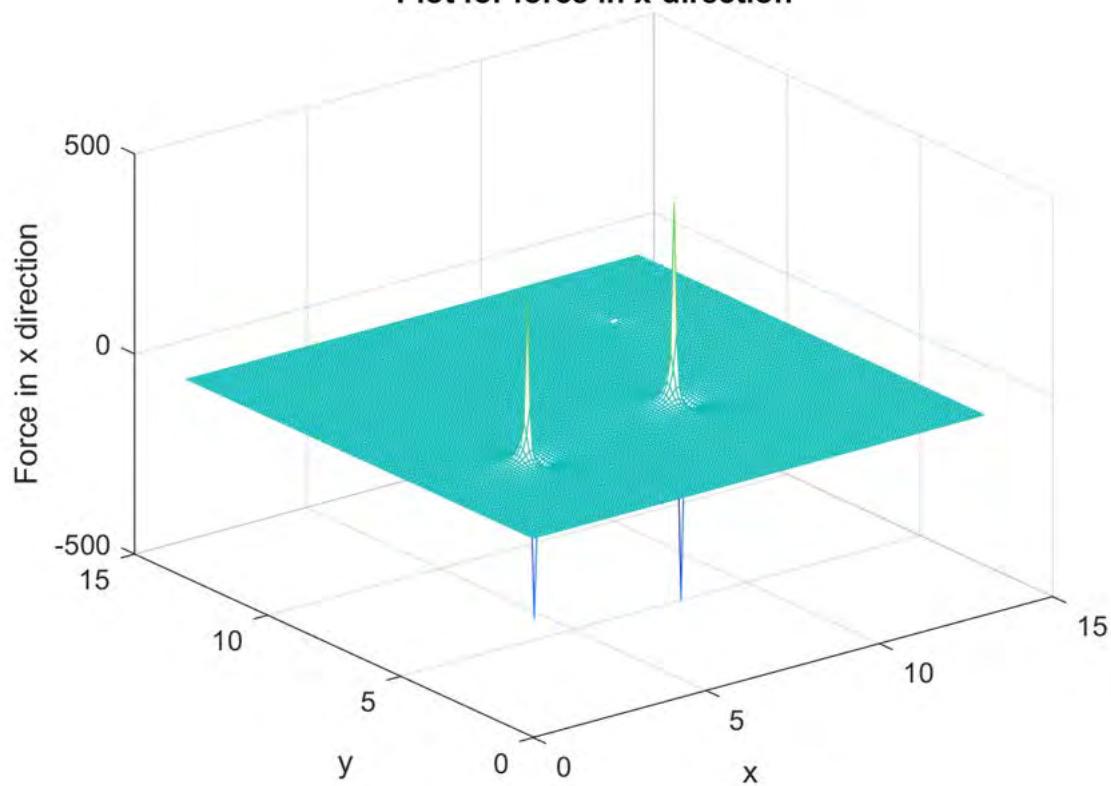
**Plot for total force**



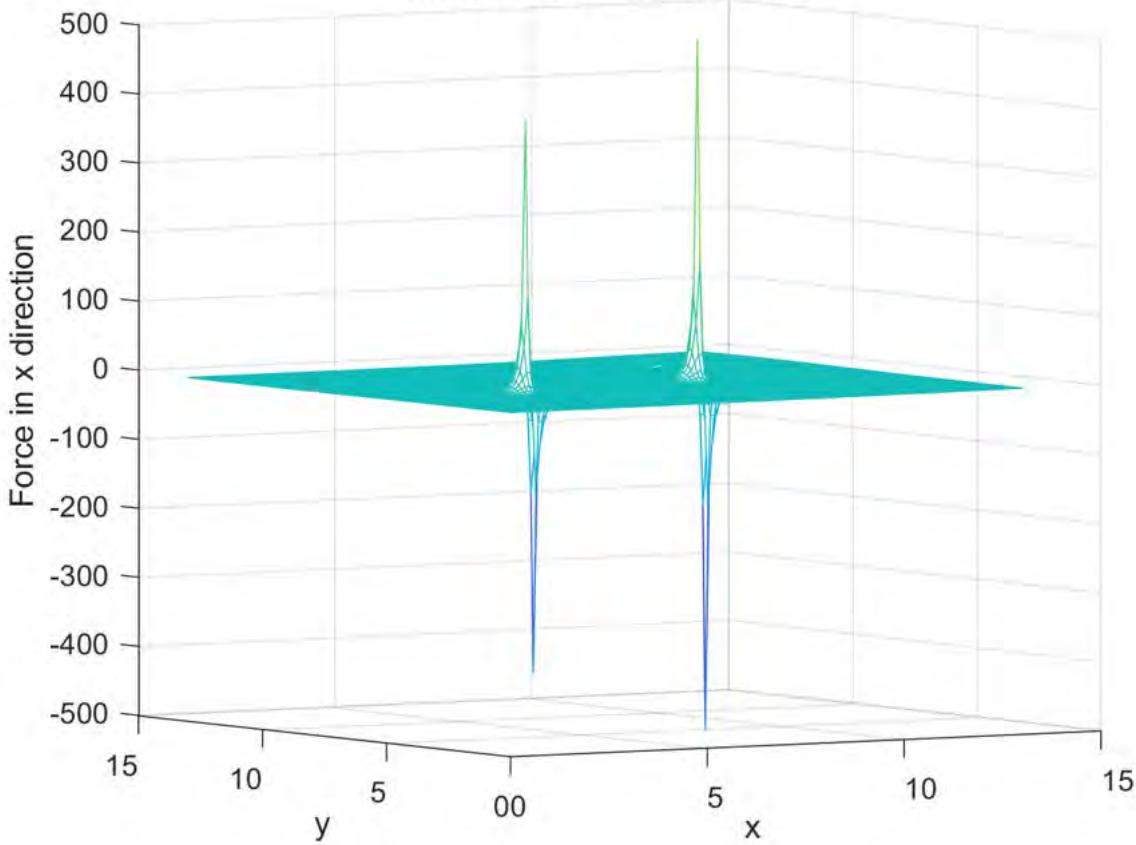
**Plot for total force**



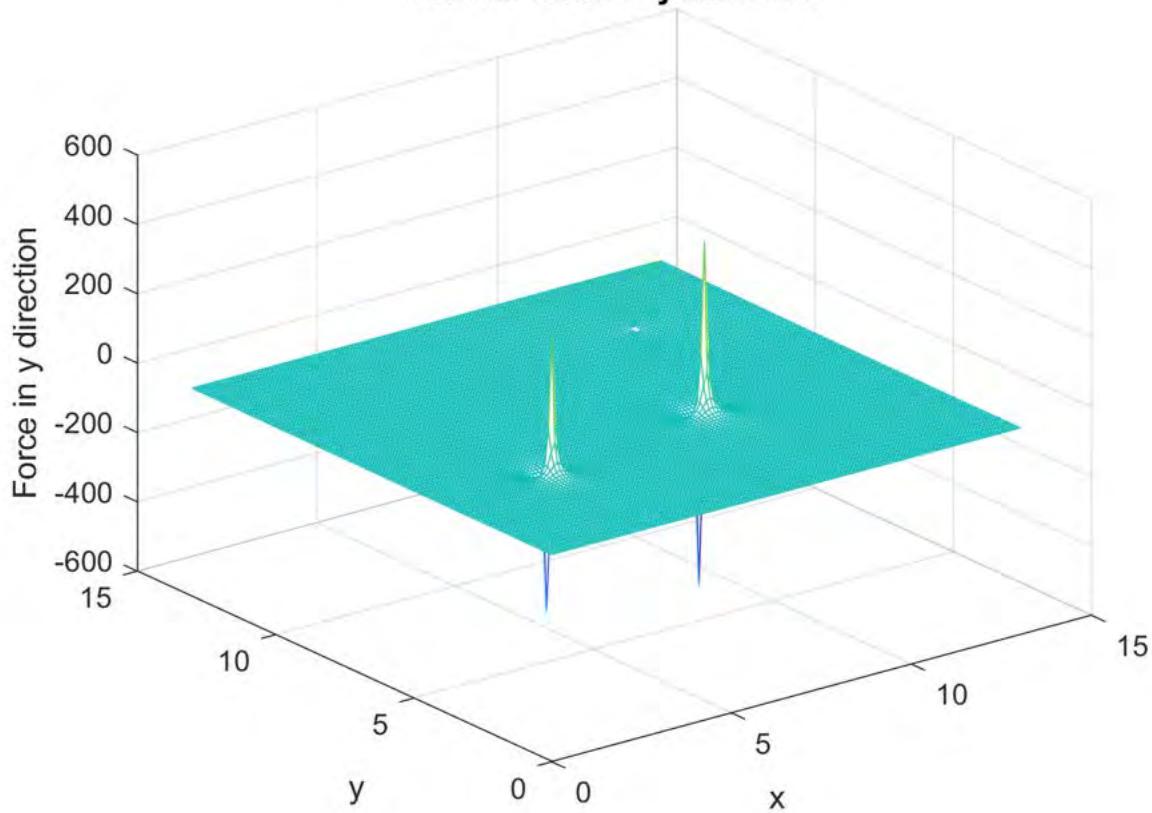
**Plot for force in x-direction**



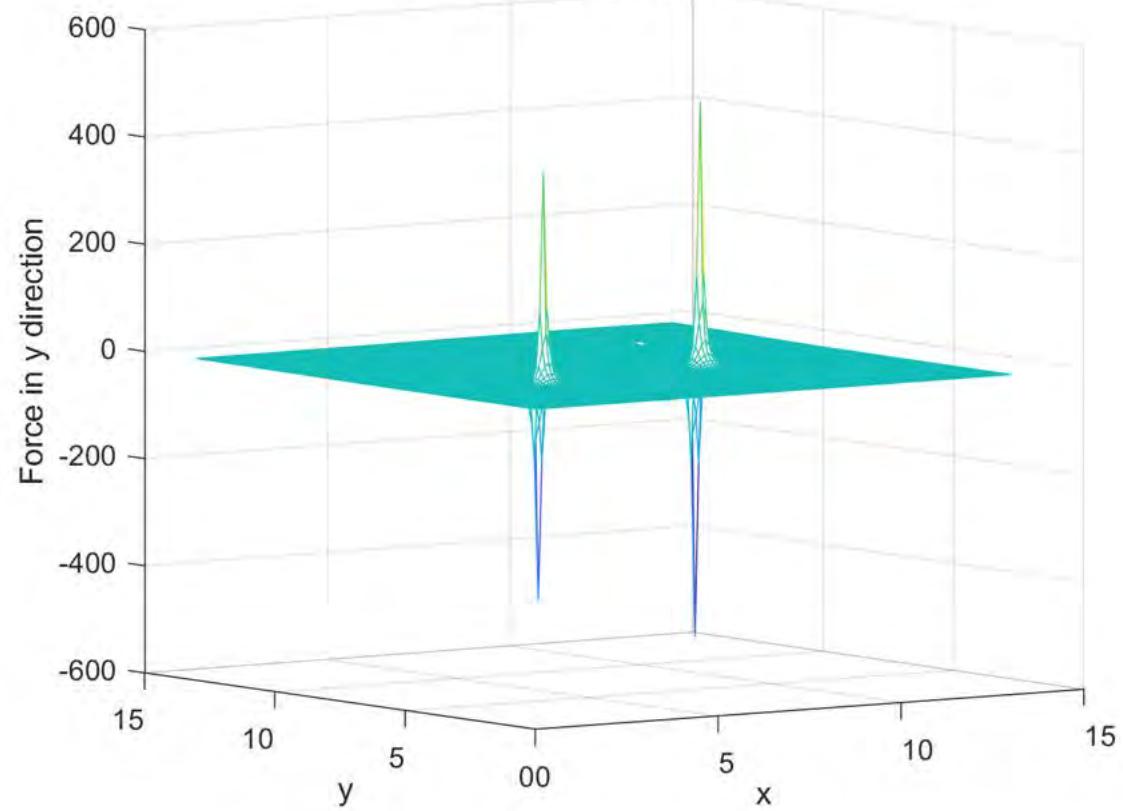
**Plot for force in x-direction**



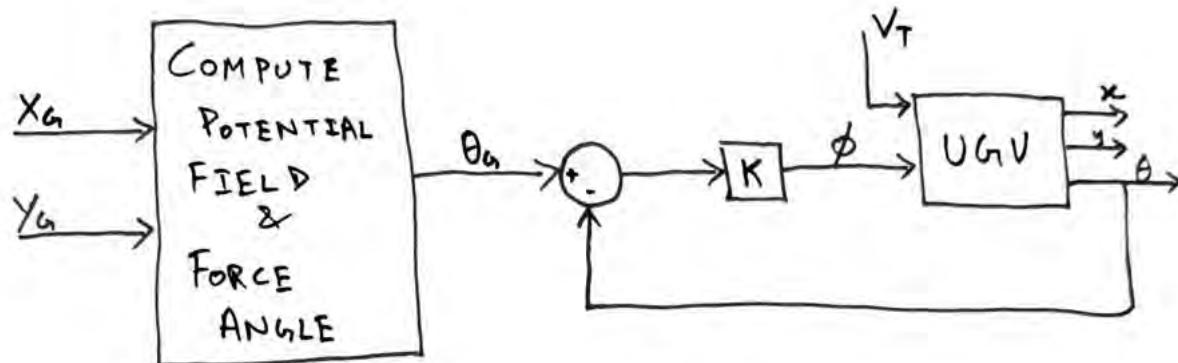
**Plot for force in y direction**



**Plot for force in y direction**



## b. Control System



### CONTROL SYSTEM

$(x_g, y_g) \rightarrow$  GOAL POSITION

$\theta_g \rightarrow$  DESIRED ANGLE

$K \rightarrow$  STEERING GAIN

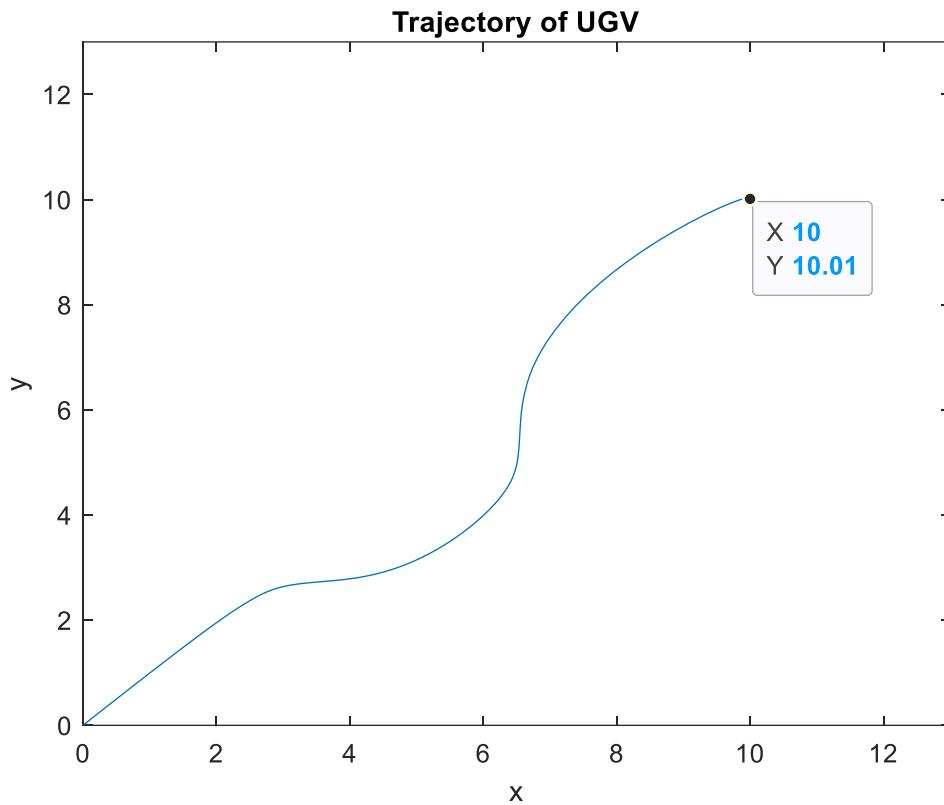
$\phi \rightarrow$  STEERING ANGLE

$v_t \rightarrow$  CONSTANT UGV VELOCITY

$(x, y) \rightarrow$  ACTUAL POSITION OF UGV

$\theta \rightarrow$  CURRENT ANGLE

### c. Simulation



### MATLAB Code

```
function hw4q2c
%work area
wa=[0:0.1:13;0:0.1:13];
%locations of obstacles and target
T=[10,10];
obs=[3,4;8,5];
n=length(wa(1,:));
%gains
kT=3;
ko=[4,5];
options=odeset('events',@StopSim);
Vp=4; kn=3; L=2;
init=[0;0;pi/4]; %initial robot condition
[t,pos]=ode45(@robot,[0 100],init,options);
figure
plot(pos(:,1),pos(:,2))
```

```

xlim([0 13]);
ylim([0 13]);

function dp=robot(t,pos)
dp=zeros(3,1);
x=pos(1);y=pos(2);theta=pos(3);
FTx=kT*(T(1,1)-x)/(sqrt(((T(1,1)-x)^2)+((T(1,2)-y)^2)));
Fo1x=-ko(1,1)*(obs(1,1)-x)/((((obs(1,1)-x)^2)+(obs(1,2)-y)^2)^1.5);
Fo2x=-ko(1,2)*(obs(2,1)-x)/((((obs(2,1)-x)^2)+(obs(2,2)-y)^2)^1.5);
Fx=FTx+Fo1x+Fo2x;
FTy=kT*(T(1,2)-y)/(sqrt(((T(1,1)-x)^2)+(T(1,2)-y)^2));
Fo1y=-ko(1,1)*(obs(1,2)-y)/((((obs(1,1)-x)^2)+(obs(1,2)-y)^2)^1.5);
Fo2y=-ko(1,2)*(obs(2,2)-y)/((((obs(2,1)-x)^2)+(obs(2,2)-y)^2)^1.5);
Fy=FTy+Fo1y+Fo2y;
thetades=atan2(Fy,Fx);
fi=kn*(thetades-theta);
dp(1)=Vp*cos(fi)*cos(theta);
dp(2)=Vp*cos(fi)*sin(theta);
dp(3)=(Vp/L)*sin(fi);

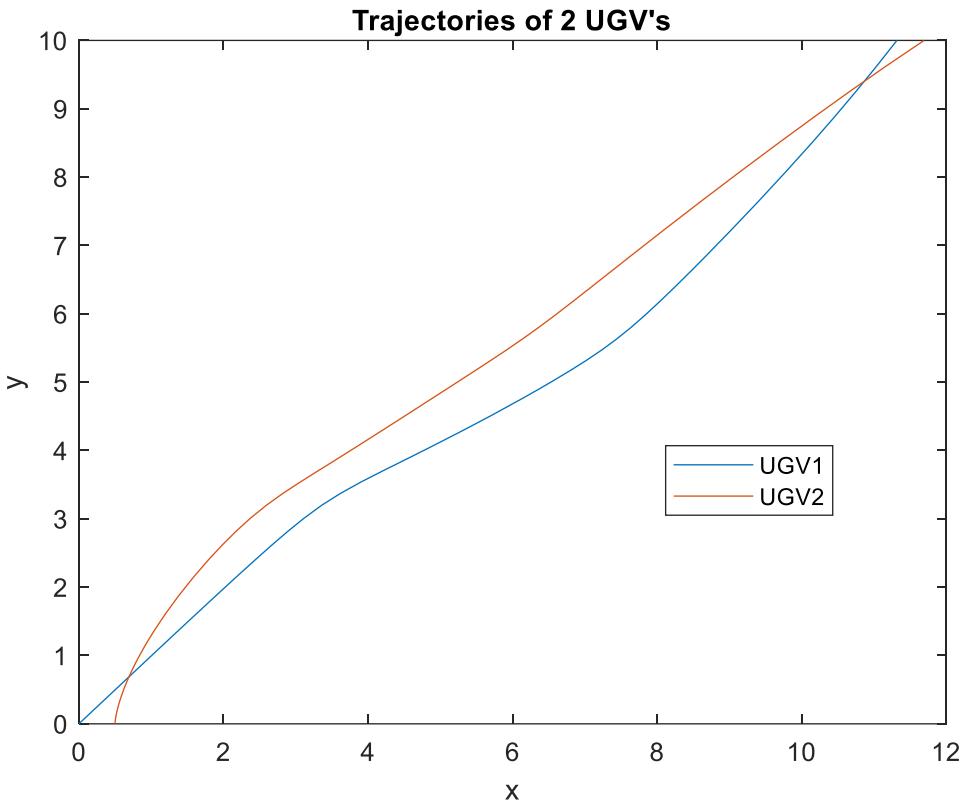
end

function [Val,Ister,Dir]=StopSim(t,pos)
Val(1)=(pos(1)-T(1,1));
Ister(1)=1;
Dir(1)=0;

end
end

```

### Problem - 3:



```
function hw4q3
%work area
wa=[0:0.1:13;0:0.1:13];
%locations of obstacles and target
T=[10,10];
obs=[3,4;8,5];
n=length(wa(1,:));
%gains
kT=3;
ko=[4,5];
options=odeset('events',@StopSim);
ktr=0.0001; m=1;
init=[0;0;0;0;0.5;0;0;0];
[t,pa]=ode45(@robot,a,[0 10],init,options);
plot(pa(:,1),pa(:,3),pa(:,5),pa(:,7))

function dpa=robot(t,pa)
dpa=zeros(8,1);
```

```

xa=pa(1);ya=pa(3); xb=pa(5); yb=pa(7);

FTax=kT*(T(1,1)-xa)/(sqrt(((T(1,1)-xa)^2)+(T(1,2)-ya)^2));
Fo1ax=-ko(1,1)*(obs(1,1)-xa)/(((obs(1,1)-xa)^2)+(obs(1,2)-ya)^2)^1.5;
Fo2ax=-ko(1,2)*(obs(2,1)-xa)/(((obs(2,1)-xa)^2)+(obs(2,2)-ya)^2)^1.5;
Frabay=-ktr*(xb-xa)/((xb-xa)^2+(yb-ya)^2)^1.5;
Fxax=FTax+Fo1ax+Fo2ax+Frabay;

FTay=kT*(T(1,2)-ya)/(sqrt(((T(1,1)-xa)^2)+(T(1,2)-ya)^2));
Fo1ay=-ko(1,1)*(obs(1,2)-ya)/(((obs(1,1)-xa)^2)+(obs(1,2)-ya)^2)^1.5;
Fo2ay=-ko(1,2)*(obs(2,2)-ya)/(((obs(2,1)-xa)^2)+(obs(2,2)-ya)^2)^1.5;
Frabay=-ktr*(yb-ya)/((xb-xa)^2+(yb-ya)^2)^1.5;
Fyay=FTay+Fo1ay+Fo2ay+Frabay;

dpa(1)=pa(2);
dpa(2)=Fxax/m;
dpa(3)=pa(4);
dpa(4)=Fyay/m;

FTbx=kT*(T(1,1)-xb)/(sqrt(((T(1,1)-xb)^2)+(T(1,2)-yb)^2));
Fo1bx=-ko(1,1)*(obs(1,1)-xb)/(((obs(1,1)-xb)^2)+(obs(1,2)-yb)^2)^1.5;
Fo2bx=-ko(1,2)*(obs(2,1)-xb)/(((obs(2,1)-xb)^2)+(obs(2,2)-yb)^2)^1.5;
Frabx=-ktr*(xa-xb)/((xa-xb)^2+(ya-yb)^2)^1.5;
Fxbx=FTax+Fo1ax+Fo2ax+Frabx;

FTby=kT*(T(1,2)-ya)/(sqrt(((T(1,1)-xa)^2)+(T(1,2)-ya)^2));
Fo1by=-ko(1,1)*(obs(1,2)-ya)/(((obs(1,1)-xa)^2)+(obs(1,2)-ya)^2)^1.5;
Fo2by=-ko(1,2)*(obs(2,2)-ya)/(((obs(2,1)-xa)^2)+(obs(2,2)-ya)^2)^1.5;
Fraby=-ktr*(ya-yb)/((xa-xb)^2+(ya-yb)^2)^1.5;
Fyby=FTay+Fo1ay+Fo2ay+Fraby;

dpa(5)=pa(7);
dpa(6)=Fxby/m;

```

```
dpa(7)=pa(8);  
dpa(8)=Fyb/m;  
end  
  
function [Val,Ister,Dir]=StopSim(t,pa)  
Val(1)=pa(3)-T(1,1);  
Ister(1)=1;  
Dir(1)=0;  
end  
end
```

## EE 5322 Homework 5

### Neural Networks

1- Consider the following training data set

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, t_1 = 0 \right\}, \left\{ \mathbf{p}_2 = \begin{bmatrix} 4 \\ -1 \end{bmatrix}, t_2 = 0 \right\}, \left\{ \mathbf{p}_3 = \begin{bmatrix} 6 \\ 0 \end{bmatrix}, t_3 = 0 \right\}, \left\{ \mathbf{p}_4 = \begin{bmatrix} 5 \\ -1 \end{bmatrix}, t_4 = 0 \right\}$$
$$\left\{ \mathbf{p}_5 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, t_5 = 1 \right\}, \left\{ \mathbf{p}_6 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, t_6 = 1 \right\}, \left\{ \mathbf{p}_7 = \begin{bmatrix} 2 \\ -2 \end{bmatrix}, t_7 = 1 \right\}, \left\{ \mathbf{p}_8 = \begin{bmatrix} 1 \\ -3 \end{bmatrix}, t_8 = 1 \right\}$$

- a- Train a perceptron to classify the data set into two classes. Plot points and decision boundaries.
- b- Use a single neuron with sigmoid activation function to do this classification problem. Plot the points and decision boundary and compare the results to part a.

2- Consider the dynamical system

$$\ddot{x} = -0.1x^3 + f(x)$$

where  $f(x) = x^2 + 2\cos(x)$ .

- a) Approximate the function  $f(x)$  using an MLP neural network and plot the function and the estimation on the same graph.
- b) Simulate the system response for exact  $f(x)$  and the approximation. Use different initial conditions. Compare the results.

**Homework Pledge of Honor** On all homeworks in this class - YOU MUST WORK ALONE. Any cheating or collusion will be severely punished. It is very easy to compare your software code and determine if you worked together Or if you found code online written by someone else. It does not matter if you change the variable names. Please sign this form and include it as the first page of all of your submitted homeworks.

.....  
Typed Name: Atul Shrotriya \_\_\_\_\_

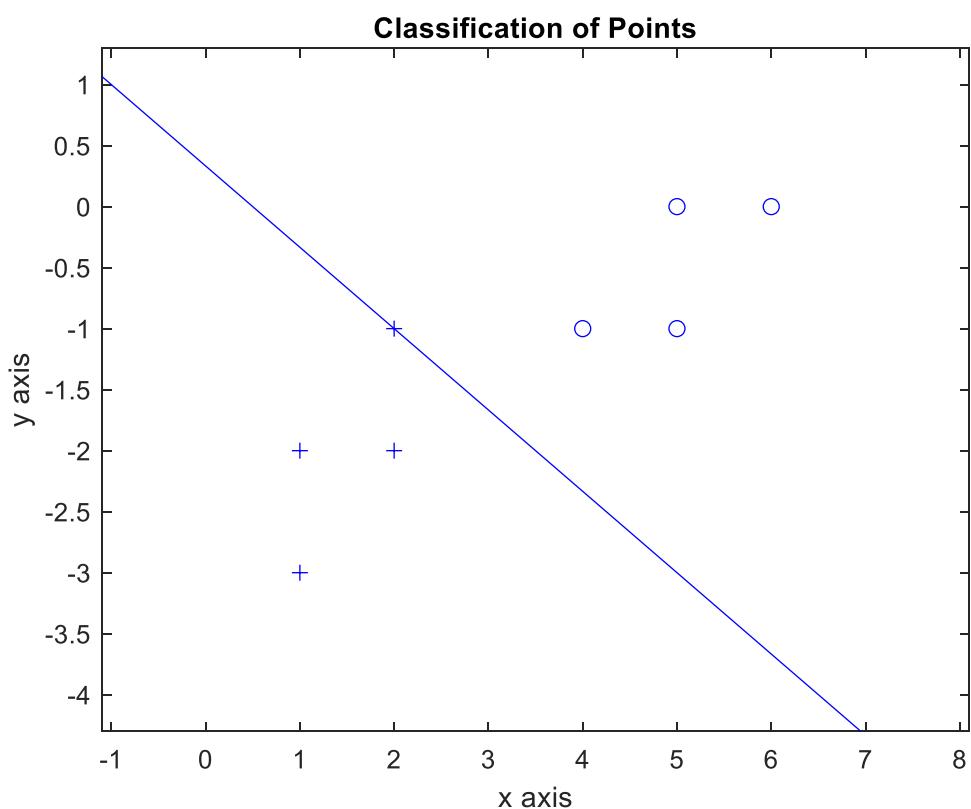
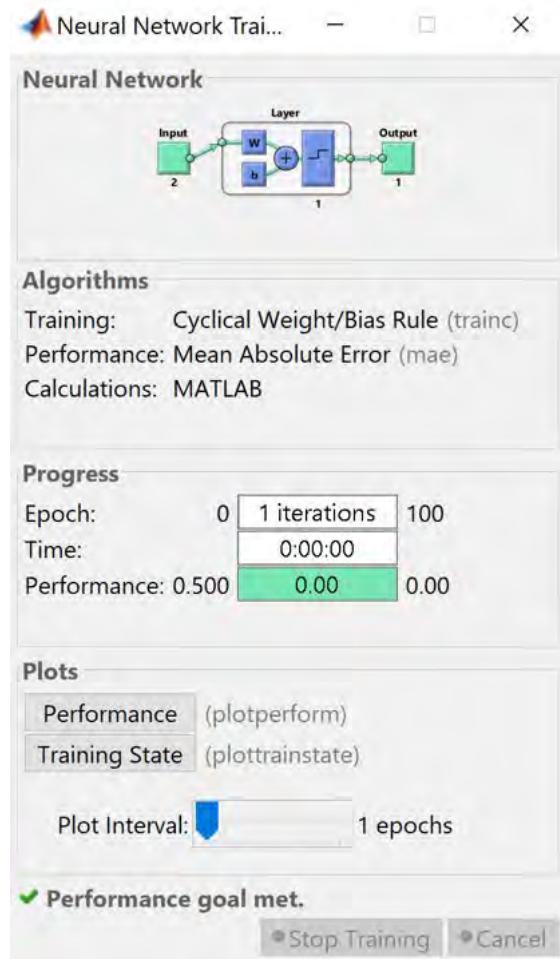
Pledge of honor: "On my honor I have neither given nor received aid on this homework."

e-Signature: Atul Shrotriya \_\_\_\_\_

## **Problem - 1:**

### **a. MATLAB Code**

```
function neuralnet1  
  
p(:,1)=[5;0];p(:,2)=[4;-1];p(:,3)=[6;0];p(:,4)=[5;-1];  
p(:,5)=[2;-1];p(:,6)=[1;-2];p(:,7)=[2;-2];p(:,8)=[1;-3];  
  
t(:,1)=[0];t(:,2)=[0];t(:,3)=[0];t(:,4)=[0];  
t(:,5)=[1];t(:,6)=[1];t(:,7)=[1];t(:,8)=[1];  
  
net1=newp(minmax(p),1);  
out=sim(net1,p)  
weight=net1.IW{1,1}  
bias=net1.b{1}  
  
net1.trainParam.epochs=100;  
net1=train(net1,p,t);  
  
out=sim(net1,p)  
weight=net1.IW{1,1}  
bias=net1.b{1}  
  
figure  
plotpv(p,t)  
plotpc(weight,bias)  
end
```



## b. Done using two similar methods

### Method 1 MATLAB code:

```
function neuranlet2
p(:,1)=[5;0];p(:,2)=[4;-1];p(:,3)=[6;0];p(:,4)=[5;-1];
p(:,5)=[2;-1];p(:,6)=[1;-2];p(:,7)=[2;-2];p(:,8)=[1;-3];

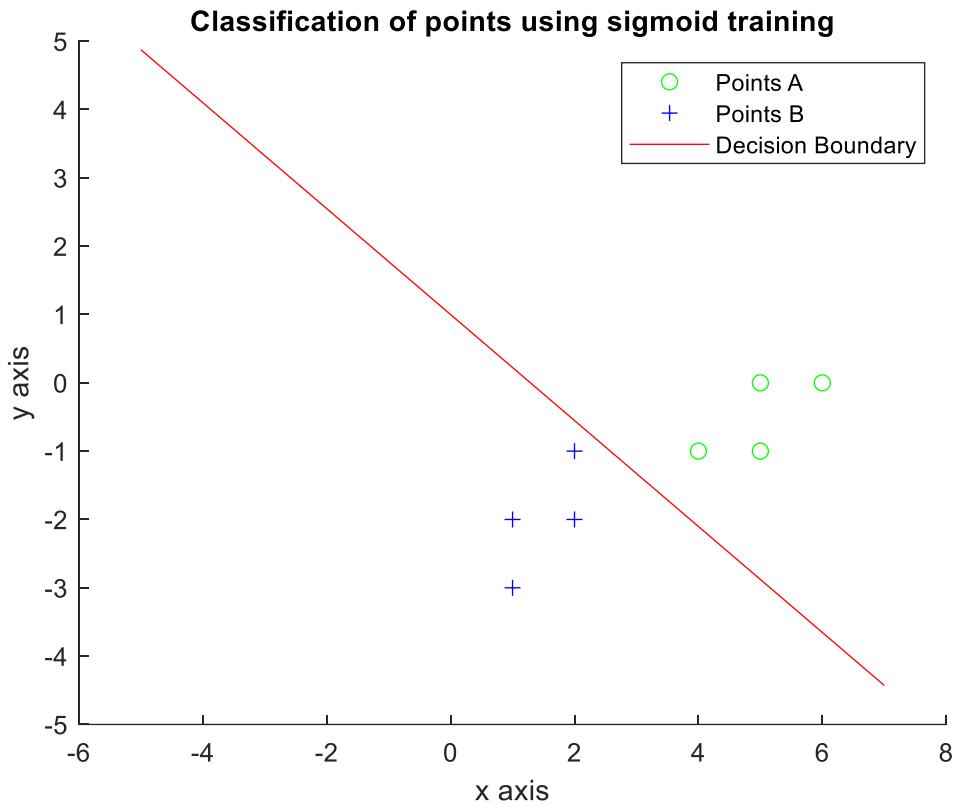
t(:,1)=[0];t(:,2)=[0];t(:,3)=[0];t(:,4)=[0];
t(:,5)=[1];t(:,6)=[1];t(:,7)=[1];t(:,8)=[1];

weight=rand(3,1);
thresh=1;
error=ones(8,1);
e=0.5;

while norm(error)>1e-5
    for k=1:8
        P=[p(:,k);thresh];
        tnet=hardlim(weight'*P);
        error(k)=t(:,k)-tnet;
        if abs(error(k))>1e-6
            weight=weight-sign(weight'*P)*e*P;
        end
    end
end

it=-5:0.1:7;
out=(-weight(1)*it-weight(3))/weight(2);
hold on
plot(p(1,1:4),p(2,1:4),'go',p(1,5:8),p(2,5:8),'b+')
plot(it,out,'r')
end
```

**Note:** The slope of the decision boundary can be altered using the it values



## Method 2: MATLAB Code

```

function neuralnet5
p=[5 4 6 5 2 1 2 1;0 -1 0 -1 -1 -2 -2 -3];
D=[5 4 6 5 2 1 2 1;0 -1 0 -1 -1 -2 -2 -3]';
target=[0 0 0 0 1 1 1 1]';
[m, n] = size(D);
D = [ones(m, 1) D];
Vin = zeros(n + 1, 1);
[cost, gradient] = costFn(Vin, D, target); %costFn defined in Appendix
%Initial cost and gradient
fprintf('Initial Cost %f', cost);
fprintf('\n Initial Gradient \n');
fprintf(' %f \n', gradient);
%Using fminunc for gradient descent
options = optimset('GradObj', 'on', 'MaxIter', 10);
[V, cost] = fminunc(@(t)(costFn(t, D, target)), Vin, options);

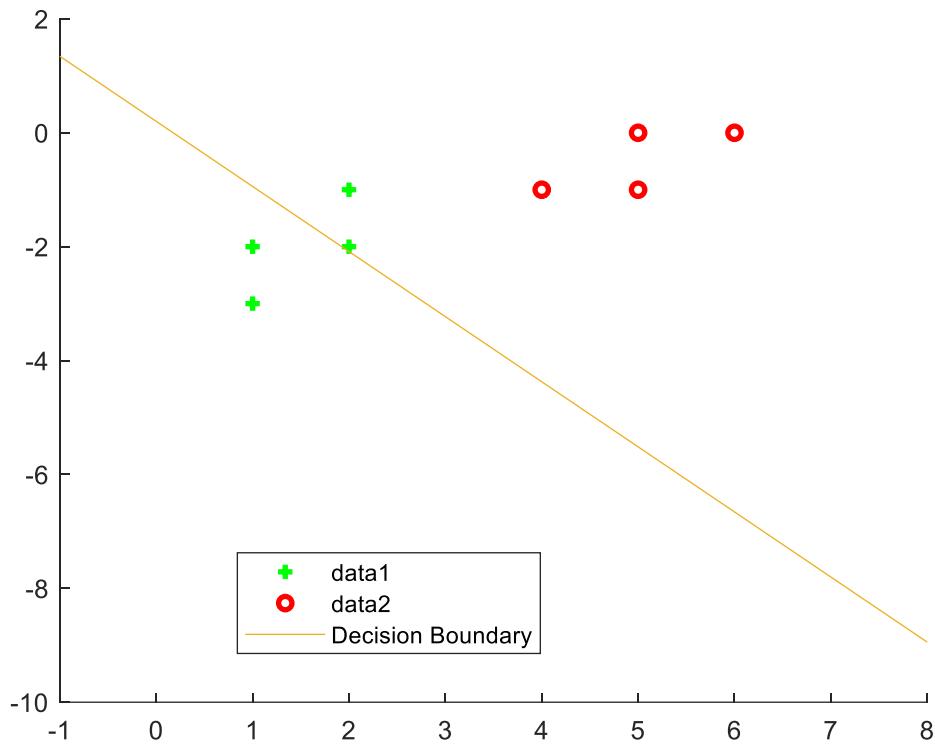
```

```

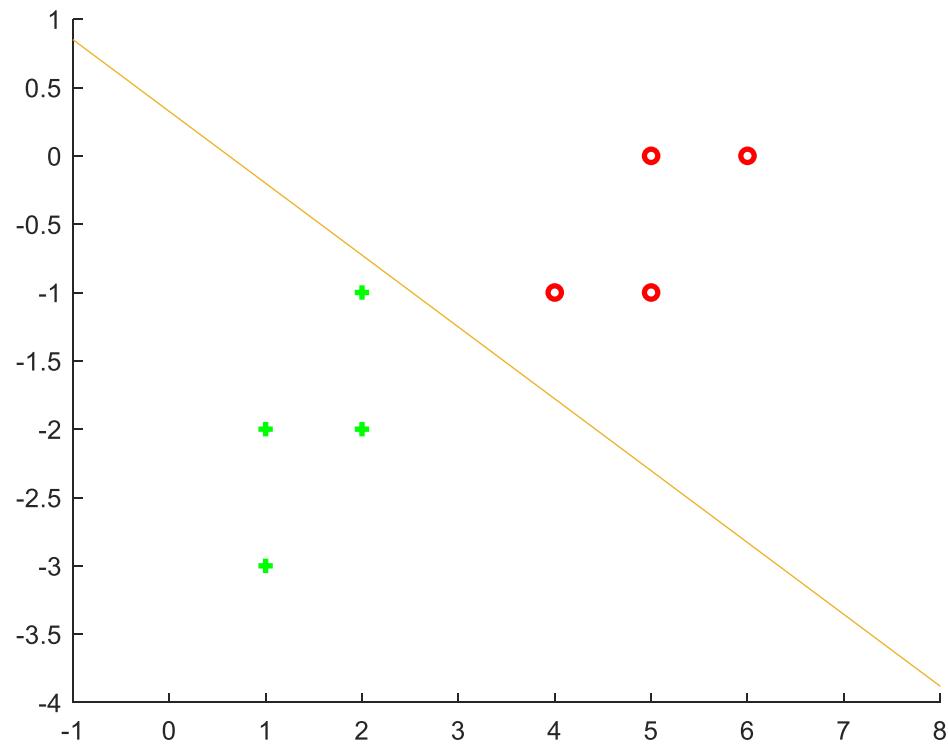
% Final cost
fprintf('Cost at final weights %f\n', cost);
fprintf('Weights and threshold: \n');
fprintf(' %f \n', V);
function plotDB(Vw, D, target)
    function plotD(x,y)
        figure;
    end
    plotD(D(:,1:2), target);
    hold on
    plot(p(1,5:8),p(2,5:8), 'g+', 'LineWidth', 2, 'MarkerSize', 5)
    plot(p(1,1:4),p(2,1:4), 'ro', 'Linewidth', 2, 'MarkerSize', 5)
    if size(D, 2) <= 3
        %evaluating and plotting the boundary line
        xa = [min(D(:,2))-2, max(D(:,2))+2];
        ya = (-1./Vw(3)).*(Vw(2).*xa + Vw(1));
        plot(xa, ya)
    else
        u = linspace(-10, 0.1, 200);
        v = linspace(-10, 0.1, 200);
        z = zeros(length(u), length(v));
        for i = 1:length(u)
            for j = 1:length(v)
                z(i,j) = mapFeature(u(i), v(j))*Vw;
            end
        end
        z = z';
        contour(u, v, z, [0, 0], 'LineWidth', 2)
    end
    hold off
end
plotDB(V, D, target);
end

```

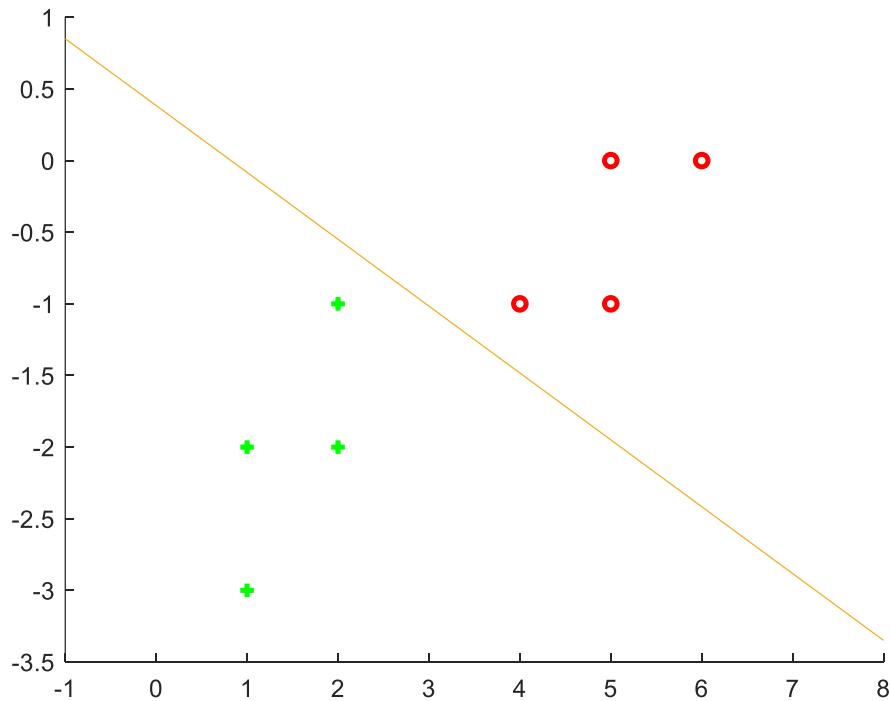
**Plot of Decision Boundary after 1 iteration of fminunc**



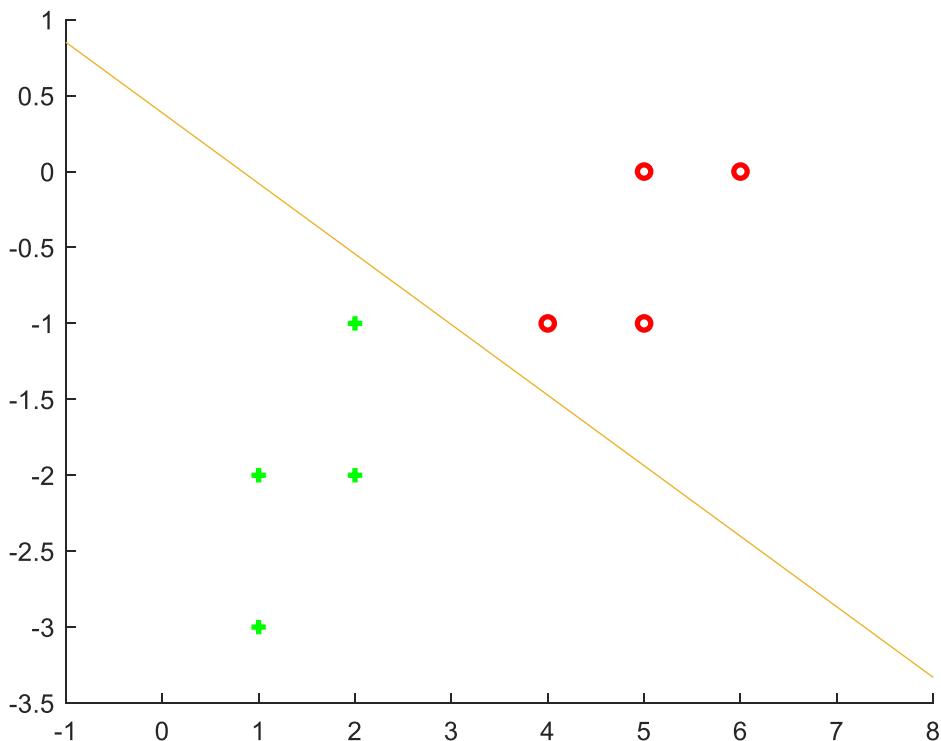
**Plot of Decision Boundary after 3 iterations of fminunc**



**Plot of Decision Boundary after 10 iterations of fminunc**



**Plot after 15 iterations, mimima reached**

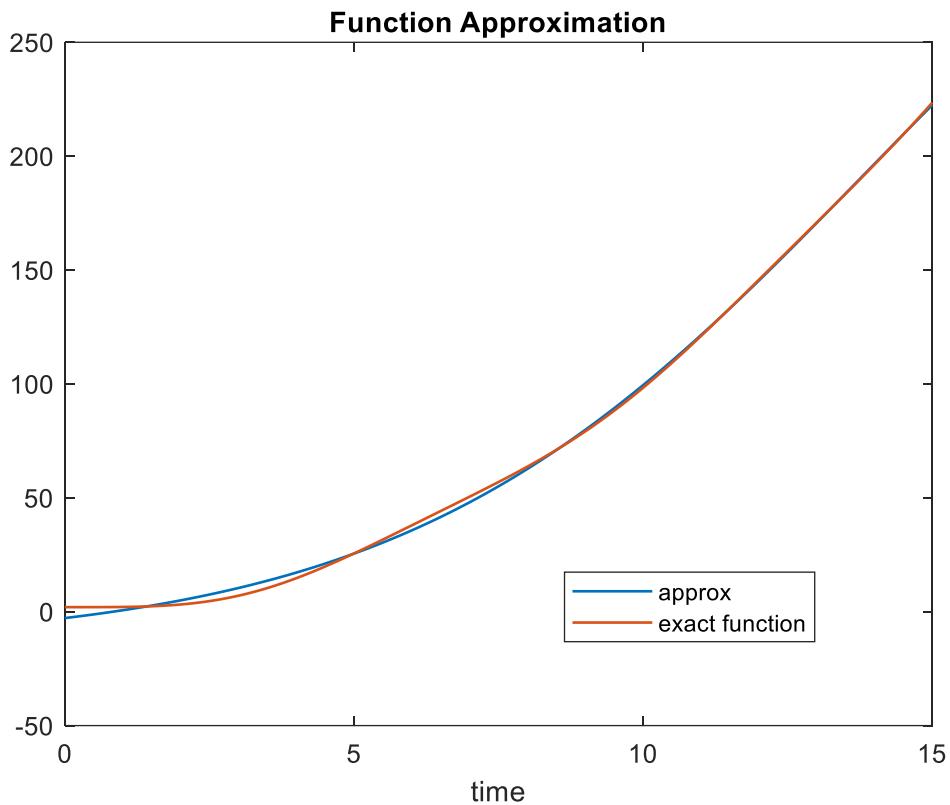


**Conclusion** - It is apparent that the second method is better at classification because one point in part a almost touches the decision boundary. This does not happen in part b once the appropriate minima location is reached. Thus, part b provides better results.

