Modern Control Project

Controller Design Based on Modern Control Theory

Student Name:

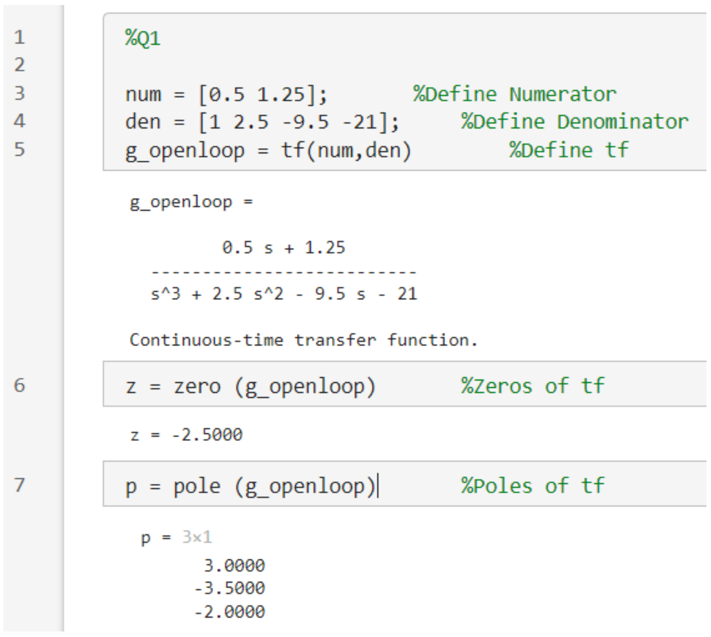
Amirreza Zarrabinia

Professor Name:

Dr. Hossein Bolandi

July 2024

Q1:



```
%Q1

num = [0.5 1.25];        %Define Numerator
den = [1 2.5 -9.5 -21];      %Define Denominator
g_openloop = tf(num,den)         %Define tf
```

```
g_openloop =

        0.5 s + 1.25
  --------------------------
  s^3 + 2.5 s^2 - 9.5 s - 21

Continuous-time transfer function.
```

```
z = zero (g_openloop)       %Zeros of tf
```

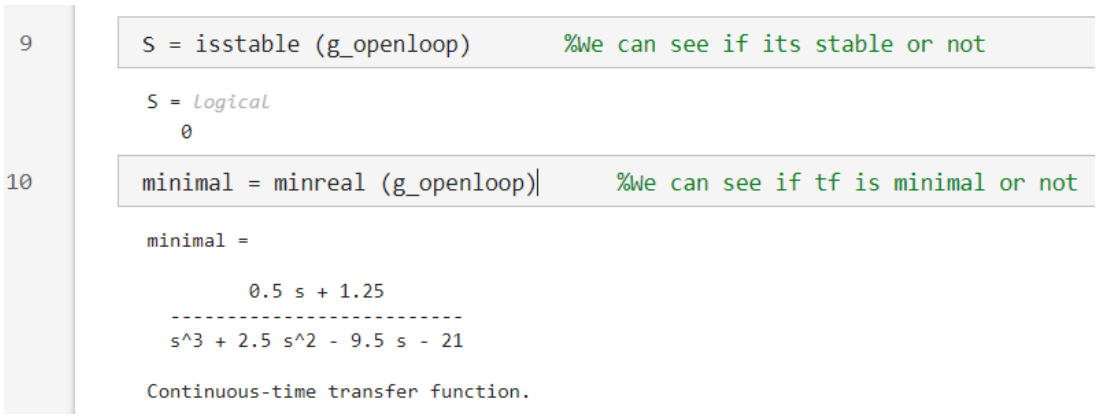
```
z = -2.5000
```

```
p = pole (g_openloop)       %Poles of tf
```

```
p = 3x1
     3.0000
    -3.5000
    -2.0000
```

First, we define the numerator and denominator of the transfer function and then convert it into a transfer function in the MATLAB environment using the “tf” command.

Next, we can observe the zeros and poles of the open-loop system using the zero and pole commands. We notice that the system has a pole in the right half of the imaginary axis, indicating the instability of the system.



```
S = isstable (g_openloop)       %We can see if its stable or not
```

```
S = logical
   0
```

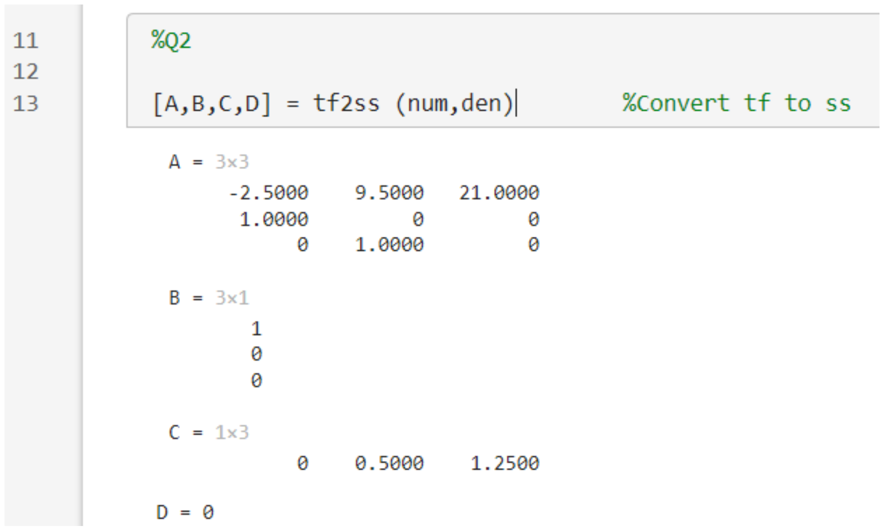
```
minimal = minreal (g_openloop)      %We can see if tf is minimal or not
```

```
minimal =

        0.5 s + 1.25
  --------------------------
  s^3 + 2.5 s^2 - 9.5 s - 21

Continuous-time transfer function.
```

Additionally, we can use the “isstable” command to check the stability of the system, which returns a logical output of zero or one. A zero indicates instability, while a one indicates stability.

To check the minimality of the system, we use the “mineral” command, which provides the minimal realization of the system. It can be seen that this is the same as the original transfer function. Therefore, the transfer function is minimal.

Q2:

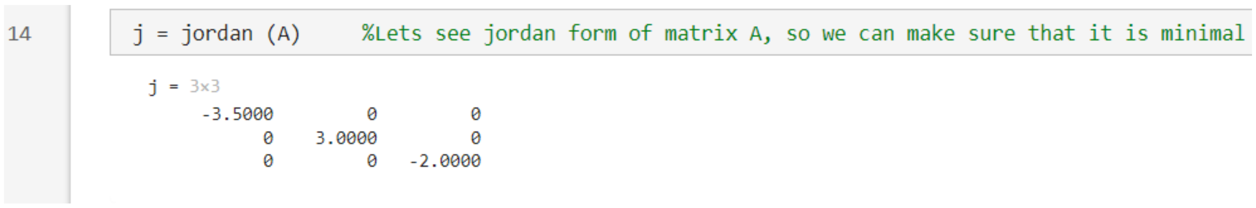


```
%Q2

[A,B,C,D] = tf2ss (num,den)        %Convert tf to ss
```

```
A = 3x3
     -2.5000    9.5000   21.0000
      1.0000         0         0
           0    1.0000         0

B = 3x1
      1
      0
      0

C = 1x3
           0    0.5000    1.2500

D = 0
```

Using the “tf2ss” command, we can easily obtain the state-space representation of the system from the transfer function.

To ensure the minimality of the system, we can display the Jordan matrix A and examine it, as shown below:

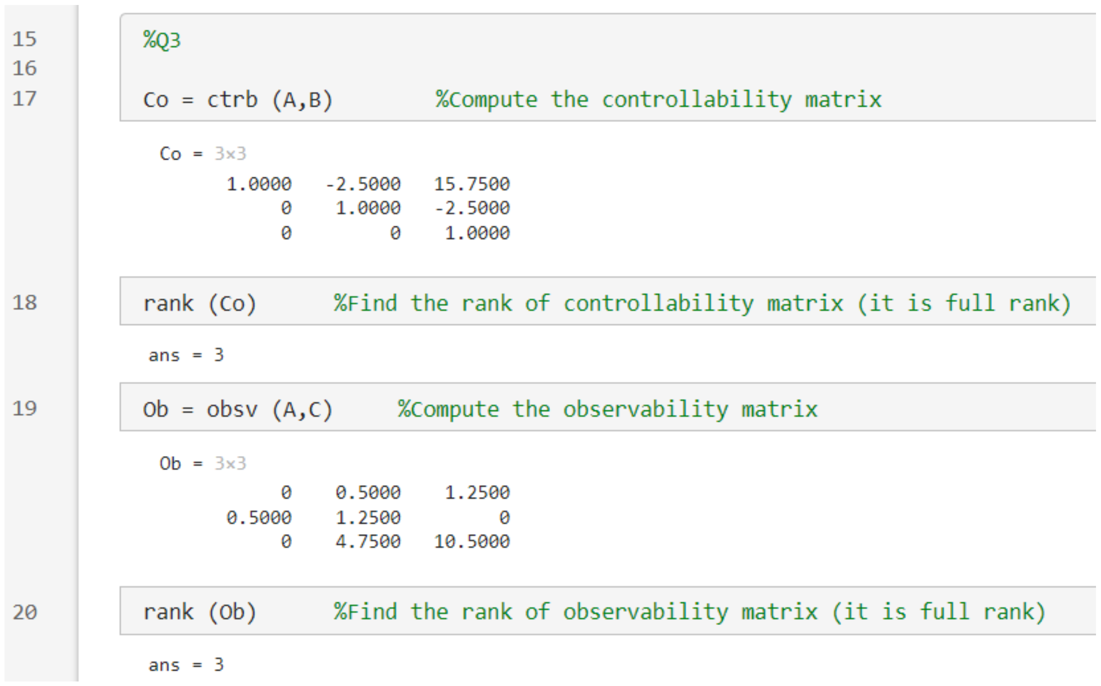


```
j = jordan (A)     %Lets see jordan form of matrix A, so we can make sure that it is minimal
```

```
j = 3x3
   -3.5000         0         0
         0    3.0000         0
         0         0   -2.0000
```

It can be seen that the system is minimal.

Q3:



```
%Q3

Co = ctrb (A,B)         %Compute the controllability matrix
```

```
Co = 3x3
      1.0000   -2.5000   15.7500
           0    1.0000   -2.5000
           0         0    1.0000
```

```
rank (Co)      %Find the rank of controllability matrix (it is full rank)
```

```
ans = 3
```

```
Ob = obsv (A,C)     %Compute the observability matrix
```

```
Ob = 3x3
           0    0.5000    1.2500
      0.5000    1.2500         0
           0    4.7500   10.5000
```

```
rank (Ob)      %Find the rank of observability matrix (it is full rank)
```

```
ans = 3
```

For this question, we construct the controllability and observability matrices using the “ctrb” and “obsv” commands, respectively, and examine the rank of each. We know that if these matrices are full rank, it indicates controllability and observability.

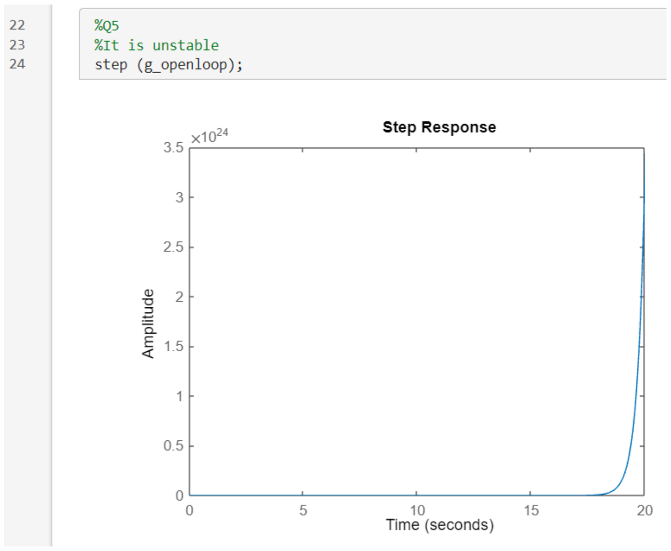
Given that the system possesses both controllability and observability properties, it is possible to design a state feedback controller and a state observer for this system.

Q4:

Since the system is both controllable and observable, the state-space representation obtained in the previous section is minimal.

Q5:

We know that the system is unstable, but let's observe its response to a step input.



```
%Q5
%It is unstable
step (g_openloop);
```

The instability of the system is clearly evident with the step input.

Q6:

```
%Q6

g_closedloop = feedback (g_openloop,1)      %Compute closed loop tf
```

```
g_closedloop =

         0.5 s + 1.25
  ---------------------------
  s^3 + 2.5 s^2 - 9 s - 19.75

Continuous-time transfer function.
```

We obtain the closed-loop transfer function of the system using the “feedback” command.

```
z1 = zero (g_closedloop)       %Zeros of closed loop tf
```

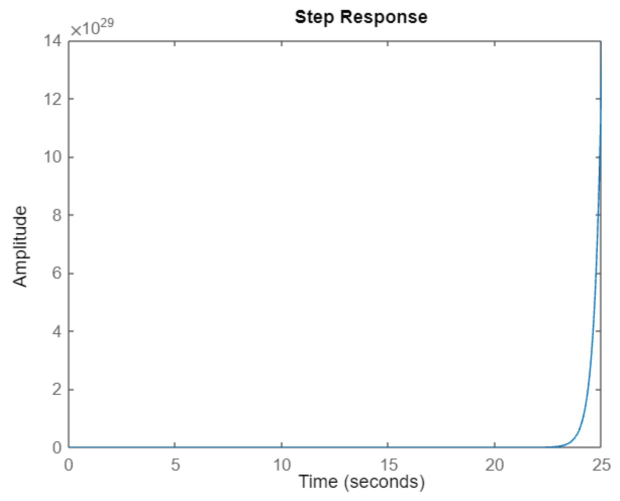
```
z1 = -2.5000
```

```
p1 = pole (g_closedloop)       %Poles of closed loop tf
```

```
p1 = 3x1
      2.9141
     -3.4492
     -1.9649
```

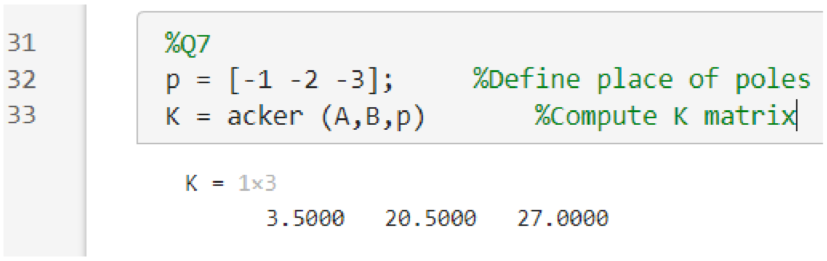
We display the zeros and poles of the system.

```
step (g_closedloop)        %Step response of closed loop tf
```



The step response will also be plotted as shown above, using the step command.