## Problem - 2:

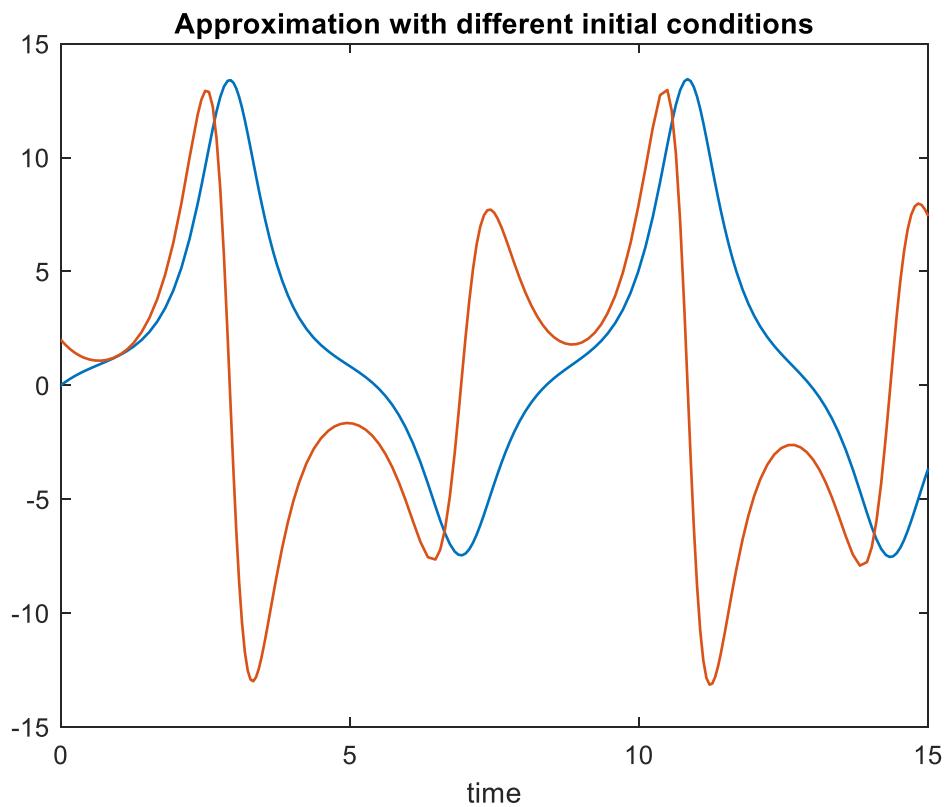
### a. MATLAB Code:

```
function approx1
x=0:0.01:15;
fun=(x.^2)+2*cos(x);
approx=feedforwardnet(1);
approx.trainParam.epochs=500;
approx.layers{1}.transferFcn='logsig';
approx.trainParam.goal=0.01;
approx=train(approx,x,fun);
nefn=approx(x);
plot(x,nefn,x,fun)
end
```



## b. MATLAB Code

```
function approx1
x=0:0.01:15;
fun=(x.^2)+2*cos(x);
approx=feedforwardnet(1);
approx.trainParam.epochs=500;
approx.layers{1}.transferFcn='logsig';
approx.trainParam.goal=0.01;
approx=train(approx,x,fun);
nefn=approx(x);
plot(x,nefn,x,fun)
approximate=@(t,x)[x(2);(-0.1*x(1)^3+sim(approx,x(1)))];
[time,fn]=ode45(approximate,[0 15],[0 2]);
plot(time,fn);
end
```



## Appendix

### CostFn.m

```
function [J, gradient] = costFn(angle, X, target)
m = length(target);
J = 0;
gradient = zeros(size(angle));
J=1./m*sum((-target).*log(1./(1+exp(-angle'*X'))))-((1-target).*log(1-1./(1+exp(-angle'*X'))));
gradient=1./m*(1./(1+exp(-angle'*X')) -target')*X;
end
```

## References

1. Coursera Machine Learning by Andrew NG

**EE 5322 Exam 2**  
**Adaptive Control, Robust Control**

1. A system is given by

$$\dot{x} = Ax + Bu = \begin{bmatrix} 0 & 1 \\ -25 & -2 \end{bmatrix}x + \begin{bmatrix} 0 \\ 1 \end{bmatrix}u, \quad y = Cx = \begin{bmatrix} 1 & 0 \end{bmatrix}x$$

- a. Find poles and transfer function
- b. Simulate the system with  $u(t) = \text{unit step}$  for 10 sec.

2. Write the system as  $\ddot{y} + a_1\dot{y} + a_2y = u$  and assume the coefficients are unknown.

Design an adaptive controller as described in class. Use desired trajectory of  $5u_{-1}(t)$ , that is 5 times the unit step. Here are some hints -

Write the system error dynamics as  $\dot{r} = f(x) - v$  with  $f(x) = a_1\dot{y} + a_2y = [a_1 \quad a_2] \begin{bmatrix} y \\ \dot{y} \end{bmatrix} = W^T \phi(x)$

Tune the unknown parameter vector using  $\dot{\hat{W}} = F\phi(x)r^T$ .

To simulate, take states 1 and 2 as the system dynamics. You will need to define states 3 and 4 to be the adaptive controller dynamics.

3. Write the system as  $\ddot{y} + a_1\dot{y} + a_2y = u$  with unknown coefficients same as above values. Assume the estimated coefficients are  $\hat{a}_1 = 1, \hat{a}_2 = 20$ .

Design a robust controller as described in class. Use desired trajectory of  $5u_{-1}(t)$ , that is 5 times the unit step. Here are some hints -

Write the estimate  $\hat{f}(x) = \hat{a}_1\dot{y} + \hat{a}_2y = [\hat{a}_1 \quad \hat{a}_2] \begin{bmatrix} y \\ \dot{y} \end{bmatrix} = \hat{W}^T \phi(x)$

Find the estimation error  $\tilde{f}(x) = f(x) - \hat{f}(x) = [a_1 \quad a_2] \begin{bmatrix} y \\ \dot{y} \end{bmatrix} - [\hat{a}_1 \quad \hat{a}_2] \begin{bmatrix} y \\ \dot{y} \end{bmatrix}$

Take its known norm bound as  $F(x) = [1 \quad 5] \begin{bmatrix} y \\ \dot{y} \end{bmatrix}$

The controller you will implement in MATLAB is

$$\tau = \hat{f}(x) + K_v r - v_r$$

$$v_r = \begin{cases} -r \frac{F(x)}{\|r\|}, & \|r\| \geq \varepsilon \\ -r \frac{F(x)}{\varepsilon}, & \|r\| < \varepsilon \end{cases}$$

It is nondynamic and has no states.

## **EE 5322 Intelligent Control- Exam 2**

**Fall 2020**

1. This is a take home exam. YOU MUST WORK ALONE.

***Any cheating or collusion will be severely punished.***

2. To obtain full credit, show all your work. No partial credit will be given without the supporting work.
3. Please sign this form and include it as the first page of your submitted exam.

.....  
Typed Name: ATUL SHROTRIYA

Pledge of honor: "On my honor I have neither given nor received aid on this examination."

e-Signature: ATUL SHROTRIYA

## Problem - 1

### MATLAB Code

```
function systemsimulation
u=1;
A=[0 1;-25 -2];
B=[0;1];
C=[1 0];
D=0;
[Num,Den]=ss2tf(A,B,C,D);
G=tf(Num,Den)
damp(G)
[t,x]=ode45(@system,[0 10],[0 0]);
plot(t,x(:,1))

function dx=system(t,x)
dx=zeros(2,1);
dx(1)=x(2);
dx(2)=-25*x(1)-2*x(2)+u;
end
end
```

### a. Poles and transfer function

```
G =
1
-----
s^2 + 2 s + 25
```

Continuous-time transfer function.

Pole	Damping	Frequency	Time Constant
		(rad/seconds)	(seconds)
-1.00e+00 + 4.90e+00i	2.00e-01	5.00e+00	1.00e+00
-1.00e+00 - 4.90e+00i	2.00e-01	5.00e+00	1.00e+00

Here, the transfer function is G and the poles are  $-1 \pm 4.9i$

$$\dot{x} = Ax + Bu = \begin{bmatrix} 0 & 1 \\ -25 & -2 \end{bmatrix}x + \begin{bmatrix} 0 \\ 1 \end{bmatrix}u$$

$$y = Cx = \begin{bmatrix} 1 & 0 \end{bmatrix}x$$

Solving these, we get

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -25x_1 - 2x_2 + u$$

Substituting, we get

$$\ddot{x}_1 = -25x_1 - 2\dot{x}_1 + u$$

$$\Rightarrow \ddot{x}_1 + 2\dot{x}_1 + 25x_1 = u$$

Taking the Laplace Transform, we get:

$$X_1(s) = \frac{u(s)}{s^2 + 2s + 25}$$

Transfer function is

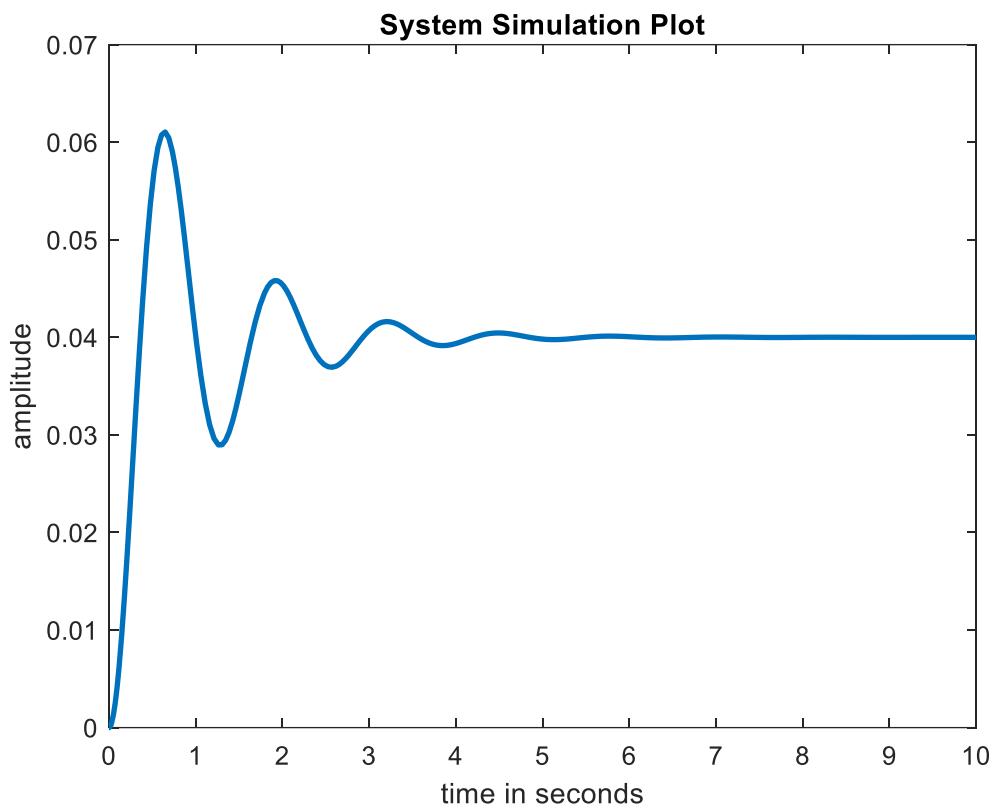
~~$$\text{Output}(s) = \frac{X_1(s)}{u(s)} = \frac{1}{s^2 + 2s + 25}$$~~

Poles are the roots of the denominator,

Solving for the roots, we get them as

$$\boxed{-1 \pm 2i\sqrt{6}}$$

## b. System Simulation



The plot is correct as the steady state gain is  $1/25$  for this system.

## Problem - 2

### MATLAB Code

#### Adaptive Controller

```
function dx = adaptivecontrol(t,x)
yd=5;      %givens
lam=10; Kv=30; Fx=10*eye(2); %controller parameters
pyd=0;
ppyd=0;

%tracking errors
e=yd-x(1);
pe=pyd-x(2);
r=pe+lam*e;

phi=[x(1);x(2)];
W=[x(3);x(4)];

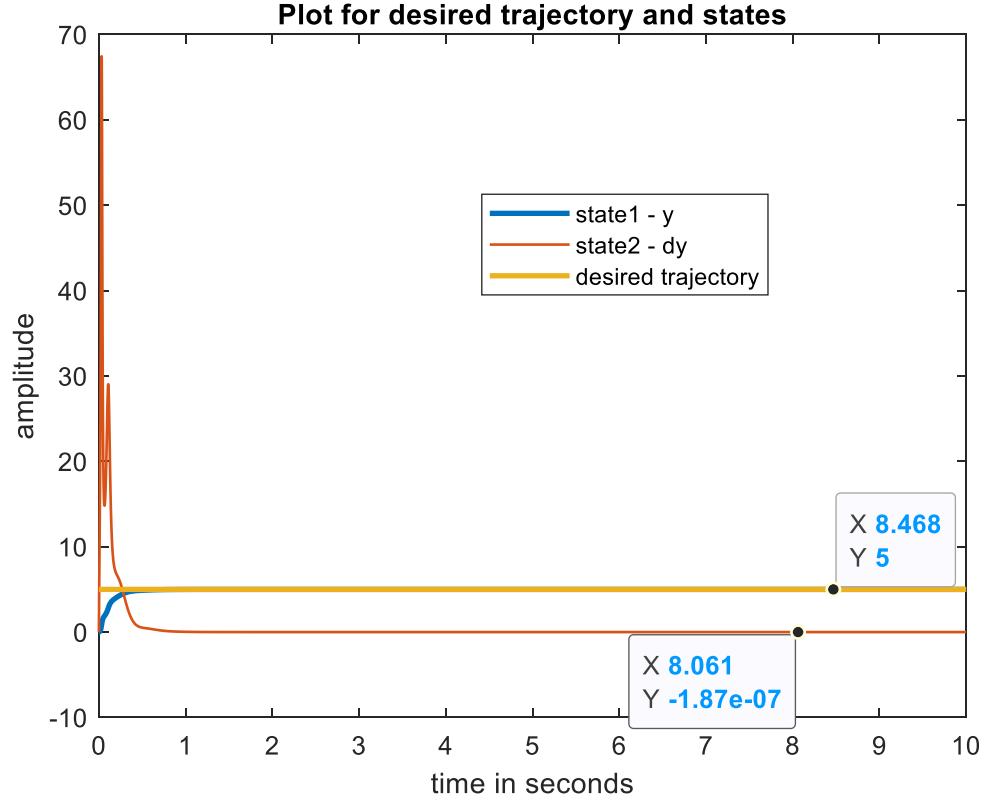
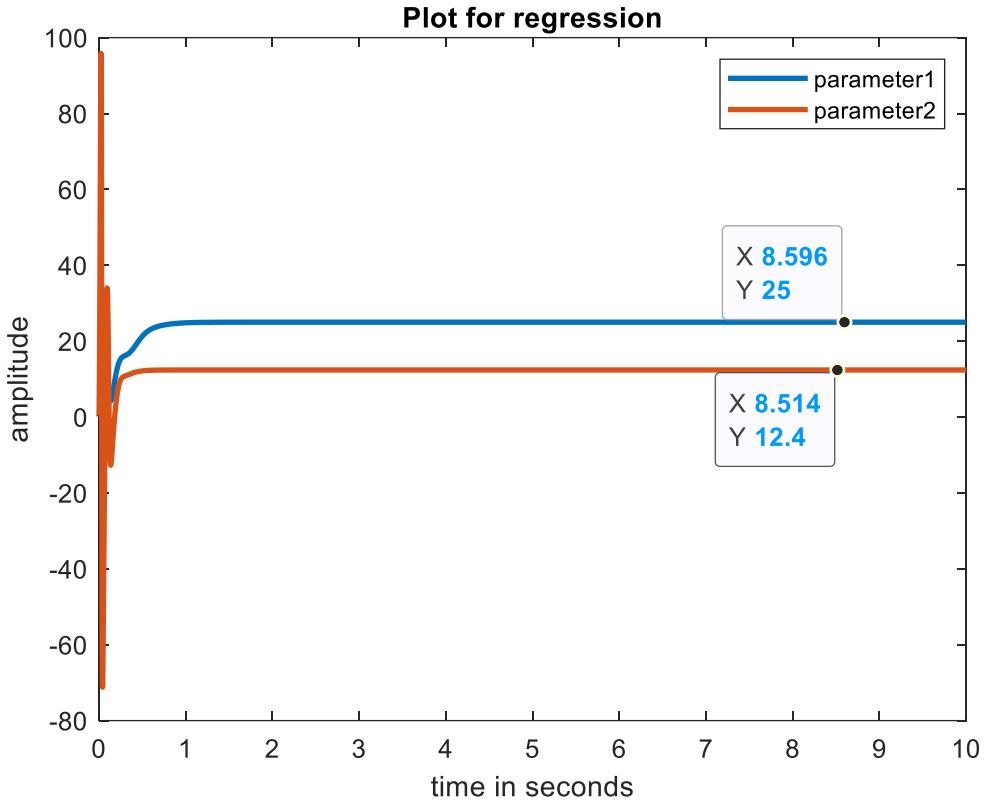
%regression
v=W'*phi+Kv*r;
u=v+ppyd+lam*pe;
dx(1)=x(2);
dx(2)=-25*x(1)-2*x(2)+u;

%parameter update
dW=Fx*phi*r;
dx=[dx(1);dx(2);dW];
end
```

#### Adaptive Controller Application

```
function adaptivecontrolapplication
yd=5;
[t,x]=ode45('adaptivecontrol',[0 10],zeros(4,1));
figure()
plot(t,x(:,3),t,x(:,4))
figure()
plot(t,x(:,1),t,x(:,2))
hold on
plot(t,5*ones(size(t)))
end
```

## Output Plots



From the second plot for desired trajectory and states, it is observed that the error goes to zero and we end up on the desired trajectory which is 5 in this case.

## Problem - 3

### MATLAB Code

#### Robust Controller

```
function dx=robustcontrol(t,x)
W=[1;20]; yd=5; %givens
lam=10; Kv=30; %controller parameters
phi=[x(1);x(2)];
Fx=[1 5]*phi; %bounding function
pyd=0;
ppyd=0;

%tracking errors
e=yd-x(1);
pe=pyd-x(2);
r=pe+lam*e;

%control input
E=0.05;
div=max([norm(r) E]');
v=-r*Fx/div;
T=W'*phi+Kv*r-v;
u=T+ppyd+lam*pe;
dx(1)=x(2);
dx(2)=-25*x(1)-2*x(2)+u;
dw=Fx*phi*r;
dx=[dx(1);dx(2);dw];
end
```

#### Robust Controller Application

```
function robustcontrolapplication
yd=5;
[t,x]=ode45('robustcontrol',[0 10],zeros(4,1));
```

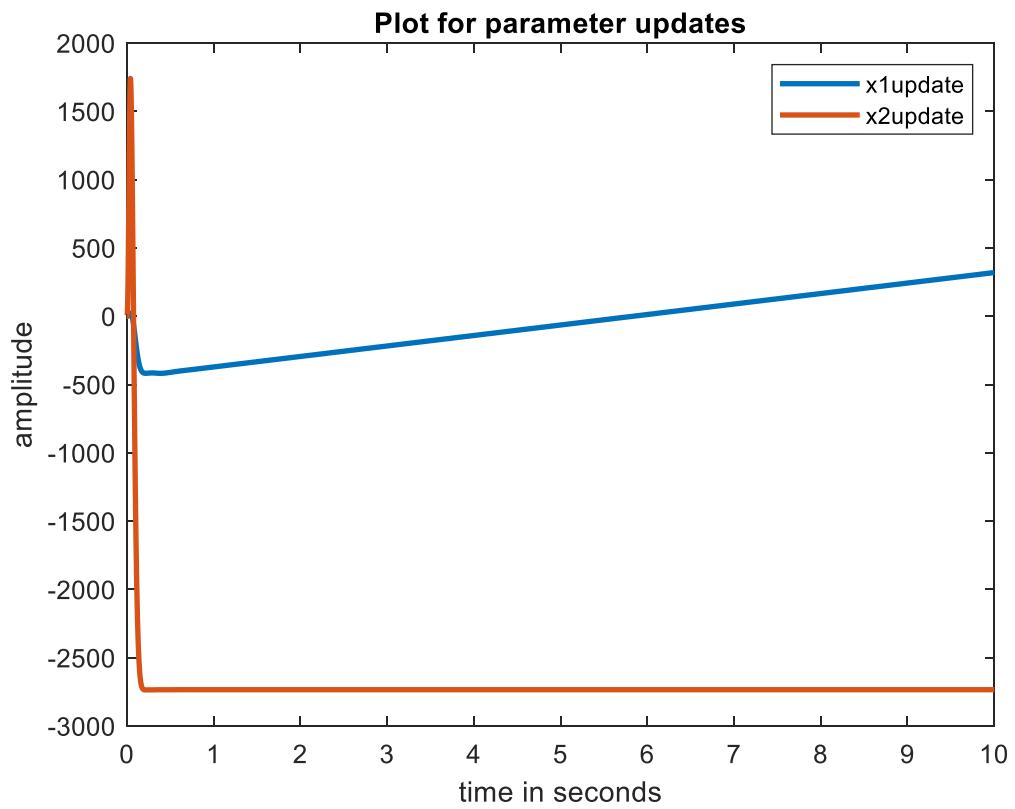
```

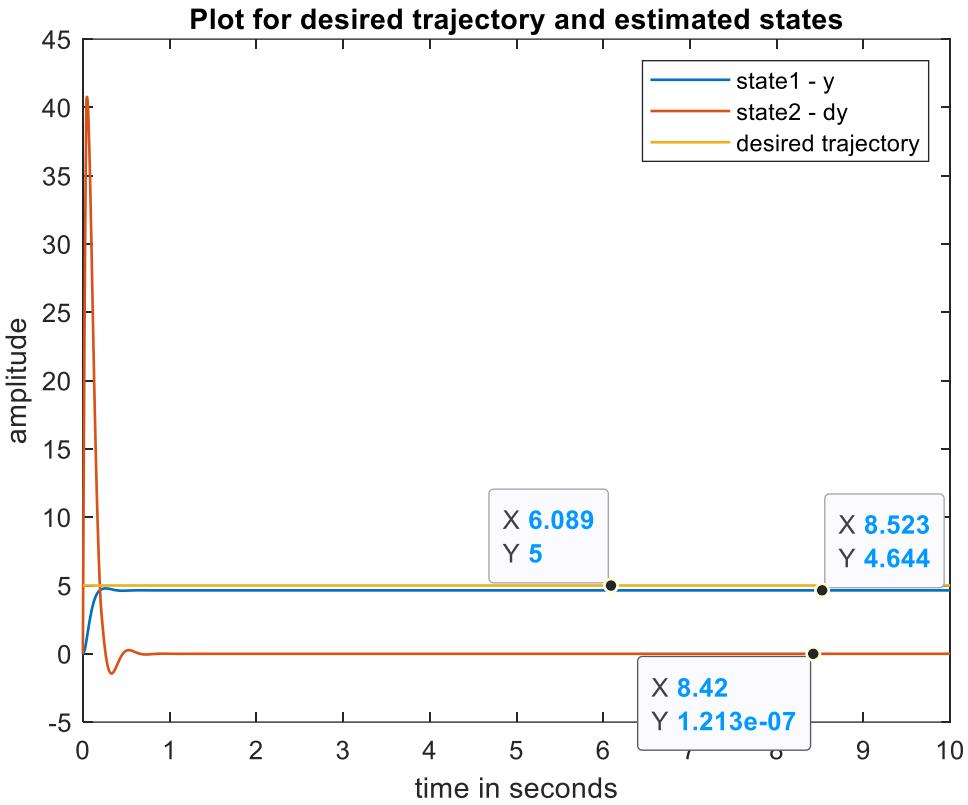
figure(1)
plot(t,x(:,3),t,x(:,4))

figure()
plot(t,x(:,1),t,x(:,2))
hold on
plot(t,5*ones(size(t)))
end

```

## Output Plots





From the plot for desired trajectory and estimated states, it is observed that the error does not go to zero and we end up close to the desired trajectory (5 in this case) but not on the exact trajectory as seen in adaptive control.

## **References**

1. Hu, Chenyuan – Doctoral Student, Electrical Engineering Department, University of Texas at Arlington
2. Dynamic Systems Modeling – Dr. David A. Hullender, Professor, Department of Mechanical and Aerospace Engineering, The University of Texas at Arlington
3. Engineering Analysis – Dr. Seiichi Nomura, Professor, Department of Mechanical and Aerospace Engineering, The University of Texas at Arlington
4. System Identification and Estimation – Dr. Michael A. Niestroy, Other Faculty, Electrical Engineering Department, University of Texas at Arlington
5. Intelligent Control Systems – Dr. Frank Lewis, Professor, Electrical Engineering Department, University of Texas at Arlington

# Comparison of different neural and fuzzy models for recognizing vowels within uncontrolled environments

Atul Shrotriya

Department of Mechanical and Aerospace Engineering

The University of Texas at Arlington

Arlington, United States

atul.shrotriya@mavs.uta.edu

**Abstract**—Speech recognition has long been used and implemented in controlled environments. It is quickly becoming mainstream with the advancements in deep learning techniques. This paper compares different basic neural and fuzzy logic models to access which basic techniques are best suited for recognizing vowels which are the core of speech processing in many methodologies. With conventional equipment used for recording in an uncontrolled environment, this paper defines the challenges which can be faced in developing reliable and robust models without using large amounts of pre-recorded data. It is found that fuzzy systems with 3 generalized bell membership functions in two middle layers provide the closest outputs to the expected values.

## I. INTRODUCTION

Speech recognition is quickly becoming mainstream in the today's world with a wide variety of devices understanding the user's words for different purposes. An audio interface allows the users to keep their hands and eyes free and focus on other tasks. [8] Although speech recognition has existed and been used for many years in controlled environments, recent developments in the field of deep learning has allowed it to be included in the mainstream use with the help of conventional devices. [9] The availability of large amounts of data allows various organizations to train the deep learning models which can provide a good accuracy even in the presence of noise. These models are regularly updated with edge cases to include different accents and speech patterns used by users. [8,9]

A basic part of speech recognition is vowels. These can be analyzed to check different factors for example, the same vowel may be spoken differently by the same person under different conditions. [10] Vowels can also be accessed for improving speech [11]. Improvement in the basic ASR pipeline is made by replacing existing machine learning models with deep learning models [10]. This paper aims to create different models using feedforward neural networks with 3 and 10 neurons and fuzzy logic systems with triangular and generalized bell membership functions. These methods are chosen as they can encompass a variety of basic

techniques. Determining this will help in applying deep learning models in a robust method and provide more accurate outputs.

## II. VOWEL RECOGNITION USING UNIFORM DATA

Voice recorded in a controlled environment is often different than natural voice recorded by usual microphones. Upon plotting, it can be observed that the amplitudes are very similar with the bin size remaining constant.

### A. *Determination of Vowel Sequence using Visual Inspection*

The audio files can be easily extracted in the form of information which is understandable by MATLAB. This can be achieved through the use of audioread() command. For this step, the data was already available in an excel sheet [1]. The following steps were taken to recognize the vowels:

1. The signal is divided into 8 equal bins of 1 second each. With each bin containing 1000 data points
2. Discrete Fourier transform (DFT) is done on each bin using the fft() command.[2] The number of points used for calculating the Fourier transform should be a power of 2 [3] and is hence taken as 1024 i.e.,  $2^{11}$ . The sampling rate was taken as 0.001 seconds or 1 millisecond.
3. Absolute values of the FFT are plotted for half the frequencies as it repeats [4].
4. Peaks can be easily observed on the plots and the top two peaks mark the formant frequencies required to determine the sequence of vowels.

### B. *Determination of vowel sequence using findpeaks command*

MATLAB has a useful findpeaks() command which allows to determine the peaks within a dataset. Due to the presence of noise, there are a lot of peaks however, these can

be separated easily by applying a magnitude threshold of 200 for consideration.

Based on the sequence of peaks, a threshold value of 10 is applied to the frequency to determine the vowel sequence. The `findpeaks()` function provides peaks that are slightly shifted to higher frequencies. Thus, only the higher end needs to be considered for recognizing the vowel.

### III. VOWEL RECOGNITION USING NON-UNIFORM DATA

Non-uniform data was recorded for all vowels A,E, I, O and U with 10 samples of each vowel. The samples were varied in pitch and length to observe the effects on detection. The following steps were taken to analyze the samples.

#### A. Preparation of Training Data for the networks

1. Audio file was converted into a recognizable matrix format for MATLAB
2. The audio recorded was in stereo format. Convolution of audio was done to get a single matrix. Other software and techniques can also be used to convert stereo signal to mono if the sampling rate is different due to any reason.
3. Discrete Fourier Transform (DFT) was done on the new data using the `fft()` command. Different number of points were tried to calculate the FFT. Some files provided a flat graph when the number of points was below 4096 or  $2^{13}$ . To ensure no loss in data a minimum of 8192 points were required. However, this resulted in some noise being magnified and thus 16384 points were used.
4. Signals of different length, cannot be stored in the same matrix without the addition of zero values. Another method was used to avoid adding synthetic data to the original signal. Here, each matrix was created differently and called through a loop using cell arrays.
5. Comparison of Power Spectral Density (PSD) plots was done between convoluted data and by neglecting the second channel. Both helped in filtering the data but magnified the differences in amplitudes of recordings due to squaring of data.
6. The `findpeaks()` command, was tried again to separate the peaks. It failed due to the following reasons, the threshold value on magnitudes could not be applied as some magnitudes of some files were so small that they were less than the noise of other files. There were multiple peaks near the formant frequencies. It may be possible to introduce another threshold to neglect values if they are within a certain frequency range. To evade the issue with varying magnitudes, normalization can be applied.
7. Sample rate, number of points were changed to compare their effects on the plots. Reducing the sample rate reduced the difference between formant

frequencies and thus, sample rate was kept at 8000 samples to discern the frequencies aptly. Reducing the number of points used for calculating the FFT resulted in loss of data and was thus kept at 16384.

8. The sample size was standardized at 15000 points because it was able to collect most of the relevant information. It is not feasible to do a discrete Fourier transform on the entire data because it identifies the frequency of consistent background noise. While most noise is ideally considered white noise, the region where the non-uniform data was recorded was not controlled. Thus there were consistent frequencies from the exhaust fans which defeats the purpose discrete Fourier transform irrelevant

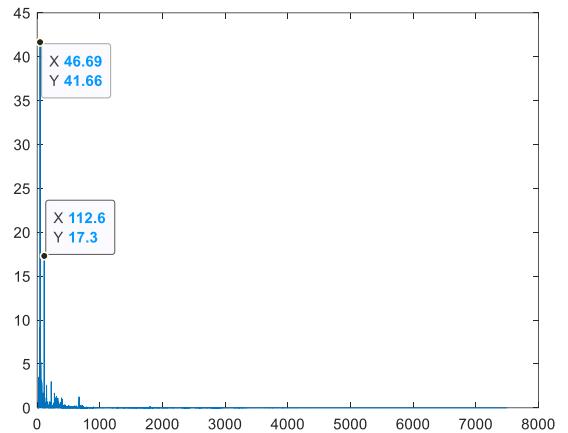


Figure 1. Discrete Fourier Transform (DFT) recognizing exhaust fan frequencies from the background noise

9. The start of points had to be identified visually because the vowels were spoken at different times with different tones and volumes. For comparison, both Fig. 2 and Fig. 3 consist of the same vowel "A" but have vastly varying amplitudes as observed.

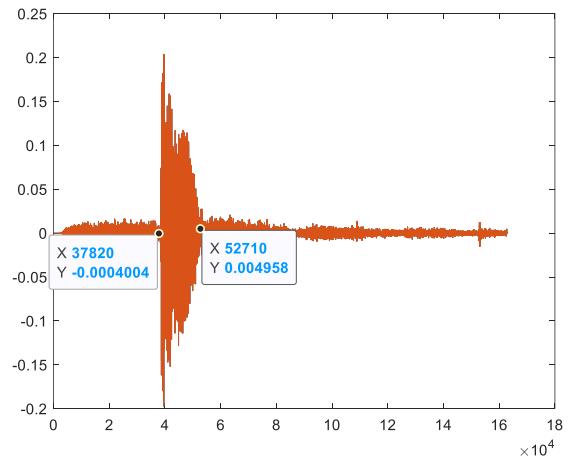


Figure 2. Plot of recording of vowel "A" with amplitudes reaching till 0.2 in magnitude

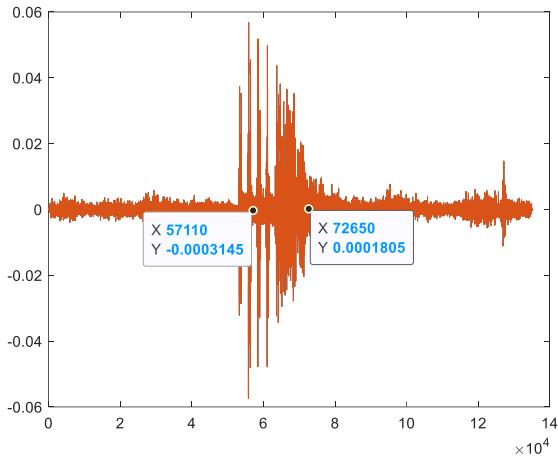


Figure 3. Plot of recording of vowel “A” with amplitudes reaching only till 0.06 in magnitude

10. Data from different channels both channels was compared and it was observed that they overlap closely. Thus, one channel was neglected and only one channel was retained for performing the discrete Fourier transform (DFT). It was also observed that by using a regular laptop microphone, there were some glitches in recording. As seen in Figure 3, the voice was recorded in fragmented intervals. Thus, these were neglected and data was taken once the signal was stable as shown in Figure 4.
11. Formants were identified using visual inspection method. Since the data set was brought to the same size, it was put into the same matrix for ease of processing.

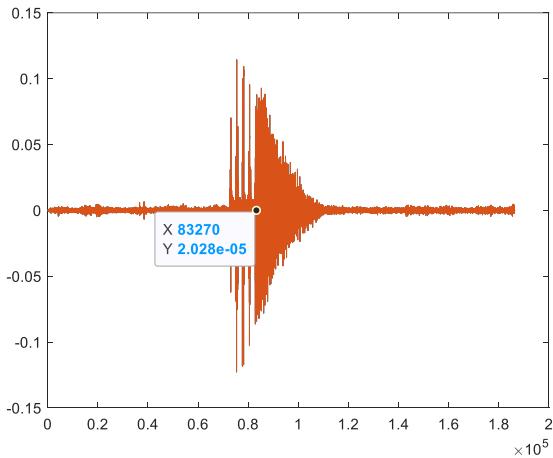


Figure 4. Starting point for considering data in case of fragmented recording

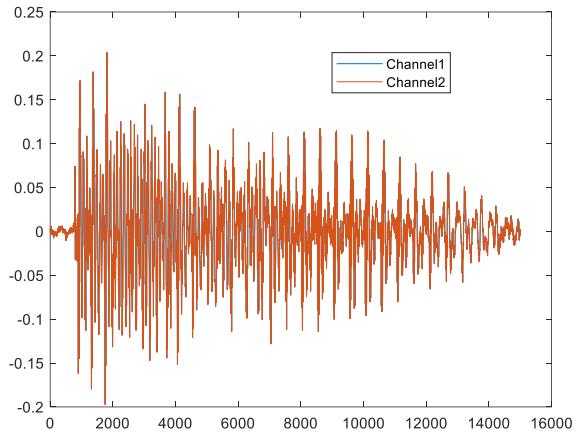


Figure 5. Plot of left and right channels of recording showing a perfect overlap of the signals

12. The peaks of were entered into the training matrix and a corresponding target vector was also formed. These were used for training the neural networks described next in this paper. The middle of the peaks were considered to obtain the average frequency of the vowel formants. The varying pitch, styles and amplitudes of the recorded vowels cause a variation of around 25 percent in the formant frequencies. It was also observed that the comparison of amplitudes at the formant frequencies is reversed depending on how the vowel was spoken or which parts were stressed on. The maximum variation was observed in the first formant of the vowel “I” with the variation of 36.3 percent. It was sometimes recorded as “aye” and at other times as “eye” to encompass different speaking styles.

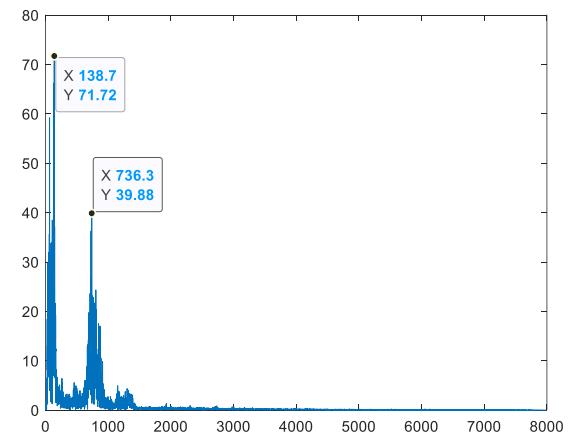


Figure 6. Depiction of visual method used for extracting the formant frequencies from the plots

#### IV. TRAINING THE MODELS

Four different types of networks were trained to form models and recognize the vowels. These models were based on neural and fuzzy logic.

##### A. Neural Network based on 3 perceptrons

- The training vector was plotted in the 2D plane and it was observed that the vowel "I" varies the most which can also be predicted by looking at the formant frequencies.

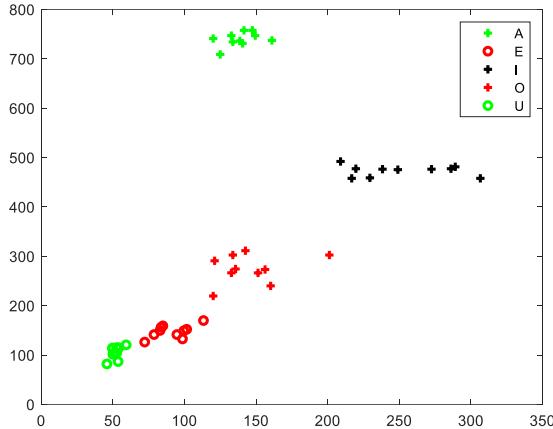


Figure 7. Plot the formants in a 2-dimensional plane

- The network is trained with 3 perceptrons for 100 epochs and the output is obtained. However, the output is not entirely accurate and only has an accuracy as follows:
  - 100 percent for the vowel "A"
  - 50 percent for the vowel "E"
  - 90 percent for the vowel "I"
  - 0 percent for the vowel "O"
  - 60 percent for the vowel "U"
- The accuracy of output for vowel "O" is zero because the network classified it as one of the other four vowels and didn't assign a unique output for it.
- The network is trained again for 10000 epochs and the following results are obtained:
  - 100 percent for the vowel "A"
  - 60 percent for the vowel "E"
  - 60 percent for the vowel "I"
  - 60 percent for the vowel "O"
  - 90 percent for the vowel "U"
- The reduction in the accuracy of the vowel "I" may be caused due to overfitting by the neural network over too many epochs.

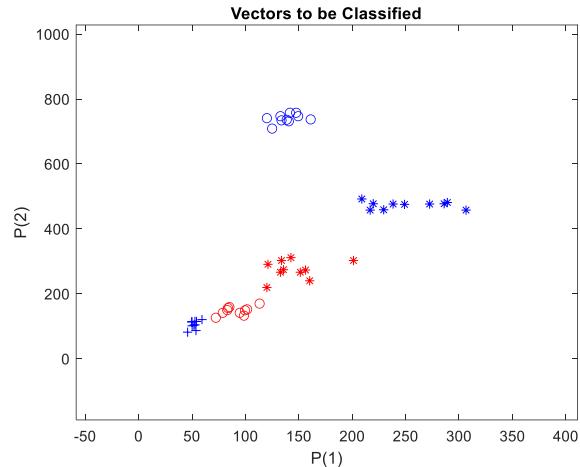


Figure 8. Simulation of the perceptron model with 3 perceptrons

##### B. Generating models using the neural network fitting toolbox in MATLAB

- The data is first arranged into different input and output matrices as required by the neural network toolbox. For training all fuzzy and neural models, the following convention is used to classify the outputs A=1, E=2, I=3, O=4, U=5.
- The network is trained using 3 neurons and the training stopped after 34 iterations or epochs with the mean squared error being 0.42.

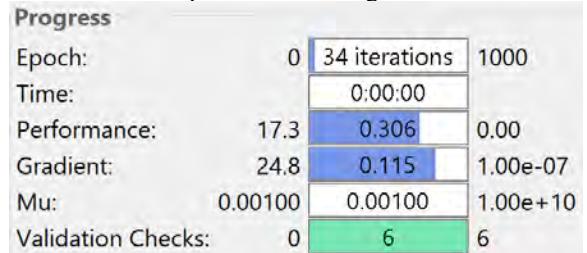


Figure 9. Output provided by the toolbox on completion of training the feed forward neural network with 3 neurons

Results			
	Samples	MSE	R
Training:	34	4.27839e-1	8.80658e-1
Validation:	8	2.42446e-1	9.55555e-1
Testing:	8	8.68484e-1	7.69583e-1

Figure 10. Error data provided by the toolbox on completion of training the feed forward neural network with 3 neurons

- The training algorithm used is Levenberg-Marquardt as it is the most effective option available.[5] It combines gradient descent and Gauss Newton methods to fit curves.[6]
- Plots for error histogram and regression are obtained using the toolbox buttons.

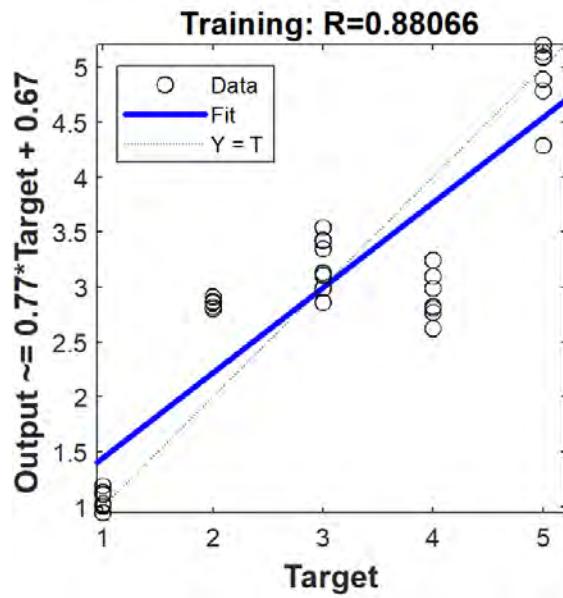


Figure 11. Regression plot for training data of feed forward neural network with 3 neurons

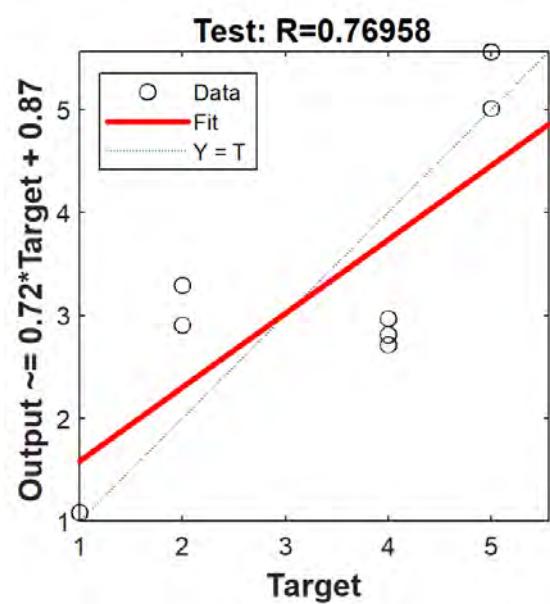


Figure 13. Regression plot for validation data of feed forward neural network with 3 neurons

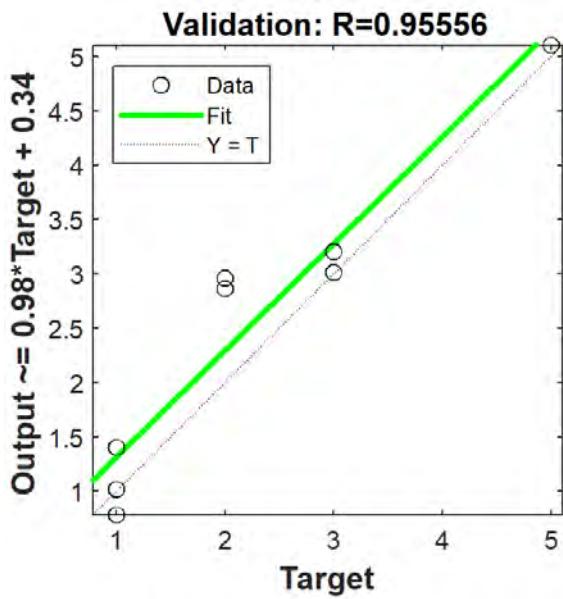


Figure 12. Regression plot for validation data of feed forward neural network with 3 neurons

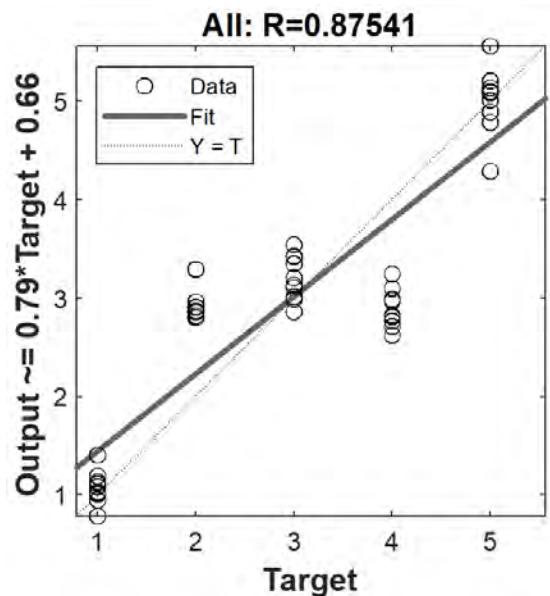


Figure 14. Regression plot for all data of feed forward neural network with 3 neurons

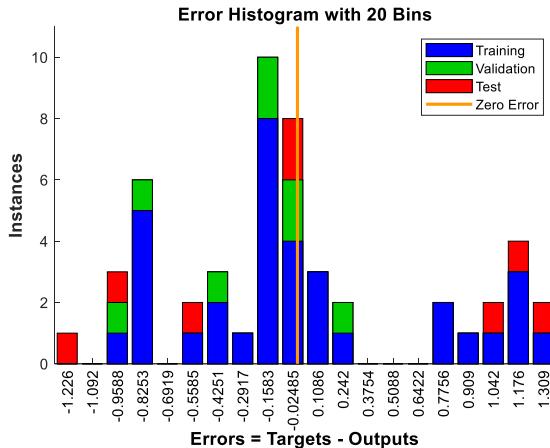


Figure 15. Error histogram for feed forward neural network with 3 neurons

5. The generated model is retrieved in the form of a Simulink block for testing further.

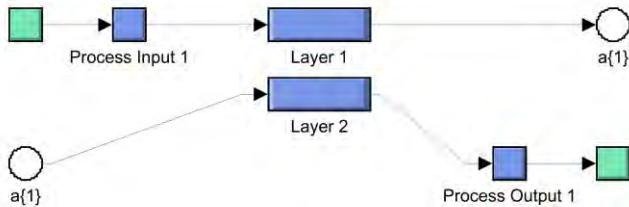


Figure 16. Neural Network Model in Simulink for feed forward neural network with 3 neurons

6. A simple Simulink model is developed for testing the generated neural network as shown below.

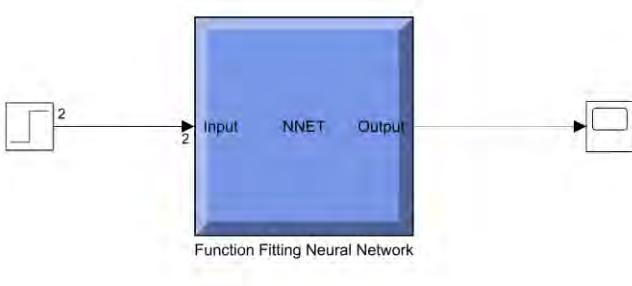


Figure 17. Simulink diagram for testing of feed forward neural network with 3 neurons

7. The system is easily able to recognize the formant frequencies near the vowel "A" but doesn't do an effective job at recognizing the other variables. However, it remains closer to the expected value instead of a wrong value.

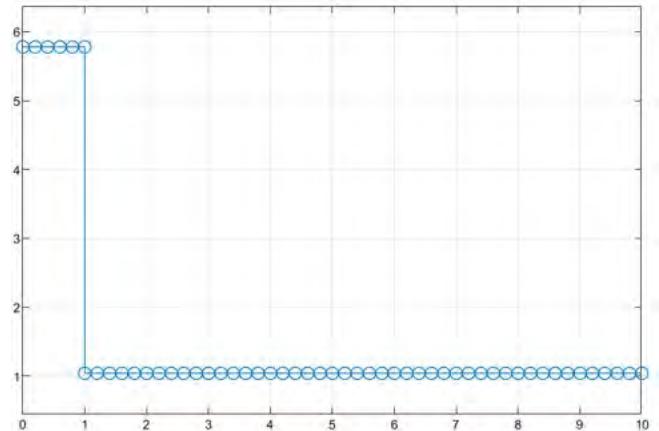


Figure 18. Neural Network output for input values near vowel "A"

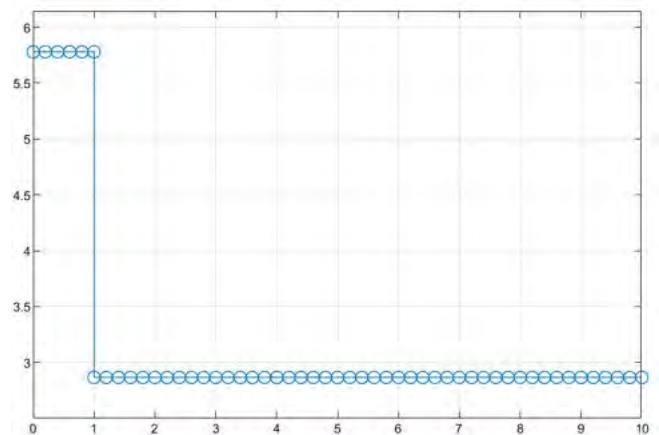


Figure 19. Neural Network output for values near vowel "E"

8. A new model is trained consisting of 10 neurons using the same training dataset. The training stops after 14 iterations or epochs and a mean squared error of 0.3105 is obtained.

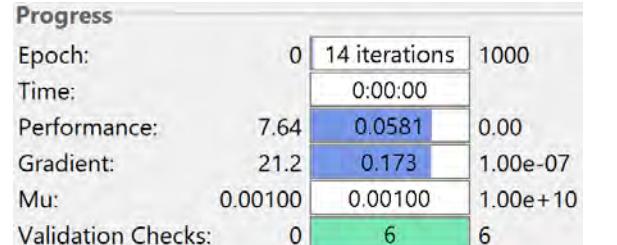


Figure 20. Output provided by the toolbox on completion of training the feed forward neural network with 10 neurons

9. The Simulink block is tested and the output is closer to the expected values than previously for the same input values. The values used for testing the model are not present in the training data.

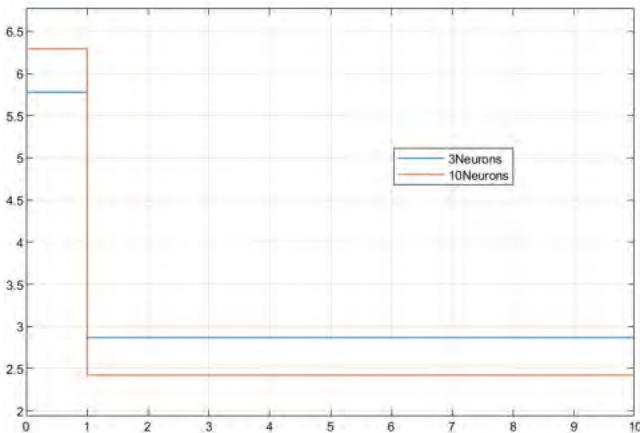


Figure 21. Output comparison for the same values of vowel "E"

### C. Generating models using the Neuro-Fuzzy Toolbox in MATLAB

1. Data is combined into a single set acceptable by the neuro-fuzzy toolbox with the last column representing the outputs.
2. A Fuzzy Inference System with 3 3 triangular membership functions (TMF) is used to train the model. The membership function type is kept linear throughout all the models.
3. Average testing error of 0.27 is observed in this case.

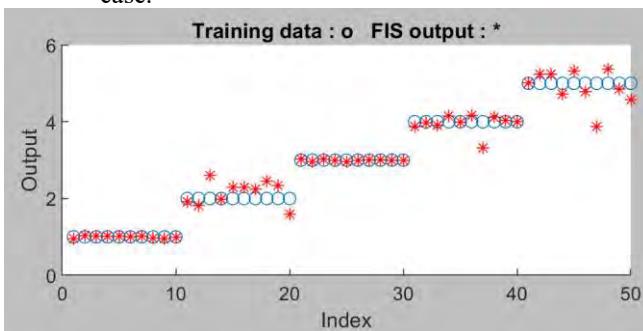


Figure 22. Output plot of training data for 3 3 TMF

4. A new model is trained using 5 5 triangular membership functions (TMF) and the same dataset.
5. Average testing error is reduced to 0.183 for this case.

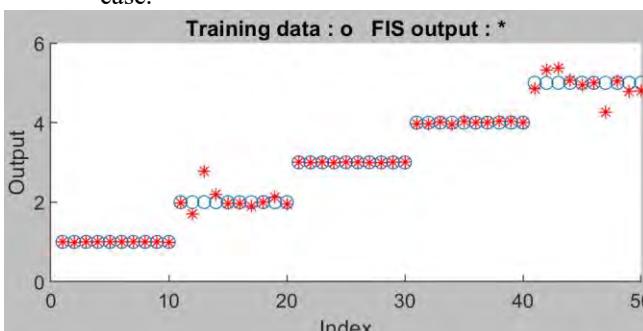


Figure 23. Output plot of training data for 5 5 TMF

6. Triangular Membership functions have firm boundaries and are not as good as the generalized bell functions (GBELL) or other membership functions with smooth boundaries. [7] Two more systems are trained with the same dataset using the generalized bell functions as 3 3 and 5 5 respectively.
7. The average testing error for the generalized bell (GBELL) model with 3 3 functions was 0.249

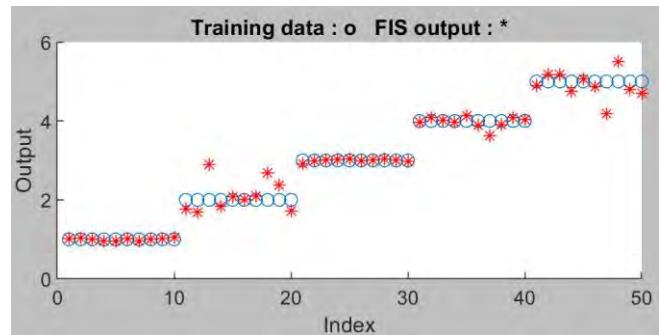


Figure 24. Output plot of training data for GBELL 3 3

8. The average testing error for the generalized bell (GBELL) model with 5 5 functions was 0.045

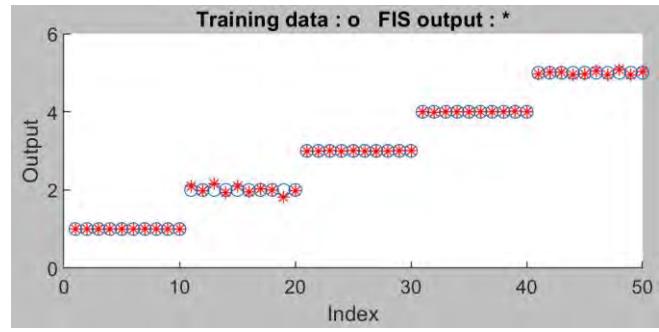


Figure 25. Output plot of training data for GBELL 5 5

9. All the models were exported to workspace and tested with the same testing data used on the neural networks.
10. On checking the output, almost all the fuzzy systems are closer to the output as compared to the neural network models. As expected from the average testing error values, the best output for the testing data is provided by the GBELL 55 model followed by the TMF 55 model. This is due to the larger number of membership functions which help in fitting more data accurately into the model.

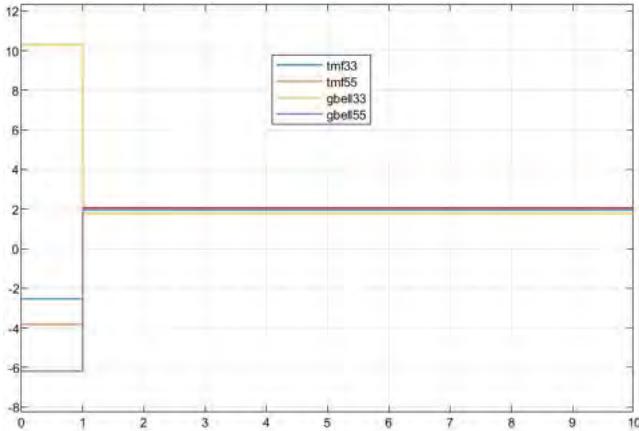


Figure 26. Output comparison for Fuzzy systems

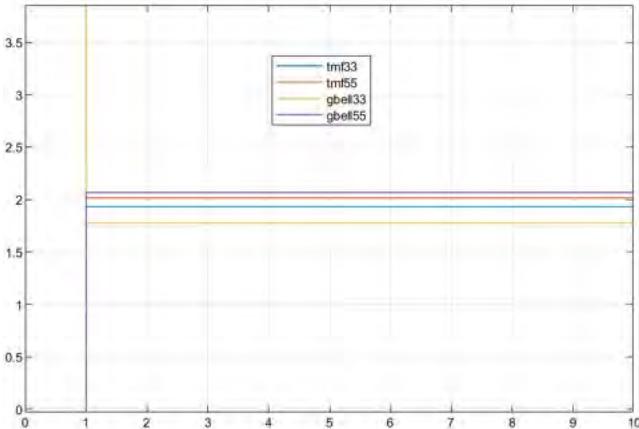


Figure 27. Output comparison for fuzzy systems with improved resolution

- ## V. COMPARISON OF ALL THE IMPLEMENTED METHODS
1. The neural network and fuzzy models are imported into the Simulink diagram and their outputs are plotted against each other. The testing inputs are synthetically designed to be within the range of the expected formant ranges which were observed in the training data.
  2. A simple Simulink block with step input was used to compare all the models against each other.
  3. 10 different testing values were used to observe the outputs and make comparisons. The testing values were taken randomly between the vowels, “E”, “I”, “O” and “U”. The values near the vowel “A” were not used for testing as it is classified effectively by all methods.
  4. The variation in amplitude from the actual expected amplitude was noted to check for the best and least effective models.

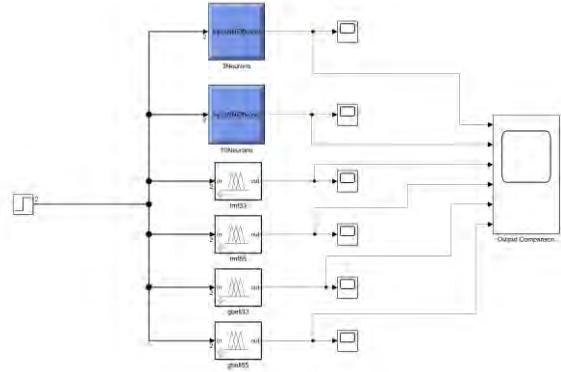


Figure 28. Simulink model for comparison of outputs

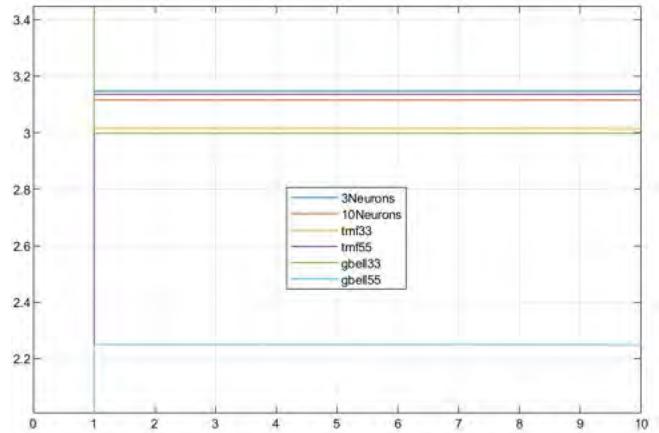


Figure 29. Magnified plot of output comparison using new testing data

## VI. CONCLUSIONS AND DISCUSSION

Based on the comparison plots, it was observed that the overall best model for estimating the outputs was the GBELL 33 closely followed by TMF 55. It should be noted that for a few cases, the TMF 55 provided the closest output so the random selection of testing data may have slightly effected the results. However, both models predicted the values accurately with GBELL 33 remaining within 0.2 of the expected output for all cases.

The values given by models vary a lot before the input is provided. This is due to each model fitting the data in a different manner. The neural networks also provide acceptable outputs remaining within 0.3 of the expected output for all cases.

The GBELL 55 function does not perform well despite having the lowest average mean squared error during training. This occurs due to overfitting the data. This has also been observed in other previous experiments [7].

Based on the results obtained, it is concluded that the generalized bell membership functions with 3 functions in each layer are best suited for recognizing vowels within uncontrolled environments. These functions can be further improved and coupled with deep learning methods to improve the conventional ASR pipelines. [8]

These systems can be implemented with limited resources and can be trained with relatively small amounts of data. A good application of this might be implemented in space systems to save energy on communication and processing hardware.

#### ACKNOWLEDGMENT

Thanks to the University of Texas at Arlington for providing the necessary resources to perform this research. I would like to thank my research advisor Dr. David A. Hullender and my academic supervisor Dr. Seiichi Nomura for introducing me to the concept of Discrete Fourier Transforms and signal processing in detail which proved to be crucial for this paper. I'm immensely grateful to my professors, Dr. Frank Lewis, Dr. Michael Niestroy, Dr. Kamesh Subbarao and Dr. Robert Woods for explaining the concepts to me in a practical manner which allowed me to implement them in a versatile manner on this project.

#### REFERENCES

- [1] F. L. Lewis and C. He, "Homework 3 - DFT Analysis for speech recognition and Fault diagnosis," Intelligent Control Systems EE 5322, Department of Electrical Engineering, The University of Texas at Arlington, Arlington, Sep., 24, 2020
- [2] Mathworks, "FFT Documentation," retrieved on Dec. 5, 2020 from <https://www.mathworks.com/help/matlab/ref/fft>
- [3] D. A. Hullender, "Stochastic systems," Dynamic Systems Modeling ME 5305, Department of Mechanical and Aerospace Engineering, The University of Texas at Arlington, Nov. 14, 2019
- [4] S. Nomura – "Discrete Fourier Transforms," Analytic Methods in Engineering ME 5331, Department of Mechanical and Aerospace Engineering, The University of Texas at Arlington, Oct., 2019
- [5] M. A. Niestroy – "Neural Networks," System Identification and Estimation EE 5327, Department of Electrical Engineering, The University of Texas at Arlington, Nov., 10, 2020
- [6] Marquardt, D., "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," SIAM Journal on Applied Mathematics, Vol. 11, No. 2, June 1963, pp. 431–441
- [7] A. Shrotriya and M. A. Niestroy, "Homework 6 – Fuzzy Systems," System Identification and Estimation EE 5327, Department of Electrical Engineering, The University of Texas at Arlington, Arlington, Dec., 01, 2020
- [8] A. Geitgey, "Machine Learning is Fun," second edition, United States, 2018, pp. 158-170
- [9] A. Coates (Baidu Inc.), "Speech Recognition", Bay Area Deep Learning School Day 2 at CEMEX auditorium, Stanford University, Stanford, California, Sep., 25, 2016
- [10] Adi Y, Keshet J, Goldrick M. VOWEL DURATION MEASUREMENT USING DEEP NEURAL NETWORKS. IEEE Int Workshop Mach Learn Signal Process. 2015; 2015:10.1109/MLSP.2015.7324331.
- [11] C. Kim and W. Sung, "Vowel Pronunciation Accuracy Checking System Based on Phoneme Segmentation and Formants Extraction," Seoul National University, Seoul, Korea

Shrotriya, Atul

The schematic for a suspension is shown below. The input is the road elevation (displacement)  $u(t)$ . The output of interest is the acceleration of the mass  $\ddot{y}(t)$ . The five modeling equations for this system are given below.

- a. List the unknowns.

Unknowns are  $y$  and  $v$ .

- b. Express the equations in state variable format and use ode45 for a simulation to get a plot of  $\ddot{y}(t)/g$ . Assume the following parameters:  $M = 2.33 \text{ kg}$ ,  $K_1 = 201 \text{ N/m}$ ,  $K_2 = 153 \text{ N/m}$ ,  $b = 1.75 \text{ Ns/m}$ . Assume  $u(t) = 0.05\sin(6t)$ . Assume all initial conditions are zero. It should take about 40 seconds for the simulation to reach a steady state sinusoidal response. What is the peak value of the acceleration in g's after the response reaches steady state, i.e.  $(\ddot{y}/g)_{\text{peak}} = ?$

$$2.33\ddot{y} + f1 + f2 = 0 \quad (\text{Equation } - 1)$$

$$f1 = k2 \times (y - v) \quad (\text{Equation } - 2)$$

$$f2 = b \times (\dot{y} - \dot{v}) \quad (\text{Equation } - 3)$$

$$f3 = k1 \times (v - u) \quad (\text{Equation } - 4)$$

$$f1 + f2 - f3 = 0 \quad (\text{Equation } - 5)$$

From the above equations, we can derive:

$$2.33\ddot{y} + f1 + f2 = 0$$

$$\ddot{y} = -(f1 + f2)/M \quad (\text{Equation } - 6)$$

And also,

$$\dot{v} = \dot{y} - f2/b$$

Now, taking,

$$y = x1; \dot{y} = x2 \text{ and } v = x3$$

Therefore,

$$f1 = k2(y - v)$$

$$f3 = k1(v - u)$$

$$f2 = f3 - f1$$

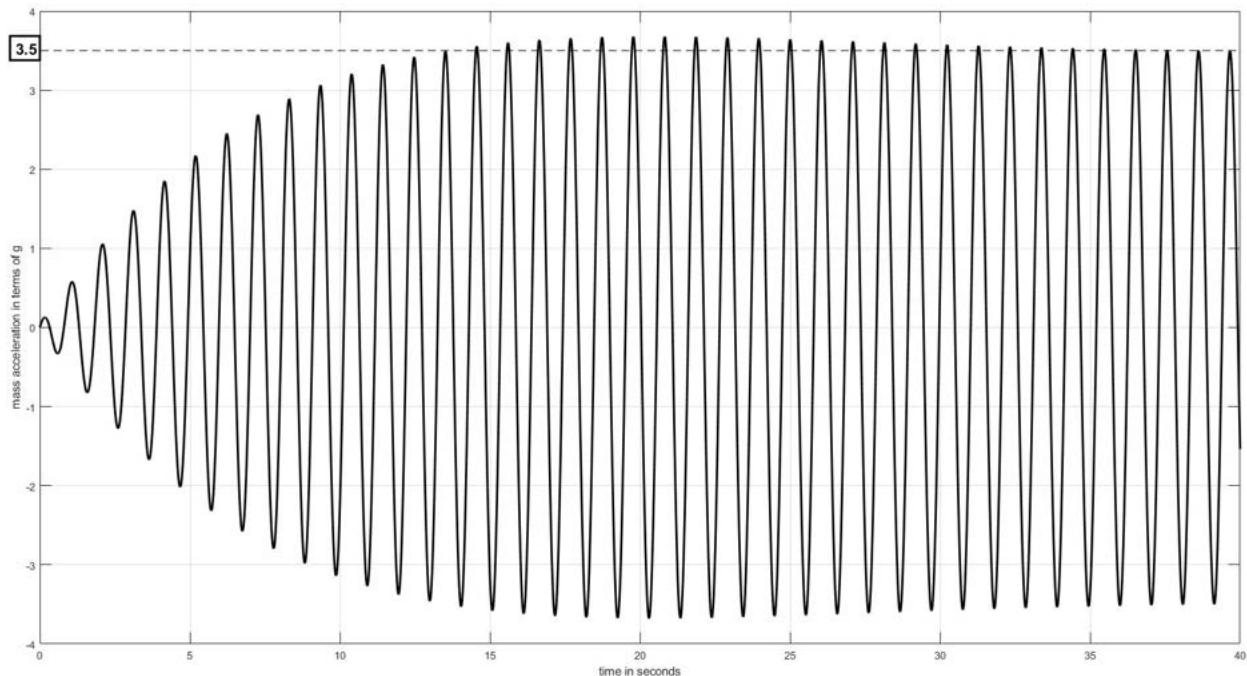
$$\dot{x1} = x2$$

$$\dot{x2} = -\frac{f1 + f2}{M} \quad \dot{x2}(0^-) = 0$$

$$\dot{x3} = \dot{y} - \frac{f2}{b} \quad \dot{x3}(0^-) = 0$$

Therefore, the m-file to plot graph for  $\frac{\ddot{y}}{g}(t)$  using ode45 in MATLAB is:

```
function homework2part2
M=2.33; k1=201; k2=153; b=1.75; g=9.81;
[t,x]=ode45(@Equation,[0 40],[0 0 0]);
d2y=-(k1*x(:,3)-k1*0.05*sin(6*t))/(g*M);
plot(t,d2y,'k','LineWidth',2)
xlabel('time in seconds')
ylabel('mass acceleration in terms of g')
grid
function dx=Equation(t,x)
dx=zeros(3,1);
y=x(1); dy=x(2); v=x(3);
u=0.05*sin(6*t);
f1 = k2*(y-v);
f3=k1*(v-u);
f2=f3-f1;
dx(1)=x(2);
dx(2)=- (f1+f2)/M;
dx(3)=dy-(f2/b);
end
end
```



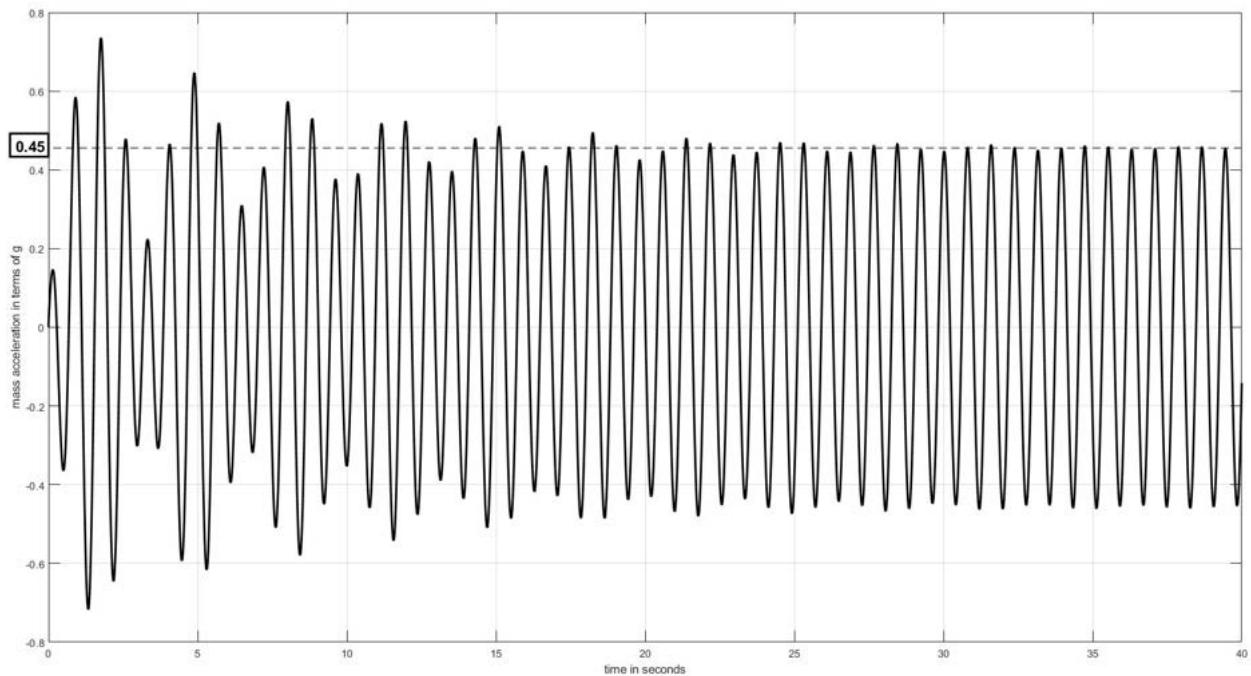
Upon studying the above graph, it becomes apparent that the peak values of  $\frac{\ddot{y}}{g}(t)/g$  are somewhere near 3.5. Therefore,  $(\frac{\ddot{y}}{g})_{peak} \approx 3.5g$

- c. Repeat the simulation in part b but this time change in the input frequency from 6 rad/s to 8 rad/s. What is the peak value of the acceleration in g's after the response reaches steady state, i.e.  $(\frac{\ddot{y}}{g})_{peak} = ?$  Explain why there is such a large difference in the peak values even though

there is a small change in the road frequency; to answer this question, you will probably need to first work part e below which gives the eigenvalues for this system.

The m-code for generating the plot of  $\ddot{y}(t)/g$  using ode45 in MATLAB is:

```
function homework2part3
M=2.33; k1=201; k2=153; b=1.75; g=9.81;
[t,x]=ode45(@Equation,[0 40],[0 0 0]);
d2y=-(k1*x(:,3)-k1*0.05*sin(8*t))/(g*M);
plot(t,d2y,'k','LineWidth',2)
xlabel('time in seconds')
ylabel('mass acceleration in terms of g')
grid
function dx=Equation(t,x)
dx=zeros(3,1);
y=x(1); dy=x(2); v=x(3);
u=0.05*sin(8*t);
f1 = k2*(y-v);
f3=k1*(v-u);
f2=f3-f1;
dx(1)=x(2);
dx(2)=-(f1+f2)/M;
dx(3)=dy-(f2/b);
end
end
```



Through this graph, we observe peak values of  $\ddot{y}(t)/g$  to be around 0.45. Therefore,  $(\frac{\ddot{y}}{g})_{peak} \approx 0.45g$ . Despite a small change in frequency, there's such a large change in amplitude because the undamped natural frequency of the system is 6.11 rad/sec as observed in **part e** of this problem. Therefore, this leads to very less damping and high magnitudes in the system at a nearby frequency of 6 rad/sec.

- d. Assuming all initial conditions are zero and assuming all displacements  $y$ ,  $v$ , and  $u$  are assumed to be zero at equilibrium, Laplace transform these equations using  $U(s)$  for the Laplace of  $u(t)$ . What is the transfer function relating  $\ddot{Y}(s)$  to  $U(s)$ , i.e.  $\ddot{Y}(s) = [ ? ] U(s)$ . Hint:  $\ddot{Y}(s) = s^2 Y(s)$ .

Considering  $\dot{y}(s) = sy$ ;  $\ddot{y}(s) = s^2 y$  and  $\dot{v} = sv$ . Therefore, equation 1 becomes

$$Ms^2y + f_1 + f_2 = 0 \quad (\text{Equation - 7})$$

Substituting values of  $(f_1+f_2)$  as  $f_3$ , the above equation can also be written as:

$$Ms^2y = -k_1(v - u) \quad (\text{Equation - 8})$$

Similarly,  $\dot{v}(s) = sv$ . Then, substituting values of  $f_1$ ,  $f_2$  and  $f_3$  from equations 2, 3 and 4 respectively into Equation – 5.

$$k_2(y - v) + b(sy - sv) = k_1(v - u)$$

$$k_1v + k_2v + bsy = k_2y + bsy + k_1u$$

$$v = \frac{k_1u + k_2y + bsy}{k_1 + k_2 + bs}$$

Putting the value of  $v$  obtained above into equation 8,

$$Ms^2y = k_1\left(u - \frac{k_1u + k_2y + bsy}{k_1 + k_2 + bs}\right)$$

$$Ms^2y(k_1 + k_2 + bs) + k_1k_2y + k_1bsy = (k_1k_2 + k_1bs)u$$

$$y = \frac{k_1k_2 + k_1bs}{Ms^3b + (k_1 + k_2)Ms^2 + k_1bs + k_1k_2}u$$

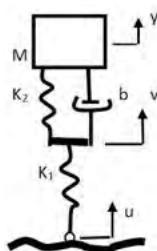
Now,  $\ddot{Y}(s) = s^2 Y(s)$ . Therefore,

$$\ddot{Y}(s) = \frac{k_1bs^3 + k_1k_2s^2}{Ms^3b + (k_1 + k_2)Ms^2 + k_1bs + k_1k_2}U(s)$$

On substituting the values of  $M$ ,  $k_1$ ,  $k_2$  and  $b$  into the above equation we get:

$$\ddot{Y}(s) = \left( \frac{351.75s^3 + 30753s^2}{4.0775s^3 + 824.82s^2 + 351.75s + 30753} \right) U(s)$$

- e. Use the ‘damp’ command to get the eigenvalues? What are the damping ratio, frequencies, and time constants?



$$\begin{aligned} M\ddot{y} + f_1 + f_2 &= 0 \\ f_1 &= K_2(y - v) \\ f_2 &= b(\dot{y} - \dot{v}) \\ f_3 &= K_1(v - u) \\ f_1 + f_2 - f_3 &= 0 \end{aligned}$$

```

>> n=[351.75 30753 0 0];
>> d=[4.0775 824.82 351.75 30753];
>> G=tf(n,d)

```

G =

$$\frac{351.8 s^3 + 30753 s^2}{4.077 s^3 + 824.8 s^2 + 351.8 s + 30753}$$

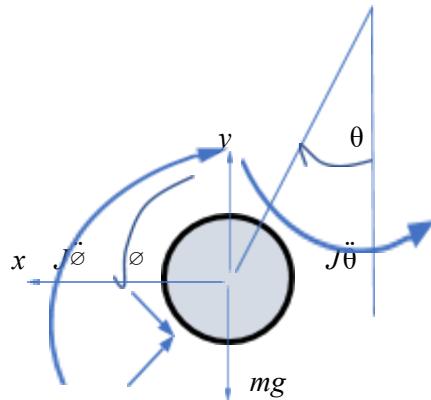
Continuous-time transfer function.

```
>> damp (G)
```

Pole	Damping (rad/seconds)	Frequency (rad/seconds)	Time Constant (seconds)
-1.21e-01 + 6.11e+00i	1.98e-02	6.11e+00	8.26e+00
-1.21e-01 - 6.11e+00i	1.98e-02	6.11e+00	8.26e+00
-2.02e+02	Not Applicable	Not Applicable	4.95e-03

Shrotriya, Atul

1. Draw the free body diagram and write the equations for this system.



for no slip motion,

$$r\dot{\phi} = R\theta \text{ and } r\ddot{\phi} = R\ddot{\theta} \quad - \text{Eq. 1}$$

$$f_d = cv|v| \quad - \text{Eq. 2}$$

$$v = (R - r)\dot{\theta} \quad - \text{Eq. 3}$$

$$(J + mr^2)\ddot{\phi} - (J + mr^2)\ddot{\theta} + rmgsin\theta + rf_d = 0 \quad - \text{Eq. 4}$$

$$\text{Equation 4 can be written as } (J + mr^2)\left(\frac{R}{r} - 1\right)\ddot{\theta} + rmgsin\theta + rf_d = 0 \quad - \text{Eq. 5}$$

2. Use ode45 to simulate and plot  $\theta(t)$  and  $\dot{\theta}(t)$  assuming  $\theta(0^-) = 0.70 \text{ rad}$  and  $\dot{\theta}(0^-) = 0$ .

What is the oscillation frequency in rad/s? What are the initial and final values of  $\dot{\theta}(t)$ ; do these values seem to be correct?

$$C = 100 \text{ Ns}^2/\text{m}^2 \quad R = 2 \text{ m} \quad J_{cg} = 0.25Mr^2 \quad r = 0.2 \text{ m} \quad M = 270 \text{ kg} \quad g = 9.81 \text{ m/s}^2$$

$$\text{Time period, } \tau = 6.114 - 3.142 = 2.972 \text{ seconds}$$

$$\text{Oscillation frequency, } \omega = \frac{2\pi}{2.972} \approx 2.11 \text{ rad/seconds}$$

The obtained plots show the initial value of  $\dot{\theta}(t)$  at 0 and the final value of  $\dot{\theta}(t)$  at approximately 0.0145 radians/second. These values seem to be correct as the initial value matches the given condition of  $\dot{\theta}(0^-) = 0$  whereas the final value is decaying towards 0 as expected from predicted results. The simulation is run for 300 seconds as the final value of  $\theta(t)$  comes within 1 percent range in that duration.

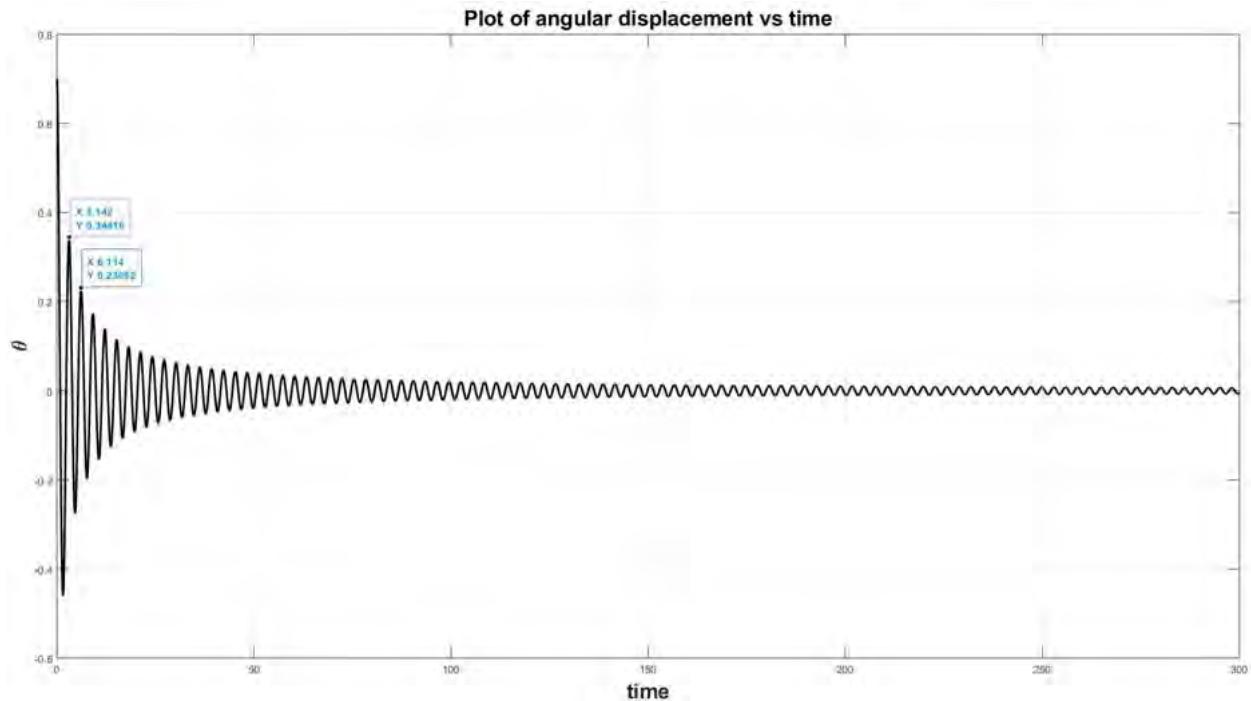
#### m-file code:

```
function dsmhw3q2
c=100; R=2; r=0.2; m=270; g=9.81; J=0.25*m*r*r;
[t,x]=ode45(@eqns,[0 300],[0.7 0]);
figure(1)
plot(t,x(:,1), 'k', 'LineWidth', 2)
```

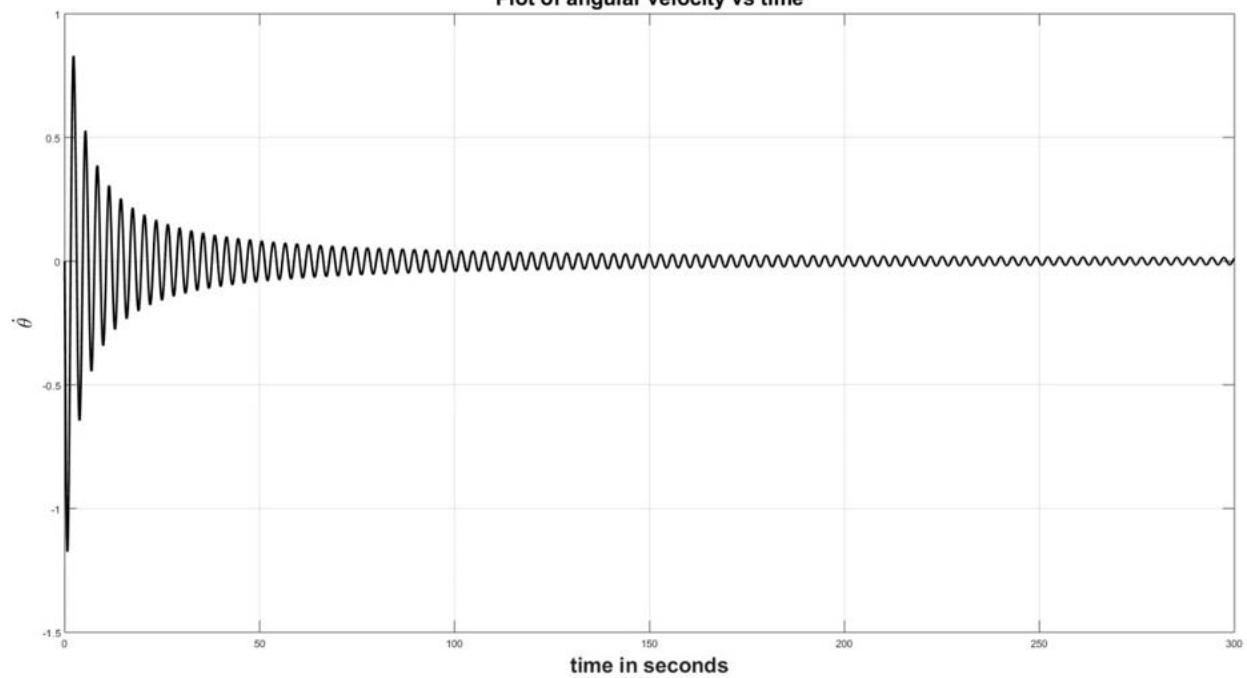
```

xlabel('time')
ylabel('\theta')
title('plot of angular displacement vs time', 'FontSize', 14)
grid
figure(2)
plot(t,x(:,2), 'k', 'LineWidth', 2)
xlabel('time')
ylabel('$$\dot{\theta}$$', 'Interpreter', 'LaTeX')
title('plot of angular velocity vs time', 'FontSize', 14)
grid
function dx=eqns(t,x)
dx=zeros(2,1);
th=x(1); dth=x(2); dx(1)=x(2);
v=(R-r)*x(2);
fd=c*v*abs(v);
dx(2)=(fd*r+m*g*r*sin(x(1)))/((1-(R/r))*(J+m*r*r));
end
end

```



**Plot of angular velocity vs time**



3. Based on observed values in your plot of  $\dot{\theta}(t)$ , obtain an equation for a straight-line to approximate  $\dot{\theta}|\dot{\theta}|$ ; also, introduce the small-angle approximation and get a linear approximation for the differential equation for  $\theta(t)$ . What are the damped natural frequency and damping ratio associated with your linearized differential equation?

$$\dot{\theta}|\dot{\theta}| = m\dot{\theta} + b$$

$$\dot{\theta}(0^-) = \dot{\theta}(\infty) = 0$$

Taking the maximum amplitude of  $\dot{\theta}$  from above plot for linear approximation,  $\dot{\theta}(0.6418) = -1.1741$ ;

$$\dot{\theta}|\dot{\theta}| \approx \dot{\theta} \quad - \text{Eq. 6}$$

Introducing small angle approximation and linearization obtained in Equation 6, Equation 4 becomes,

$$(J + mr^2)\left(\frac{R}{r} - 1\right)\ddot{\theta} + rc(R - r)^2\dot{\theta} + rm\theta = 0 \quad - \text{Eq. 7}$$

Taking the Laplace transform of Eq. 7,

$$(J + mr^2)\left(\frac{R}{r} - 1\right)(s^2\theta(s) - 0.7s) + rc(R - r)^2(s\theta(s) - 0.7) + rm\theta(s) = 0$$

$$\theta(s) = \frac{0.7(J+mr^2)\left(\frac{R}{r}-1\right)s+0.7rc(R-r)^2}{(J+mr^2)\left(\frac{R}{r}-1\right)s^2+rc(R-r)^2s+rmg} \quad - \text{Eq. 8}$$

Using the damp command in MATLAB, the damped natural frequency is obtained as 2.06 rad/seconds and damping ratio is obtained as 0.15.

```
>>c=100; R=2; r=0.2; m=270; g=9.81; J=0.25*m*r*r;
>> K=tf([0.7*(J+m*(r^2))*((R/r)-1) 0.7*r*c*((R-r)^2)],[(J+m*(r^2))*((R/r)-1) r*c*((R-r)^2) r*m*g])
K =

```

$$85.05 s + 45.36$$


---

$$121.5 s^2 + 64.8 s + 529.7$$

Continuous-time transfer function.

```
>> damp(K)
```

Pole	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-2.67e-01 + 2.07e+00i	1.28e-01	2.09e+00	3.75e+00
-2.67e-01 - 2.07e+00i	1.28e-01	2.09e+00	3.75e+00