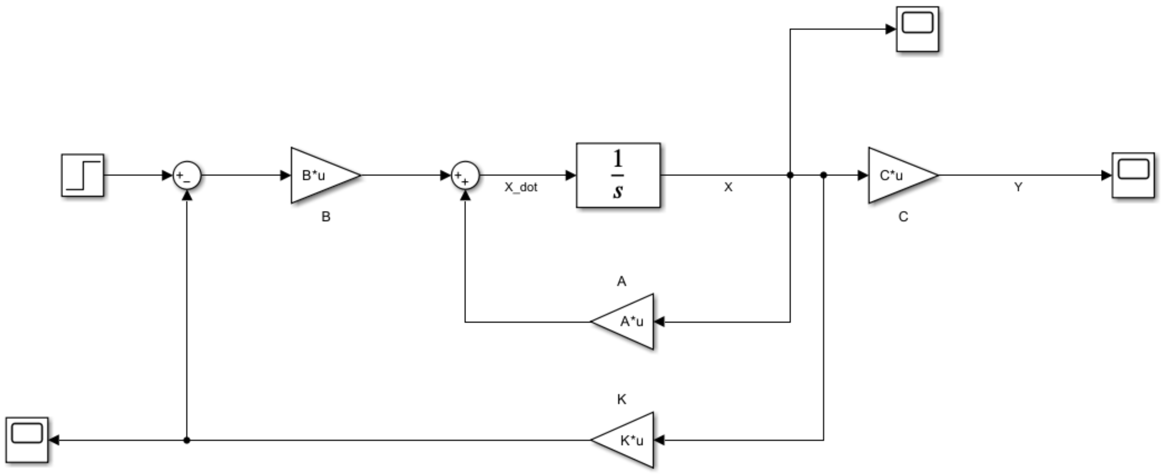
Q7:



```
%Q7
p = [-1 -2 -3];       %Define place of poles
K = acker (A,B,p)        %Compute K matrix
```

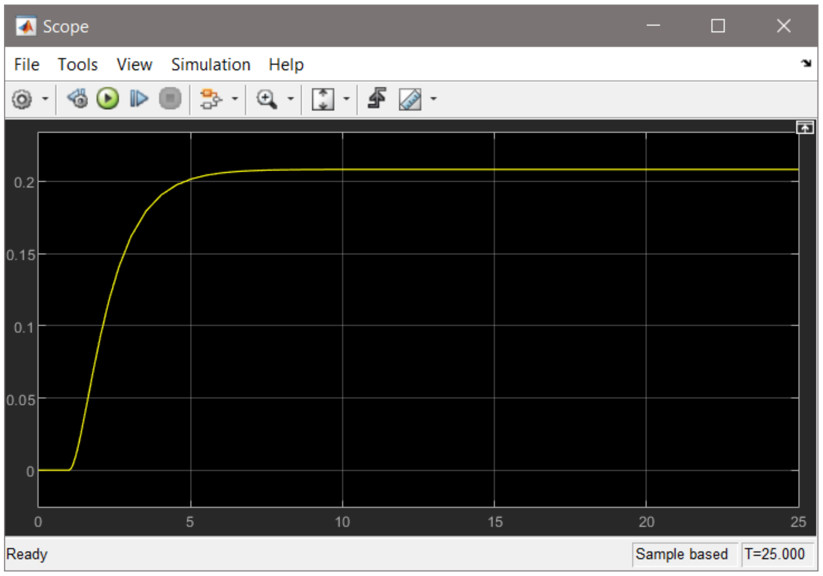
```
K = 1×3
      3.5000    20.5000    27.0000
```

First, we set the pole locations at -1, -2, and -3, and accordingly, we compute the matrix K using the “acker” command. Now, we simulate the state feedback in the Simulink environment.

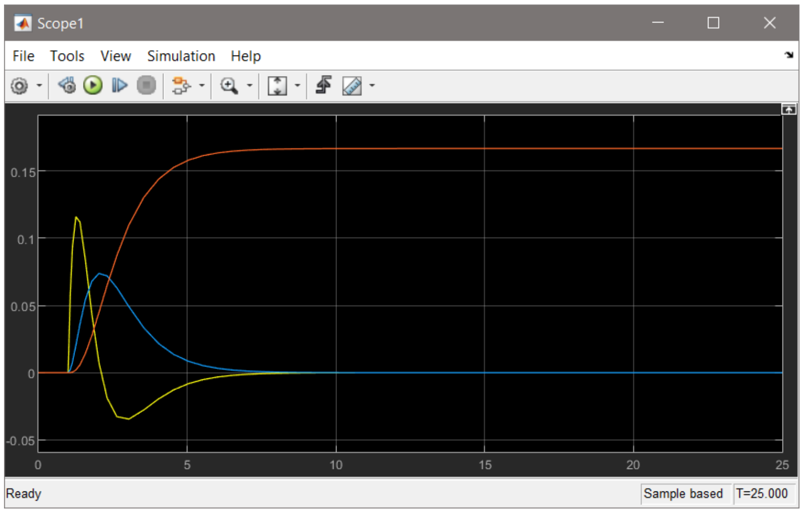


The output of the system, state variables, and the control signal are observed from the scopes and are shown below.

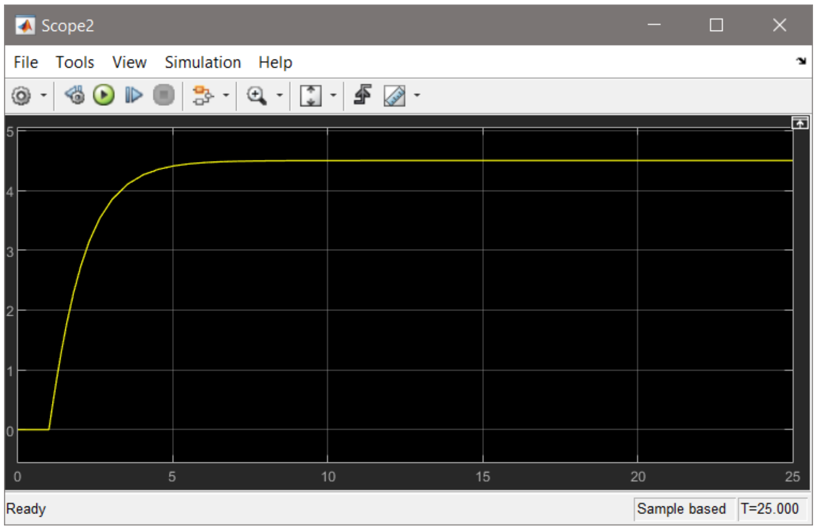
Output:



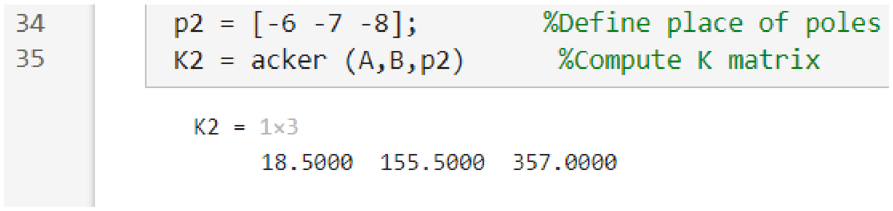
State Variables:



Control Signal:

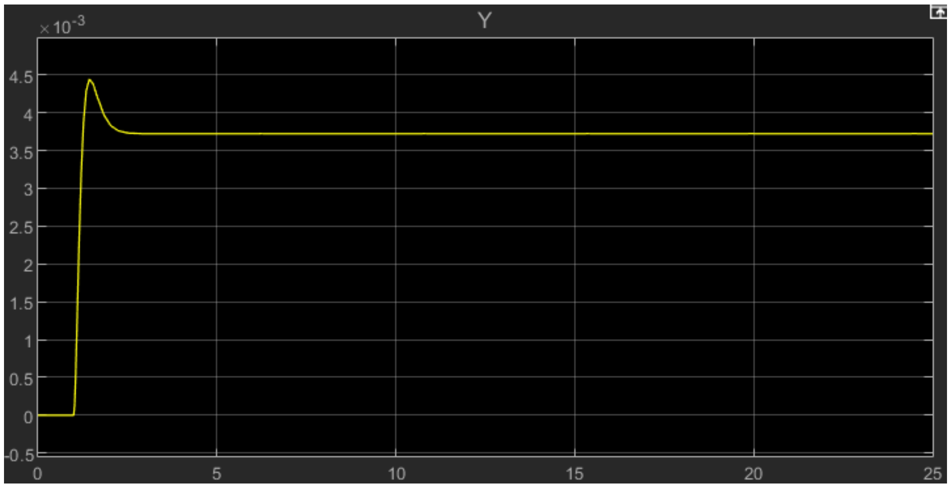


Now, we move the desired poles away from the imaginary axis and perform the simulation for the new matrix K to observe the result of this pole relocation in the scopes.



```
p2 = [-6 -7 -8];          %Define place of poles
K2 = acker (A,B,p2)        %Compute K matrix

  K2 = 1×3
       18.5000   155.5000   357.0000
```