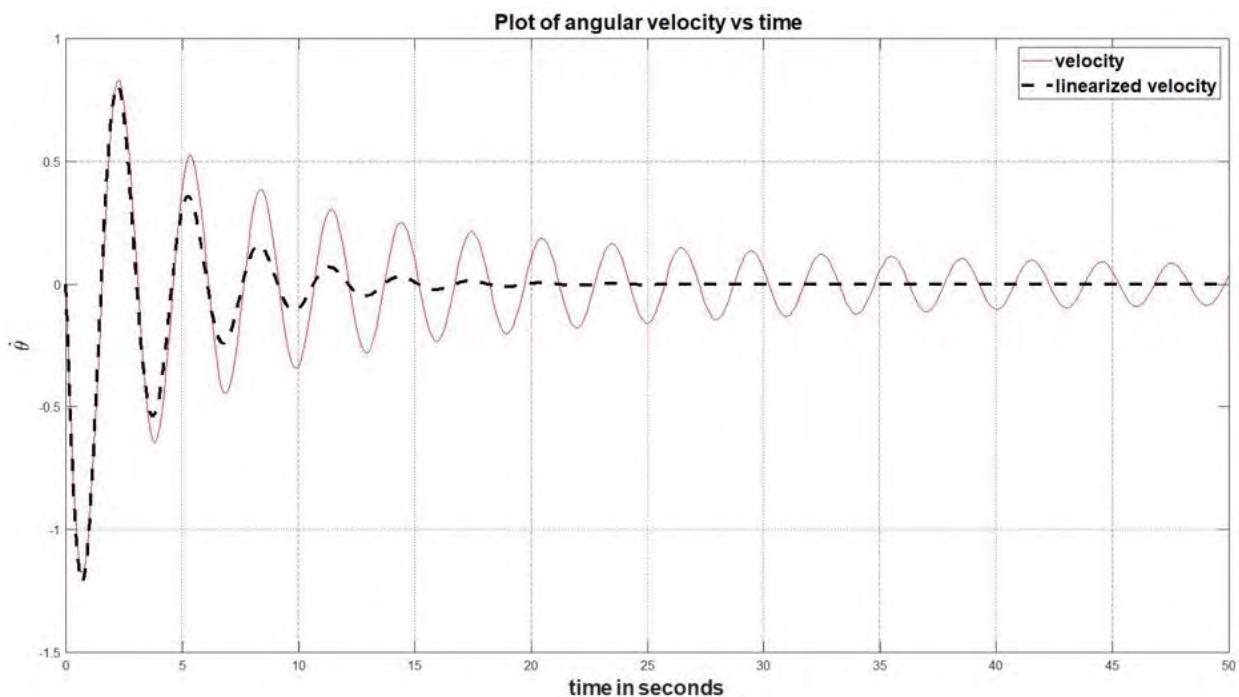
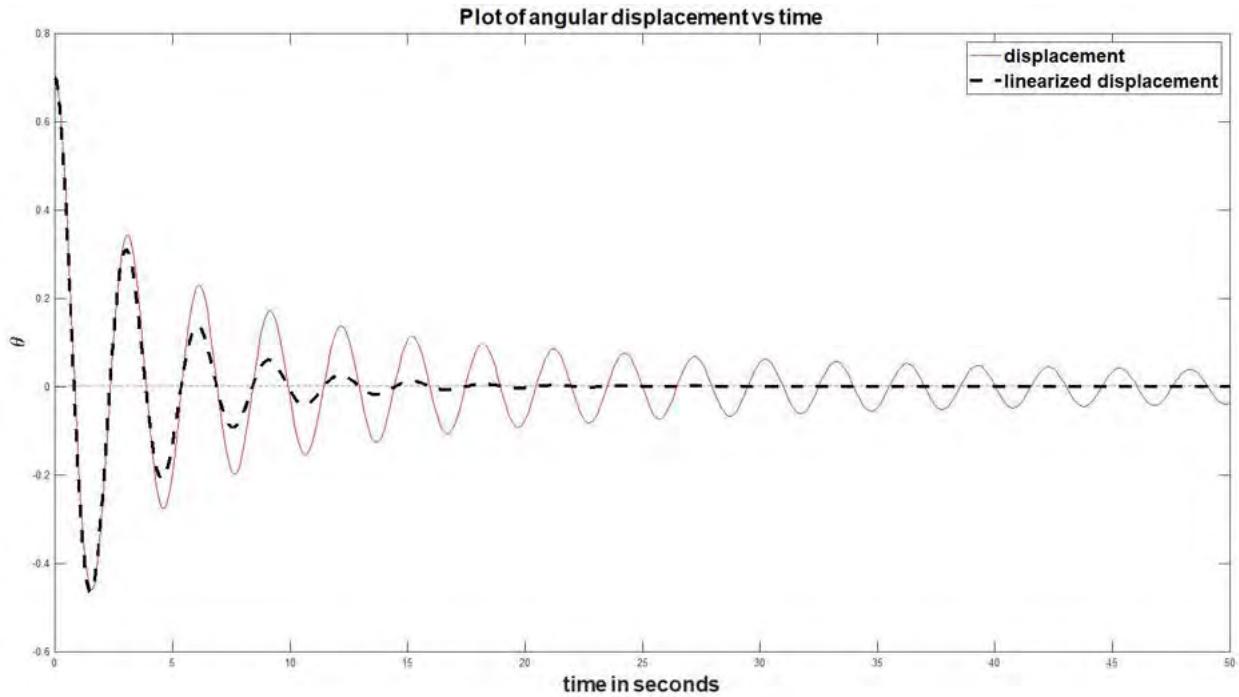
Based on the above table,

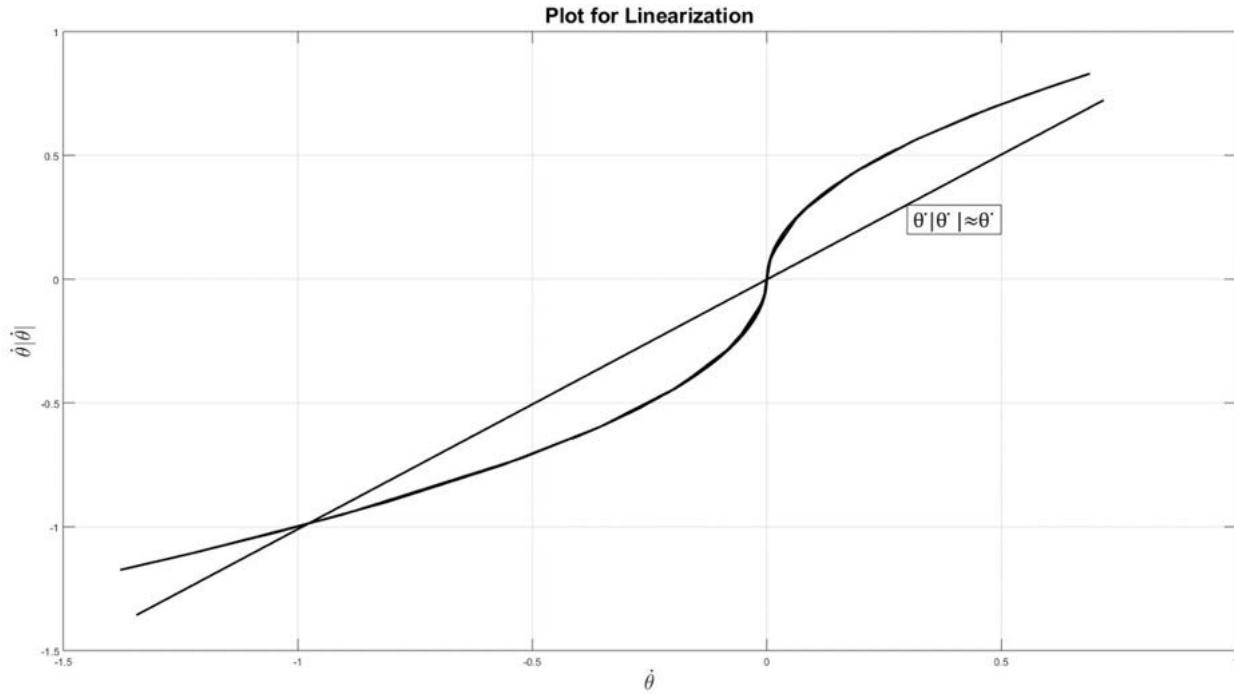
Damped natural frequency,  $\omega_d = 2.07 \text{ rad/seconds}$

Damping Ratio,  $\delta = 0.128$

4. Plot  $\theta(t)$  corresponding to parts 2 and 3 on the same graph for comparison. Plot  $\dot{\theta}(t)$  corresponding to parts 2 and 3 on the same graph for comparison. How accurate is the linear differential equation? For example, how do the oscillation frequencies, decay rates associated with the damping, and amplitudes compare?

Here, time period is only taken till 50 seconds to improve the resolution in initial 10 seconds. The desired plots and the m-file code for obtaining them are given below:





**m-file code:**

```

function dsmhw3q4
c=100; R=2; r=0.2; m=270; g=9.81; J=0.25*m*r*r;
[t,x]=ode45(@eqns,[0 50],[0.7 0]);
[Lt,Lx]=ode45(@Leqns,[0 50],[0.7 0]);
figure(1)
plot(t,x(:,1),'r',Lt,Lx(:,1),'k--','LineWidth',3)
xlabel('time')
ylabel('\theta')
title('Plot of angular displacement vs time','FontSize',20)
legend('displacement','linearized displacement')
grid
figure(2)
plot(t,x(:,2),'r',Lt,Lx(:,2),'k--','LineWidth',3)
xlabel('time')
ylabel('$$\dot{\theta}$$', 'Interpreter', 'LaTeX')
title('plot of angular velocity vs time','FontSize',20)
legend('velocity','linearized velocity')
grid
f=x(:,2).*abs(x(:,2));
Lf=x(:,2);
figure(3)
plot(f,Lf,'k','LineWidth',2)
xlabel('$$\dot{\theta}$$', 'Interpreter', 'LaTeX')
ylabel('$$\dot{\theta}|\dot{\theta}$$', 'Interpreter', 'LaTeX')
title('Plot for Linearization')
grid
function dx=eqns(t,x)
dx=zeros(2,1);
th=x(1); dth=x(2); dx(1)=x(2);
v=(R-r)*x(2);
fd=c*v*abs(v);
dx(2)=(fd*r+m*g*r*sin(x(1)))/((1-(R/r))*(J+m*r*r));

```

```

end
function dLx=Leqns(Lt,Lx)
dLx=zeros(2,1);
Lth=Lx(1); dLth=Lx(2); dLx(1)=Lx(2);
Lv=(R-r)*Lx(2);
Lfd=c*(R-r)*Lv;
dLx(2)=( (Lfd*r+m*g*r*Lx(1)) ) / ((1-(R/r))*(J+m*r*r));
end
end

```

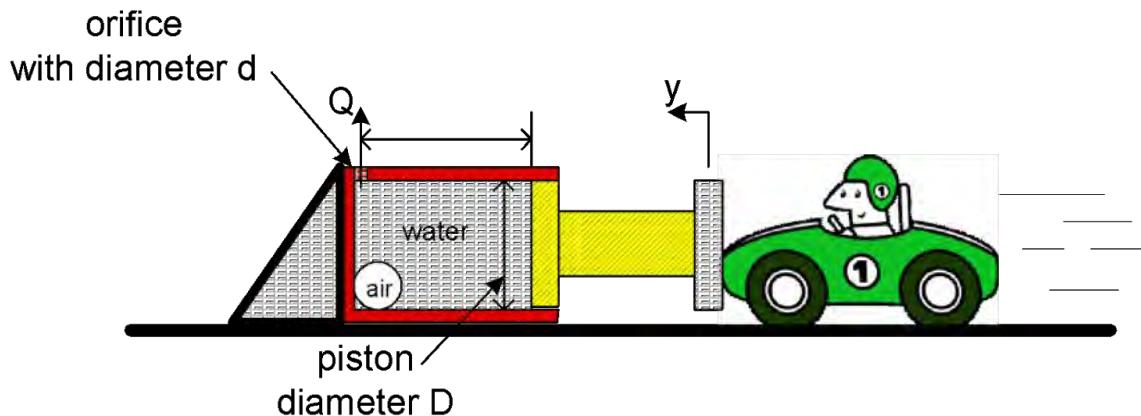
$$\tau = \left| \frac{1}{r} \right| = \left| \frac{1}{0.267} \right| = 3.745 \text{ seconds}$$

$$\tau_{max} = 5\tau = 18.726$$

Through a comparison of the plots, it seems that the linear differential equation is quite accurate. However, it is not exact because the small angle approximation results in a very slight reduction in the oscillation frequency of the linearized plot. The amplitudes are similar in the initial seconds of the simulation but decay rates are much higher which results in a rapid drop of amplitudes for the linearized equation. The amplitude values come within the 1 percent range in about 19 seconds which is further confirmed by  $\tau_{max}$  values obtained from the above table.

Shrotriya, Atul

An impact bumper system is shown below for stopping the forward motion of a car. Energy is stored in the fluid capacitance of the cylinder and is dissipated with the flow through the orifice. A rubber bladder is filled with air and placed in the water in the cylinder with the objective of reducing the effective bulk modulus of the water. You are to do ode45 simulations and select an initial volume of the air in the bladder and a diameter for the orifice with the design objective being to stop the car before the piston bottoms out ( $y(t) < y_{max}$ ) and with the peak deceleration during the stopping process being as small as possible.



Assume the following:

Car mass = 1500 kg  $D = 0.25 \text{ m}$  Bulk modulus of water =  $2e9 \text{ N/m}^2$   $y_{max} = 7.5 \text{ m}$   $P_{atm} = 1,01325e5 \text{ N/m}^2$

- (1) Write the equation for the equivalent bulk modulus of the water and air in the cylinder. Assume that the diameter of the cylinder remains constant.

$$\frac{1}{\beta_e} = \frac{\text{Water Ratio}}{\beta_w} + \frac{\text{Air Ratio}}{\beta_a}$$

Here, Air Ratio =  $\frac{\text{Volume of Air at } P}{\text{Volume in Cylinder}}$

Using Boyle's Law,

$$\text{Volume of Air at } P = \frac{\text{Volume of Air at Atmospheric Pressure * Atmospheric Pressure}}{P_{gauge}}$$

Also, Bulk modulus of air is a function of Pressure, therefore,

$$\beta_a = 1.4(P_{gauge})$$

- (2) Write the volume equation for the cylinder; use V to denote the volume.

$$V = Ay_{max} \left( 1 - \frac{y}{y_{max}} \right)$$

$$\dot{V} = -A\dot{y}$$

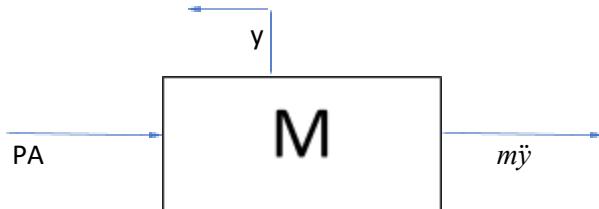
$$A = \frac{\pi D^2}{4}$$

- (3) Using  $P$  for the gage pressure of the water in the cylinder, write the lumped capacitance equation for the fluid in the cylinder.

$$-Q - \dot{V} = \frac{V}{\beta_e} \dot{P}$$

$$Q = C_d A_{orifice} \sqrt{\left(\frac{2}{\rho}\right) |P_{gauge}| \operatorname{sign}(P_{gauge})}$$

- (4) Draw the free body diagram for the forces on the car mass. Write the differential equation for  $y(t)$ . Assume  $t = 0$  when the car contacts and adheres to the bumper and assume  $\dot{y}(0^-) = 15 \text{ m/s}$  and  $y(0^-) = 0 \text{ m}$ .



$$m\ddot{y} + P_{gauge}A = 0$$

- (5) Define state variables and express your equations in the format for an ode45 simulation.

$$X1 = y; \quad X2 = \dot{y}; \quad X3 = P$$

$$dX1 = X2$$

$$dX2 = -\frac{P_{gauge}A}{m}$$

$$dX3 = \left(-Q - \dot{V}\right) \frac{\beta_e}{V}$$

- (6) Use trial-and-error to select an initial air volume and an orifice diameter to meet your design specifications. Use an event function to stop the simulation at  $y(t) = y_{\max}$  or  $\dot{y}(t) = 0$  whichever occurs first. Fill in the blanks below for your final design.

Orifice diameter 0.077 m

Peak deceleration 9.298 g's

Initial air volume in the bladder 0.004 m<sup>3</sup>

- (7) Show plots of  $P(t)/P_{\text{atm}}$ ,  $y(t)/y_{\max}$ ,  $\dot{y}(t)/15$ , and  $\ddot{y}(t)/g$  corresponding to your final design.

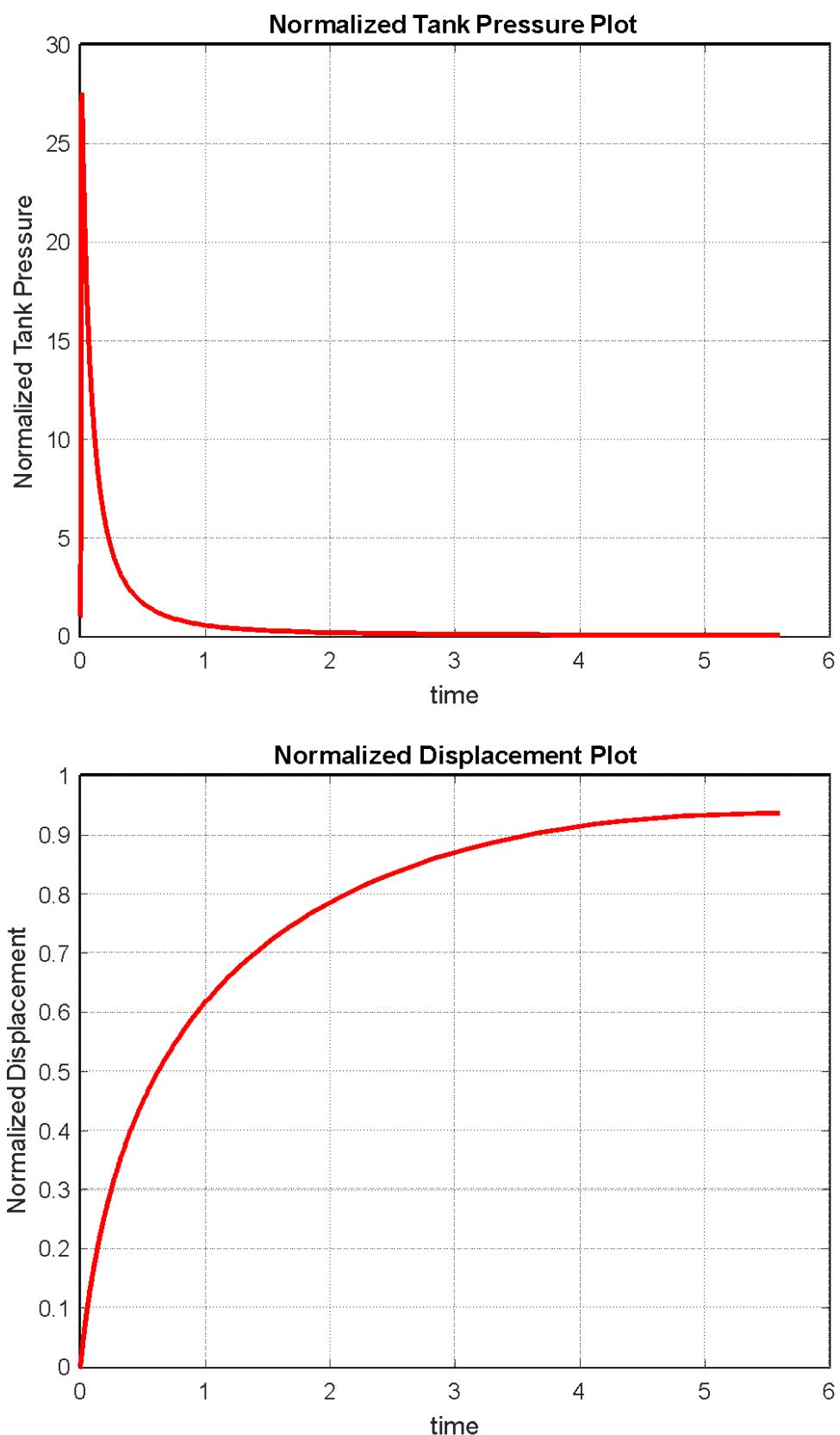
The m-file code for plotting the required values is given below:

```
function hw4q7
m=1500; D=0.25; Bw=2e9; ymax=7.5; Pa=101325; Ao=1.9*10^(-2); rho=1000;
Vai=0.004; Cd=0.5; g=9.81;
options=odeset('events',@StopSim);
[t,x]=ode45(@Eqns,[0 10],[0 15 101325],options);
Pt=x(:,3)/Pa;
Yt=x(:,1)/ymax;
```

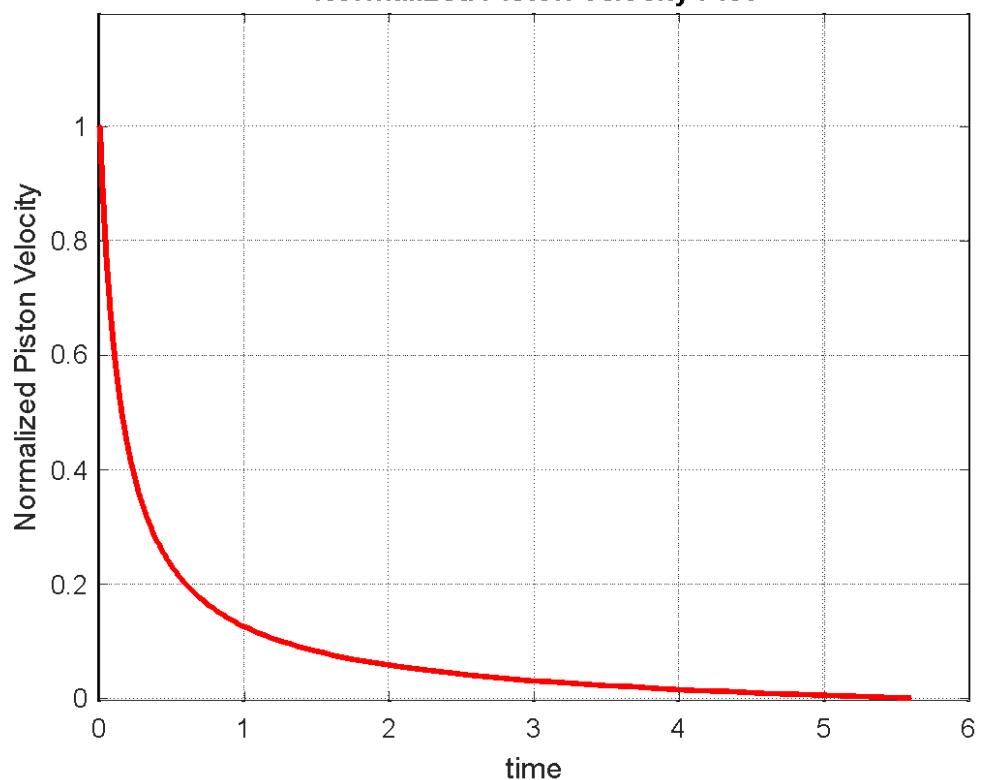
```

dyt=x(:,2)/15;
d2y=(-(x(:,3))*A) / (m*g);
figure(1)
plot(t,Pt,'r','Linewidth',2);
xlabel('time');
ylabel('Normalized Tank Pressure');
title('Normalized Tank Pressure Plot');
grid
figure(2)
plot(t,Yt,'r','Linewidth',2);
xlabel('time');
ylabel('Normalized Displacement');
title('Normalized Displacement Plot');
grid
figure(3)
plot(t,dyt,'r','Linewidth',2);
xlabel('time');
ylabel('Normalized Piston Velocity');
title('Normalized Piston Velocity Plot');
grid
figure(4)
plot(t,d2y,'r','Linewidth',2);
xlabel('time');
ylabel('Normalized Piston Acceleration');
title('Normalized Piston Acceleration Plot');
grid
function dx=Eqns(t,x)
dx=zeros(3,1)
y=x(1); dy=x(2); P=x(3);
dx(1)=x(2);
A=(pi*D*D)/4;
V=A*ymax*(1-(y/ymax));
Ba=1.4*(P);
Va=(Pa*Vai)/(P);
Ar=Va/V;
Lr=1-Ar;
Be=1/((Lr/Bw)+(Ar/Ba));
dV=-A*dy;
Q=Cd*Ao*((2/rho)*abs(P))^(0.5)*sign(P);
dx(2)=(-P*A)/m;
dx(3)=(Be/V)*(-Q-dV);
end
function [V,I,D]=StopSim(t,x)
I(1)=1;
V(1)=x(1)-7.5;
D(1)=0;
I(2)=2;
V(2)=x(2);
D(2)=0;
end
end

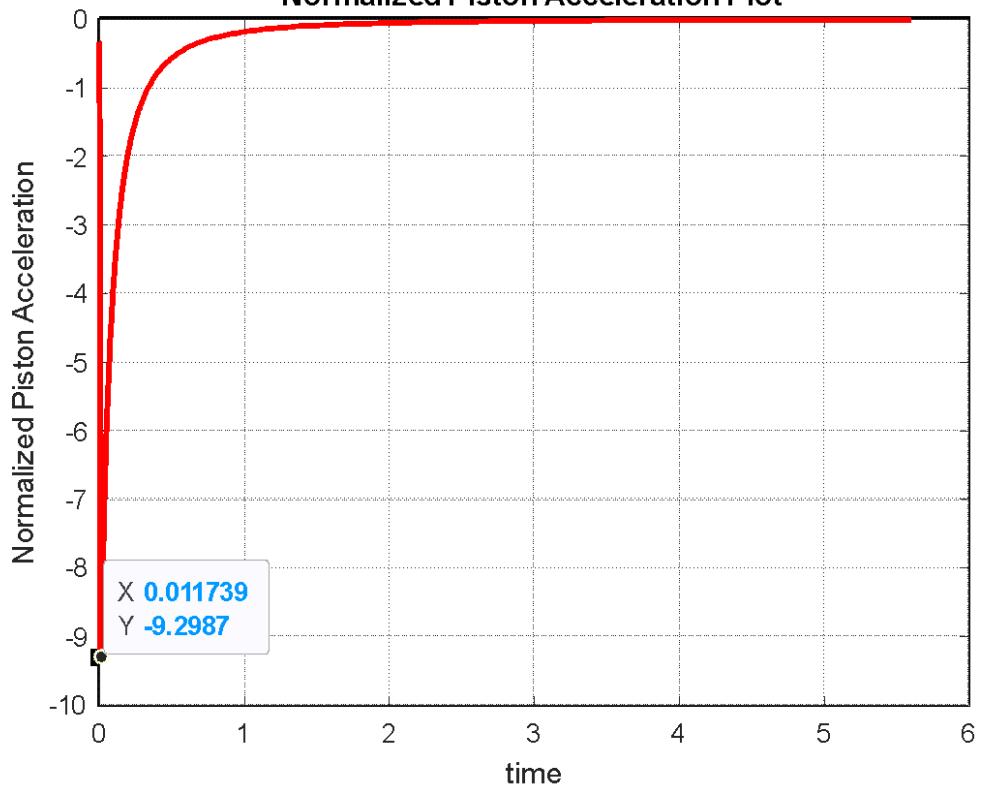
```



### Normalized Piston Velocity Plot

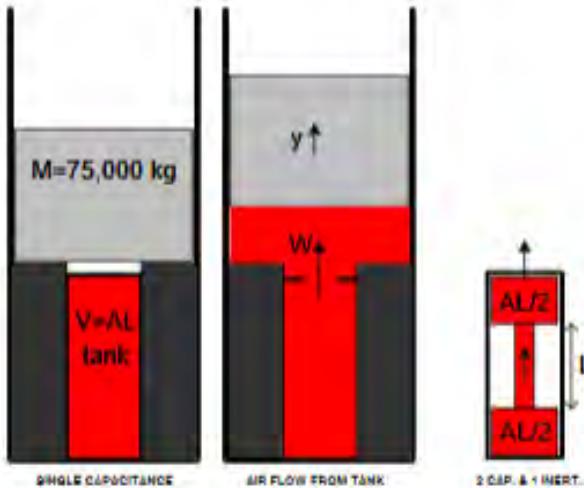


### Normalized Piston Acceleration Plot

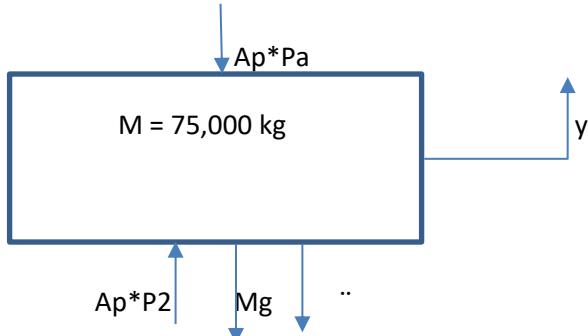


Last name, first \_\_\_\_\_

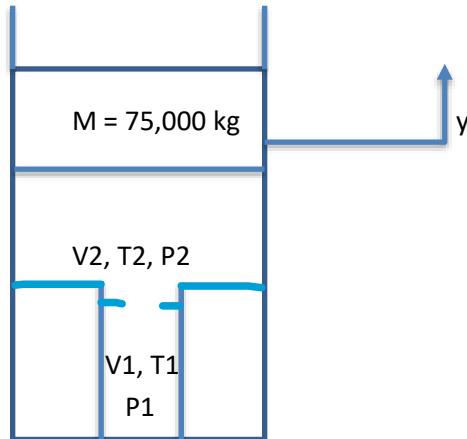
- An air tank is initially pressurized to 200 bar at 300 degrees K;  $A = 0.07 \text{ m}^2$  and  $L = 2 \text{ m}$ . At time  $t = 0$ , air starts flowing through an orifice (diameter = 0.2 m) into a chamber beneath a circular mass in a circular pipe causing the mass to accelerate upward. The objective is to use a lumped simulation model to estimate the maximum height  $y_{\max}$  that can be achieved. Assume the air below the mass (initial volume of  $0.0028 \text{ m}^3$ ) is initially atmospheric and the air above the mass remains atmospheric. Also, assume there is no wall friction and no leakage on the sides of the mass; the pipe diameter is 1 m. Use ode45 and stop the simulation at  $y_{\max}$ ; plot  $y(t)$ ,  $\dot{y}(t)$ , and the air pressure and temperature in the tank. Calculate the initial internal energy of the air in the tank and compare it with the potential energy of the mass at  $y_{\max}$ . Does the comparison make sense?
- Concerned with the accuracy of the assumption of a single lumped capacitance for the tank, a second simulation is to be conducted using two-lumped capacitances in series with a lumped inertance for the air in the tank. Plot  $y(t)$ ,  $\dot{y}(t)$ , and both air pressures and temperatures in the two capacitances. What are the effects of changing the model for the air in the tank; that is, how do the values of  $y_{\max}$  and potential energy compare with those of the single capacitance model?



### 1. Free Body Diagram



Tank Diagram:



Equations:

$$D_p(\text{pipe diameter}) = 1\text{m}; A_p(\text{pipe area}) = \frac{\pi \cdot 1^2}{4} \text{m}^2$$

$$D_o(\text{orifice diameter}) = 0.2\text{m}; A_o(\text{orifice area}) = \frac{\pi \cdot 0.2^2}{4} \text{m}^2$$

$$A_1(\text{tank area}) = 0.07\text{m}^2; L_1=2\text{m}; V_1(\text{tank volume}) = 0.07 \cdot 2 = 0.14\text{m}^3$$

$$\frac{P_1}{P_2} + \frac{2}{1} - \left( \frac{T_1}{T_2} - \frac{V_1^2}{2g} + \frac{V_2^2}{2g} \right) = 0$$

$$-\frac{P_1}{P_2} = \frac{1}{\sqrt{\frac{T_1}{T_2}} - \frac{V_1^2}{2g} + \frac{V_2^2}{2g}}$$

$$\frac{P_1}{P_2} = \frac{1}{\sqrt{\frac{T_1}{T_2}} - \frac{V_1^2}{2g} + \frac{V_2^2}{2g}}$$

$$-\frac{P_1}{P_2} = \frac{1}{\sqrt{\frac{T_1}{T_2}} - \frac{V_1^2}{2g} + \frac{V_2^2}{2g}}$$

$$\frac{P_1}{P_2} = \frac{2}{\sqrt{\frac{T_1}{T_2}} - \frac{V_1^2}{2g} + \frac{V_2^2}{2g}}$$

If  $P_1 > P_2$

$$= \frac{0.0404}{\sqrt{\frac{T_1}{T_2}} - \frac{V_1^2}{2g} + \frac{V_2^2}{2g}} \quad \text{Choked Flow}$$

If  $0.52828 < \frac{P_2}{P_1} < 1$

$$= 0.0835 \sqrt{\frac{2(1 - \frac{P_2}{P_1})}{\frac{T_1}{T_2}}} \quad \text{Unchoked Flow}$$

If  $P_2 > P_1$

$$= -\frac{0.0404}{\sqrt{\frac{T_1}{T_2}} - \frac{V_1^2}{2g}} \quad \text{Reverse Choked Flow}$$

If  $\frac{P_1}{P_2} > 0.52828$

$$= -0.0835 \quad \sqrt{\frac{1(2-1)}{2}} \textbf{Reverse Unchoked Flow}$$

$$2 = 0.0028 + \left( \frac{\frac{*}{*}}{4} \right)$$

$$\cdot 2 = \left( \frac{\frac{*}{*}}{4} \right) \cdot$$

$$\frac{\cdot 2}{2} - \cdot 2 = \frac{2}{2} \cdot 2$$

$$\frac{T_1}{T_2} * \left( \frac{T_1}{T_2} \right) - \cdot 2 = \frac{2}{2} \cdot 2$$

If P2>P1,

$$-\frac{2}{1} * \left( \frac{2}{1} \right) = \frac{1}{1} \cdot 1$$

State Variables and their Equations:

$$y=X1, \quad \cdot = 2, \quad P1=X3, \quad 1 = 4, \quad P2=X5, \quad 2 = 6$$

$$\begin{aligned} \cdot 1 &= X2 = \cdot; \\ \cdot 2 &= (Ap * \frac{P2 - Pa}{M}) - g \\ \cdot 3 &= \frac{-n * P1 * W}{1 * V1} \\ \cdot 4 &= -\frac{W}{V1} \\ \cdot 5 &= (n * \frac{P2}{V2}) * (\frac{W}{2} * (\frac{T1}{T2}) - dV2) \\ \cdot 6 &= \frac{2}{V2} * (\frac{W}{2} - dV2) \end{aligned}$$

**m-file code:**

```
function dsmhaw6q1
P10=200e5; T10=300; P20=101325; T20=300; g=9.81; M=75000; R=286.8; Cd=0.6;
Do=0.2; Dp=1; n=1.4;
Ao=pi*Do*Do/4;
Ap=pi*Dp*Dp/4;
A1=0.07; L1=2;
rho10=P10/(R*T10);
rho20=P20/(R*T20);
options=odeset('events',@StopSim);
[t,X]=ode45(@eqns,[0 5],[0 0 P10 rho10 P20 rho20],options);
plot(t,X(:,1), 'r', 'linewidth', 2)
xlabel('time, sec.')
ylabel('displacement of mass, m')
figure
```

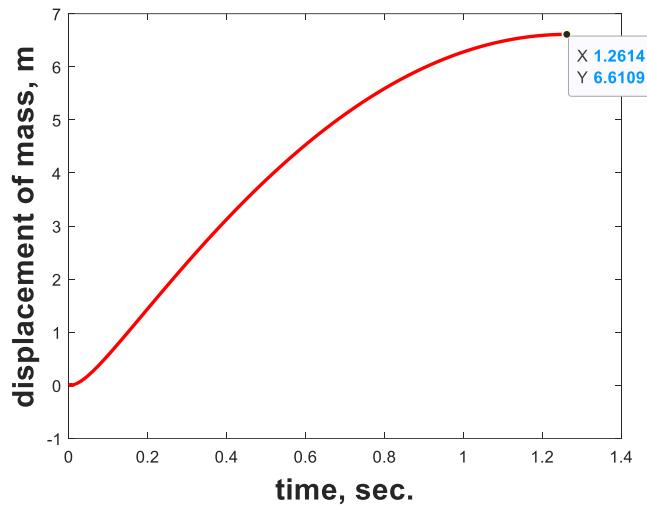
```

plot(t,X(:,2), 'r', 'linewidth', 2)
xlabel('time, sec.')
ylabel('velocity of mass, m/sec')
figure
plot(t,X(:,3), 'r', 'linewidth', 2)
xlabel('time, sec.')
ylabel('tank pressure, N/m^2')
T1=X(:,3)./(R*X(:,4));
figure
plot(t,T1, 'r', 'linewidth', 2)
xlabel('time, sec.')
ylabel('tank temperature, deg.K')
function dX=eqns(t,X)
dX=zeros(6,1);
y=X(1); dy=X(2); P1=X(3); rho1=X(4); P2=X(5); rho2=X(6);
T1=P1/(R*rho1);
T2=P2/(R*rho2);
V2=0.0028+Ap*y;
dV2=Ap*dy;
V1=A1*L1;
if P1>=P2
W=(0.0404*Cd*Ao*P1)/((T1)^0.5); %choked flow
if 0.52828<(P2/P1)<1
W=0.0835*Cd*Ao*((P2*(P1-P2))/T1)^0.5); %un choked flow
end
end
if P1<P2
W= -0.0404*Cd*Ao*P2/sqrt(T2);%choked reverse flow which will not happen as
simulation stops at ymax
if (P1/P2)>0.52828
W= -0.0835*Cd*Ao*sqrt(P1*(P2-P1)/T2);%un choked reversed flow may occur if P1
near P2
end
end
if (Ap*(P2-101325))<=(M*g)
X(1)=0;
end
dX(1)=X(2);
dX(2)=(Ap*(P2-101325)/M)-g;
dX(3)=(-n*P1*W)/(rho1*V1);
if P2>P1
dX(3)=((-n*P1*W)*(T2/T1))/(rho1*V1);
end
dX(4)=-W/V1;
dX(5)=((n*P2)/V2)*(((W/rho2)*(T1/T2))-dV2);
dX(6)=(rho2/V2)*((W/rho2)-dV2);
end
function [V,I,D]=StopSim(t,X)
I(1)=1;
V(1)=X(2);
D(1)=-1;
end
end

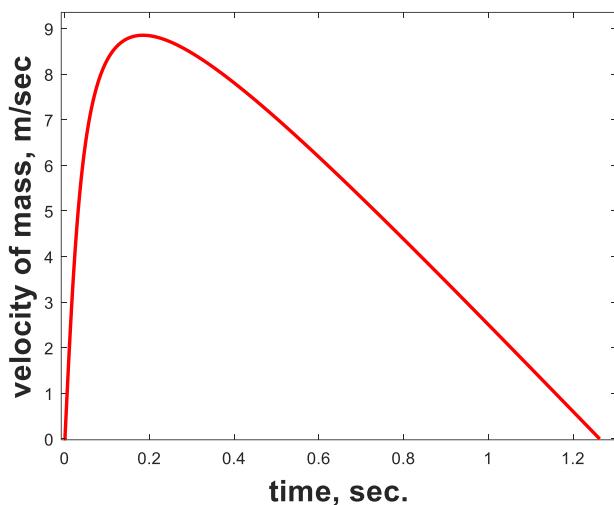
```

**Plots:**

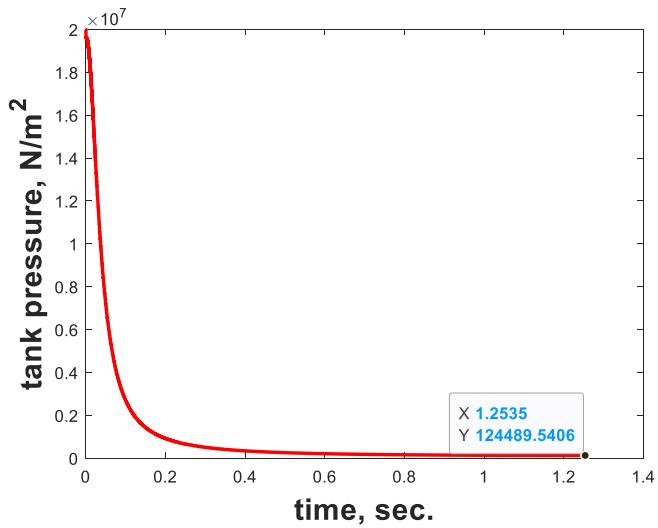
**Displacement Plot**



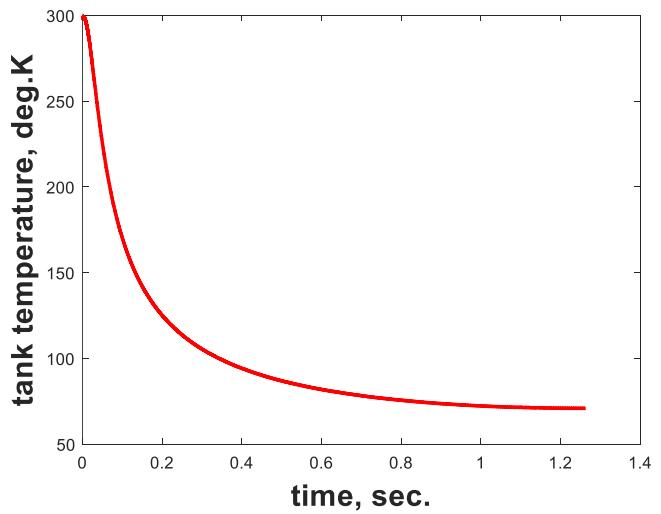
**Velocity Plot**



**Tank Pressure Plot**



**Tank Temperature Plot**



### Comparison of internal energy and potential energy:

Internal energy of a gas system can be given by

$$= \frac{V_{cv} C_v P_{cv}}{R}$$

Where,  $\gamma = /(-1)$

$$\text{Therefore, } \gamma = \frac{V_{cv} P_{cv}}{-1}$$

- Initial internal energy of the tank:

$$K_{\text{air}} = 1.4$$

$$= \frac{V_1(0^-) * P_1(0^-)}{k - 1} = \frac{0.07 * 2 * 200}{1.4 - 1} = 70 \text{ J}$$

- Internal energy of small volume:

$$= \frac{(0^-) * P_a(0^-)}{k - 1} = \frac{0.0028 * 101325}{1.4 - 1} = 709.275 \text{ J}$$

- Final Potential Energy of Mass:

$$= * * h = 75000 * 9.81 * 6.61 = 4863307.5 \text{ J}$$

- Final Potential Energy of Gas:

$$= \frac{(\infty) * P_1(\infty)}{k - 1} = \frac{((0.07 * 2) + 0.0028 + (\frac{(\pi * 1 * 1)}{4} * 6.56)) * 124489.5}{k - 1} \\ = 1660155 \text{ J}$$

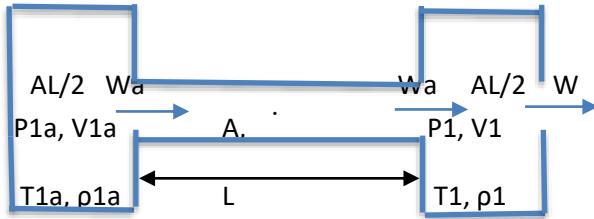
- Work Done Against Atmosphere:

$$= * * = 101325 * (\frac{* 1 * 1}{4}) * 6.61 = 526,026.8$$

**Total Initial Energy = 7000709 J**

**Total Final Energy + Work Done = 7,049,489.3 J**

Ratio of initial energy to final energy and work = **99.3 percent**. This makes sense as the initial and final values are very similar to each other.



**Equations:**

**Note – Variables in 2<sup>nd</sup> capacitance are kept same to avoid making excessive changes in M-file code.**

$$V1a = V1 = A1 * L1 / 2$$

$$-\frac{1}{1} = \frac{1}{1} \cdot i$$

$$1 = \frac{1}{1}$$

$$-\frac{1}{1} = \frac{1}{1} \cdot i$$

$$\frac{1}{1} * (\frac{T1a}{T1}) - \frac{1}{1} = \frac{1}{1} \cdot i$$

$$1 = \frac{1}{1}$$

$$\frac{1}{1} - \frac{1}{1} = \frac{1}{1} \cdot i$$

$$1 - 1 = \frac{1}{1} \cdot i$$

If  $P1 > P1a$ ,

$$1 - 1 = \frac{1}{1} \cdot i$$

As the line Volume doesn't change,  $Wa = W$

Additional state variables are:

$$Wa = X(7); \quad P1a = X(8); \quad 1 = (9);$$

New State Variable Equations are:

$$\begin{aligned} \dot{1} &= X2 = \\ \dot{2} &= (Ap * \frac{P2 - Pa}{M}) - g \end{aligned}$$

$$\begin{aligned}
\cdot 3 &= \frac{-n * P1}{V1} * \left( \frac{1}{1} * \left( \frac{T1a}{T1} \right) - \frac{1}{1} \right) \\
\cdot 4 &= \frac{Wa - W}{V1} \\
\cdot 5 &= \left( n * \frac{P2}{V2} \right) * \left( \frac{W}{2} * \left( \frac{T1}{T2} \right) - dV2 \right) \\
\cdot 6 &= \frac{2}{V2} * \left( \frac{W}{2} - dV2 \right) \\
\cdot 7 &= \left( \frac{1}{1} \right) * \left( 1 - 1 \right) \\
\cdot 8 &= \frac{-n * P1a * Wa}{1 * V1a} \\
\cdot 9 &= -\frac{Wa}{V1a}
\end{aligned}$$

### M-file code:

```

function dsmhw6q2
function Homework_6AB_5305_Fall_2019
R=286.8; % Gas constant for air, m^2/K*s^2
Cd=1; % Assumed Orifice discharge Coefficient
M=75000; % Mass, Kg
A=0.07; % Aera, m^2
L=2.0; % Length, m
d=0.2; % Orifice diameter, m
Ao=0.25*pi*d^2; % Orifice area, m^2
D=1.0; % Pipe diameter, m
Vi=0.0028; % Initial volume under mass, m^3
Patm=101325; % Atmospheric pressure, N/m^2
Pi= 200e5; % Initial pressure in tank, N/m^2. Note, 1 bar = 1e5 N/m^2
Ti= 300; % Initial Temperature in tank, K
Ap=0.25*pi*D^2; % Aera of pipe, m^2
g=9.81; % gravity, m/s^2
r=0.; % Heat transfer ratio, r = 0 for too fast for any heat transfer
k=1.4; % ratio of specific heats for air
n=k*(1-r)+r;
Cv=R/(k-1); % constant volume specific heat for air, J/(kg K)
V1=A*L; % tank volume, m^3
den1i=Pi/(Ti*R); % initial density of air in tank, kg/m^3
den2i=Patm/(Ti*R); % initial density of air under mass, kg/m^3
options=odeset('events',@StopSim);
[t,x,tE,xE]=ode45(@EqnsA,[0 5],[0 0 Pi Patm den1i den2i],options);

V1=A*L/2; % volume top half of tank, m^3
V0=A*L/2; % volume bottom half of tank, m^3
den1i=Pi/(Ti*R); % initial density of air in top half of tank, kg/m^3
den2i=Patm/(Ti*R); % initial density of air under mass, kg/m^3
den0i=Pi/(Ti*R)% initial density in bottom half of tank kg/m^3
[T,X,TE,XE]=ode45(@EqnsB,[0 5],[0 0 Pi Patm den1i den2i Pi den0i 0],options);

figure(1)
plot(t,x(:,1), 'r', T,X(:,1), 'k:', 'linewidth', 2)

```

```

xlabel('time,s')
ylabel('height of mass, m')
legend('one tank capacitance','two tank capacitances','location','best')
grid
figure(2)
plot(t,x(:,2),'r',T,X(:,2),'k:', 'linewidth',2)
xlabel('time, s')
ylabel('velocity of mass, m/s')
legend('one tank capacitance','two tank capacitances','location','best')
grid
figure(3)
plot(T,X(:,7), 'k--',t,x(:,3), 'r',T,X(:,3), 'r:',T,X(:,4), 'k:', 'linewidth',2)
xlabel('time,s')
ylabel('pressure, N/m^2')
legend('half volume at bottom of tank','single volume tank model','half volume at top
of tank',...
      'volume under mass','location','best')
grid
axis([0 0.01 1.85e7 2.05e7])
figure(4)
plot(T,X(:,3)./(R*X(:,5)), 'r',T,X(:,7)./(R*X(:,8)), 'k--'
  ,T,X(:,4)./(R*X(:,6)), 'k:', 'linewidth',2)
xlabel('time,s')
ylabel('air temperature, K')
legend('top of tank','bottom of tank','under mass','location','best')
grid
figure(5)
plot(T,X(:,3)./(R*X(:,5)), 'r',T,X(:,7)./(R*X(:,8)), 'k--'
  ,T,X(:,4)./(R*X(:,6)), 'k:', 'linewidth',2)
xlabel('time,s')
ylabel('air temperature, K')
legend('top of tank','bottom of tank','under mass','location','best')
grid
axis([0 0.0002 290 450])
figure(6)
plot(T,X(:,7), 'k--',t,x(:,3), 'r',T,X(:,3), 'r:',T,X(:,4), 'k:', 'linewidth',2)
xlabel('time,s')
ylabel('pressure, N/m^2')
legend('half volume at bottom of tank','single volume tank model','half volume at top
of tank',...
      'volume under mass','location','best')
grid

Initial_Internal_energy_J=Cv*den1i*Ti*(V1+V0) % initial internal energy in tank, J
Final_internal_energy_J=Cv*(XE(3)*V1+XE(7)*V0+XE(4)*(Vi+Ap*XE(1)))/R %final internal
energy all 3 volumes
Potential_energy_of_Mass_J=M*g*XE(1)
Total_energy_at_end_divided_initial_internal_energy=(Potential_energy_of_Mass_J ...
    +Final_internal_energy_J)/Initial_Internal_energy_J
Potential_energy_of_mass_divided_by_initial_internal_energy=M*g*XE(1)/Initial_Internal
energy_J
function dx=EqnsA(t,x)
    dx=zeros(6,1);
    y=x(1);dy=x(2);P1=x(3);P2=x(4);den1=x(5);den2=x(6);
    V2=Vi+(Ap*y);

```

```

dV2=Ap*dy;
T1=P1/(R*den1);
T2=P2/(R*den2);
if P1>=P2
    W12=0.0404*Cd*Ao*P1/sqrt(T1);
    if P2/P1>0.528
        W12=0.0835*Cd*Ao*sqrt(P2*(P1-P2)/T1);
        end
    end
if P1<P2
    W12=-0.0404*Cd*Ao*P2/sqrt(T2);
    if P1/P2>0.528
        W12=-0.0835*Cd*Ao*sqrt(P1*(P2-P1)/T2);
        end
    end
dx(1)=x(2);
dx(2)=Ap*(P2-Patm)/M-g;
    if dx(2)<0;% must check and see if mass has left anvil
        if x(1)<= 0
            dx(2)=0;% mass still on anvil
        end
    end
% must check to see if reversed flow to correctly use temperature
% ratio in the pressure capacitance equations
if W12 >=0 % flow going from tank to under mass
C12=(k*(1-r)*T1/T2+r)/(k*(1-r)+r);
dP1=-n*P1*W12/(V1*den1);
dP2=(n*P2/V2)*((W12*C12/den2)-dV2);
else % flow going from under mass to tank
C21=(k*(1-r)*T2/T1+r)/(k*(1-r)+r);
dP1=-n*P1*C21*W12/(V1*den1);
dP2=(n*P2/V2)*((W12/den2)-dV2);
end
dx(3)=dP1;
dx(4)=dP2;
dx(5)=-W12/V1;
dx(6)=((W12-(den2*dV2))/V2);
end
function dx=EqnsB(t,x)
dx=zeros(9,1);
y=x(1);dy=x(2);P1=x(3);P2=x(4);den1=x(5);den2=x(6);
P0=x(7);den0=x(8);W01=x(9);
V2=Vi+(Ap*y);
dV2=Ap*dy;
T1=P1/(R*den1);
T2=P2/(R*den2);
T0=P0/(R*den0);
if P1>=P2
    W12=0.0404*Cd*Ao*P1/sqrt(T1);
    if P2/P1>0.528
        W12=0.0835*Cd*Ao*sqrt(P2*(P1-P2)/T1);
        end
    end
if P1<P2
    W12=-0.0404*Cd*Ao*P2/sqrt(T2);

```

```

if P1/P2>0.528
W12=-0.0835*Cd*Ao*sqrt(P1*(P2-P1)/T2);
end
end
dx(1)=x(2);
dx(2)=Ap*(P2-Patm)/M-g;
if dx(2)<0;% must check and see if mass has left anvil
if x(1)<= 0
dx(2)=0;% mass still on anvil
end
end
% must check to see if reversed flow to correctly use temperature
% ratio in the pressure capacitance equations
C12=(k*(1-r)*T1/T2+r)/(k*(1-r)+r);
C01=(k*(1-r)*T0/T1+r)/(k*(1-r)+r);
C10=(k*(1-r)*T1/T0+r)/(k*(1-r)+r);
C21=(k*(1-r)*T2/T1+r)/(k*(1-r)+r);
if W12 >=0 % flow going from top of tank to under mass
dP2=(n*P2/V2)*((W12*C12/den2)-dV2);
if W01>=0
dP1=n*P1*(W01*C01-W12)/(V1*den1);
dP0=-n*P0*W01/(V0*den0);
dW01=A*(P0-P1)/L;
else
dP1=n*P1*(W01-W12)/(V1*den1);
dP0=-n*P0*W01*C10/(V0*den0);
end
else % flow going from under mass to top of tank
dP2=(n*P2/V2)*((W12/den2)-dV2);
if W01>=0
dP1=n*P1*(W01*C01-C21*W12)/(V1*den1);
dP0=-n*P0*W01/(V0*den0);
else
dP1=n*P1*(W01-C21*W12)/(V1*den1);
dP0=-n*P0*W01*C10/(V0*den0);
end
end
dx(3)=dP1;
dx(4)=dP2;
dx(5)=(W01-W12)/V1;
dx(6)=((W12-(den2*dV2))/V2);
dx(7)=dP0;
dx(8)=-W01/V0;
dx(9)=A*(P0-P1)/L;%+W01*dx(8)/den0;
end
function[Val,Ister,Dir]=StopSim(t,x)
Val(1)=x(2);Ister(1)=1;Dir(1)=-1; %stops simulation if velocity=0
%Val(2)=0.01-t;Ister(2)=1;Dir(2)=0;
end
end

```

In order to keep the inverted pendulum shown below vertical and return it to vertical from an initial non-zero angle, the pendulum is attached to a cart that moves laterally with displacement  $z$ . The lateral movement results from a force  $F_i$  acting on the cart. Since we are trying to keep the pendulum vertical, then the small angle assumption is sufficient since the angle should always be near zero. The small angle approximation differential equations for the pendulum and cart are

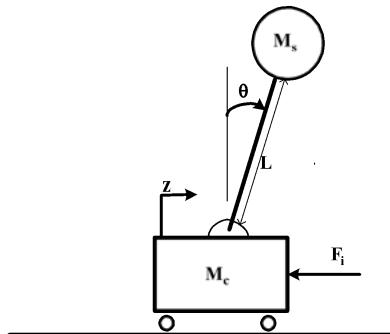
$$\ddot{\theta} - 5\theta = -0.5\ddot{z} \quad \text{new equation for pendulum considering } z\ddot{z} = -0.2\ddot{\theta} - 0.1F_i \quad \text{differential equation for the cart}$$

Demonstrate that the two equations above are equivalent to the following transfer function

$$\begin{aligned}\ddot{\theta} - 5\theta &= -0.5(-0.2\ddot{\theta} - 0.1F_i) \\ 0.9\ddot{\theta} - 5\theta &= 0.05F_i\end{aligned}\quad \text{Eqn - A}$$

Taking the Laplace transform of the above equation would give the same equation as below:

$$\theta(s) = G(s)F_i(s) = \left[ \frac{0.05}{0.9s^2 - 5} \right] F_i(s) \quad (1)$$



Assume that the output  $\theta_m$  of a transducer measuring the angle  $\theta$  contains noise, i.e.

$$\theta_m = \theta + n. \quad (2)$$

This measurement is used to generate the feedback force  $F_i$  on the cart using the following transfer function

$$F_i(s) = H(s)\theta_m(s) = -150 \left[ \frac{s+6}{s+8} \right] \theta_m(s) \quad (3)$$

Note, this transfer function for  $F_i$  is a continuous (analog) transfer function such as might be generated using an electrical operational amplifier circuit or a valve-controlled actuator system.

- (a) Express this system in state variable format, i.e.  $\dot{X} = AX + Bn$   $y = CX + Dn$   
where the output of interest  $y$  is  $\theta(t)$ .

Converting equation 1 to state variable format:

$$\begin{aligned}X_1 &= \theta; & X_2 &= \dot{\theta} = \dot{X}_1 \\ \dot{X}_2 &= \frac{5}{0.9}X_1 + \frac{0.05}{0.9}F_i \\ [\dot{X}_1 \dot{X}_2] &= [0 \ 1 \ \frac{5}{0.9} \ 0] [X_1 \ X_2] + [0 \ \frac{0.05}{0.9}] [F_i]\end{aligned}$$

$$y = \theta = [1 \ 0] [X_1 \ X_2] + [0] [F_i]$$

Converting equation 3 to state variable format:

$$\begin{aligned} F_i(s) &= -150(-\frac{2}{s+8} + 1)\theta_m(s) \\ \dot{[X_3]} &= [-8]X_3 + [1]\theta_m \\ y_n = F_i &= [300]X_3 + [-150]\theta_m \\ \text{eliminating } F_i \text{ and } \theta_m \text{ we get} \\ \dot{X}_1 &= X_2 \\ \dot{X}_2 &= \frac{5}{0.9}X_1 + \frac{0.05}{0.9}(300X_3 - 150(\theta + n)) \\ \dot{X}_2 &= -\frac{2.5}{0.9}X_1 + \frac{15}{0.9}X_3 - \frac{7.5}{0.9}n \\ \dot{X}_3 &= X_1 - 8X_3 + n \\ \dot{[X_1 \ X_2 \ X_3]} &= [0 \ 1 \ 0 \ -\frac{2.5}{0.9} \ 0 \ \frac{15}{0.9} \ 1 \ 0 \ -8] [X_1 \ X_2 \ X_3] + [0 \ -\frac{7.5}{0.9} \ 1] [n] \\ y_{th} = \theta &= [1 \ 0 \ 0] [X_1 \ X_2 \ X_3] + [0] [n] \end{aligned}$$

(b) What are the eigenvalues of this system? What are the magnitudes of the eigenvalues?

Eigen values can be obtained through following commands in MATLAB Command Window:

```
>> A=[0 1 0;-2.5/0.9 0 15/0.9;1 0 -8];
```

```
>> B=[0;-7.5/0.9;1];
```

```
>> C=[1 0 0];
```

```
>> D=[0];
```

```
>> Y=ss(A,B,C,D)
```

Y =

A =

x1	x2	x3	
x1	0	1	0
x2	-2.778	0	16.67
x3	1	0	-8

B =

u1

x1 0

x2 -8.333

x3 1

C =

x1 x2 x3

y1 1 0 0

D =

u1

y1 0

Continuous-time state-space model.

>> damp(Y)

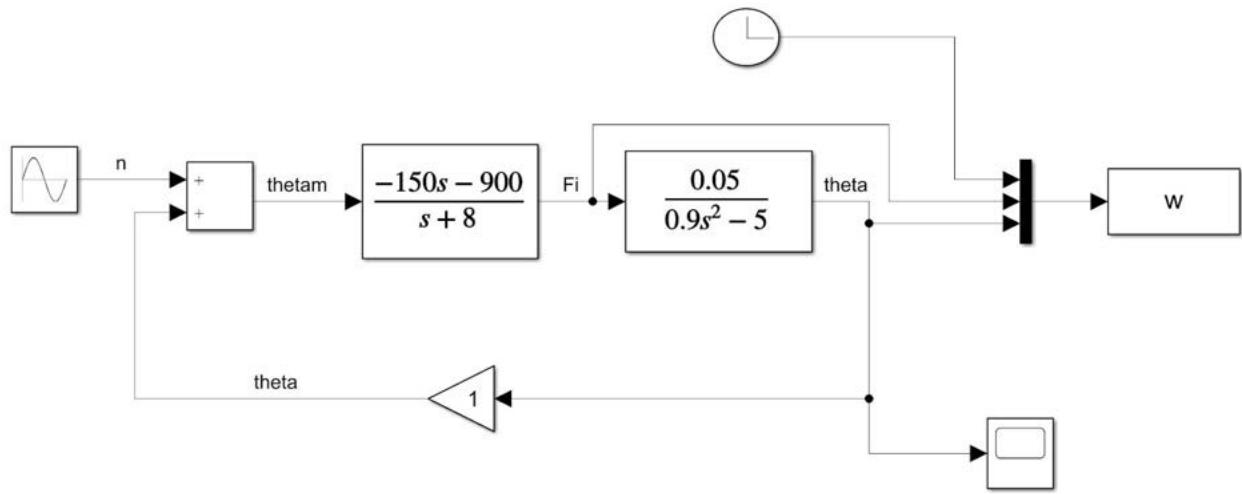
Pole	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-1.33e-01 + 8.37e-01i	1.57e-01	8.48e-01	7.51e+00
-1.33e-01 - 8.37e-01i	1.57e-01	8.48e-01	7.51e+00
-7.73e+00	Not Applicable	Not Applicable	1.29e-01

The eigen values for the system are **-0.133 ± 0.837i** and **-7.73**

Their magnitudes are **0.8475** and **7.73**

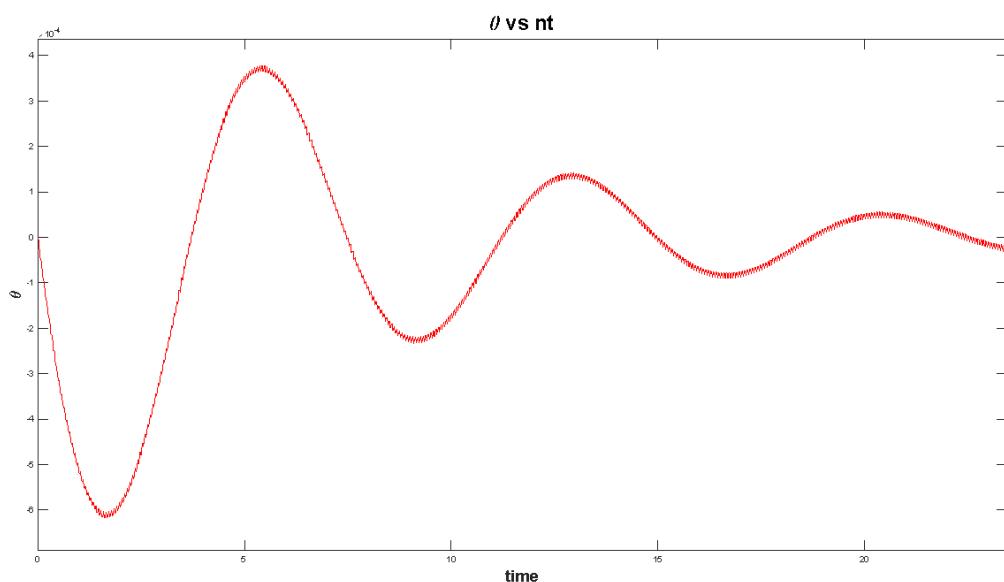
- (c) Evaluate the ability of the force feedback system to keep the pendulum approximately vertical even though the transducer output  $\theta_m(t)$  contains noise  $n(t)=.01\sin(100t)$ . You will need to generate plots of  $\theta(t)$  and  $F_i(t)$  for this  $n(t)$ . Note, even though all initial conditions are zero, the noise will continuously cause the pendulum to attempt to move off vertical. Explain what you learn from examining the plot.

### SIMULINK diagram:



The following commands in MATLAB provide plots for  $\theta(t)$  and  $F_i(t)$ :

```
>> plot(out.w(:,1),out.w(:,3),'r');
>> xlabel('time, sec.');
>> ylabel('\theta');
>> title('\theta vs nt')
```

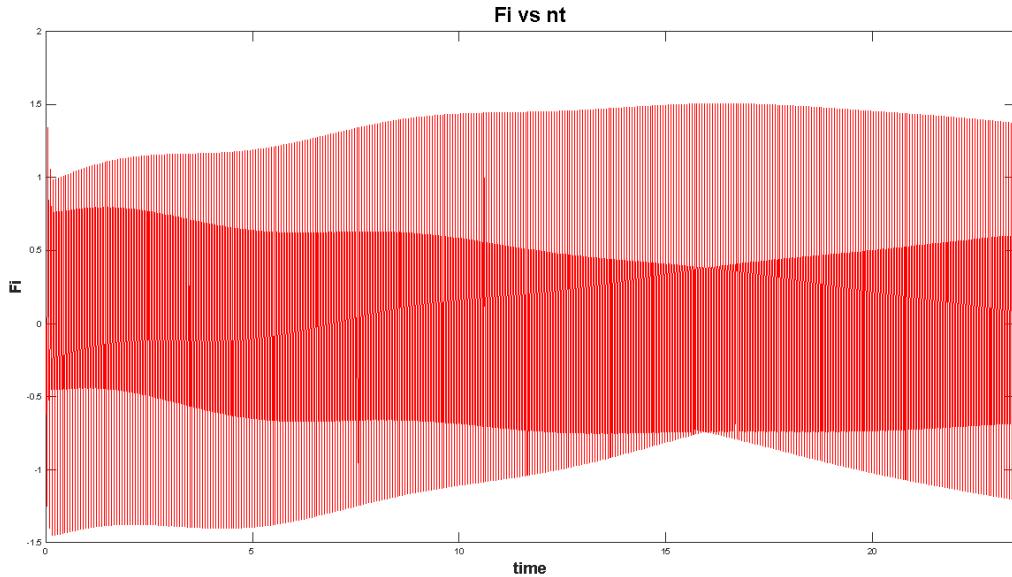


```
>> plot(out.w(:,1),out.w(:,2),'r');
```

```

>> xlabel('time, sec.');
>> ylabel('Fi');
>> title('Fi vs nt')

```



**Trying to achieve the following graphs using lsim gives the exact same plot:**

```

>> [num,den]=ss2tf(A,B,C,D)
num =

```

0 0 -8.3333 -50.0000

```

den =

```

1.0000 8.0000 2.7778 5.5556

```
>> Ytf=tf(num,den)
```

```

Ytf=

```

-8.333 s - 50

---

$s^3 + 8 s^2 + 2.778 s + 5.556$

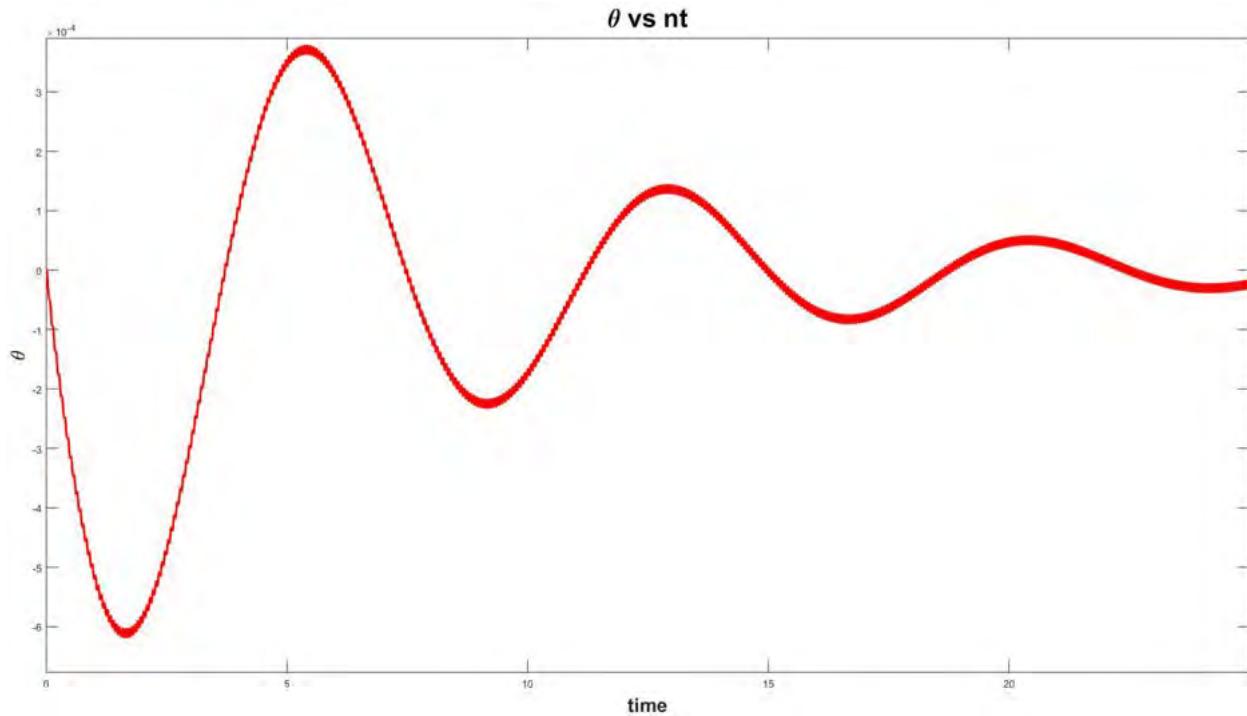
Continuous-time transfer function.

```
>> t=0:0.001:25; %time step of 0.001 is considered for better accuracy as the input frequency is
100 rad/sec.
```

```

>> n=0.01*sin(100*t);
>> [thplot,time]=lsim(Ytf,n,t);
>> plot(time,thplot,'r')

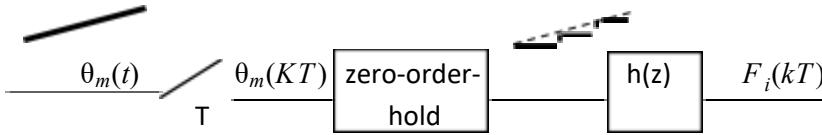
```



### Observation:

Upon observing the plots, it becomes apparent that the value of  $\theta(t)$  decays over time. While it is initially zero, the force  $F_i$  moves the pendulum but lowers the deflection with time. However, due to the noise in transducer, the value of  $\theta(t)$  never goes to a stable zero value and the pendulum continues to oscillate a few micrometers.

- (d) Instead of the force on the cart being generated by an analog system, suppose the force on the cart is to be generated by a digital computer using a digital version of the analog transfer function  $H(s)$  given above in Eq (3). Thus, the transducer output  $\theta_m$  will be sampled every  $T$  seconds using an A/D converter providing values of  $\theta_m(kT)$ . If we assume  $\theta_m$  is constant and equal to  $\theta_m(kT)$  until a new sample  $\theta_m(kT + T)$  is obtained, then  $\theta_m$  is converted from a smooth continuous function into a staircase function which is the input to a digital version of  $H(s)$ ,  $h(z)$ . This process can be achieved with a line of digital code in a micro-processor; the line of code is the inverse Z-transform of a transfer function obtained using the MATLAB command 'c2d'. Note, the default D/A in 'c2d' is the zero-order-hold.



Select a sample interval  $T$  that is sufficiently small to give a good digital approximation of the analog transfer function; use  $T \leq \frac{1}{10M}$  where  $M$  is the largest eigenvalue magnitude; use the eigenvalues corresponding to the analog feedback system. Use the MATLAB 'c2d' command with your value of  $T$  and the transfer function in Eq. (3) to get  $h(z)$  where

$$F_i(z) = h(z)\theta_m(z)$$

Using the inverse Z-transform, write the corresponding line of digital code for  $F_i(k)$ . Your code will have the following format:

$$F_i(k) = aF_i(k-1) + b\theta_m(k) + c\theta_m(k-1) \quad a = ? \quad b = ? \quad c = ?$$

$$F_i(s) = H(s)\theta_m(s) = -150 \left[ \frac{s+6}{s+8} \right] \theta_m(s)$$

$$T \leq \frac{1}{10M}; \quad \frac{1}{10M} = \frac{1}{10} * 7.73 = 0.01293$$

However, since there is also an input frequency also present for the system. The sample time is taken according to that. For a frequency of 100 rad/sec, a sample time of 0.001 would give 10 points per cycle. Therefore,  $T=0.001$  would provide us with an accurate solution

Taking  $T=0.001$ , the following MATLAB commands are used to obtain  $h(z)$ :

```
>> Fi=tf([-150 -900],[1 8])
Fi =
-150 s - 900
-----
s + 8
```

Continuous-time transfer function.

```
>> Fiz=c2d(Fi,0.001)
Fiz =
-150 z + 149.1
-----
z - 0.992
```

Sample time: 0.001 seconds

Discrete-time transfer function.

$$F_i(z) = \frac{-150z+149.1}{z-0.992} \theta_m(z)$$

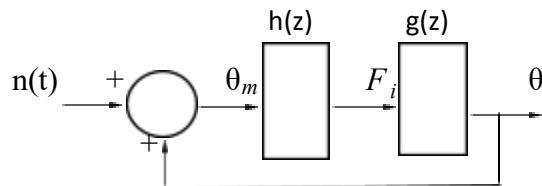
$$h(z) = \frac{-150z+149.1}{z-0.992}$$

Taking inverse Z-transform, the following equation is obtained:

$$F_i(k) = 0.992F_i(k-1) - 150\theta_m(k) + 149.1\theta_m(k-1)$$

Where,  $a = 0.992$ ,  $b = -150$  and  $c = 149.1$

- (e) Using the same T, use 'c2t' to get the digital version  $g(z)$  of the transfer function  $G(s)$  in Eq. (1). Use SIMULINK to get plots of  $\theta(kT)$  and  $F_i(kT)$  assuming  $n(t) = 0.01\sin(100t)$ . How do the plots compare with those in (c)?



$g(z)$  is obtained using the following MATLAB commands:

```
>> th=tf(0.05,[0.9 0 -5])
```

th =

0.05

-----  
0.9 s^2 - 5

Continuous-time transfer function.

```
>> thz=c2d(th,0.001)
```

thz =

2.778e-08 z + 2.778e-08

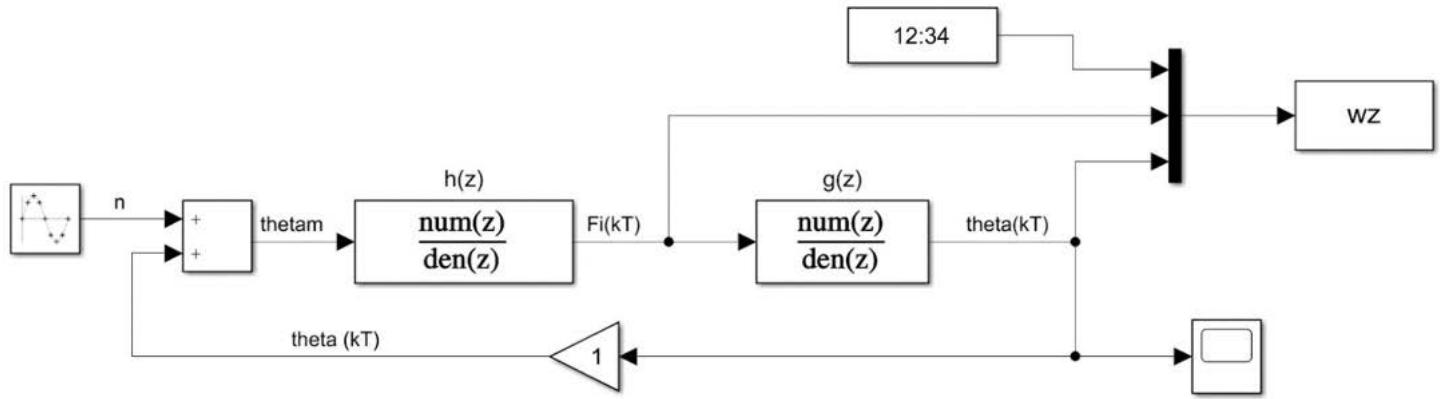
-----  
z^2 - 2 z + 1

Sample time: 0.001 seconds

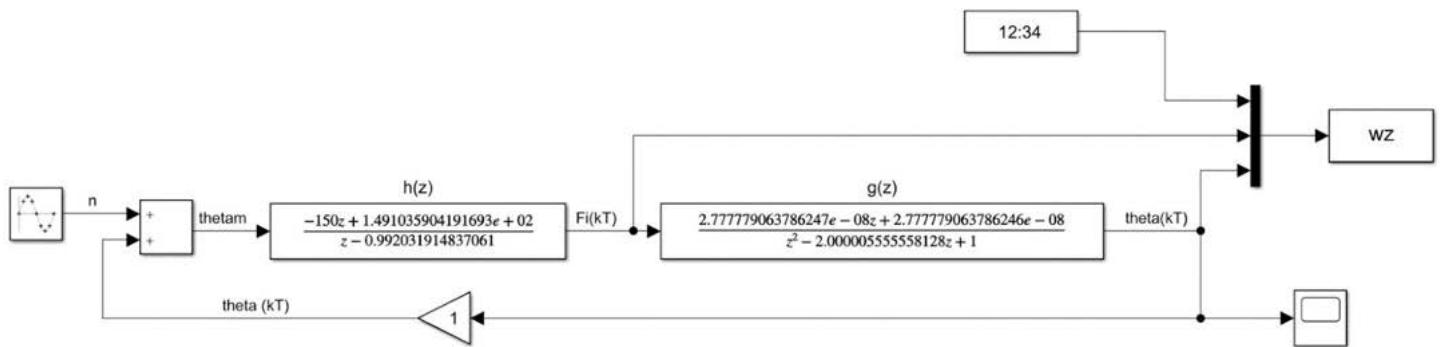
Discrete-time transfer function.

$$g(z) = \frac{2.778e-08 z + 2.778e-08}{z^2 - 2 z + 1}$$

### SIMULINK diagram:

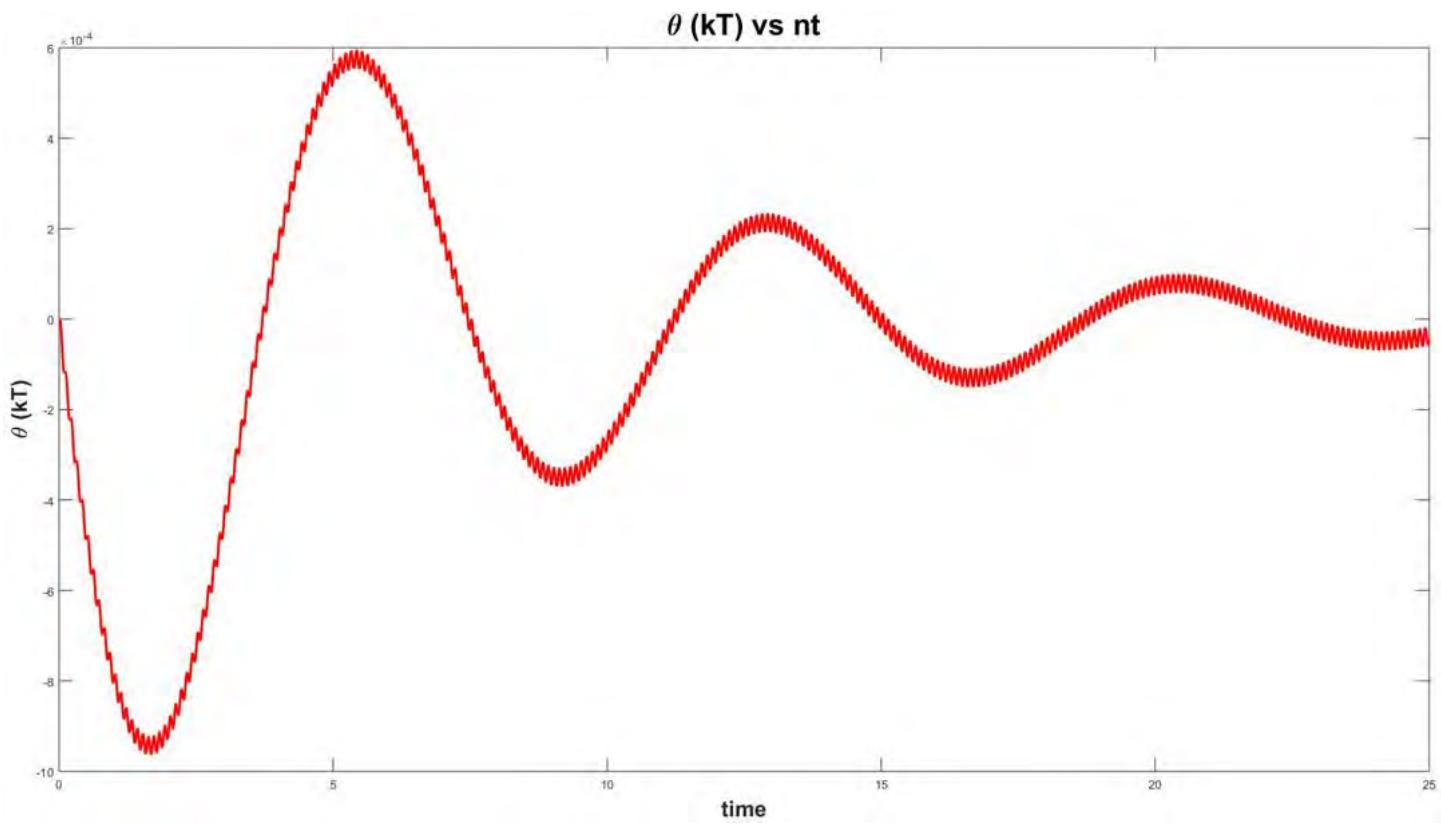


### Exact SIMULINK diagram:

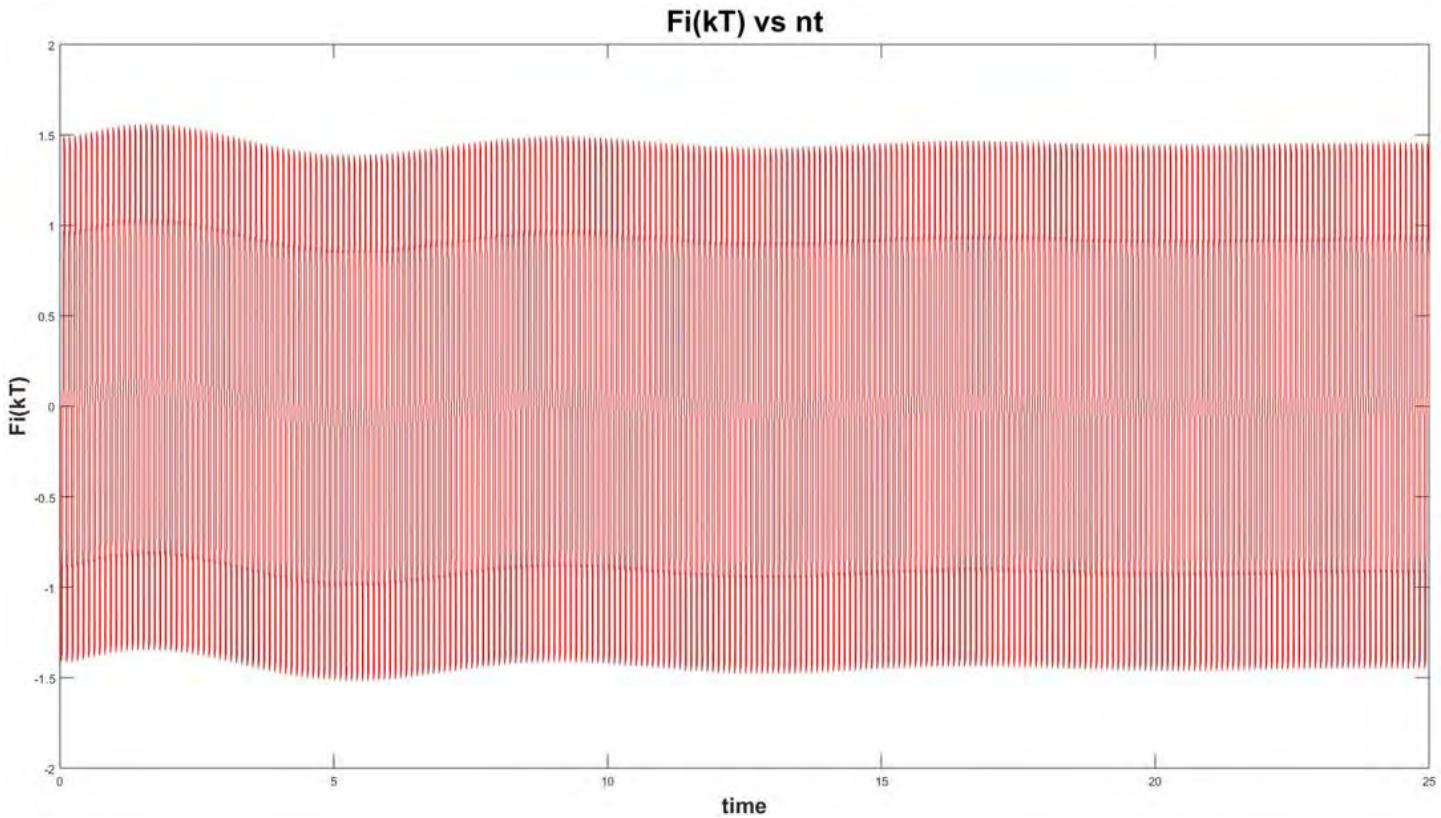


The following commands in MATLAB provide plots for  $\theta(kT)$  and  $F_i(kT)$  sample time = 0.001

```
>> plot(out.wz(:,1),out.wz(:,3),'r')
```



```
>> plot(out.wz(:,1),out.wz(:,2),'r')
```



**Using LSIM to get the plot for  $\theta(kT)$  :**

```
>> Yz=c2d(Ytf,0.001)
```

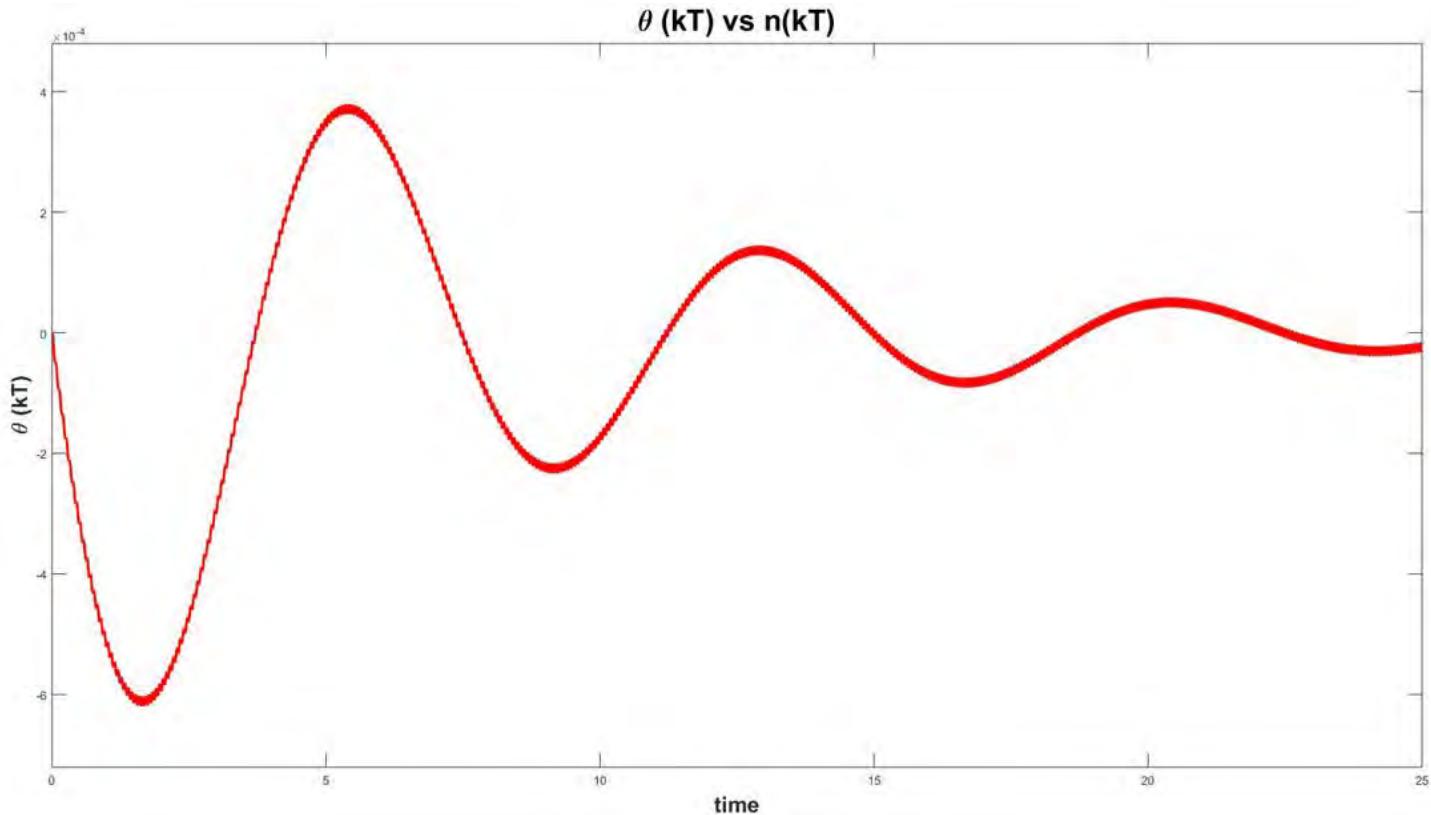
Yz =

$$-4.164e-06 z^2 - 2.213e-08 z + 4.136e-06$$

$$-----$$
$$z^3 - 2.992 z^2 + 2.984 z - 0.992$$

Sample time: 0.001 seconds  
Discrete-time transfer function.

```
>> [thzplot,time]=lsim(Yz,n,t);  
>> plot(time,thzplot,'r')
```



The plot obtained for  $\theta(kT)$  in part e through lsim is very similar to the plots obtained for  $\theta(T)$  in part c. The plot obtained for  $\theta(kT)$  through lsim has slightly larger amplitudes which probably occur due to the approximation of numerator and denominator values. The effects on  $\theta(kT)$  due to noise are almost consistent with the previous plots. The magnitudes of  $F_i(kT)$  are similar to  $F_i$  plotted in part c.

PROB. 6. SHOWN BELOW IS A TYPICAL MULTIPORT SYSTEM.

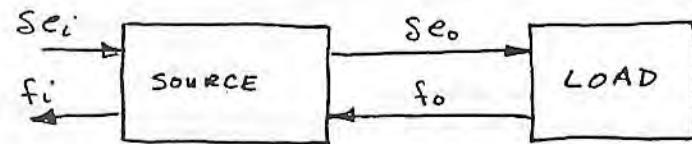
- DERIVE MATHEMATICAL MODELS FOR THE EFFORT GAIN,  $G_e$ , THE FLOW GAIN,  $G_f$ , AND THE POWER GAIN,  $G_{\pi}$ .
- SHOW THAT (DERIVE MATHEMATICALLY) MAXIMUM POWER TRANSFER OCCURS WHEN SOURCE AND LOAD IMPEDANCES ARE MATCHED (i.e.,  $R_s = R_L$ ).
- SHOW IN A TABLE WHAT HAPPENS TO  $G_e$ ,  $G_f$ , AND  $G_{\pi}$  FOR  $R_s/R_L = \infty$ , 1, AND 0. DISCUSS WHEN EACH GAIN IS MAXIMUM.