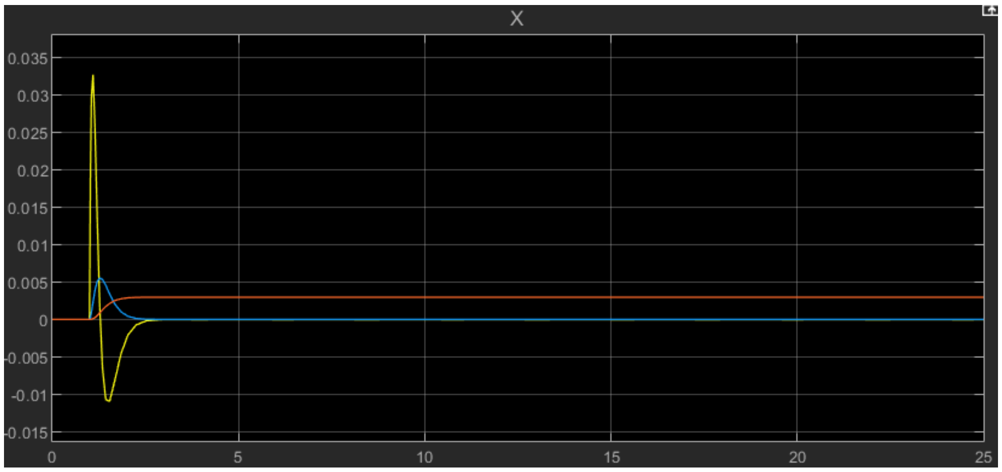
We have shifted the poles and computed the new matrix K.



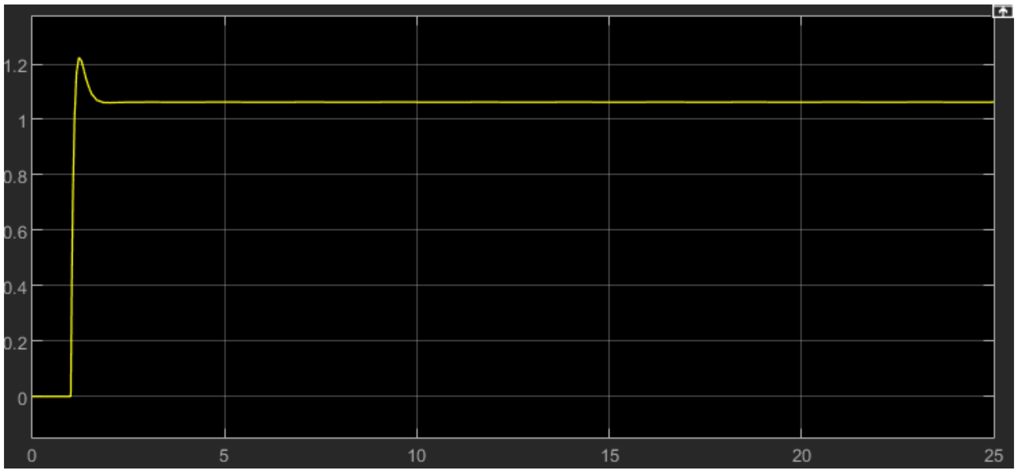
The output appears as shown above. It can be concluded that by moving the poles farther away, the overshoot of the system increases, and the gain decreases. Here, we have reached a much lower steady-state value compared to when the poles were closer.

We also knew that the farther the poles are, the faster the system becomes, which is also observable. However, if the poles are moved too far from the imaginary axis, they will become ineffective.

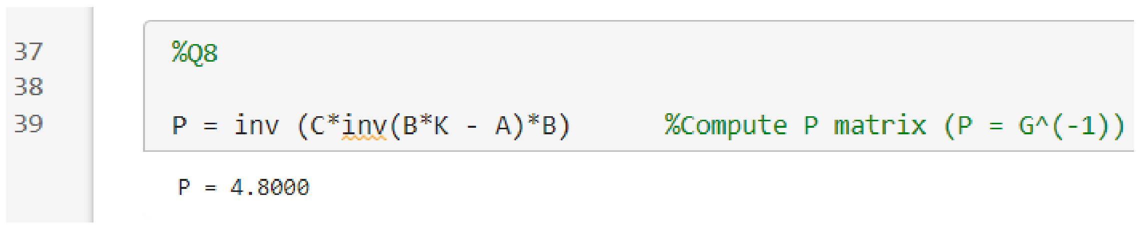
State Variables:



Control Signal:



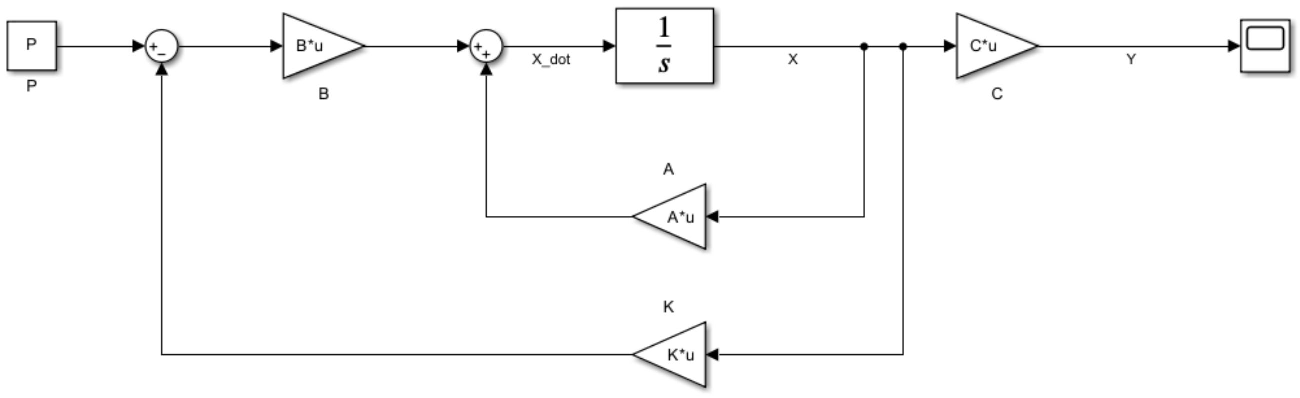
Q8:



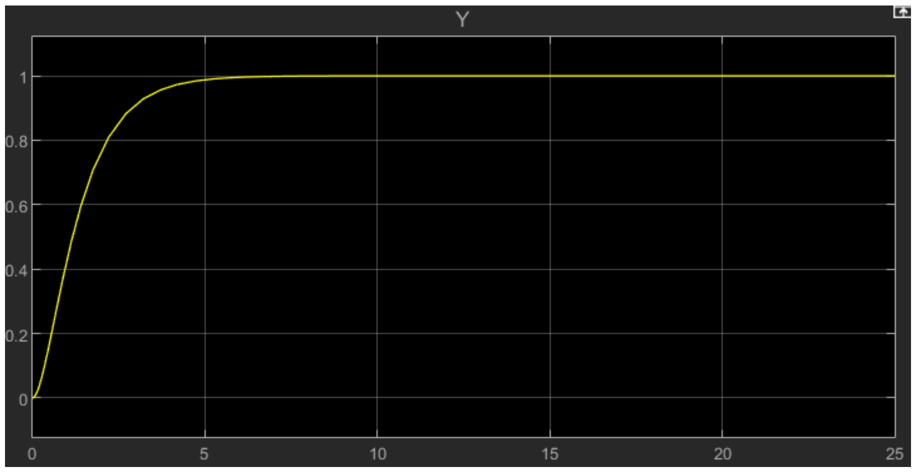
```
%Q8

P = inv (C*inv(B*K - A)*B)      %Compute P matrix (P = G^(-1))
```

```
P = 4.8000
```

Assuming the desired pole locations are -1, -2, and -3, we calculate P using the obtained K and perform the simulation below.