HINT: BE SURE TO DERIVE A COMPLETE EXPRESSION FOR THE VARIABLES OF INTEREST USING ALL EQUATIONS BEFORE FINDING GAINS. (e.g., FIND  $S_{eo} = F(S_{ei})$  BEFORE FINDING  $G_e$ )

$$G_e \triangleq \frac{\partial S_{eo}}{\partial S_{ei}}$$

$$G_{\pi} = \frac{\partial (S_{eo} f_o)}{\partial (S_{ei} f_i)}$$

$$G_f \triangleq \frac{\partial f_o}{\partial f_i}$$



SOURCE:

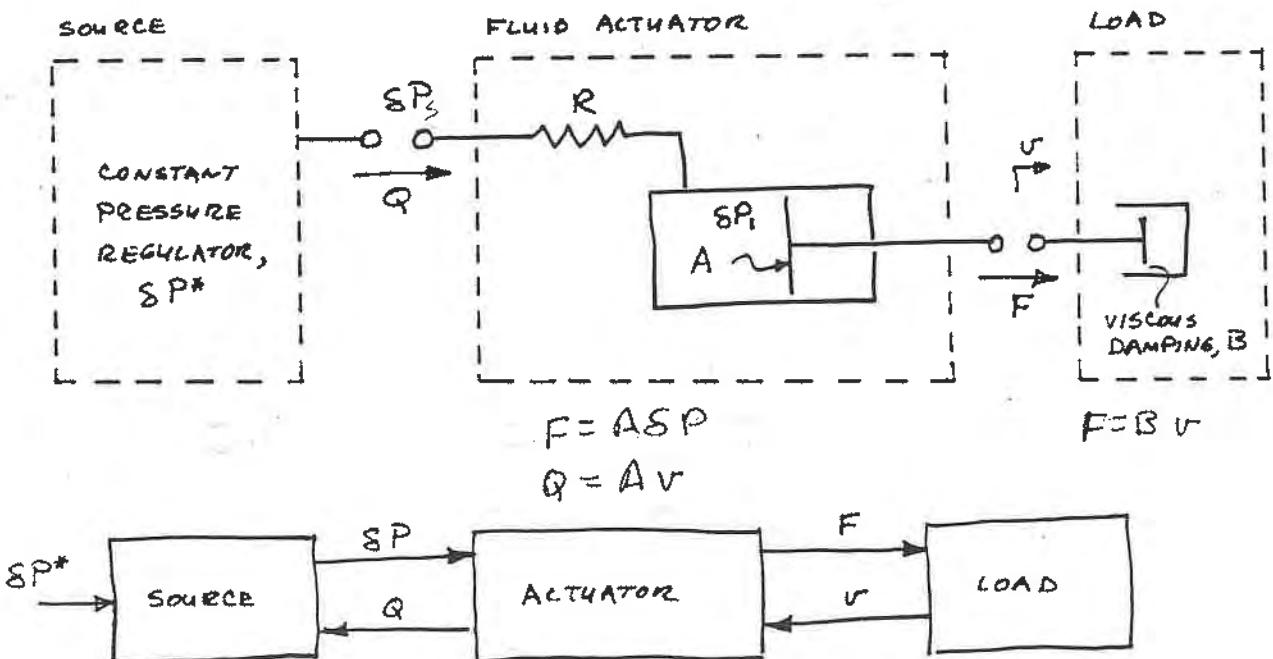
$$\begin{bmatrix} S_{ei} \\ f_i \end{bmatrix} = \begin{bmatrix} G_{eo} & -R_o \\ \frac{1}{R_s} & 0 \end{bmatrix} \begin{bmatrix} S_{eo} \\ f_o \end{bmatrix}$$

LOAD:

$$f_o = \frac{1}{R_L} S_{eo}$$

PROB 7. SHOWN BELOW IS A HYDRAULIC ACTUATOR DRIVING A MECHANICAL LOAD.

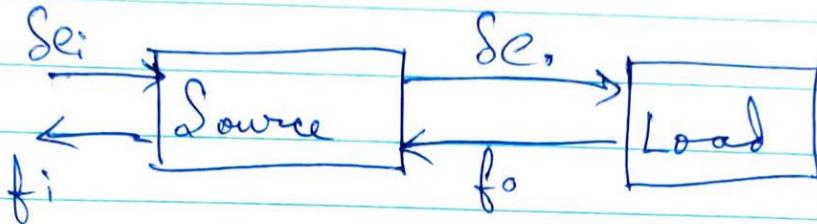
- DERIVE THE MULTIPORT MODEL (FOR LIMITED MOTION) AT THE INDICATED INTERFACES.
- SKETCH THE ACTUATOR OUTPUT CHARACTERISTICS (AT CONSTANT PRESSURE) AND THE LOAD INPUT CHARACTERISTICS. INDICATE ALL OF THE GAINS, IMPEDANCES, ETC. ON THE GRAPHS.
- SHOW HOW THE AREA OF THE PISTON,  $A$ , CAN BE USED TO EFFECT AN IMPEDANCE MATCH AND STATE THE RESULTING EXPRESSION FOR  $A$ .
- CALCULATE THE NUMERICAL VALUE OF  $A$  REQUIRED FOR AN IMPEDANCE MATCH USING THE VALUES SHOWN BELOW.
- SHOW WHAT HAPPENS TO THE  $\frac{\text{ACTUATOR}}{\text{LOAD}}$  INPUT CHARACTERISTICS AS THE AREA VARIES FROM 0 TO  $\infty$ .
- WHAT IS THE EXPRESSION FOR THE  $\frac{\text{ACTUATOR}}{\text{LOAD}}$  INPUT CHARACTERISTIC WHEN AN IMPEDANCE MATCH EXISTS BETWEEN THE ACTUATOR AND THE LOAD?



$$R = 1.0 \cdot 10^6 \frac{\text{KN S}}{\text{MS}}$$

$$B = 250 \frac{\text{KN S}}{\text{M}}$$

### PROBLEM - 6



$$\begin{bmatrix} \text{Se}_i \\ f_i \end{bmatrix} = \begin{bmatrix} G_{\text{eas}} & -R_s \\ \frac{1}{R_s} & 0 \end{bmatrix} \begin{bmatrix} \text{Se}_o \\ f_o \end{bmatrix}$$

Load:

$$f_o = \frac{1}{R_L} \text{Se}_o \quad \text{--- (3)}$$

$$\text{Se}_o = G_{\text{eas}} \text{Se}_i - R_o f_o \quad \text{--- (1)}$$

$$f_i = \frac{\text{Se}_i}{R_s} \quad \text{--- (2)}$$

(a) Substituting equation (3) in (1),

$$\text{Se}_o = G_{\text{eas}} \text{Se}_i - \frac{R_o \text{Se}_o}{R_L}$$

Substituting equation (2) above

$$\text{Se}_o = G_{\text{eas}} f_i R_s - \frac{R_o \text{Se}_o}{R_L}$$



(2)

Thus,

$$S_{e_0} = \frac{\text{Gear } f_i \cdot R_i}{\left(1 + \frac{R_o}{R_L}\right)} = \frac{\text{Gear } S_{e_i}}{\left(1 + \frac{R_o}{R_L}\right)}$$

$$G_e = \frac{S_{e_0}}{S_{e_i}} = \frac{\text{Gear}}{\left(1 + \frac{R_o}{R_L}\right)}$$

$$f_o = \frac{\text{Gear } S_{e_i}}{\left(1 + \frac{R_o}{R_L}\right)} = \frac{\text{Gear } e_i - R_o f_o}{R_L}$$

✓ (i)  $G_f = \frac{f_o}{f_i} = \frac{\text{Gear } S_{e_i} \times R_i}{\left(1 + \frac{R_o}{R_L}\right) \times S_{e_i}} = \frac{\text{Gear } R_i}{\left(1 + \frac{R_o}{R_L}\right)}$

$$\frac{f_o}{f_i} = G_f = \left( \frac{\text{Gear } e_i - R_o f_o}{R_L} \right) \times \frac{R_i}{S_{e_i}}$$

(2) —  ~~$f_i$~~   $f_o \left(1 + \frac{R_o R_i}{R_L S_{e_i}}\right) = \left(\frac{\text{Gear } e_i R_i}{R_L S_{e_i}}\right) f_i$

$$\frac{f_o}{f_i} = G_f = \frac{S_{e_i} \text{Gear } R_i}{R_L S_{e_i} + R_o R_i}$$

On L.H.P.  $\frac{\text{Gear } R_i}{R_L}$

(3)

$$G_{POWER} = G_f \times G_e$$

$$G_{POWER} = \frac{G_{eas} R_i}{\left(1 + \frac{R_o}{R_L}\right)} \times \frac{G_{eas}}{\left(1 + \frac{R_o}{R_L}\right)}$$

$$G_{POWER} = \frac{G_e^2}{\left(1 + \frac{R_o}{R_L}\right)^2} R_i = \left[ \frac{G_{eas}}{\left(1 + \frac{R_o}{R_L}\right)} \right]^2 \times R_i$$

(b) Effort gain determines max power transfer.

$$G_e = \frac{G_{eas}}{\left(1 + \frac{R_o}{R_L}\right)}$$

if  $\frac{R_o}{R_L} = 0$ ,  $G_e = G_{eas}$   
but the system is stalled

if  $\frac{R_o}{R_L} \gg 1$ ,  $G_e \rightarrow 0$

~~Thus~~, when  $\frac{R_o}{R_L} = 1$ ,  $G_e = \frac{G_{eas}}{2}$

Or, Power = Effort  $\times$  Flow  
only for Linear System  $\frac{\partial(E.F)}{\partial F} = 0$  for max power,  
 $(1/2 \text{ MaxPower} \times 1/2 \text{ MaxFlow})^{387}$

(4)

$$\text{Power} = S_{ei} \times f_o$$

$$\text{Power} = \left( \frac{S_{e_0} + R_{of}}{G_{cas}} \right) \left( \frac{S_{e_0}}{R_L} \right)$$

$$\text{Power} = \frac{S_{e_0}^2}{G_{cas} R_L} + \left( \frac{R_o}{R_L} \right) S_{e_0} f_o$$

if  $R_o \gg R_L$  then Power  $\approx R_o S_{e_0} f_o$

if  $R_o \ll R_L$  then Power  $\approx 0$

$$\frac{\partial (EF)}{\partial F} = 0 \quad \cancel{\frac{R_o S_{e_0}}{R_L} > 0}$$

(for max power)

Since  $S_{e_0} = G_{cas} S_{ei} - R_{of}$ ,

it can be proven that  
max power is when  $R_L = R_o$

(c)

$R_L/R_o$	$G_e$	$G_f$	$G_{POWER}$
$\infty$	$G_{cas}$	$G_{cas} R_i$	$G_{cas}^2 R_i$
0	0	0	0
1	$\frac{G_{cas}}{2}$	$\frac{G_{cas} R_i}{2}$	$\frac{G_{cas}^2 R_i}{4}$

(5)

## PROBLEM - 7

(a)

Impedance ( $R$ ) causes resistance in flow ( $Q$ ). So,

$$S_P = \text{Source} (P_s) - RQ$$

$$S_P = \frac{F(\text{force})}{A} \text{ and } Q = A V$$

$$\text{thus, } \frac{F}{A} = P_s - AVR$$

$$F = AP_s - A^2 VR$$

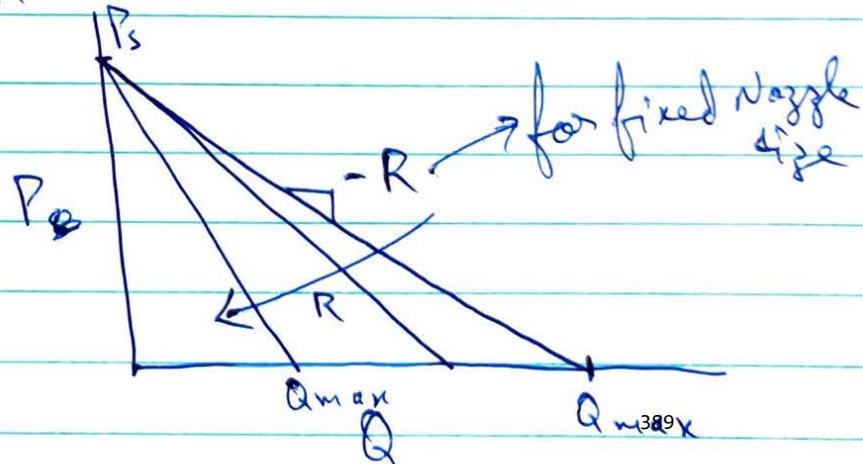
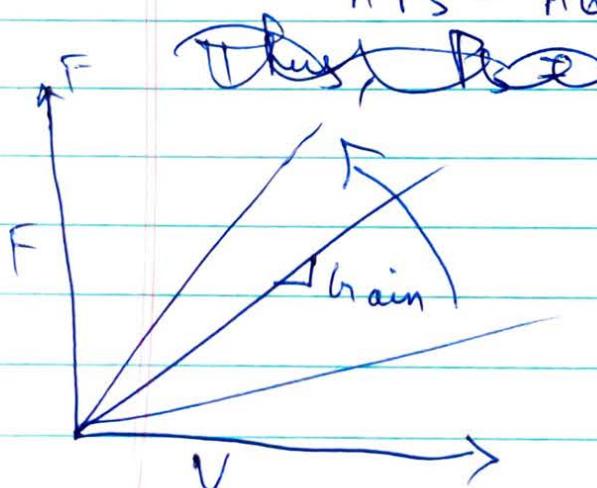
$$\begin{bmatrix} F \\ Q \end{bmatrix} = \begin{bmatrix} A & -A^2 R \\ 0 & A \end{bmatrix} \begin{bmatrix} P_s \\ V \end{bmatrix}$$

(b)

From above equations,

$$AP_s = A^2 VR \quad \text{when force is 0}$$

$$AP_s = AQR$$



$$G_{POWER} = G_f \times G_e$$

$$G_{POWER} = \frac{G_{eas} R_i}{\left(1 + \frac{R_o}{R_L}\right)} \times \frac{G_{eas}}{\left(1 + \frac{R_o}{R_L}\right)}$$

$$G_{POWER} = \frac{G_{eas}^2 R_i}{\left(1 + \frac{R_o}{R_L}\right)^2} = \left(\frac{G_{eas}}{\left(1 + \frac{R_o}{R_L}\right)}\right)^2 R_i$$

(\*)

From the  $G_{POWER}$  Equation above,

as  $\frac{R_o}{R_L} \gg 1$ ,  $G_{POWER} \rightarrow 0$

as  $\frac{R_o}{R_L} \ll 1$ ,  $G_{POWER} \rightarrow G_{eas}^2 R_i$   
but flow  $\rightarrow 0$

(\*)

However, at  $\frac{R_o}{R_L} = 1$ ,  $G_{POWER} =$

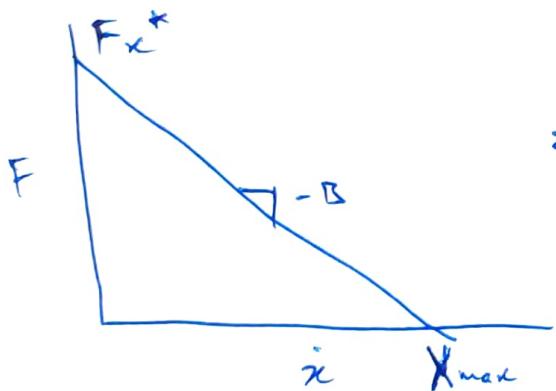
Project:	Computed:	Date
Subject:	Checked:	Date:
Task:	Page:	of:
Job #:	No:	

## Handheld Sprayer

Design  $\rightarrow$  lever ratio; piston pump area; nozzle dia to get pressure-flow characteristic for a sprayer of own specifications.

Drive simplified equations for flow delivery neglecting things done to refill the pump.

$$F_x = F_x^* - Bx$$



$\Rightarrow$  Here, max Power is achieved at  $F = \frac{F_x^*}{2}$  and  $X = \frac{X_{max}}{2}$

Neglecting the pump refill terms,

$$\dot{y} = \dot{a}x$$

and

$$F_x = \dot{a}F_y$$

$$P = \frac{F_y - k(y + y_0)}{A_p}$$

$$\dot{y} = \frac{Q}{A_p}$$

where  $A_p$  is the area of the piston.

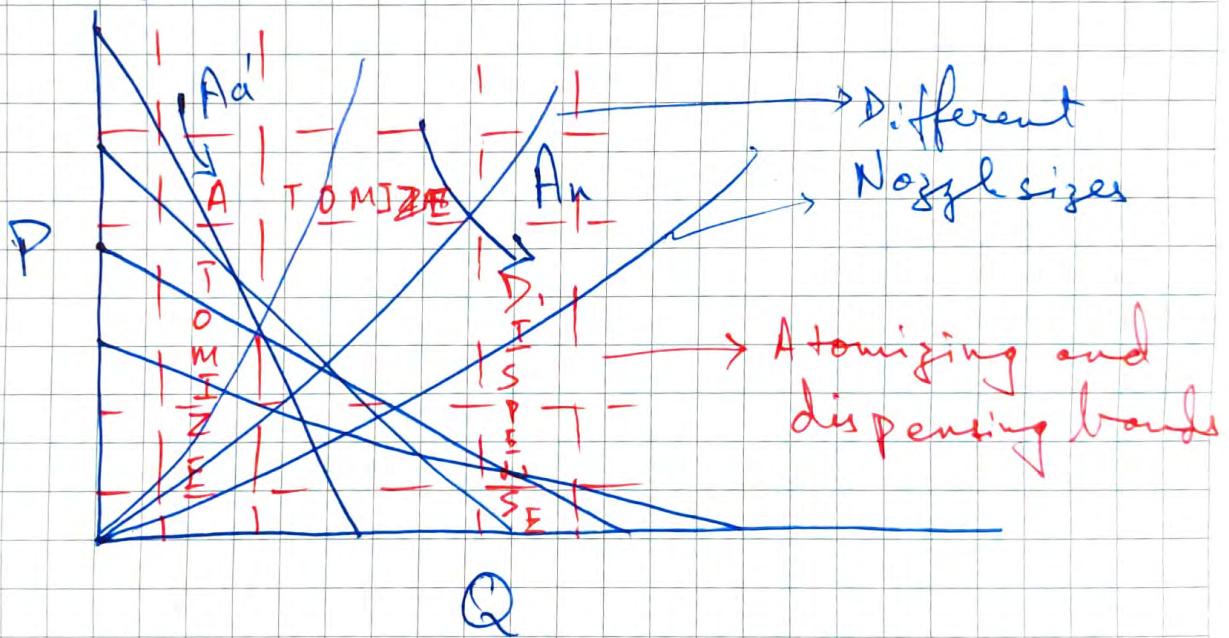
~~I will neglect the area of piston (changes in  $A_p$ ) and take it as fixed because it wasn't asked in class neither mentioned under considerations in class~~ 391

$$P = \frac{F_x - k_a(y + y_p)}{\dot{a} A_p}; \quad Q = \dot{a} A \dot{x}$$

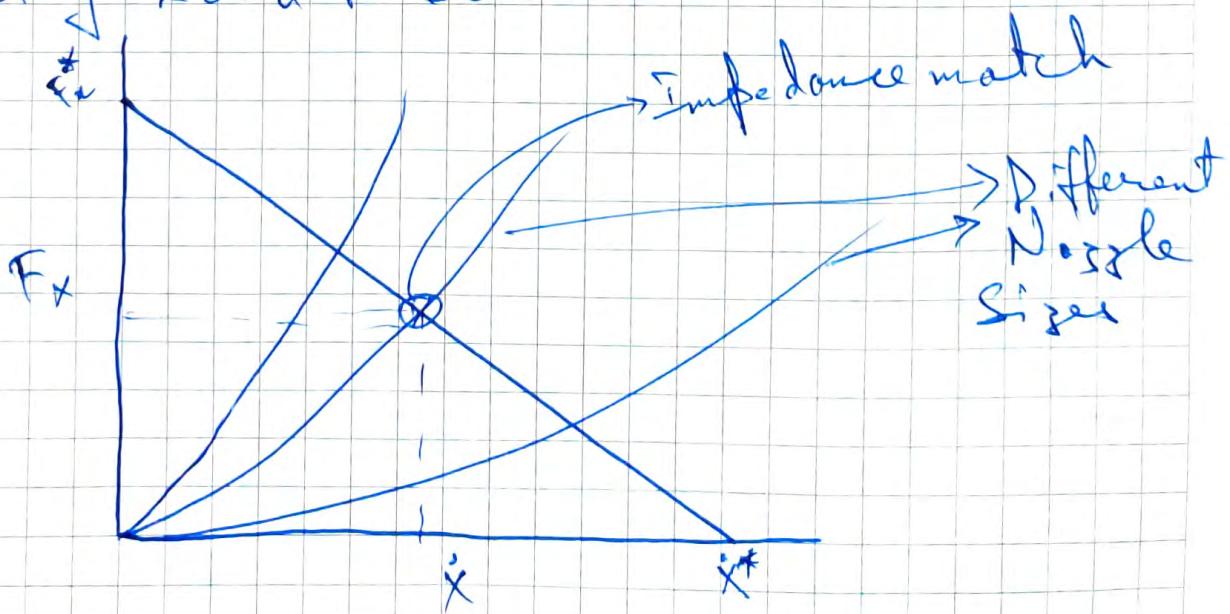
(2)

$$Q_n = C_d A_n \sqrt{2 S P}$$

Reflecting Source to Load



Reflecting Load to Source



Project:	Computed.	Date
Subject:	Checked.	Date:
Task:	Page: ③	of
Job #:	No:	

The source to load plot tells about pressure and flow. It is marked with atomizing (small droplets) bonds which require low flow but high pressure.

Dispensing bonds are also marked for dispensing large amounts of fluid which require high flow and low pressure.

The load to source plot tells how it will feel when operating the sprayer.

From the previous mathematical model,

$$F_x = F_x^* - Bx$$

$$\Rightarrow \dot{a}(PA_p) = F_x^* - B\left(\frac{y}{\dot{a}}\right) - \dot{a}k(y+y_0)$$

$$\Rightarrow \dot{a}PA_p = F_x^* - \frac{B}{\dot{a}}\left(\frac{Q}{A_p}\right) - \dot{a}k(y+y_0)$$

Note, I have used  $Q$  and  $\dot{Q}$  interchangeably they represent "flow through nozzle"

$$\Rightarrow P = \frac{F_x^* - \dot{a}k(y+y_0)}{\dot{a}A_p} \frac{B}{(\dot{a}A_p)^2} Q$$

(4)

$P^*$  is max pump pressure possible and output impedance of Pump ( $R_o$ ), then:

$$P = P^* - R_o Q \quad \text{where,}$$

$$P^* = \frac{F_x^* - \dot{a}k(y + y_0)}{\dot{a} A_p}$$

$$\text{and } R_o = \frac{B}{(\dot{a} A_p)^2}$$

I will consider better ergonomics which allows pump operators to stroke their fingers by about  $2''$ .

Picking  $a = 0.3''$  and  $b = 1.2''$ . Then,

$$\text{a}' = 0.7142$$

$$\theta = \sin^{-1} \left( \frac{8x/2}{a+b} \right) = \sin^{-1} \left( \frac{2x}{2.2} \right)$$

$$\boxed{\theta = \sin^{-1} \left( \frac{2x}{4.2} \right) = 13.768^\circ}$$

Project:	Computed:	Date
Subject:	Checked:	Date
Task:	Page:	of
Job #:	No:	(5)

$$\delta z = a(1 - \cos \theta) = 3(1 - \cos 13.768) = 0.086 \text{ in}$$

$$\phi = \tan^{-1} \left( \frac{\delta z}{l_p} \right) = \tan^{-1} \left( \frac{0.086}{2} \right) = 2.46^\circ$$

taking max piston length as 2".

$\phi$  is much less than max allowable of  $15^\circ$ .

Thus, the pump stroke will be

$$S_p = \dot{a} \dot{x} = 0.7142 \times 2 = 1.428 \text{ inches}$$

Taking pressure during spraying at 15 psi and max pressure at 30 psi,

$$30 = P^* = \frac{F_x^* - k \dot{a}(y + y_0)}{\dot{a} A_p} = \frac{15 - k(0.7142)(3 \times 1.428)}{0.7142 \times A_p}$$

$$\Rightarrow \cancel{P^*} \frac{15 - k 3.06}{0.7142 A_p} = 30$$

$$\Rightarrow \frac{21}{A_p} - 4.2852 \frac{k}{A_p} = 30$$

assuming  $C_d = 0.95$ ,

$$\textcircled{D}_N = \sqrt{\frac{4}{\pi} C_d A_n} = 0.1527 \text{ in}$$

⑥

$$- P_s = \frac{k \beta S_p}{A_p} = \frac{k (3 \times 1.428)}{A_p} = 2$$

$$\text{Thus, } \frac{k}{A_p} = 0.466$$

Combining above equations,

$$\frac{21}{A_p} - 2 = 30$$

$$\Rightarrow A_p = \frac{21}{32} = 0.65625 \text{ in}^2$$

$$\boxed{\text{Piston diameter } (D_p) = \sqrt{\frac{4}{\pi} A_p} = 0.914 \text{ in}}$$

$$\Rightarrow k = 0.3058 \text{ } \frac{\text{lb/in}}{\text{in}}$$

$$\boxed{R_o = \frac{B}{(a' A_p)^2} = \frac{0.3}{(0.7142 \times 0.65625)^2} = 1.37}$$

$$\Rightarrow C_d A_n = \frac{62.4 \times \frac{1}{386.2} \times \frac{1}{1728} \times 30}{2 \times 1.37} = 0.0193 \text{ in}^2$$

Nozzle dia at TOP

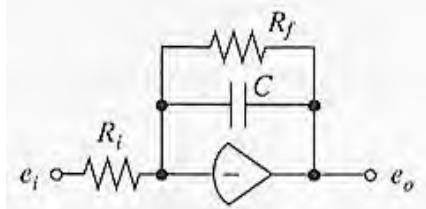
## Design a low-pass filter

It is desired to design an op-amp low-pass filter to reduce the noise in a signal above 1 KHz (6283 rad/s). Note that resistors come in relatively large steps of values and capacitors are even more coarse. The 1 kHz is an approximate number, you may design something that is close to that value without being exact. Go to Mouser.com and select  $\frac{1}{4}$  watt through-hole resistors and through-hole capacitors that will make the filter have a gain of 1.0 and an appropriate filter frequency.

The transfer function for this circuit is

$$= -\frac{1}{+1}$$

Where the break frequency is  $\omega_l = 1/\text{      } \text{(rad/s)}$ . Note that a megaohm times a microfarad has units of seconds. Use resistors in the range of  $>1 \text{ k}\Omega$  to  $<100 \text{ k}\Omega$ .



**HR**A TUL SHROTRIYA

UTA ID - 1001812437

Project:	Computed:	Date
Subject:	Checked:	Date:
Task:	Page: ①	of
Job #:	No:	

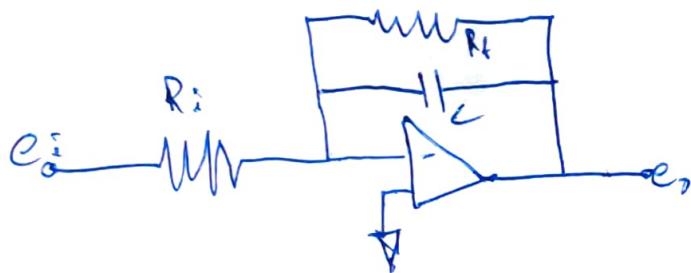
Low Pass FilterFilter above 1 kHz i.e.,  $6283 \text{ rad/s}$ 

$\text{Gain} = 1.0$

$$\frac{\text{Transfer Function } C_o}{C_i} = -\frac{R_f / R_i}{R_f CD + 1} = -\frac{R_f / R_i}{\frac{D}{\omega_1} + 1}$$

where  $\omega_1$  is the break frequency.

$\omega_1 = \frac{1}{R_f C} \text{ rad/s}$



$$\text{For Gain} = 1, \frac{C_o}{C_i} = 1 \Rightarrow -\frac{R_f}{R_i(R_f(D+1))}$$

$\Rightarrow R_i R_f D + R_i = -R_f$

$\Rightarrow R_i R_f D + R_i + R_f = 0$

This can be written as  $R_i R_f C_s + R_i + R_f = 0$ As  $C_s = 0$ , steady gain is achieved by:  

$$\boxed{R_i + R_f = 0}$$

(2)

Thus, for  $G = 1$ ,

$$R_i = -R_f$$

Now, the transfer function can be written as:

$$\frac{e_o}{e_i} = -\frac{R_f/(ER_f)}{R_f CD + 1}$$

$$\Rightarrow \boxed{\frac{e_o}{e_i} = \frac{1}{R_f CD + 1}}$$

$$\Rightarrow \boxed{\frac{e_o}{e_i} = \frac{1}{\frac{D}{\omega} + 1}}$$

$$\omega = 6283 = \frac{1}{R_f C} \text{ rad/s}$$

$$\Rightarrow R_f C = \frac{1}{6283} \text{ seconds} \approx 1.5 \times 10^{-9} \text{ seconds}$$

or 16 microseconds

---

Usually, Capacitors ~~not~~ have a value in microFarads or  $\mu F$  whereas the resistor ranges

is given in kilo ohms or k $\Omega$ .

Therefore, as long as the product of both is 16 milliseconds, any pair from below can be selected; (They can also be fine tuned if required)

- ①  $R_f = 16 \text{ k}\Omega ; C = 1 \mu\text{F}$
- ②  $R_f = 12 \text{ k}\Omega ; C = 1.33 \mu\text{F}$
- ③  $R_f = 10.6 \text{ k}\Omega ; C = 1.5 \mu\text{F}$
- ④  $R_f = 8 \text{ k}\Omega ; C = 2 \mu\text{F}$
- ⑤  $R_f = 4 \text{ k}\Omega ; C = 4 \mu\text{F}$
- ⑥  $R_f = 2 \text{ k}\Omega ; C = 8 \mu\text{F}$

Some of the possible values for  $R_f$  and  $C$  to get the desired Low Pass Filter.

The list is exhaustive and any  $C$  values within given limits can be selected as long as their product is approximately 16 milliseconds.

For fine tuning, 15915.9 microseconds can be taken as the reference value and higher decimal values of  $R_f$  and  $C$  can be multiplied to reach near the reference value.

# Servo-valve Review

The task is to search for a manufacturer who sells a servo-valve with rated pressure of 3000psi and a maximum flow of 0.5 gallons/minute

This valve is very close to the servo-valves used in Formula 1 racing. It tries to reduce the maximum flow rate in an effort to probably reduce the overall system weight. While the effort is commendable, such a valve is not mass produced as of yet.

**Moog** is the manufacturer which produces servo-valves for Formula 1 applications. Their valves that come close to this category belong to the E024 series also known as the **24 Series Servo Valves**:

- E024 Series Miniature Servovalve
- E024 - 800 Series Miniature Servovalve with LVDT
- E024 Valve with Low Flow Pilot Stage (2 versions)

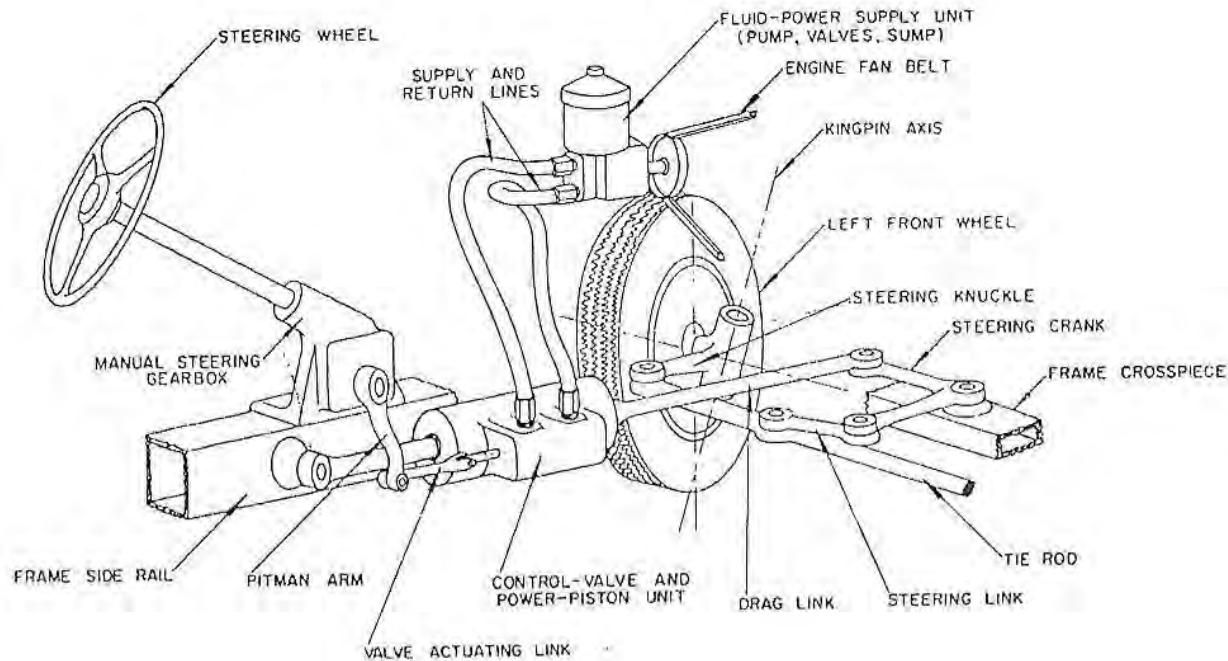
To achieve the maximum flow rate of 0.5 gallons/minute, these valves can be paired with a limiter also sold by Moog.

Other than that, if the valves are not limited by weight then another good alternative is the **30 Series Servo Valve (E030)** sold by Moog. These are usually meant for aerospace systems. However, **E-030-1001 fits all the criteria mentioned above.** It can also be used in conjunction with E24 and E32 series to achieve the desired controls.

It should be noted that these valves are susceptible to electromagnetic interference and thus should be shielded significantly. Moog provides various supporting accessories under its Motorsport catalog.

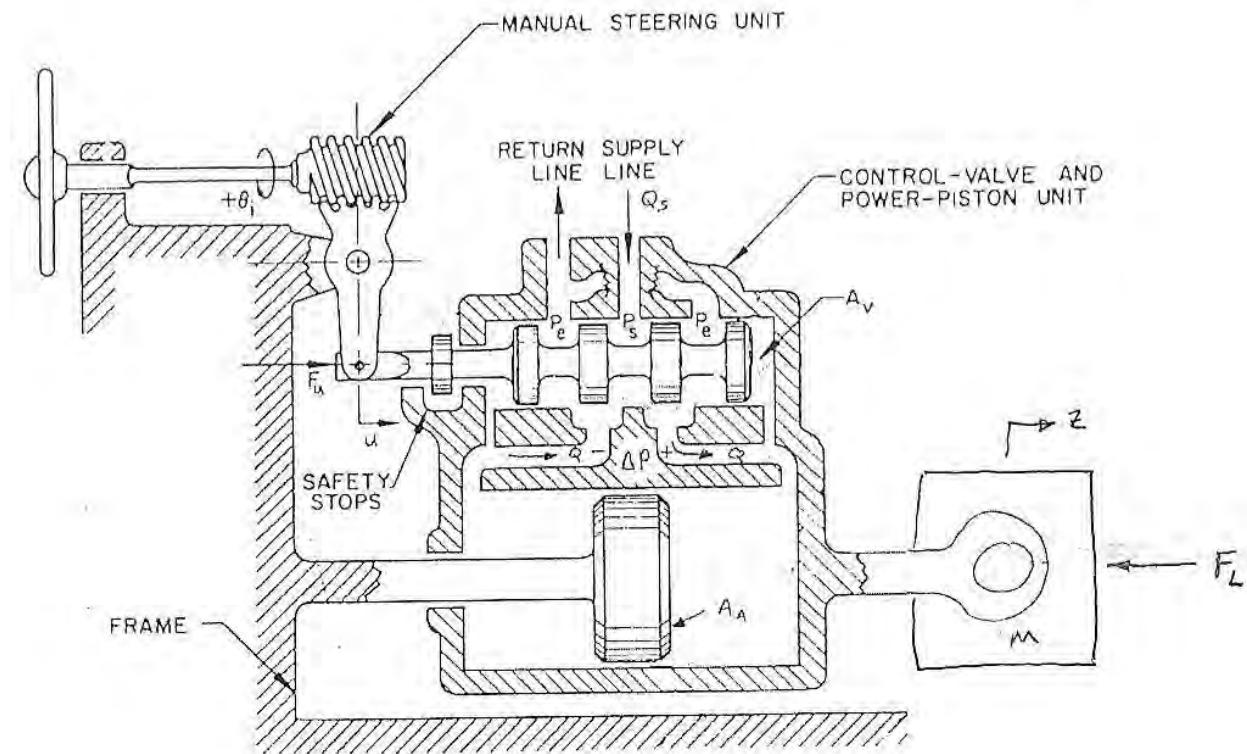
## Power Steering

Shown below is the overall configuration of a power steering system for an automobile.



Whereas it is more common to mount a hydraulic cylinder and have the piston and rod actuate the load. Notice that the piston and rod are connected to the frame and the cylinder moves to actuate the steering. Notice further that the servovalve is mounted on the cylinder so the body or ports of the servovalve move with the cylinder independently of the actual spool. This makes this a "follow-up" type of servo. An input to the spool valve will make the cylinder and hence the valve body move until the valve is centered once again.

Shown below are the internal details of the servovalve and cylinder we want to analyze.



When the spool is moved relative to the frame, the pressure and flow that acts on the ends of the cylinder to move the whole cylinder. Notice that the area of the cylinder ends is the same as the area of the piston actuator,  $A_a$ , and we can neglect the area of the rod for simplicity. The same can be said for the spool area,  $A_v$ .

Since the spool valve body is part of the cylinder housing the ports of the spool valve move while the spool remains in the commanded position. As the cylinder moves, the difference in the spool position and the ports reduce until the spool is centered and then motion stops. Hence the term "follow-up" since the cylinder housing moves the same amount as the spool input command. This will be a "unity gain" position servo.

The purpose of this power steering system is to power the position of the wheels so the driver has a reduced force from what would be in a manual steering system. In addition to this being a position servo, we are interested in how much force amplification we will get, and just as important, how much "road feel" we can get. In other words, if we have a disturbance force,  $F_L$ , what is the force transmitted back to the driver through  $F_u$ .

Notice that the output pressure of the valve is directed to the piston, but it is also fed back to the ends of the spool. This will create the force feedback or "road feel" when a disturbance force is experienced.

Notice the "Safety Stops" on the spool rod input. If there is no hydraulic pressure, the cylinder will not move by hydraulic pressure. Instead, the spool rod will hit the safety stop and the driver will be moving the wheels with no assist. Thus this will be just a mechanical steering system.

## Problem Statement

Analyze the performance of this system by writing a linearized equation for the gain and impedance of the servovalve. Notice that the net valve position will be  $u - z$ . Write the continuity equation for flow into the cylinder neglecting compressibility of the fluid. Write a force balance on the cylinder including the pressure force, the mass, and the disturbance force, neglecting Coulomb friction.

Derive a transfer function for the position of the cylinder,  $z$ , as a function of the input,  $u$ , and the disturbance force,  $F_L$ . State the performance factors for static gain, static stiffness, natural frequency, and damping ratio.

Now write a force balance on the spool of the servovalve including the force input,  $F_u$ . You will get a transfer function for  $F_u$  as a function of  $z$  and  $F_L$ . Substitute your previous transfer function for  $z$  into the transfer function for  $F_u$ . Considering no input,  $u$ , and steady-state conditions, derive the gain for  $F_u / F_L$ . It should be some function of the areas of the valve and piston actuator.

To test this force gain, park your car and feel the force or torque required on the steering wheel with the engine on and with the engine off. With the engine off, you will be turning the wheels manually because of the safety stops. What do you think the ratio of torques is? It might be better to use the inverse of the gain to state that with the engine off, it takes 5, 10, or 20 (or whatever you measure or estimate) times the torque to turn the wheel. Does that make sense with your derived gain?

①

VALVE

$$\Delta P = G_p(u-z) - RQ$$

FORCES ON VALVE BODY

$$A_A \Delta P + A_v \Delta P - K_v(z-u) - m\ddot{z} - F_L - (\pm f_c) = 0$$

$$m\ddot{z} + K_v(z) = K_v(u) + (A_A + A_v)\Delta P - F_L - (\pm f_c)$$

FORCES ON SPOOL

$$F_u - K_v(u-z) - A_v \Delta P = 0$$

$$K_v z = K_v u - F_u + A_v \Delta P$$

CONTINUITY

$$Q = \dot{V}_A + \dot{V}_v$$

where,

$$\dot{V}_A = A_A \dot{z}$$

$$\text{and } \dot{V}_v = A_v(\dot{z} - \dot{u})$$

$$Q = (A_A + A_v)\dot{z} - A_v\dot{u}$$

STARTING ANALYSIS

$$m\ddot{z} + K_v z = K_v u + (A_A + A_v)\Delta P - F_L - (\pm f_c)$$

$$m\ddot{z} + K_v z = K_v u + (A_A + A_v)[G_p(u-z) - R\{(A_A + A_v)\dot{z} - A_v\dot{u}\}] - F_L - (\pm f_c)$$

$$m\ddot{z} + R(A_A + A_v)^2 \dot{z} + [(A_A + A_v)G_p + K_v]z = K_v u + (A_A + A_v)G_p u + R A_v (A_A + A_v) \dot{u} - F_L - (\pm f_c)$$

405

$$m\ddot{z} + R(A_A + A_v)^2 \dot{z} + [(A_A + A_v)k_p + k_v]z = [k_v + (A_A + A_v)k_p]u + RA_v(A_A + A_v)\dot{u} - F_L - (\pm f_c)$$

(2)

$$Z = \frac{[k_v + (A_A + A_v)k_p]u + RA_v(A_A + A_v)\dot{u} - F_L - (\pm f_c)}{mD^2 + R(A_A + A_v)^2 D + [(A_A + A_v)k_p + k_v]}$$

$$Z = \frac{\left[ \frac{RA_v(A_A + A_v)D}{k_v + (A_A + A_v)k_p} + 1 \right]u - \frac{F_L - (\pm f_c)}{k_v + (A_A + A_v)k_p}}{\frac{mD^2}{k_v + (A_A + A_v)k_p} + \frac{R(A_A + A_v)^2 D}{k_v + (A_A + A_v)k_p} + 1}$$

$$\text{Static Gain } (G_s) = 1$$

$$\text{Static stiffness } (k_s) = k_v + k_p(A_A + A_v)$$

where  $k_v$  is very small.

$$\text{frequency } (\omega) = \sqrt{\frac{k_s}{m}}$$

$$\text{time constant } (\tau) = \frac{1}{\omega} = \sqrt{\frac{m}{k_s}}$$

### FORCE

$$F_u = k_v(u - z) + A_v \Delta P$$

$$F_u = k_v(u - z) + A_v [k_p(u - z) - R \{ (A_A - A_v)A_v \dot{z} - A_v \dot{u} \}]$$

$$F_u = [k_v + A_v k_p + RA_v^2 D]u + [-k_v - A_v k_p - R(A_A + A_v)D]z$$

$$F_u = [RA_v^2 D + (k_v + A_v g_p)] u - \frac{R(A_A + A_v)A_v D + (k_v + A_v g_p)}{z}$$

Neglecting  $k_v$  due to its lower value and substituting  $z$  derived previously.

$$F_u = [RA_v^2 D + A_v g_p] u - [RA_v(A_A + A_v)D + A_v g_p] *$$

$$\left[ \frac{\left[ \frac{RA_v(A_A + A_v)D}{k_v + (A_A + A_v)g_p} + 1 \right] u - \frac{F_L - (f_c)}{k_v + (A_A + A_v)g_p}}{\frac{m D^2}{k_v + (A_A + A_v)g_p} + \frac{R(A_A + A_v)^2 D}{k_v + (A_A + A_v)g_p} + 1} \right]$$

To obtain these equations without friction as asked in the question, just remove  $f_c$  term.

No inflat & steady state,

$$F_u = \frac{-A_v g_p F_L}{k_v + (A_A + A_v)g_p}$$

$$\boxed{\frac{F_u}{F_L} = \frac{-A_v g_p}{k_v + (A_A + A_v)g_p}}$$

# Object Recognition using Python and OpenCV

This is an independent project in progress made with the help of LinkedIn Courses on Python and OpenCV. Currently, over 200 hours of work has been put into the project and it is estimated that another 150 hours of work may be required to see it to completion. Below is a summary of the tasks that have been achieved from starting to present.

The entire project has been carried out using resources provided by the University of Texas at Arlington. These resources were obtained by Atul Shrotriya from the university as a graduate student in mechanical engineering.

The code and associated files for the project can be found by visiting my GitHub repository at <https://github.com/atulshrotriya/opencv>. The updates for the project will also be posted on this link.

## Completed tasks in increasing order:

- Red detection through 2 different methods (BGR then HSV)
- Canny Edges
- Band approximation
- Object detection based on centers
- Blur and Dilation for better contouring and removing/reducing disturbances
- Red detection through 2 different methods (BGR then HSV)
- Canny Edges
- Band approximation
- Object detection based on centers
- Blur and Dilation for better contouring and removing/reducing disturbances
- Implemented command to check for Python environment in MATLAB
- Successfully interfaced with MATLAB to provide recognized objects as outputs for Webots
- Evaluated filtering techniques using DFT and PSD analysis

## Tasks currently in progress:

- Implementation of Machine Learning models using Keras
- Further evaluation of different filtering and estimation techniques

## **Objectives in the pipeline:**

- Access benefits of dual filtering in other colors (currently only BGR)
- Currently it gets confused if a large enough object is near the vicinity. Can only be fixed through training the model over a fixed object database (learning Keras for that)
- Depth mapping through Intel RealSense or other sensor
- Figuring out how to implement algorithm in real time (it's fast but not as fast as the advanced algorithms used by leading organizations) using video feed from the camera
- Implementing algorithm in real time under different lighting conditions and surfaces

## **References**

1. Martini J. (2020). *Learning Python* [Online Course]. Retrieved from <https://www.linkedin.com/learning/learning-python-2/>
2. Crawford P. W. (2017). *OpenCV for Python Developers* [Online Course]. Retrieved from <https://www.linkedin.com/learning/opencv-for-python-developers>

# **Cast Steel Gate Valve Design and Assembly using SolidWorks and Manufacturing Layout**

This was the final project of the course Manufacturing Processes and Systems (ME 5326). A 5-person team was randomly selected by the lecturer who had to design and assemble a part using SolidWorks. Then figure out manufacturing layout for mass manufacturing according to the industry demand.

Atul Shrotriya was responsible for the design and assembly of the Cast Steel Gate Valve using SolidWorks and was assisted by group member Sayyam Papalkar. Atul Shrotriya also provided assistance with the selection of manufacturing processes and spearheaded the final presentation with the help of other group members.

The SolidWorks part files and assembly can be found at the following link - <https://github.com/atulshrotriya/Caststeelgatevalve>  
Attached below is the final report submitted to the lecturer and teaching assistant for evaluation.

**Lecturer** – Dr. Sunand Santhanagopalan, Lecturer (Other Faculty),  
Department of Mechanical and Aerospace Engineering, The University of Texas at Arlington

# **PROJECT 02: PART MANUFACTURING**

## **Cast Steel Gate Valve**

### **PROJECT GROUP 4**

---

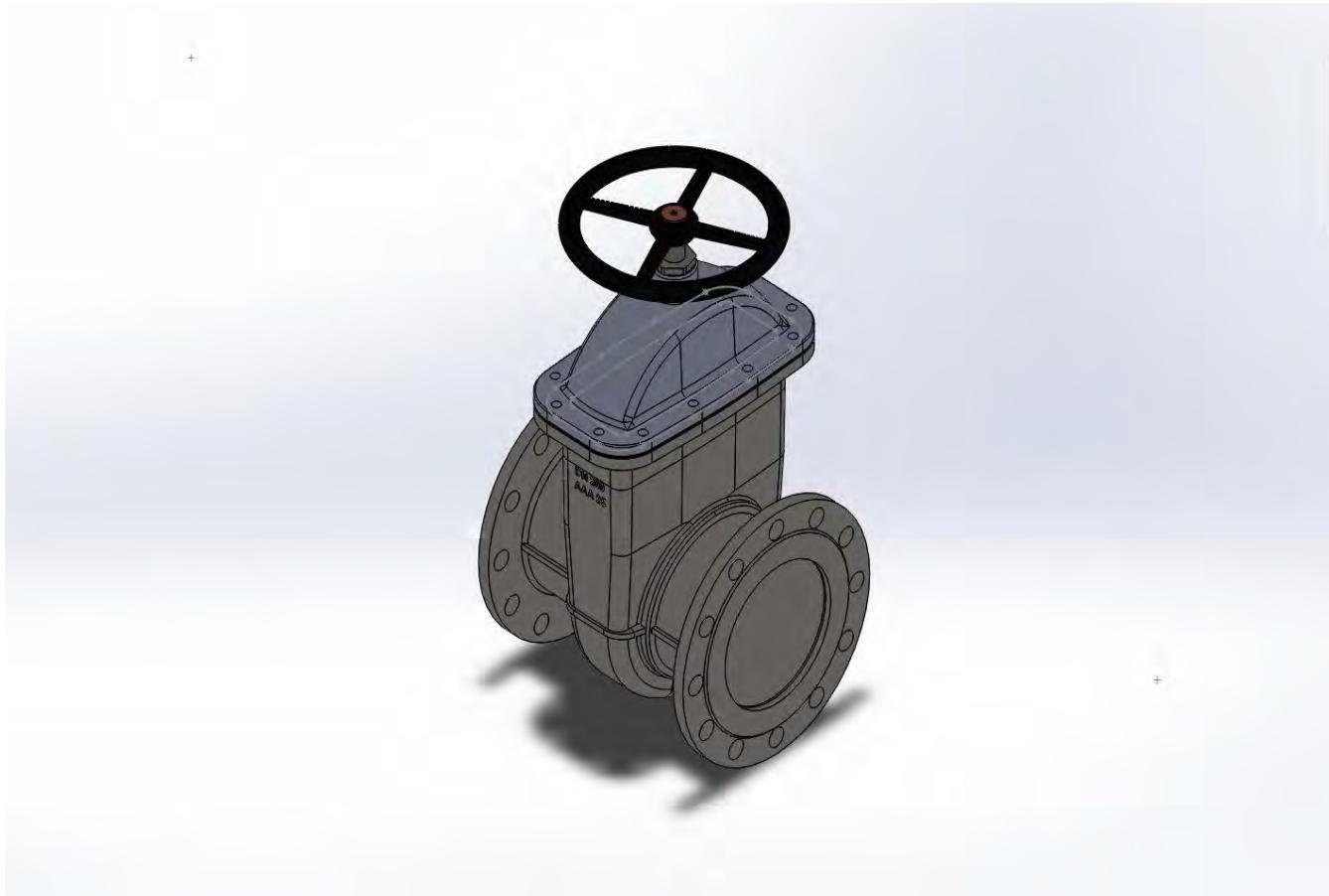
Atul Shrotriya

Baby Ramona Azad

Kari Naga Nikhil

Samarth Ramachandra

Sayyam Harshad Papalkar

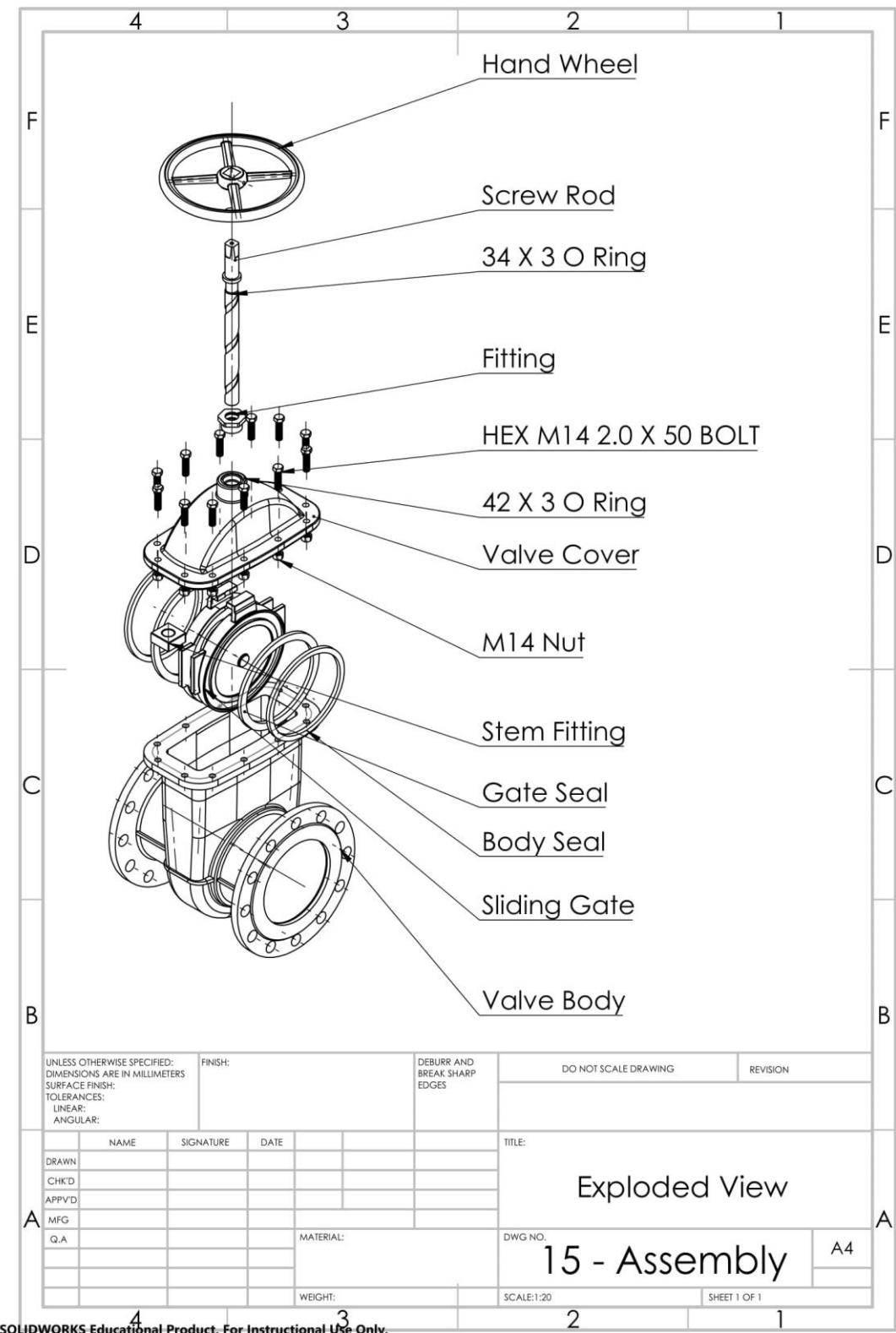


**Manufacturing Processes and systems ME 5326**

Professor: Sunand Santhanagopalan

TA: Himanth Kumar Talla

## **1. Exploded view of the Assembly**

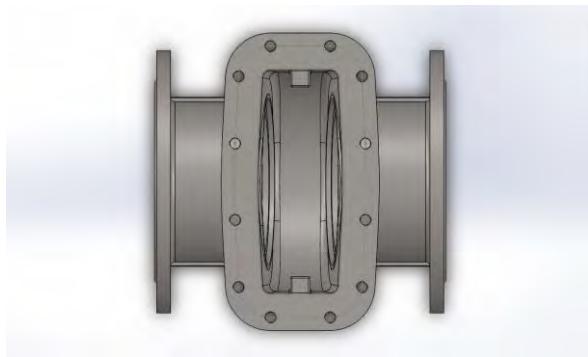


## 2. 3D drawings for each part

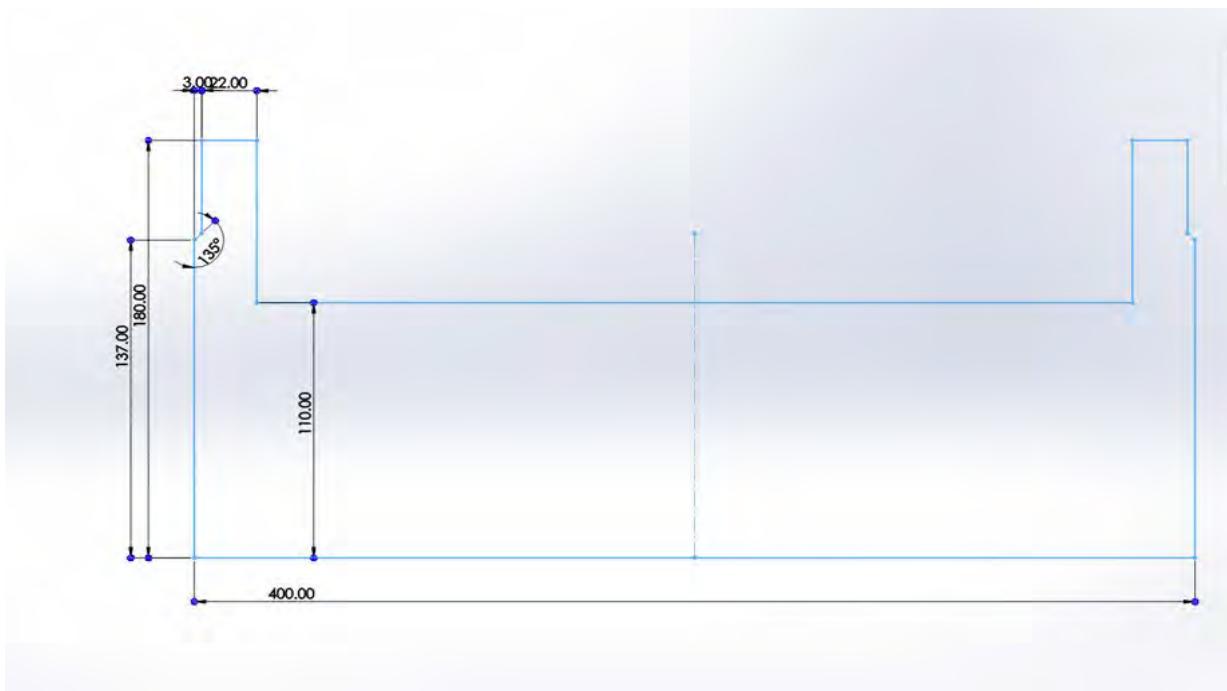
### 2.1 Valve Body



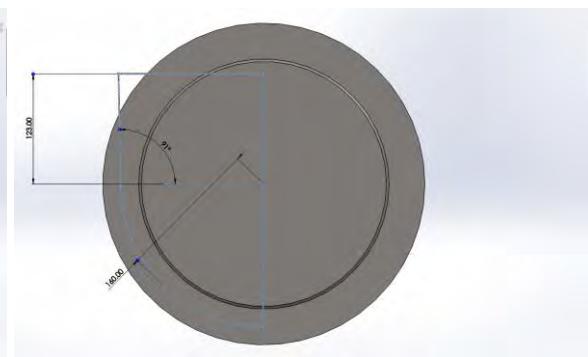
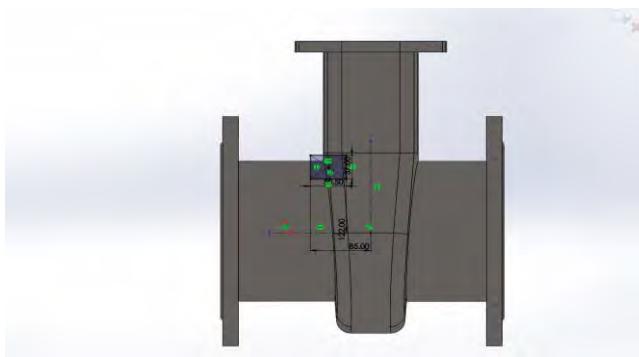
Front View



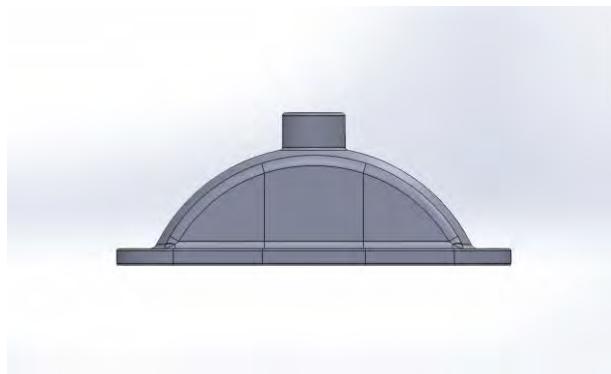
Top View



Dimensions related to the valve body



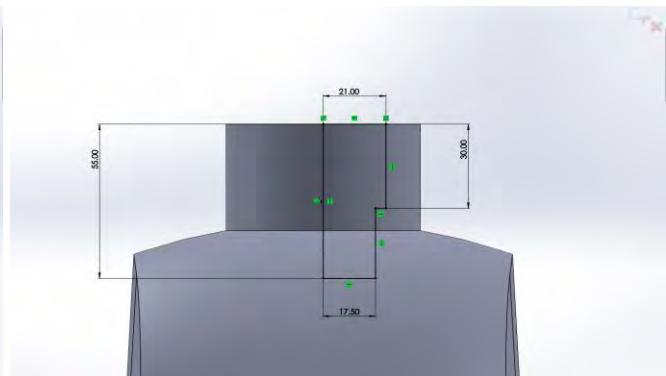
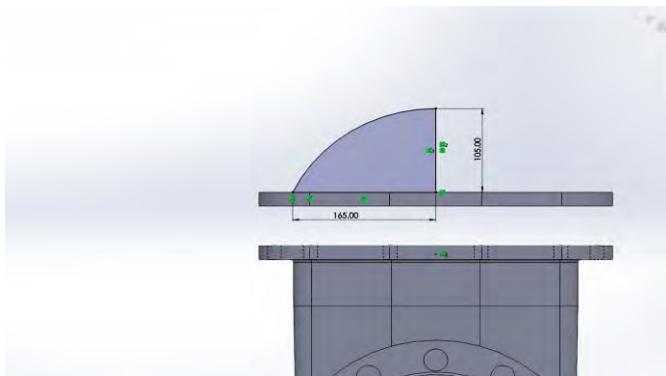
## 2.2 Valve Cover



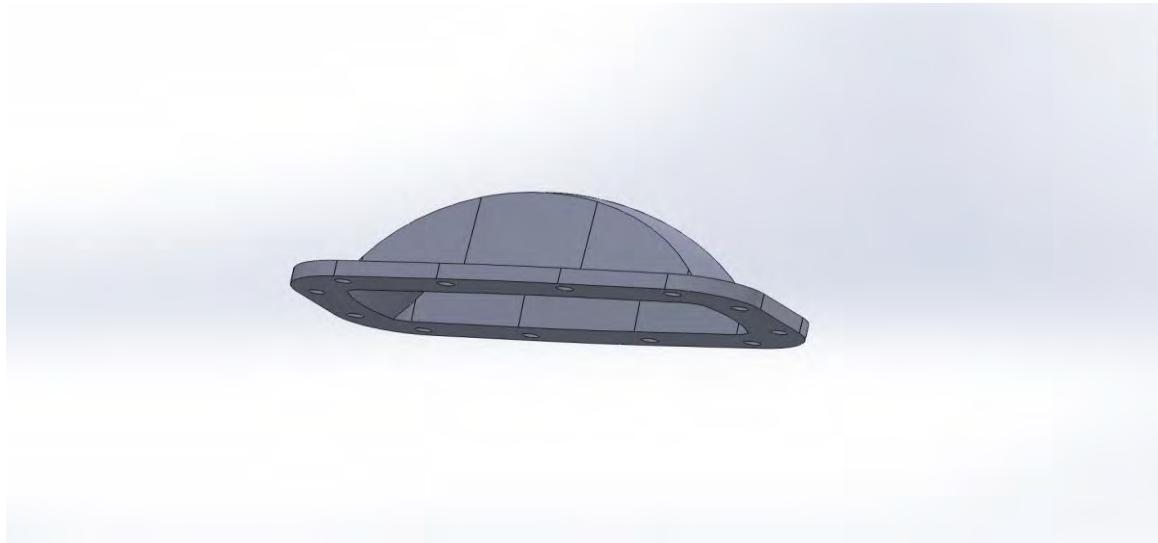
*Front View*



*Top View*

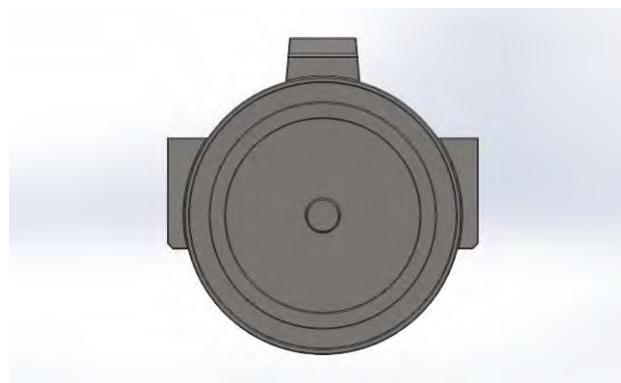


*Dimensions related to Valve Cover*

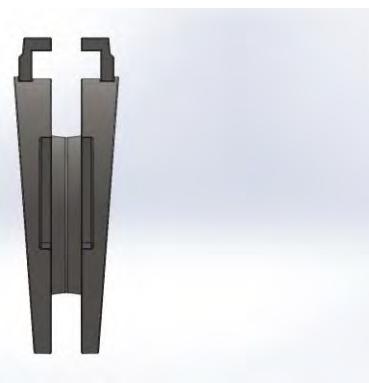


*Isometric view*

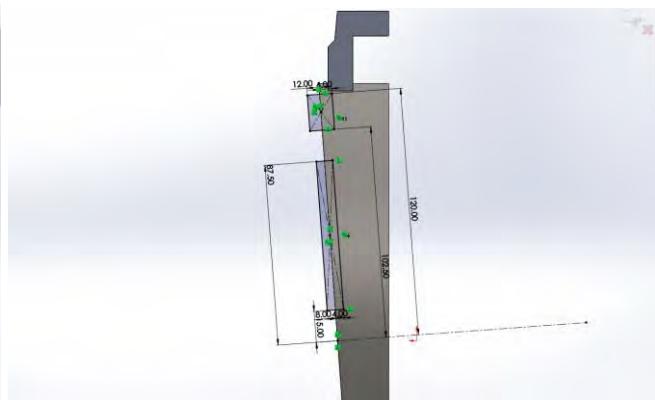
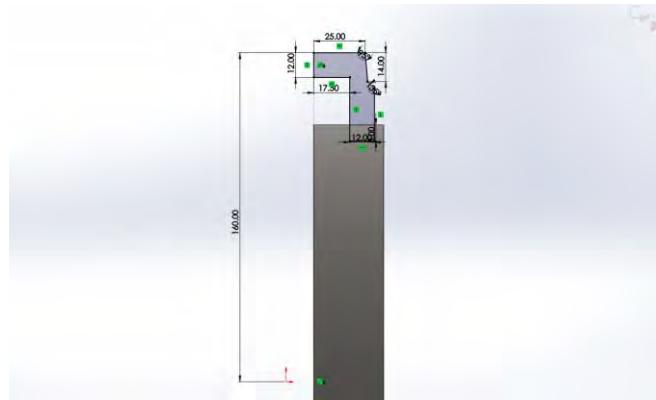
### 2.3 Sliding Gate



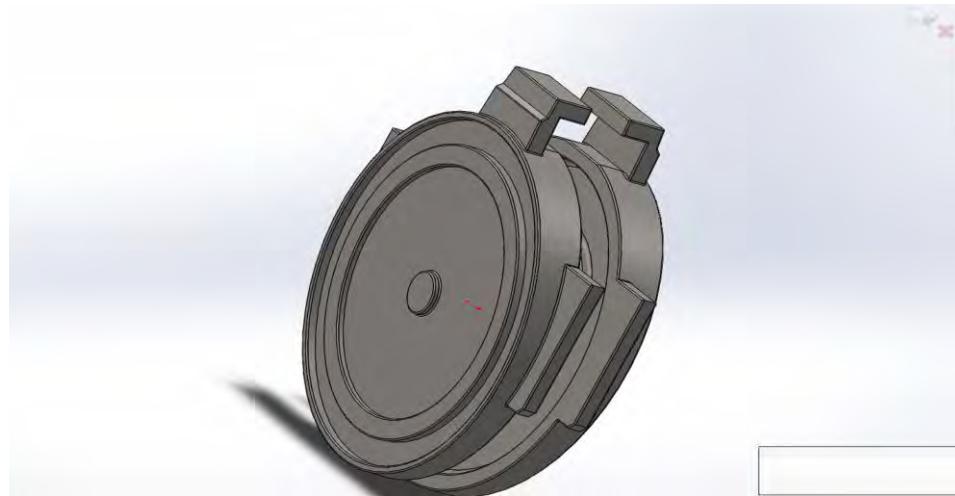
*Front View*



*Side View*



*Dimensions related to Sliding gate*

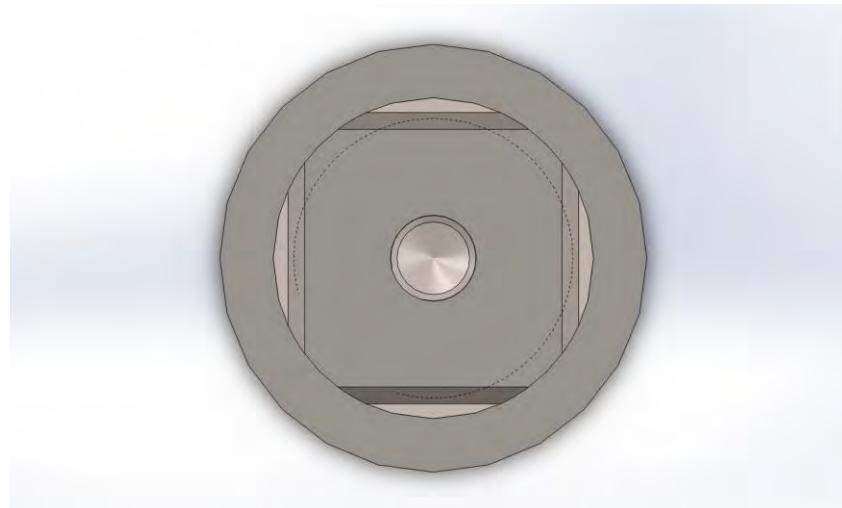


*Isometric View*

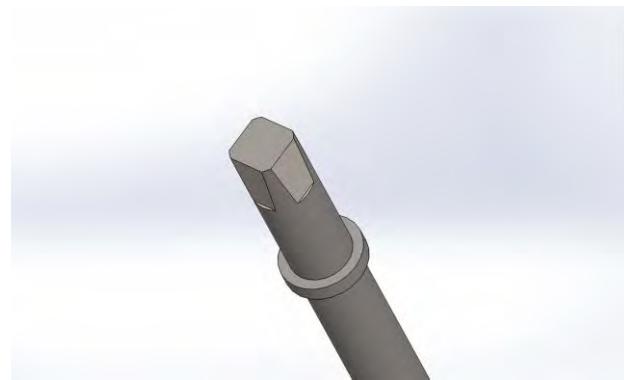
## 2.4 Screw Rod



*Front/ Side view*



*Top View*

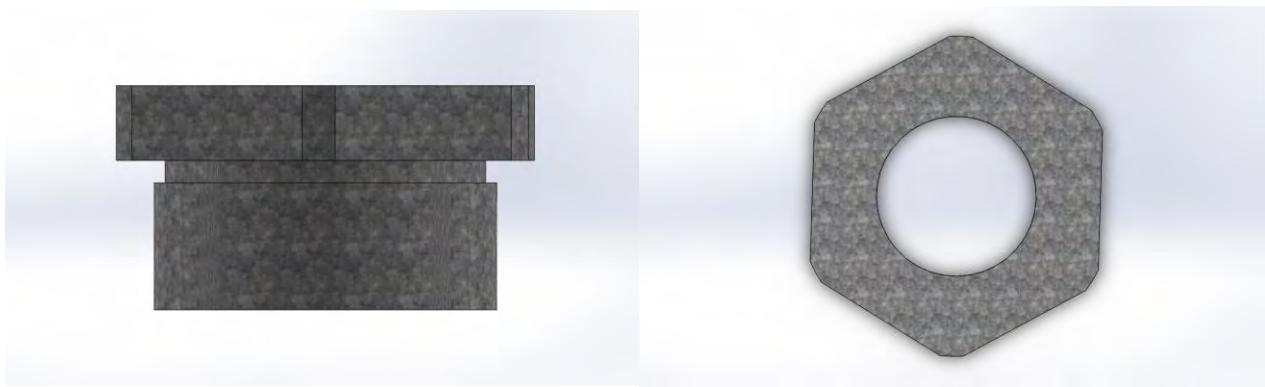


## 2.5 Packing Seal



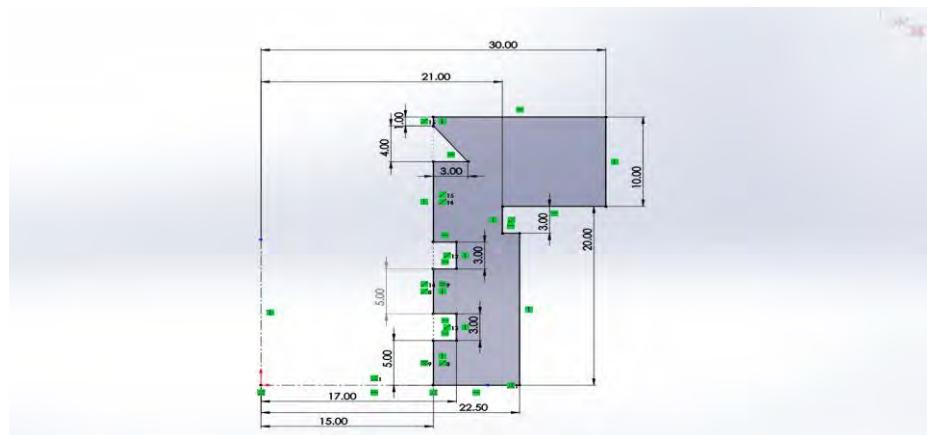
*Top View*

## 2.6 Fitting



*Front View*

*Top View*



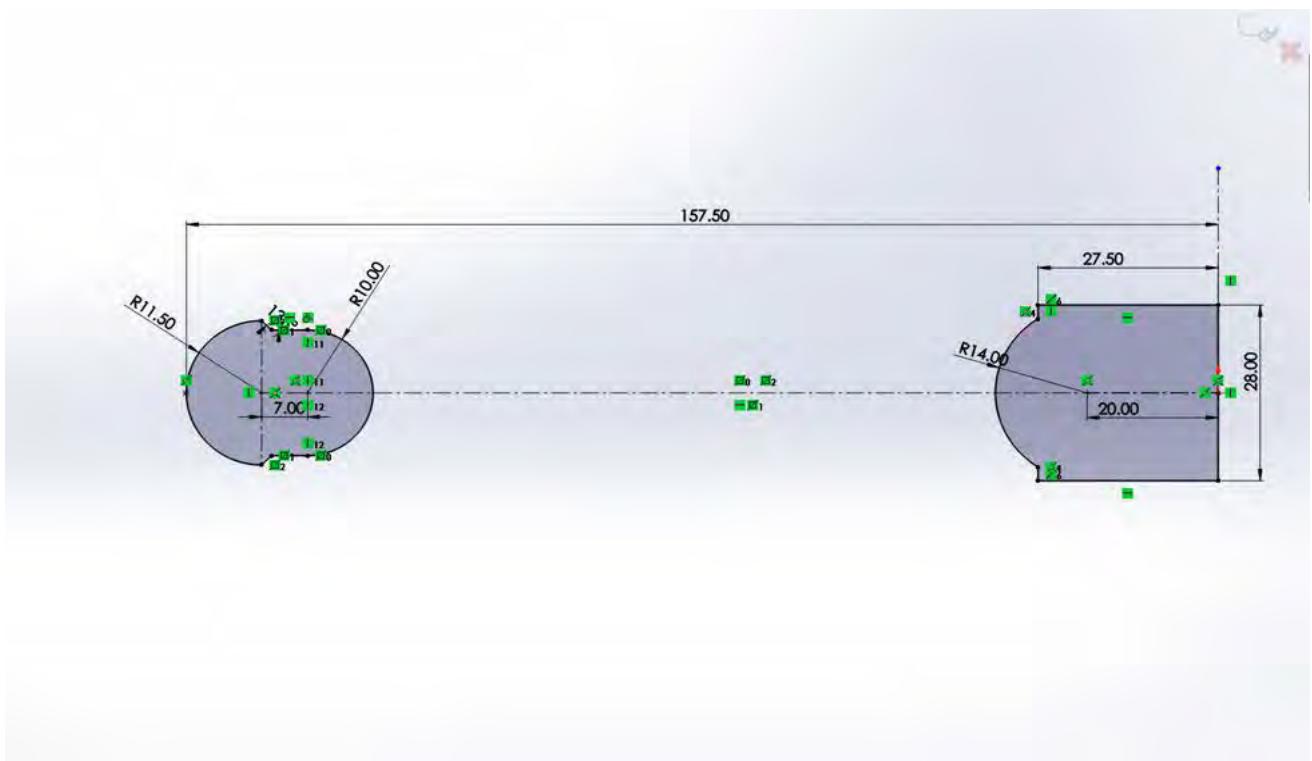
*Dimensioning*

## 2.7 Hand Wheel



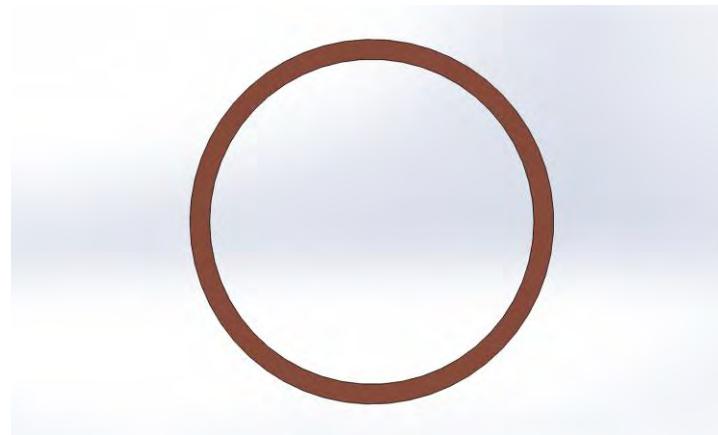
Front View

Top View



Dimensioning of Hand Wheel

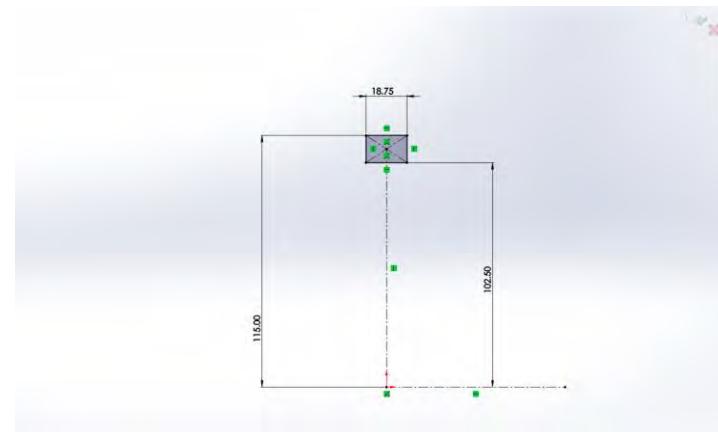
## 2.8 Body Seal



*Front View*



*Isometric View*

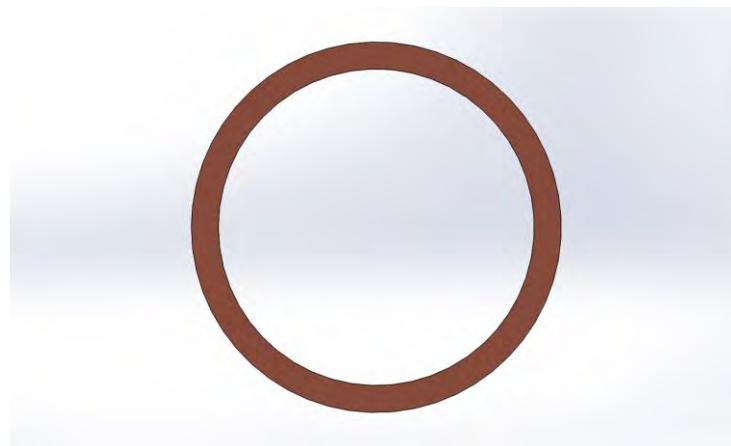


*Dimensioning*

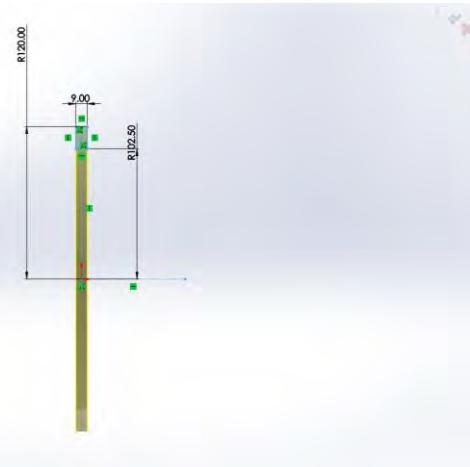
## 2.9 Gate Seal



*Isometric View*



*Front View*

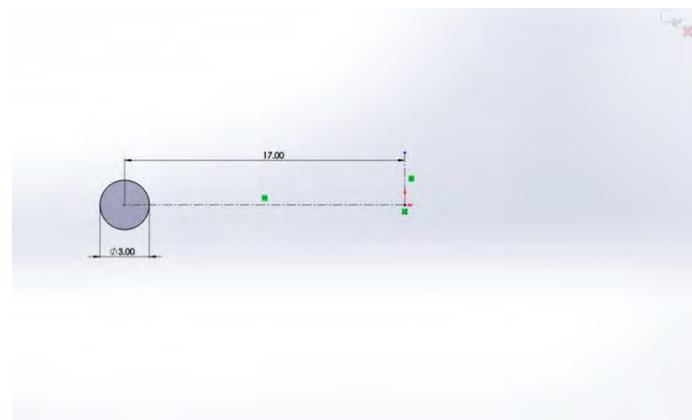


*Dimensioning*

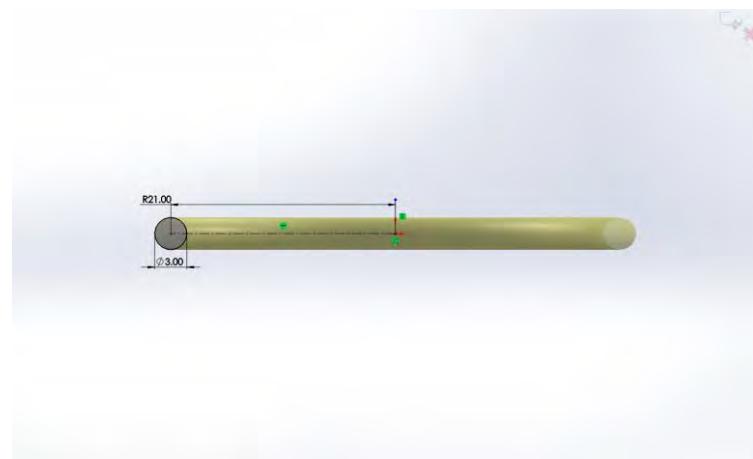
## 2.10 O – Ring



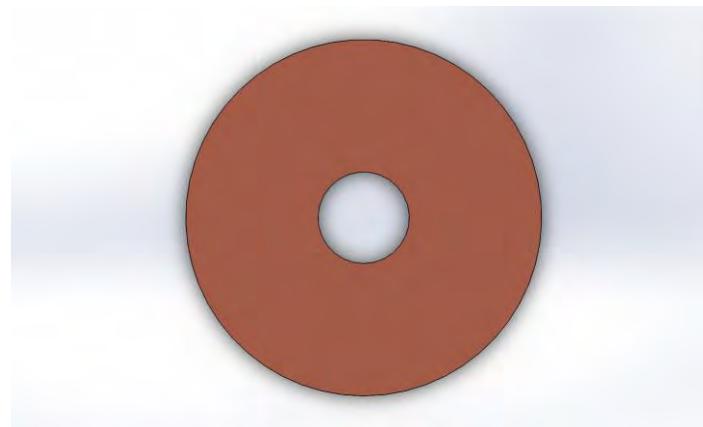
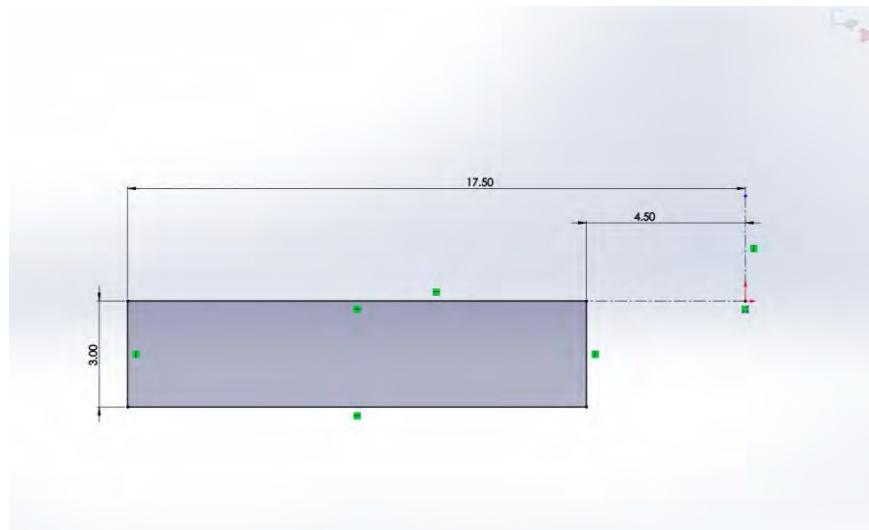
*Top View*