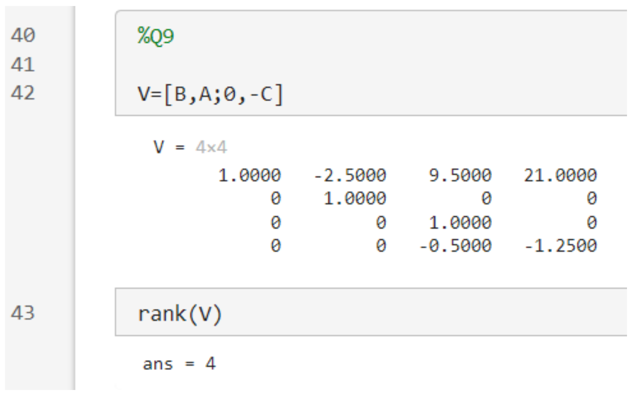
Now, let's examine the output:



As we expected, after designing the observer, the system accurately tracks the step response and reaches a value of one.

Q9:

For the design of the integral observer, we need to check whether the matrix V, defined below, is full rank.



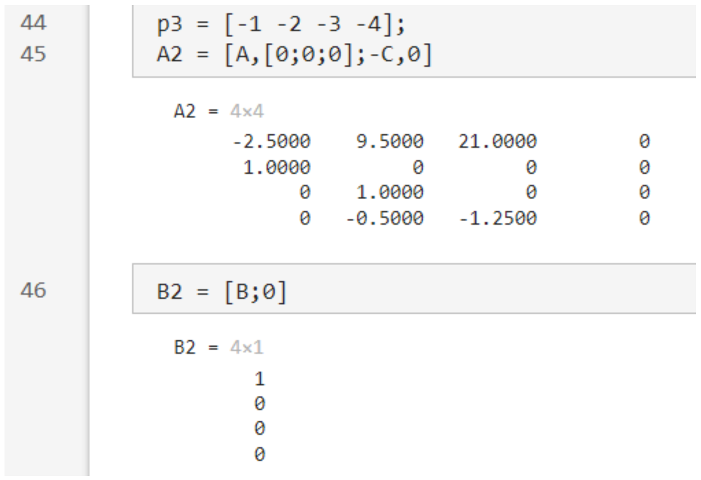
```
%Q9

V=[B,A;0,-C]
```

```
V = 4x4
     1.0000   -2.5000    9.5000   21.0000
          0    1.0000         0         0
          0         0    1.0000         0
          0         0   -0.5000   -1.2500
```

```
rank(V)
```

```
ans = 4
```

Thus, the necessary and sufficient condition is satisfied, allowing us to design the integral observer.



```
p3 = [-1 -2 -3 -4];
A2 = [A,[0;0;0];-C,0]
```

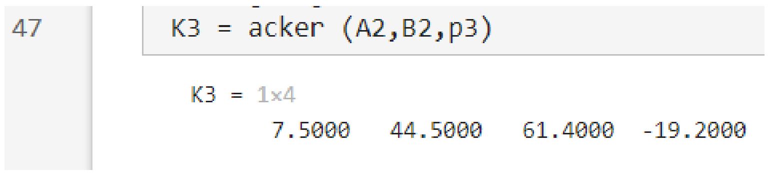
```
A2 = 4x4
    -2.5000    9.5000   21.0000         0
     1.0000         0         0         0
          0    1.0000         0         0
          0   -0.5000   -1.2500         0
```

```
B2 = [B;0]
```

```
B2 = 4x1
     1
     0
     0
     0
```

Assuming that the poles move to the locations -1, -2, -3, and -4, we calculate the new matrices A and B based on the previous knowledge as shown above.

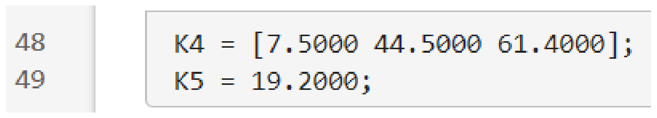
Now, we calculate K3, from which the two matrices K4 and K5 can be derived.



```
K3 = acker (A2,B2,p3)
```

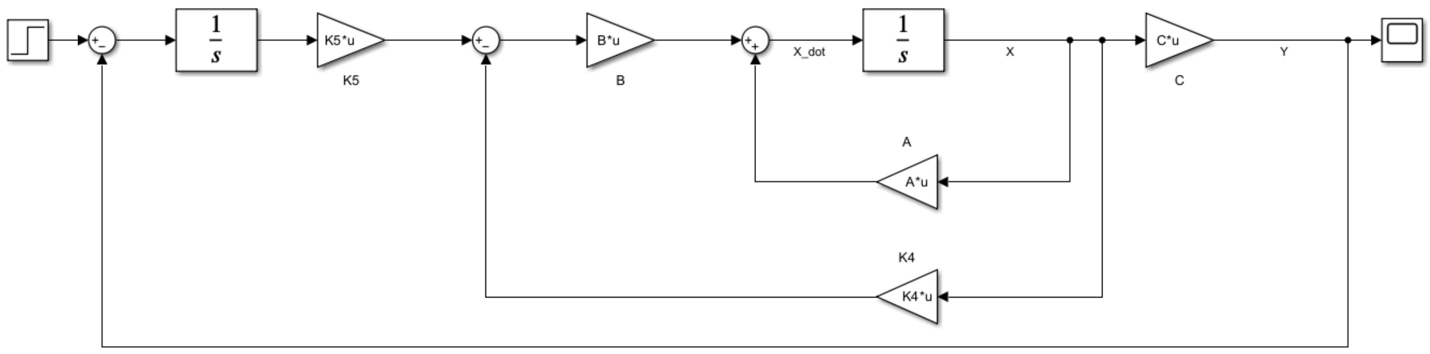
```
K3 = 1x4
     7.5000   44.5000   61.4000  -19.2000
```

Consequently, K4 and K5 are defined as follows.

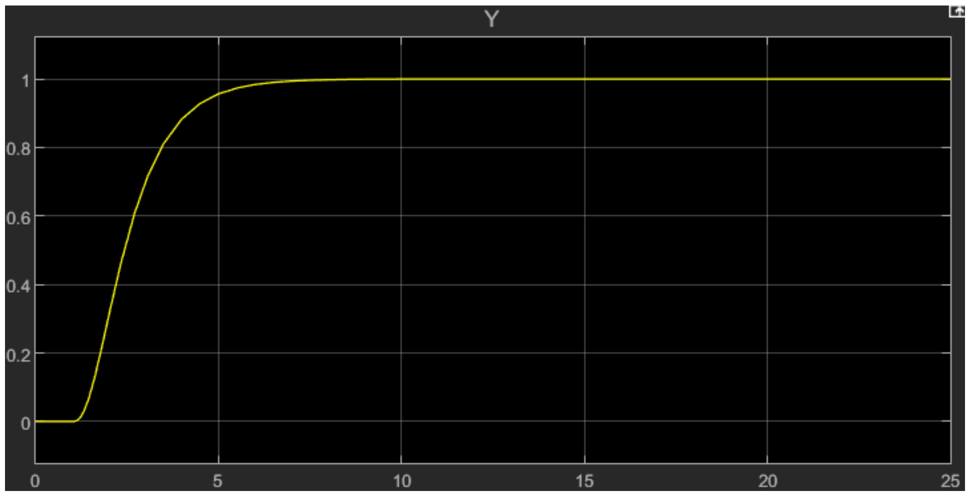


```
K4 = [7.5000 44.5000 61.4000];
K5 = 19.2000;
```

With the obtained values, we simulate the integral observer block diagram in the Simulink environment.



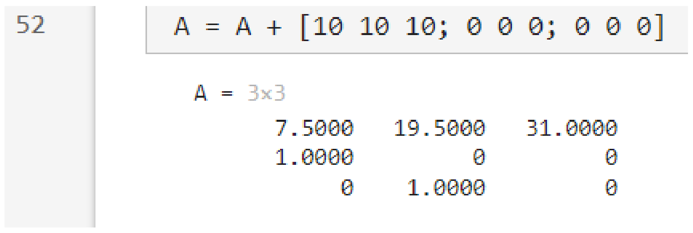
After performing the above simulation, we obtain the output shown below, indicating that the integral observer is functioning correctly and the system is accurately tracking the step input.