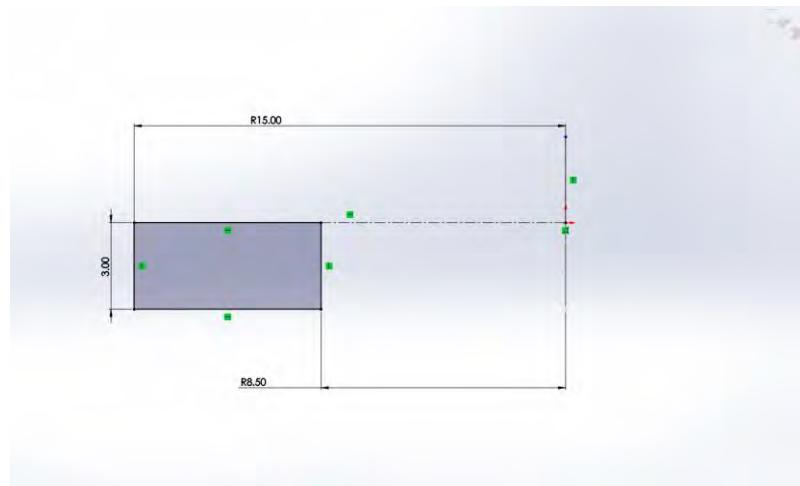


*Dimensioning*



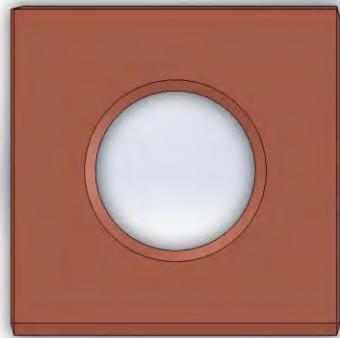
## 2.11 M8 and M16 Washer

*Top View**Isometric View**Dimensioning of M8 Washer*

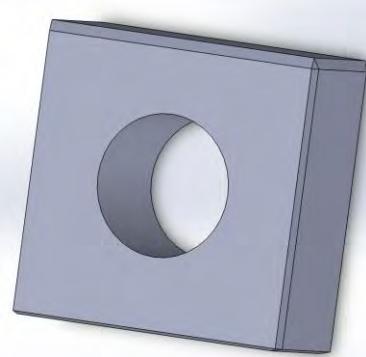


*Dimensioning of M16 Washer*

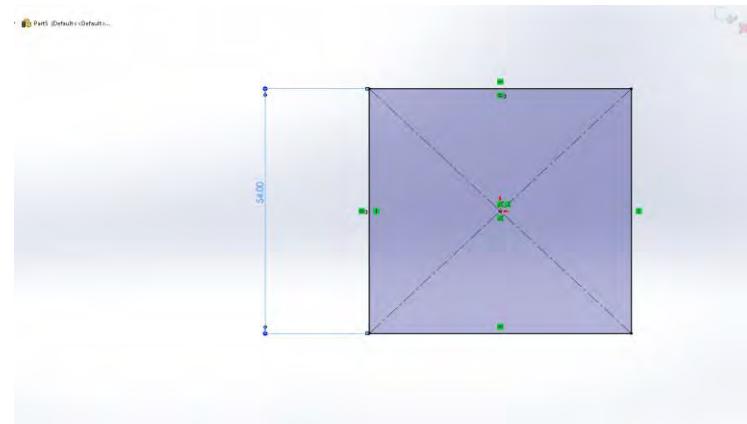
## 2.12 Stem Fitting



*Front View*



*Isometric View*

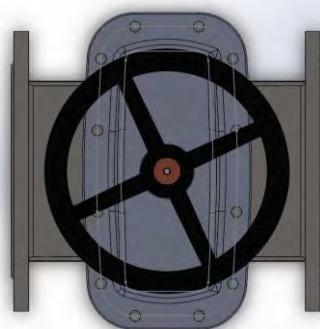


*Dimensioning*

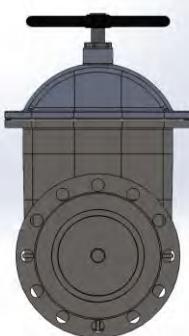
## 2.13 Assembly



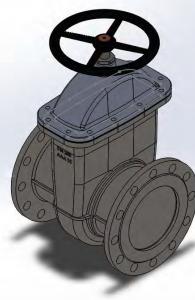
Front View



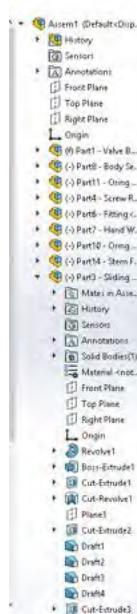
Top View



Side View

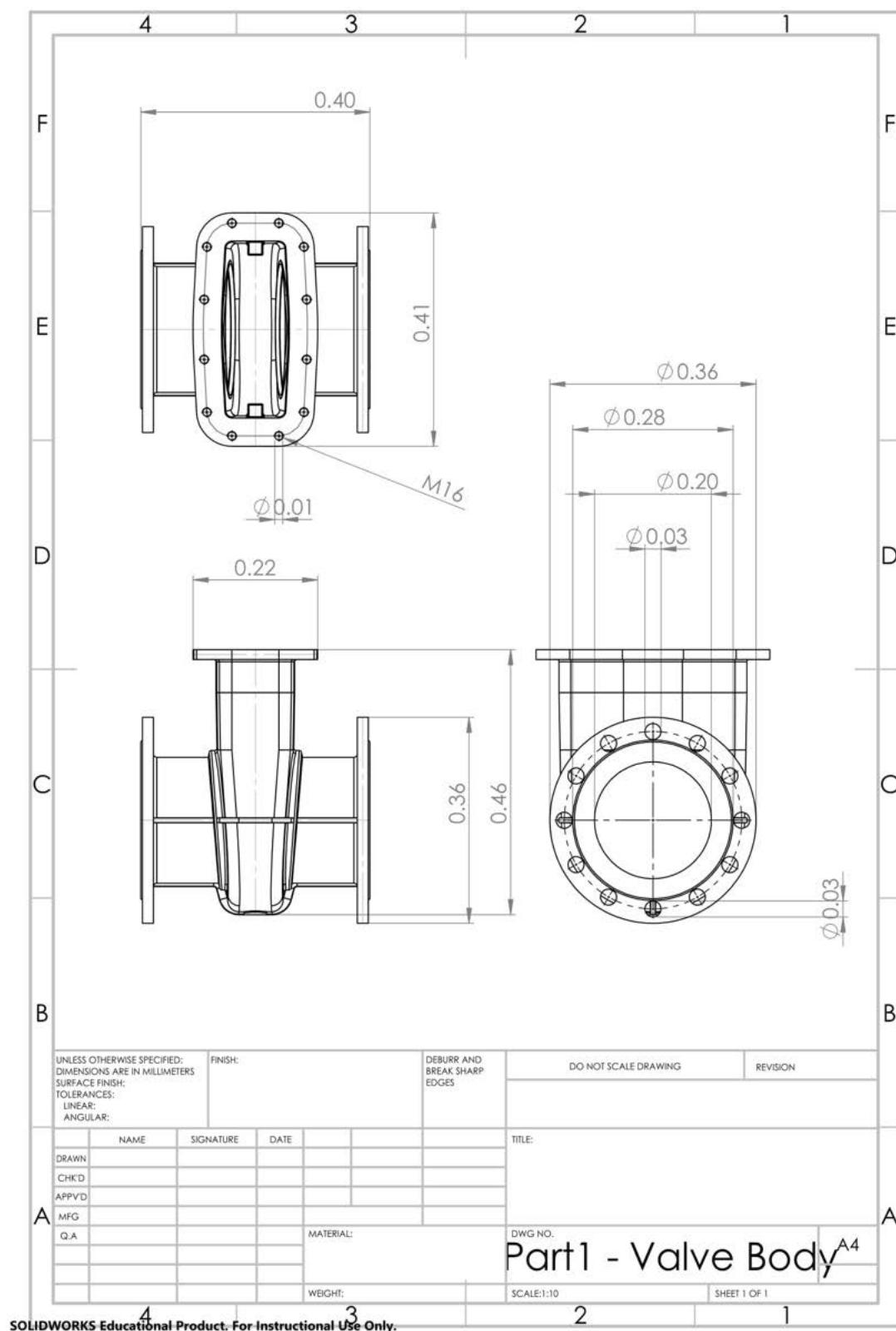


Isometric View

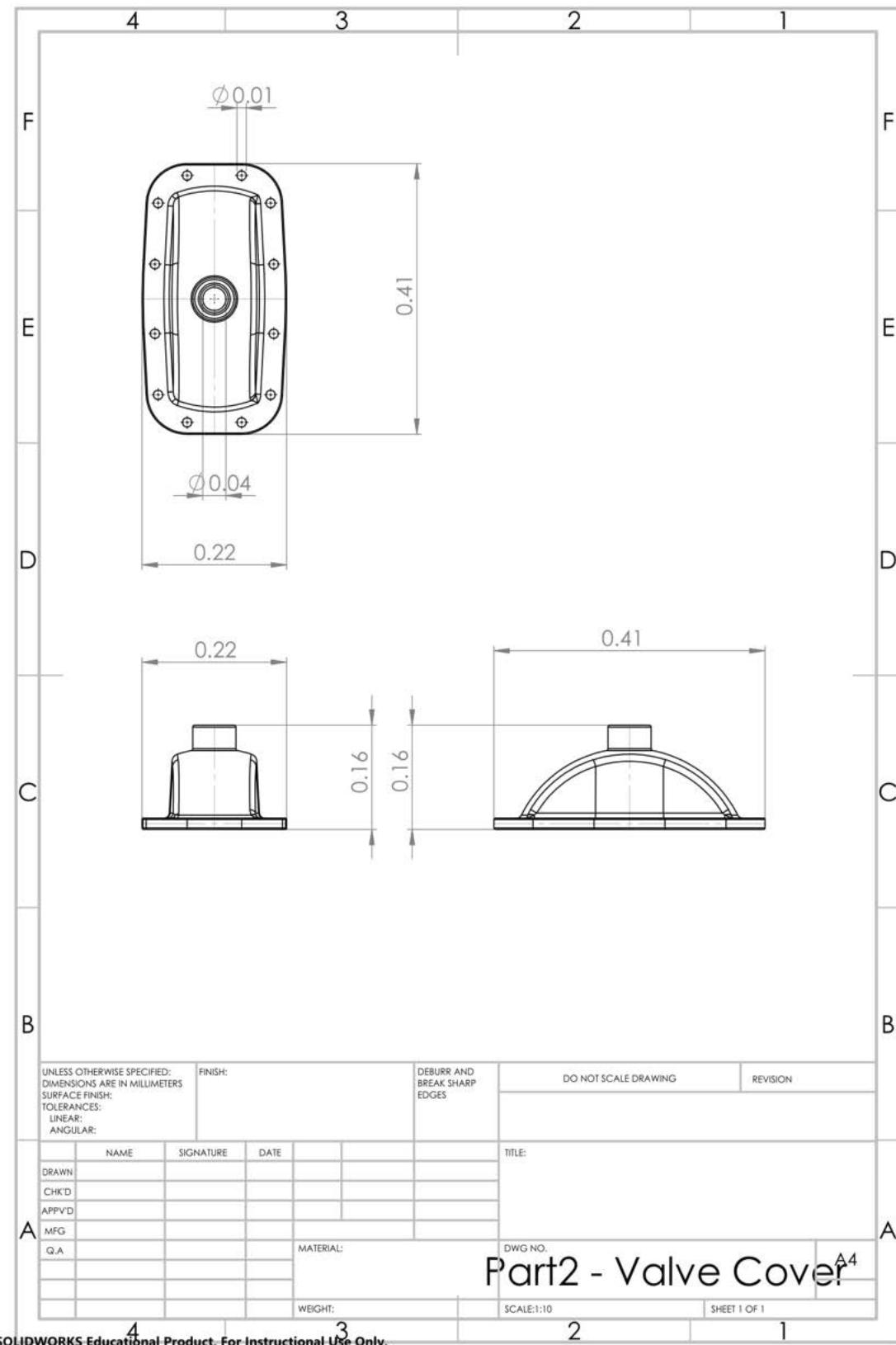


### 3. 2D drawings for each part

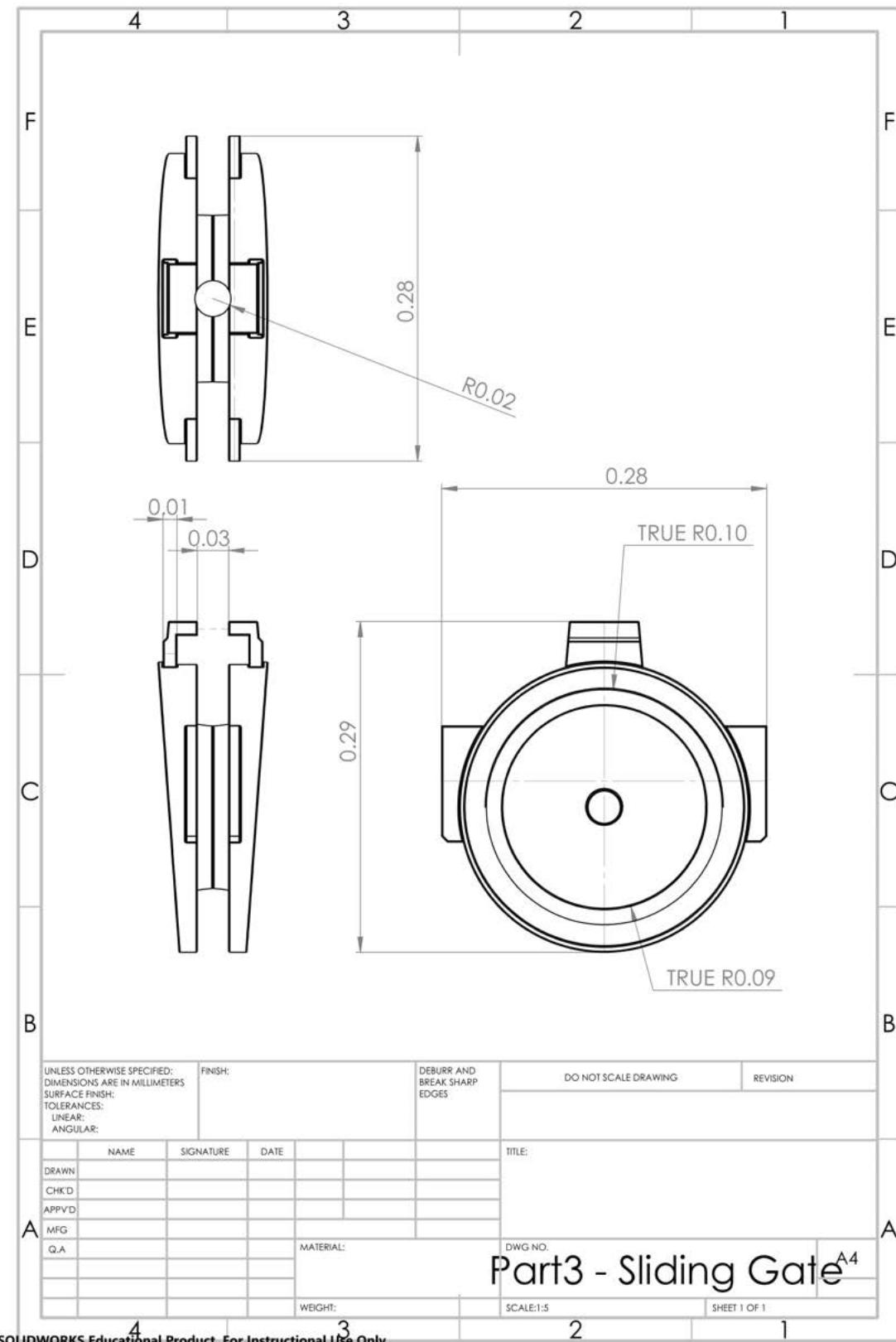
#### 3.1 Valve Body



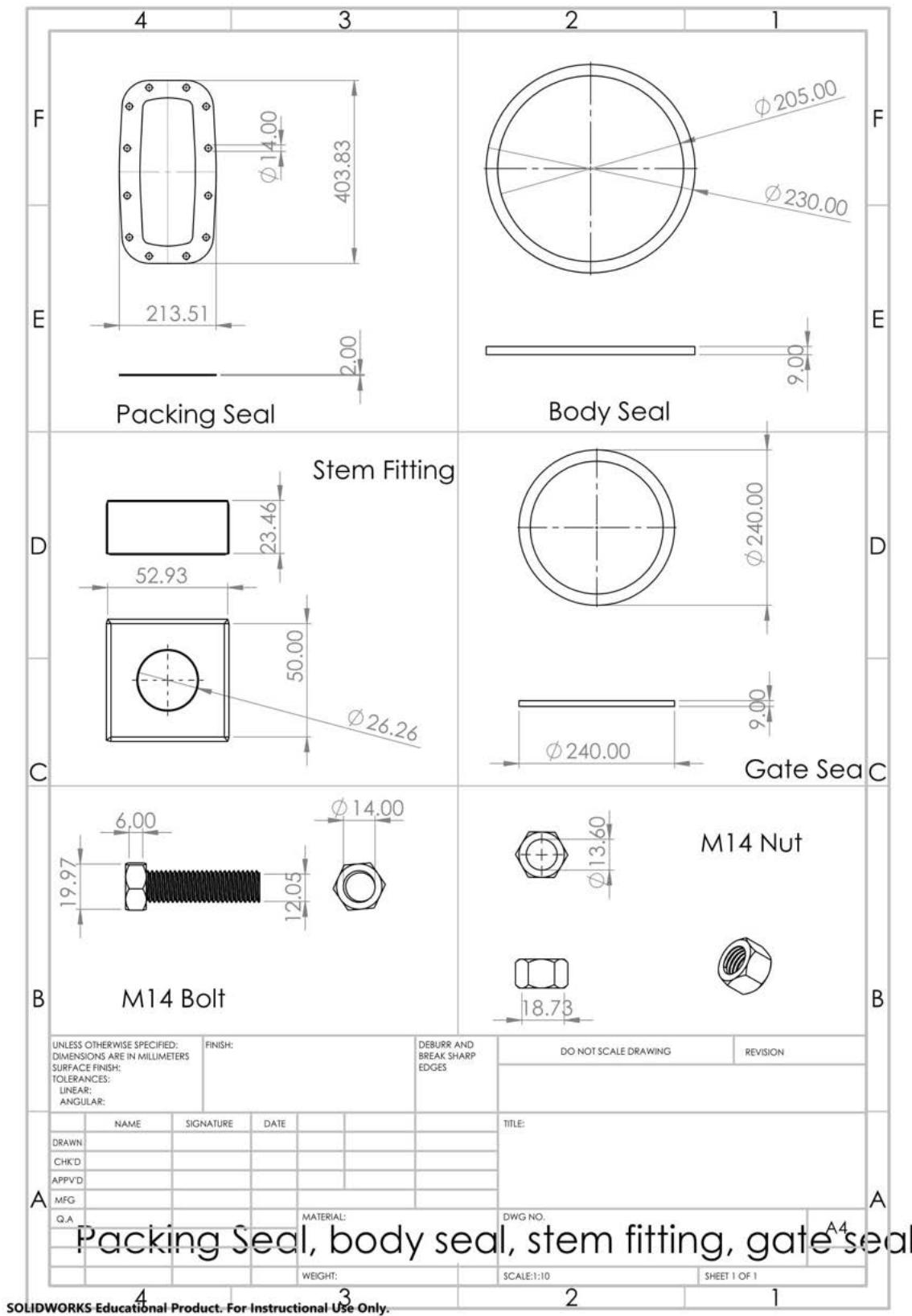
### 3.2 Valve Cover



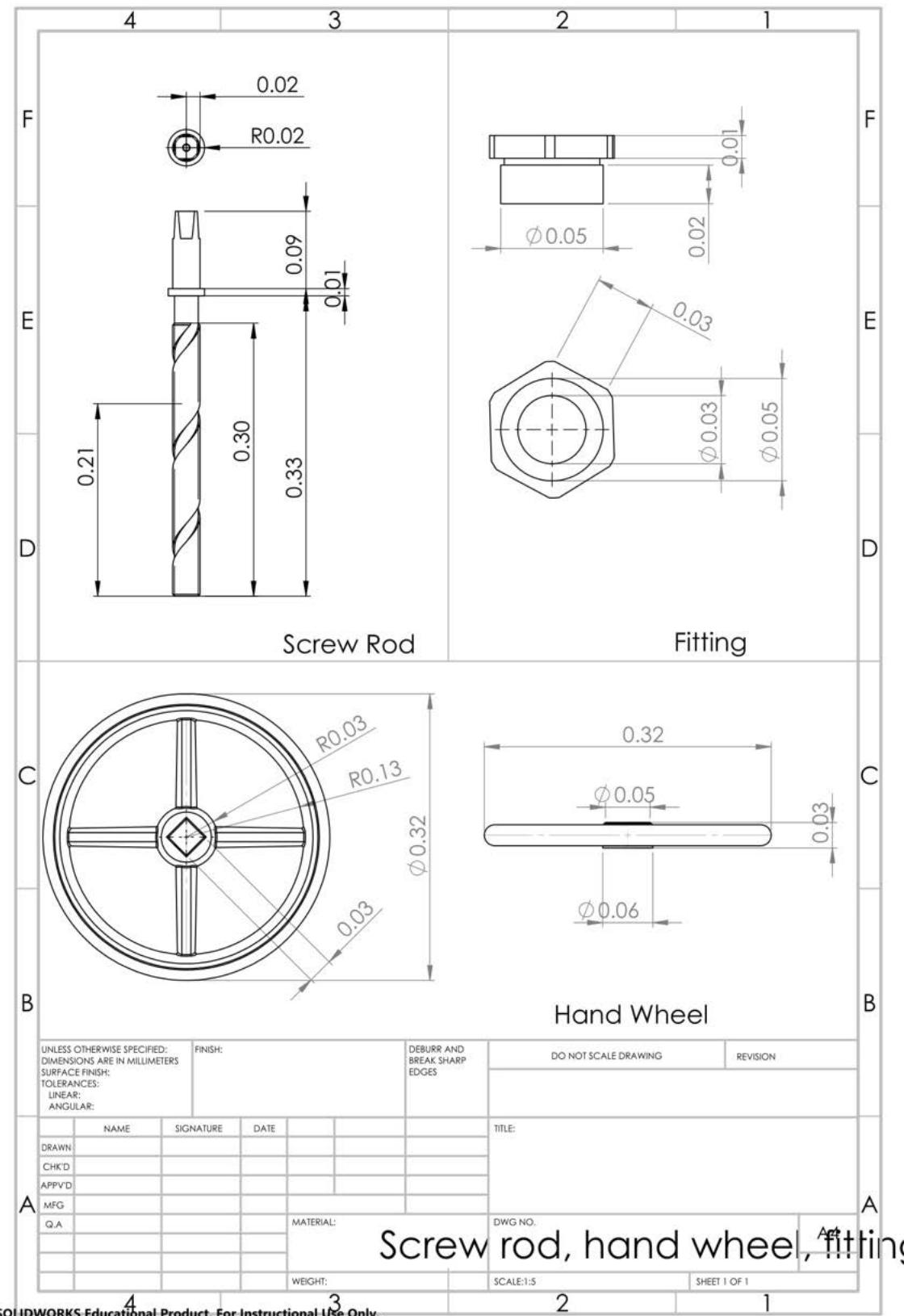
### 3.3 Sliding Gate



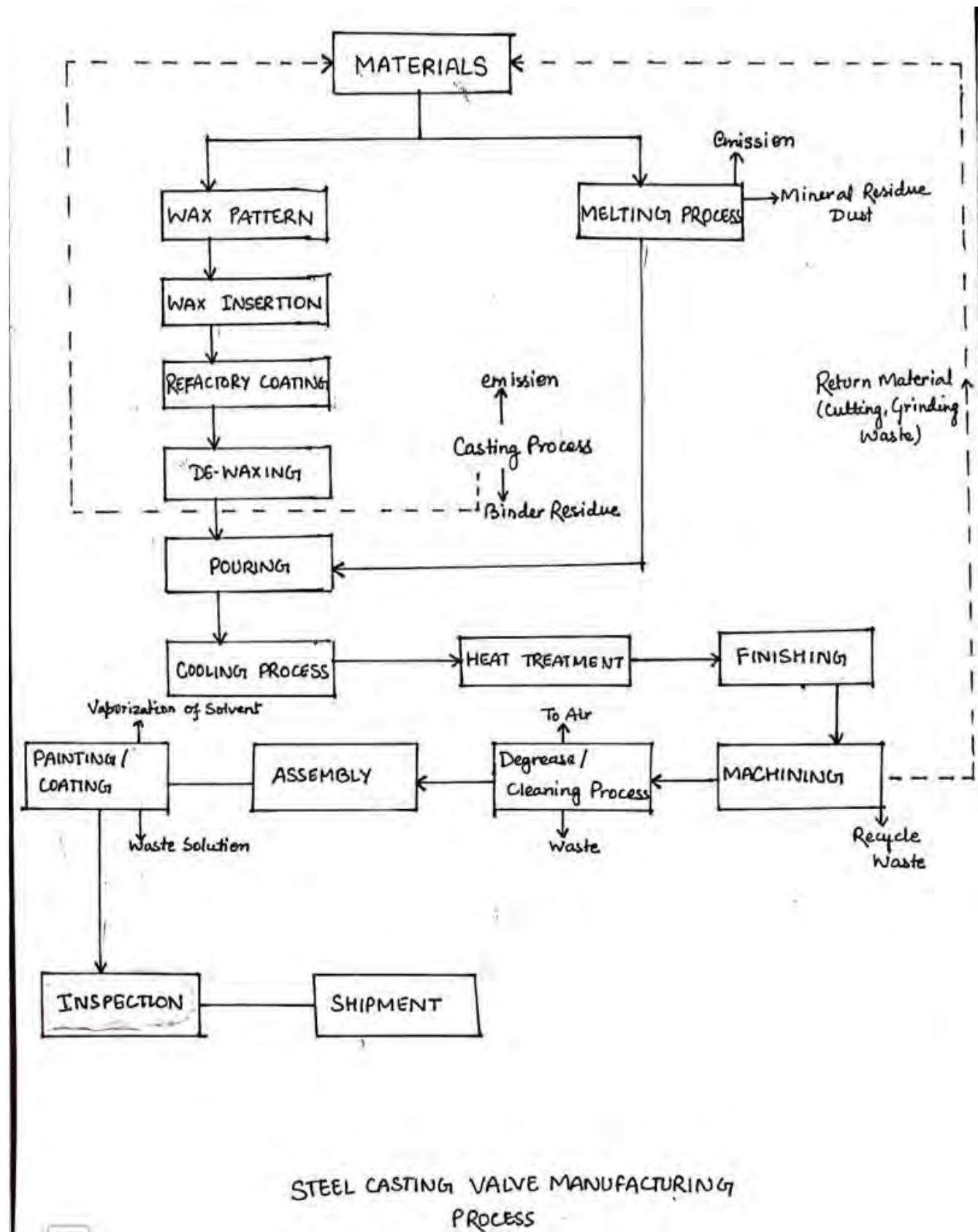
## 3.4 Packing Seal – Body Seal – Stem Fitting – Gate Seal – M14 Bolt – M14 Nut



## 3.5 Screw Rod – Fitting – Hand Wheel



#### 4. Floor Plan Layout



## 5. Cost Analysis

Assumptions: Facility efficiency: 90%

Working days per year: 305

Work Shifts: 8-hr shift per day

Production rate: 10,000 units/year

### Parts Cost

<i>Valve Body and Cover</i>	\$ 129.6
<i>Body Seal/ gate Seal</i>	\$ 21.6
<i>Hand Wheel</i>	\$ 43.2
<i>Screw Rod</i>	\$ 21.6
<i>Hex Nut/ Screw/ Bolts</i>	\$ 43.2
<i>Sliding Gate</i>	\$ 172.8
<i>Total</i>	\$ 432

Total Material/ Casting and Machining Cost \$ 432

Direct Labor Cost \$ 200

Manufacturing Overhead Cost \$ 168

Total Cost of Assembly \$ 800

## MANUFACTURING TRAVELER

---

Part no.: 0001

Description: Assembly

Start Date: 04/25/2020

Finish Date: 05/07/2020

Req. Date: 05/17/2020

Part	Operation	Quantity	Status
Valve Body & Cover	Investment Casting	1	Completed
Hand Wheel	Die Casting	1	Completed

Screw Rod	Cutting, Turning, Threading and Surface Finishing	1	Completed
Sliding Gate	Die Casting	1	Completed

## 6. Manufacturing Process for each Unique Part

### 6.1 Valve Body and Cover

Material: Cast Steel

Manufacturing Process: Investment Casting

It is the oldest known metal forming techniques, also called Lost-Wax Process. The wax process starts with wax injection into a pattern where flow channels are added onto the valve components. Each component receives 7-9 coats of refractory shell material. After that, the molds are removed in the de-wax area and is given a hot water wash to remove residual wax. Molds are then transferred to the baking furnace, brought upto a temperature of 800°C and maintained for 2 hours. Molds are then staged near the furnace for the baking process. The molten metal when it meets the standard chemical composition is then poured into the molds. This is the basic process involved in casting of valve body and cover.



#### Why Investment Casting?

- Reliability: gives a reliable control of the process and repeatability
- Tolerances: the tolerance of  $\pm .005"$  is not always possible with any other casting process.
- Lower tooling cost
- Better for environment: the use of wax patterns eliminates the expensive revisions and metal scrap. Also, investment casting produces parts that are net or near shapes which eventually reduces the secondary machining and labor cost.
- Intricate Design: features like logo, product ID etc. can easily be incorporated in the part. Holes, splines, slots, thread profiles is easy to cast with the use of investment casting to reduce the secondary machining time and total cost.

## 6.2 Body Seal

Material: Nitrile Rubber (High chemical resistivity).

Manufacturing Process: usually bought from other sources.

## 6.3 Gate Seal

Material: Stainless steel (Safe Material Interception and excellent durability)

Manufacturing process:



## 6.4 Hand Wheel

Material: Ductile Iron

Manufacturing Process: Die Casting



Why Die Casting?

It accurately produces sharp defined, smooth metal parts. Also, it's easy for mass production.

## 6.5 Screw Rod

Material: Aluminum/ Brass/ Bronze (Depending upon the gate valve)

Manufacturing Process:



## 6.6 Hex Nut

Material: Carbon Steel

Manufacturing Process:



## 6.7 Sliding Gate

Material: Cast Steel

Manufacturing Process:



### Why Die Casting?

It accurately produces sharp defined, smooth metal parts. Also, it's easy for mass production.

## References:

<https://www.lubkerdist.com/library/fastener-specifications/grade-and-material-identification-hex-nuts>

[https://www.milwaukeeprec.com/investment-castings-2.html?utm\\_expid=.5kh3RWmbR\\_GnjL18yd7NWQ.1&utm\\_referrer=https%3A%2F%2Fwww.google.com%2F](https://www.milwaukeeprec.com/investment-castings-2.html?utm_expid=.5kh3RWmbR_GnjL18yd7NWQ.1&utm_referrer=https%3A%2F%2Fwww.google.com%2F)

<https://www.mcwane.com/businesses/kennedy-valve/>

[https://en.wikipedia.org/wiki/Washer\\_\(hardware\)#Materials](https://en.wikipedia.org/wiki/Washer_(hardware)#Materials)

<https://blog.projectmaterials.com/valves/valve-parts/>

<http://www.astech-valve.com/en/2-2555-66375/product/Gate-Valve-Parts-and-MaterialListid365935.html>

<https://www.atlrod.com/metric-hex-bolt-dimensions/>

<https://www.nord-lock.com/insights/knowledge/2018/the-making-of-bolts/>

<https://www.aatprod.com/threaded-rod-materials/>

# Hydraulic Cylinder Design and Assembly

A general hydraulic cylinder was designed and assembled using SolidWorks by Atul Shrotriya. This project was a basic attempt to understand the design of hydraulic devices by Atul Shrotriya. It also helped brush up on the concepts of design and assembly using SolidWorks.

The resources for completing this project were provided by the University of Texas at Arlington. The resources were assessed by Atul Shrotriya as a graduate student in mechanical engineering.

The project may be extended further by analyzing the flow of fluids at different pressures and viscosity using ANSYS Fluent. The part and assembly files can be found using the following link -

<https://github.com/atulshrotriya/hydrauliccylinderanalysis>

Any updates will also be posted on the same link.

# Optimal Control of Cuff Pressure in Oscillometric Devices

Dr. David A. Hullender<sup>1</sup>, *Member, IEEE* and Atul Shrotriya<sup>2</sup>

*Department of Mechanical and Aerospace Engineering, The University of Texas at Arlington,  
Arlington, TX, 76013, USA*

Oscillometric devices consist of a cuff on the upper arm with a low frequency constant rate increasing or decreasing pressure plus an oscillating pressure induced by artery volume expansions and contractions caused by the blood pressure waveform in the brachial artery. Commercially available devices employ proprietary algorithms for computing diastolic and systolic blood pressure level estimates in the artery through cuff pressure measurements. Recent studies have revealed that the total arterial blood pressure waveform as well as stiffness properties of the artery can be extracted from the oscillometric cuff pressure using the extended Kalman filter. This paper attempts to improve the accuracy by optimally controlling the low frequency portion of the pressure in the cuff.

## I. Nomenclature

$a, b$	= constants used for modeling compliance of artery wall
$A_0, A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4, B_5$	= Fourier series coefficients
$f(P_t)$	= compliance function for the walls of the artery
$P(t)$	= blood pressure in the artery
$P_c(t)$	= cuff pressure
$P_{ca}(t)$	= absolute pressure in cuff
$P_{cm}$	= maximum cuff pressure
$P_t(t)$	= transmural pressure
$Q$	= air flow into the cuff
$V_o$	= initial cuff volume
$V_a(t)$	= artery volume
$V_{ao}$	= artery volume at zero transmural pressure
$V_c(t)$	= cuff volume
$V_{cm}$	= maximum cuff volume
$y(t)$	= pressure oscillations in the cuff
$\omega$	= unknown frequency

## II. Introduction

Measuring blood pressure is a basic check performed by medicine professionals. This check is essential and supplementary to a wide array of diagnostic procedures. In many cases requiring blood pressure tracking, patients often use remote kits for doing the test themselves. These readings are then reported to the consulting doctor. Modern devices including but not limited to smartwatches and fitness bands provide a convenient means to measure blood pressure. They can also be used for monitoring the blood pressure of a patient while they are sleeping or access it continuously to diagnose any irregularities such as arrhythmias.

Commercially available devices use proprietary algorithms to estimate the blood pressure using cuff pressure measurements. In a 2020 paper, Hullender showed that an extended Kalman filter yielded promising results to estimate

<sup>1</sup> Professor, Department of Mechanical Engineering.

<sup>2</sup> Graduate Student, Department of Mechanical Engineering.

the blood pressure[1]. Furthermore, it is reported that arterial stiffness and hypertension can also be calculated using the blood pressure waveform. A higher slope of the blood pressure waveform indicates thin walled arteries requiring minimal oscillations to change the pressure. Thus, the blood pressure waveform can also be used to assess the risk of heart diseases and diagnose hypertension, arterial issues etc.

However, modern Kalman algorithms utilize simultaneous signals from multiple sources such as pulse wave velocity (PWV) sensors or optical sensors found in most smartphone cameras. Blacher [1] surmised that pulse wave velocity measurements can be used to detect hypertension in patients. Various studies have found that most of these devices fail to extract full information available in the blood pressure waveform [3-5]. Furthermore, there is no method of determining if a certain type of pressure variation would result in better readings. Previous research and equations developed by Hullender [2] provides a model-based Kalman filter algorithm for estimating the total continuous blood pressure waveform in the brachial artery. The simulation is run for a total of 96 seconds allowing for 3 cycles of cuff pressure increment till 160mmHg and reduction to zero at the rate of 10mmHg/sec.

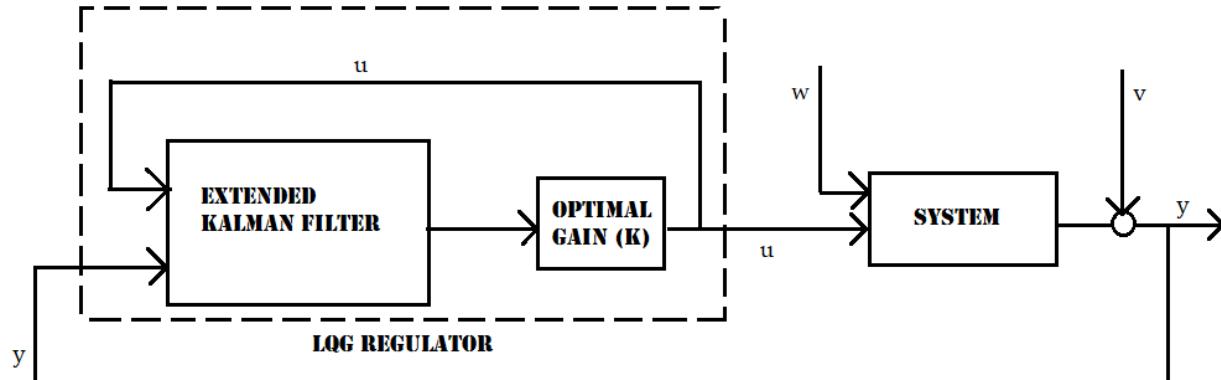
This model can be used to control the cuff pressure in the system optimally and generate better readings for the blood pressure waveform. Various approaches are evaluated for optimizing the cuff pressure.

### III. Control Methods

The system model is defined by Hullender [2] using the following equations:

$$\begin{aligned}
 \dot{p}(t) &= -\frac{5}{(t)} + (1 - \frac{5}{(t)}) \\
 \dot{\cdot}(t) &= \frac{1.4}{(t)} \cdot(t) \\
 (t) &= -\cdot(t) \quad (t) \geq 0 \\
 (t) &= \cdot(t) \quad (t) \leq 0 \\
 (t) &= p_0 + p_1 \sin(t) + p_2 \sin(2t) + p_3 \sin(3t) + p_4 \sin(4t) + p_5 \sin(5t) + p_6 \cos(t) \\
 &\quad + p_7 \cos(2t) + p_8 \cos(3t) + p_9 \cos(4t) + p_{10} \cos(5t) \\
 (t) &= (t) - (t)
 \end{aligned}$$

The Kalman filter to estimate the 16 parameters is an extended Kalman filter. This locally linearizes the non-linear equations and computes estimates and is not applicable for severely non-linear systems. To optimize this, a linear quadratic regulator (LQR) approach is attempted with constrained control as the cuff pressure input is limited by maximum pressure and pressure change rate. This system can be generally described by the below diagram:



**Figure 1. Diagram for an LQG regulator**

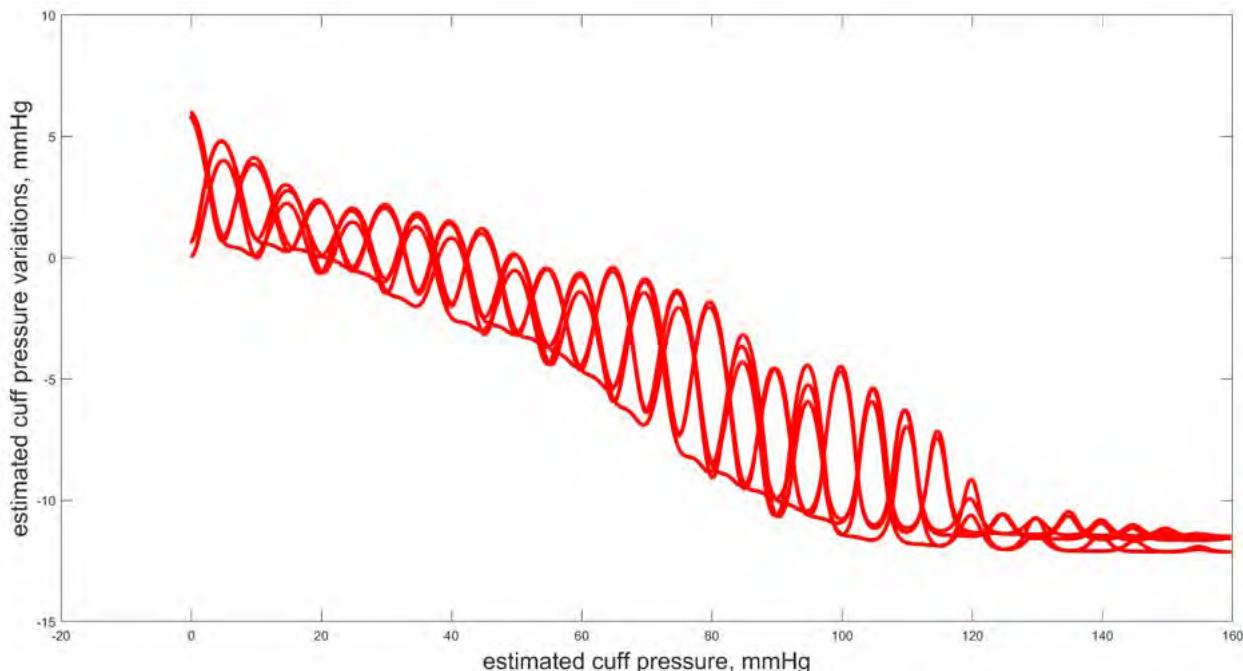
Here,  $v$  is the measurement noise and  $w$  is the process noise. The input to the system is  $u$  and the output is  $y$ . To formulate the optimal gain through LQR, the first step is to define a cost function which needs to be extremized. On evaluating the above equations, it is anticipated that optimally varying the low frequency portion of pressure inflating and deflating the cuff would lead to more accurate estimates from the Kalman algorithm. Simulations of the

combined optimally controlled low frequency portion of the cuff pressure in conjunction with the Kalman filter algorithm are used to assess the potential improvements in robustness and accuracy of the concept.

The maximum volume of cuff is  $100 \text{ cm}^3$  whereas the initial volume is  $20 \text{ cm}^3$ . The constants 'a' and 'b' may vary from equipment to equipment and are kept as 0.075 and 0.02 respectively in mmHg units. For comparison, the normal artery has the same parameters as 0.11 and 0.03. The actual artery volume at 0 transmural pressure is  $0.25 \text{ cm}^3$  and initial estimate is  $0.3 \text{ cm}^3$ .

#### IV. Controller Design

Since the measured output is cuff pressure variations, it is assumed that a higher magnitude of these variations would result in more information for the Kalman filter leading to better estimates. Therefore, a graph is plotted between estimated cuff pressure ( $P_c(t)$ ) which is the input and the estimated cuff pressure variations( $y(t)$ ).



**Figure 2. Relation of variations in cuff pressure to the cuff pressure**

From the above graph, it appears that there are three options for optimization:

1. The maximum amplitude of  $y(t)$  which is around 160 mmHg i.e., maximum cuff pressure ( $P_{cm}$ )
2. The maximum variation in amplitude of  $y(t)$  which is around 80mmHg
3. On closer inspection of the graph and comparing it to the blood pressure waveform, it is observed that the maximum amplitude difference around 80 mmHg is achieved as it corresponds to the diastolic blood pressure. This is readily observable from the ' $\cdot$ ' equation. Here, the compliance function ( $f(P_t)$ ) decreases for any value of transmural pressure above 0. Similarly, if the cuff pressure ( $P_c$ ) is stabilized then the only contribution to ' $\cdot$ ' would be through the change in blood pressure. This will reduce the bias due to input. Therefore, a controller can be designed to maintain the cuff pressure near to the diastolic pressure based on the average diastolic pressure obtained from previous Kalman estimates. As the readings proceed, the average will stabilize which will lead to better control and estimation.

Research studies show that human body can withstand sudden external pressures of nearly 40 psi or more than 2000 mmHg without damage. Air pressures of 5 psi or nearly 260 mmHg directly to the eardrum can cause damage in 1 percent of cases. From the above graph, it is apparent that increasing the cuff pressure to nearly 160 mmHg results in

the maximum magnitude of cuff pressure variations which are then used by the extended Kalman filter to generate estimates.

However, a cuff pressure of 160 mmHg which is around 3psi doesn't cause any harm to humans even if it's applied suddenly. It must be noted that sudden increase doesn't correspond to an impulse as it would generate a pressure increment in the form of blast overpressure which can cause injuries at values of 3psi. Since this is not applicable practically, the only limitation to reach a pressure of 160mmHg is the maximum air input to the cuff.

The quick pressure increment may not be suitable for a few extremely fragile patients but in that case even the conventional oscillometric devices may not be a good choice. Hence, a cost function with input constraint based on maximum air input to cuff can be defined to bring the cuff pressure to 160mmHg within a finite duration. This duration would be the time step used by the Kalman filter so that the next incoming value doesn't interfere with the current optimal input.

Cost functions to be maximized are:

$$1 = \int_0^h (\dot{P})^2 dt, \quad P(0) = 160; \quad 1(h)$$

$$2 = \int_0^h \frac{(\dot{P})^2}{P} dt = \int_0^h \left( \frac{\dot{P}}{P} \right)^2 P dt, \quad P(0) = 80; \quad 2$$

$$\dot{P} \leq | \ddot{P} |;$$

$P_{qm}$  is determined by  $Q_{max}$  or maximum air flow rate into the cuff.

Cost function to be minimized is:

$$3 = P_d(t) + \int_0^h (P - P_d)^2 dt, \quad h \quad h$$

$$(P) \leq | \ddot{P} |; \quad P = 1, \quad \dot{P}(0) = 0, \quad \ddot{P}(0) > 0, \quad h, \quad ,$$

$$(P) = \frac{1}{n} \sum_{i=1}^{n-1} (P_i)$$

Here,  $P_d(t)$  is the average diastolic pressure based on previous values and  $n$  is defined by the number of estimations available. The input constraint remains the same as it is only dependent on maximum air input. Square values are taken so that the magnitude is minimized instead of the system trying to make the cuff pressure zero. Also, a soft constraint is specified to ensure that the system moves in the correct direction this may not be necessary as the diastolic pressure will dictate the direction adequately.

#### **Noise Cancellation Approach:**

After discussion with Dr. Robert L. Woods, it was concluded that the cuff system cannot be modeled as an isolated lumped parameter system containing gases with variations caused due to blood pressure oscillations as noise. Thus, the system may not be adequately controllable using the Linear Quadratic Tracker approach.

A better approach would be the use of Model Predictive Control (MPC) techniques as they can handle interaction between input and outputs. However, they have limitations for long calculations as explained by Dr. Kamesh Subbarao in the Optimal Controls class. MPC can be integrated into the Kalman filter over a few steps to provide feasible solution. This would require extensive modification of MPC and local linearization of function.

The system is remodeled to exclude the variations considering them to be small and non-detrimental to the average pressure within the cuff.

### System remodeling:

The system is remodeled using the lumped parameter modeling approach [12]. The equations for temperature and density variation are neglected considering that they don't have much impact. Implementing the capacitance equation for cuff pressure gives us:

$$\dot{P}_c = 1.2 * \frac{(-0.18)}{0 + } \quad (1)$$

Here,  $P_c$  is the cuff pressure and  $u$  is the input provided to optimize it. Note, the heat transfer coefficient is kept 1.2 to give mid-range values. The above equation is not linear or an ordinary differential equation as there are two unknowns being multiplied. Therefore, the traditional methods are not applicable in this case.

Combining different approaches taught by Dr. Subbarao, it can be derived:

$$= 1 * \left( \frac{0.22}{0 + } \right) \quad (2)$$

Assuming the system takes 4 seconds to reach the pressure of 80mmHg, calculating and substituting the initial and final conditions,

$$1.609 = -0.22(\ln(4 * ) + \ln( )) \quad (3)$$

This gives  $u \sim 0$  which cannot take the system anywhere in the desired time. Therefore, the usual methods are not applicable in this case.

On trying the conventional two point boundary value approach,

$$\begin{aligned} ( ) &= ( - 80)^2 + \left( - \frac{0.22}{0 + } \right) \\ \frac{d}{dt} &= - \cdot = 2(-80) - \left( \frac{0.22}{0 + } \right) \\ \frac{d}{dt} &= - \frac{0.22}{0 + } = 0 \end{aligned}$$

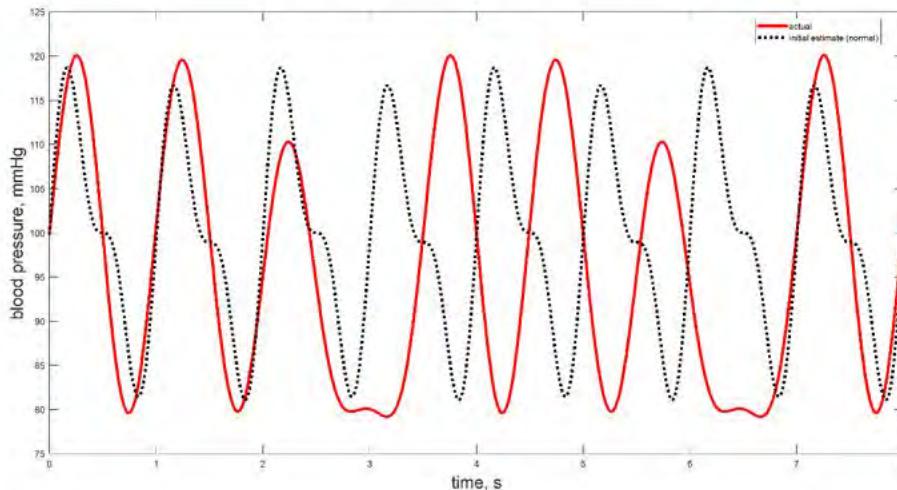
This implies that  $\lambda = 0$  and doesn't provide a reliable solution.

### MATLAB loop

The final approach is attempted by simply choosing control effort ( $u$ ) as the difference between current and target value (80mmHg) of cuff pressure. This is implemented using a for loop as follows:

```
for P_c>80
    u=-|P_c-80|;
    for P_c<80
        u=|P_c-80|;
    else u=0;
```

The above loop was added to Kalman filter at the position where it stores values for  $P_c$  and the plots were obtained:



**Figure 3. Plot for real and estimated blood pressure values after modification of extended Kalman filter**

From the above plot, it appears that there is a slight improvement in the rate of convergence. On the other hand, the estimate values for coefficients are thrown off slightly. This seems to be a result of non-robust coupling of an estimator and controller. It may also be the result of extended computation as the program keeps running for over an hour while updating the plots slowly.

## V. Conclusion

Effectively, it is concluded that the rate of pressure change cannot be linearized properly and is a highly nonlinear function. Therefore, the controller has to be redesigned using nonlinear approach for local linearization along with a backstepping algorithm approach. Then the robustness has to be ascertained so that the estimated values converge to the true values of system.

## References

- [1] Blacher J., Asmar R., Djane S., London G. M., and Safar M. E., “Aortic Pulse Wave Velocity as a Marker of Cardiovascular Risk in Hypertensive Patients,” *Hypertension AHA Journal*, Vol. 33, No. 5, 1999, pp. 33:1111-1117.  
doi: 10.1161/01.HYP.33.5.1111
- [2] D. A. Hullender and O. R. Brown, “Blood Pressure, Atrial Fibrillation and Arterial Stiffness Data Extracted from Oscillometric Waveform by Kalman Filtering,” *IEEE Transactions on Biomedical Engineering (TBME)*, 2020.  
doi: 10.2514/1.C033040
- [3] Montfrans, G., “Oscillometric blood pressure measurement: Progress and problems,” *Blood pressure monitoring*, Vol. 6, 2002, 287-90.  
doi: 10.1097/00126097-200112000-00004.
- [4] Miranda J. J., Stanojevic S., Bernabe-Ortiz A., Gilman R. H., and Smeeth L., “Performance of oscillometric blood pressure devices in children in resource-poor settings,” *Eur J Cardiovasc Prev Rehabil.*, Vol. 15, No. 3, 2008, pp : 362–364.  
doi:10.1097/HJR.0b013e3282f738b8
- [5] Czarkowski, M., Staszkow, M., Kostyra, K., Shebani, Z., Rozanowski, K., Matuszkiewicz-Rowinska, J., and Niemezyk, S., “Limitations of the of the oscillometric method for blood pressure measurements in dialyzed patients,” *Medical science monitor: international medical journal of experimental and clinical research*, Vol. 17, No. 4, MT35–MT40. <https://doi.org/10.12659/msm.881701>.
- [6] *Physics in Medicine*, University of Notre Dame, 2003, Chap 3. URL: <https://www3.nd.edu/~ns1/Lectures/mphysics/>
- [7] Why Do We Really Need Pressure Suits? , NASA, Section I, [https://www.nasa.gov/sites/default/files/atoms/files/dressing\\_for\\_altitude.pdf](https://www.nasa.gov/sites/default/files/atoms/files/dressing_for_altitude.pdf)
- [8] *Physics of Diving*, National Oceanic and Atmospheric Administration, US Dept. of Commerce. Pp. 2-1. URL: <https://www.ehs.ucsb.edu/files/docs/ds/physics.pdf>
- [9] Crawford, H., *Survivable Impact Forces on Human Body Constrained by Full Body Harness*, Health and Safety Executive, UK, URL: [https://www.hse.gov.uk/research/hsl\\_pdf/2003/hsl03-09.pdf](https://www.hse.gov.uk/research/hsl_pdf/2003/hsl03-09.pdf)
- [10] Hoback, W. W., Pursley A., Farnsworth-Hoback K., and Higley L. G., “A Laboratory Exercise to Explore Sustained Gravitational Force Tolerance by Insects,” *The American Biology Teacher*, Vol. 77, No. 9, 2015 pp. 707-709.  
doi: 10.1525/abt.2015.77.9.11
- [11] Zipf, R. K., Cashdollar, K. L., “Effects of blast pressure on structures and the human body”, Explosions and Refuge Chambers, Center for Disease Control and Prevention, URL: <https://www.cdc.gov/niosh/docket/archive/pdfs/niosh-125/125-explosionsandrefugechambers.pdf>
- [12] Hullender, D. A., *Dynamic Systems Modeling and Simulation*, 18<sup>th</sup> ed., July 2019, pp. 223