Q10:

Tracking Performance:
In terms of performance, it can be said that both observers have similar performance and are suitable; however, the static observer is slightly faster.

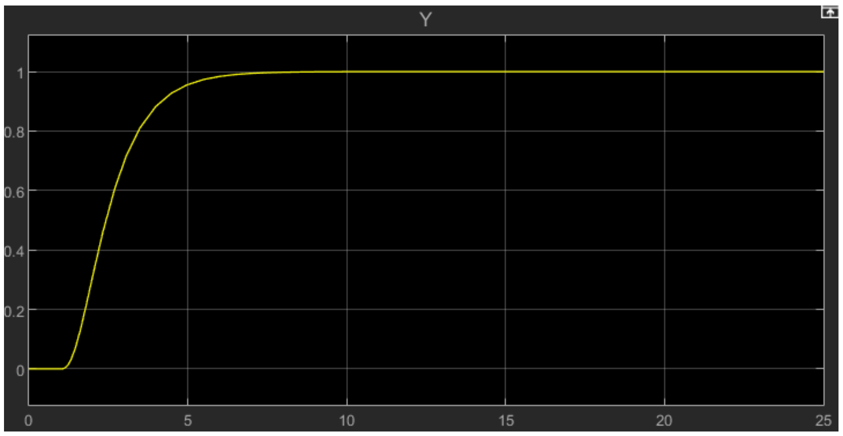
Robustness in the presence of parameter changes in the model:
To test this, we add 10 units to the first row of matrix AAA and compare the outputs of the observers.



```
A = A + [10 10 10; 0 0 0; 0 0 0]
```

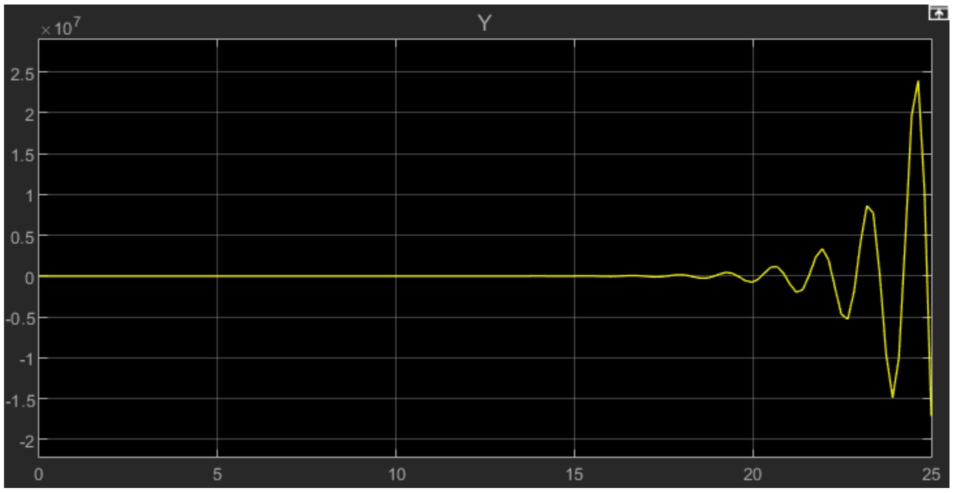
```
A = 3x3
       7.5000   19.5000   31.0000
       1.0000         0         0
            0    1.0000         0
```

For the integral observer, the output becomes as follows:



It is clear that changes in A do not significantly affect this observer.

However, the output of the static observer is as follows:

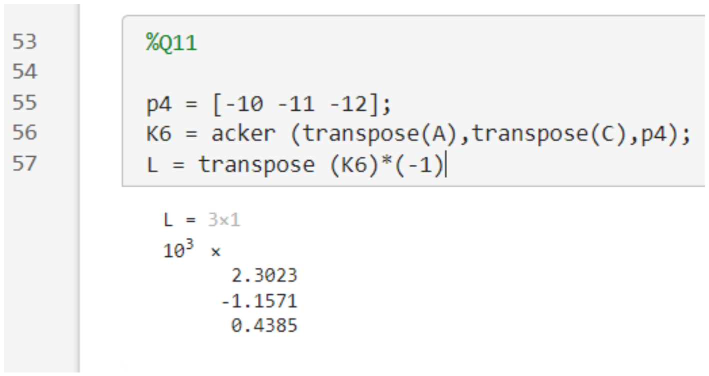


It can be seen that this observer does not perform well with changes in system parameters.

Closed-loop system performance when subjected to constant disturbances:
We know that the static observer is designed for steady-state tracking, and changes in the system and disturbances affect it. However, this is not the case for the integral observer.

Q11:

In the first step, we choose the poles to be closer. For the design of the observer, the poles should be at least 4 to 5 times farther than normal, and if feedback is included, they should be significantly farther or faster. This is necessary so that the observer can react more quickly than the feedback, allowing for proper performance.

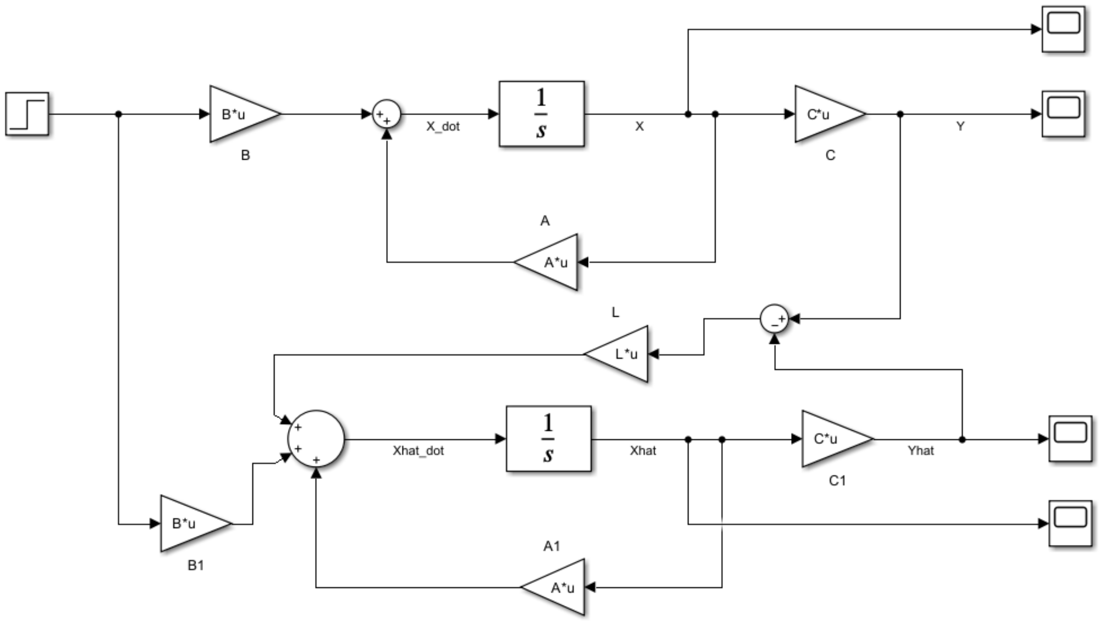


```
%Q11

p4 = [-10 -11 -12];
K6 = acker (transpose(A),transpose(C),p4);
L = transpose (K6)*(-1)
```

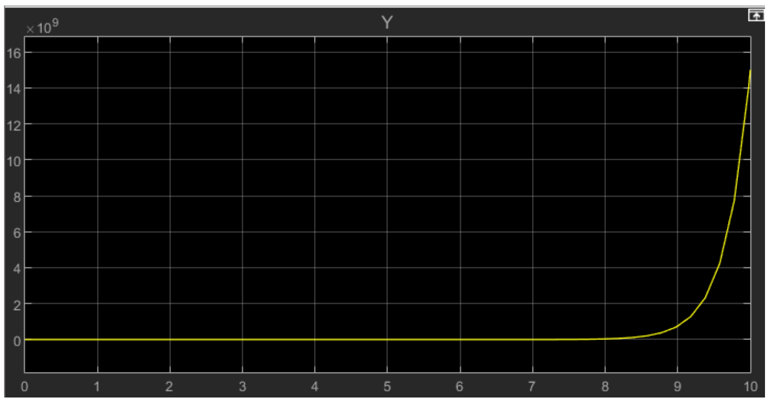
```
L = 3×1
10^3 ×
    2.3023
   -1.1571
    0.4385
```

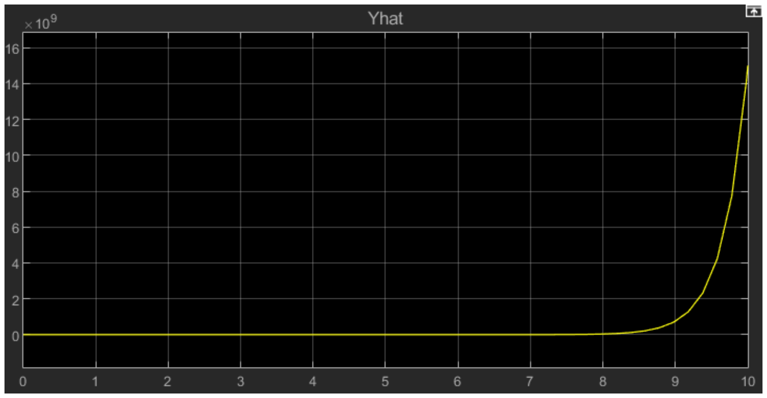
At this stage, we also calculate the matrix L. Then, using the formula we learned in class, we perform the observer simulation in the Simulink environment

$$\boxed{\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x})}$$



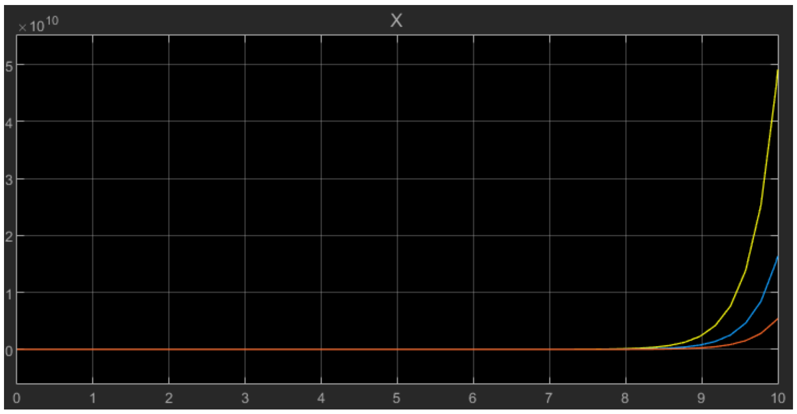
As expected, the outputs y and yhat should be similar.

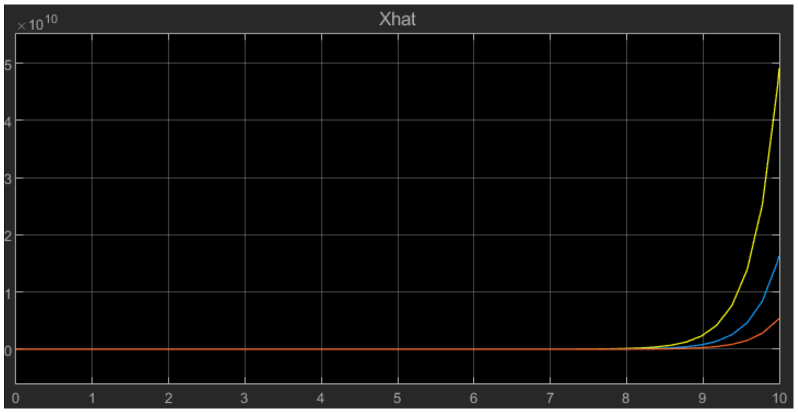




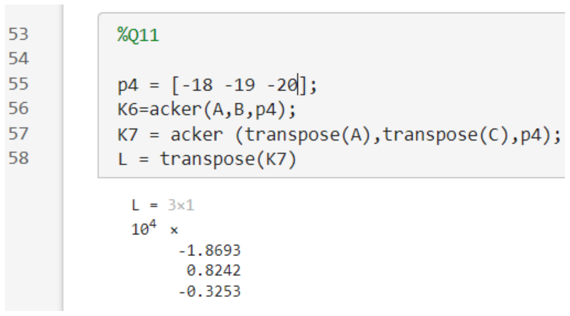
It can be observed that both are indeed similar.

The state variables, denoted as x and xhat, are presented below:





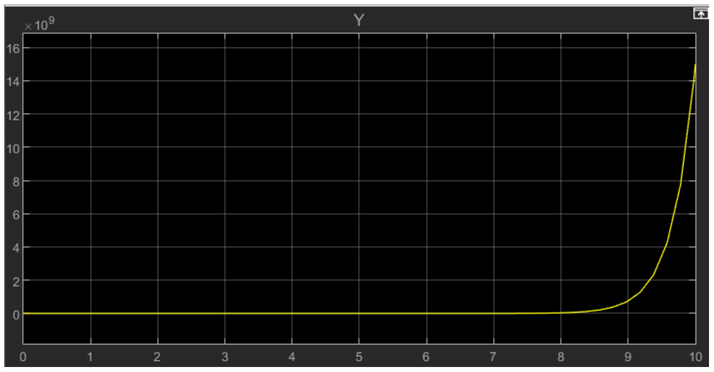
Now, we will perform the same steps for poles that are farther away and compare the results.

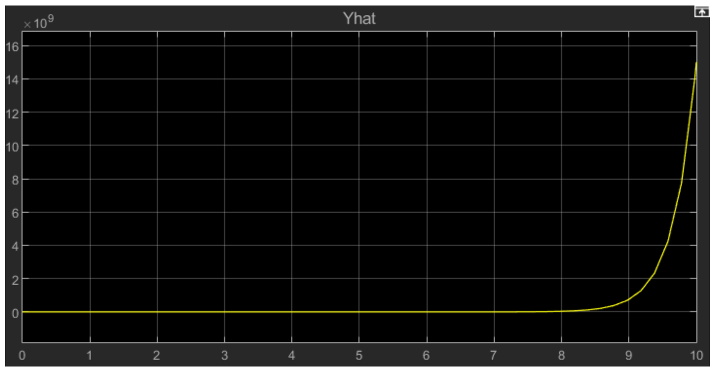


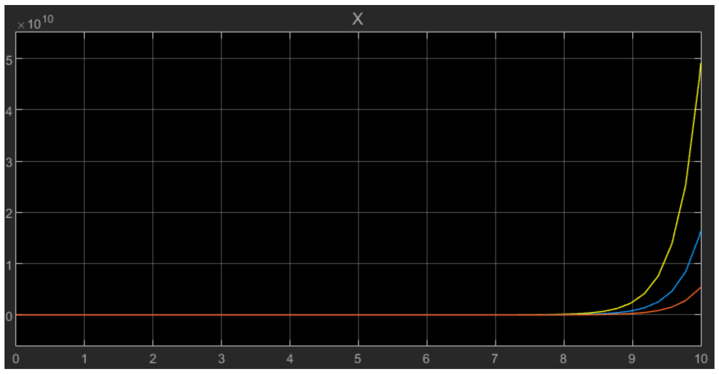
```
%Q11

p4 = [-18 -19 -20];
K6=acker(A,B,p4);
K7 = acker (transpose(A),transpose(C),p4);
L = transpose(K7)
```

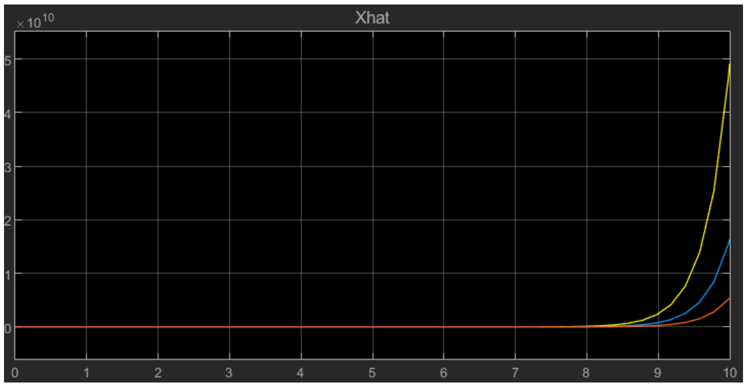
```
L = 3×1
10^4 ×
   -1.8693
    0.8242
   -0.3253
```

After conducting the simulation, we will have:





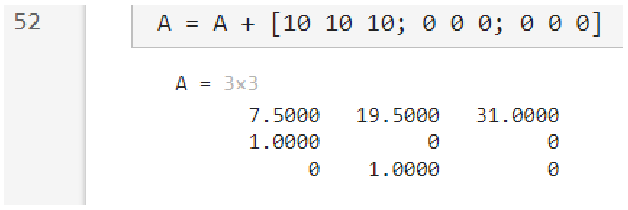




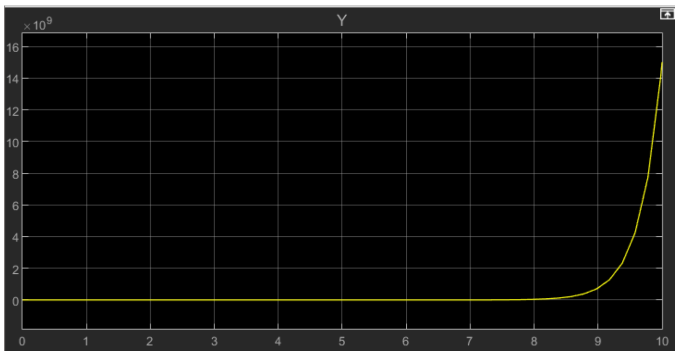
The estimator estimates with greater speed.
The outputs remain unchanged. The matrix L changes. In general, as the poles move farther away, the speed increases, and the error decreases.

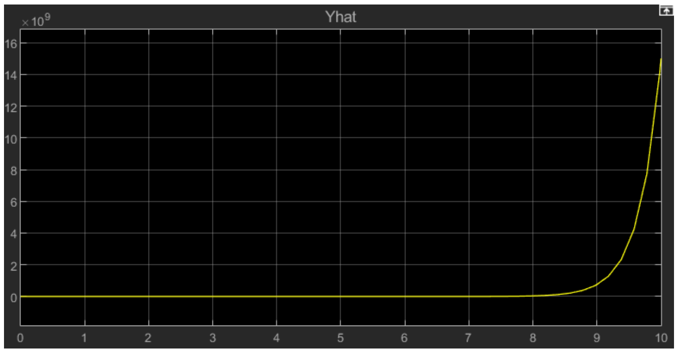
Q12:

To test this, we add 10 units to the first row of matrix A and compare the outputs of the estimators.



```
A = A + [10 10 10; 0 0 0; 0 0 0]

A = 3x3
      7.5000   19.5000   31.0000
      1.0000         0         0
           0    1.0000         0
```

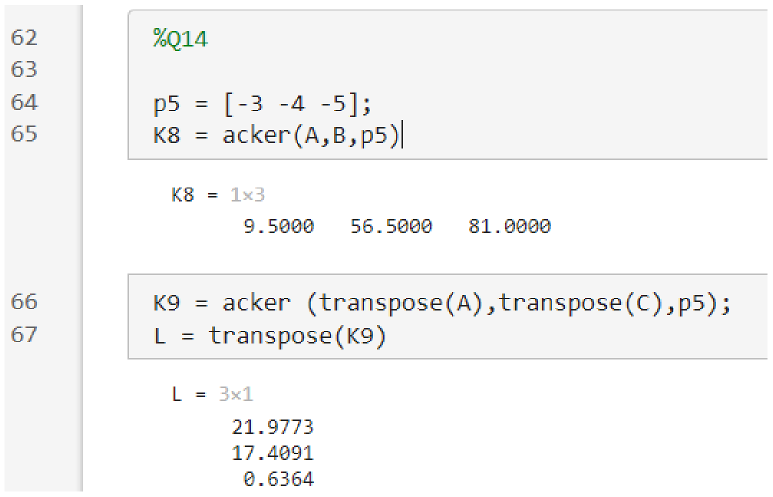




It can be seen that the outputs remain unchanged. The system is resilient to changes in parameters.

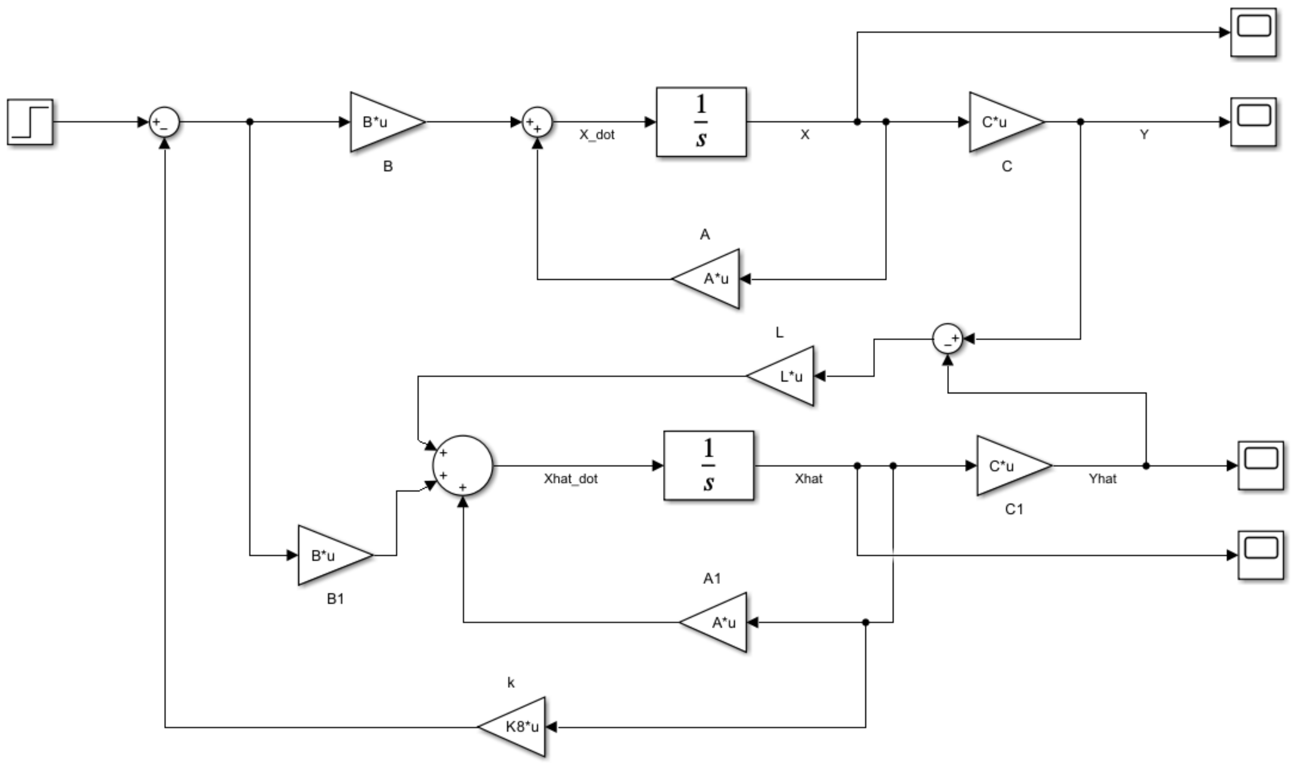
Q13:

First, we calculate the values of K and L as shown below and import the values into Simulink.

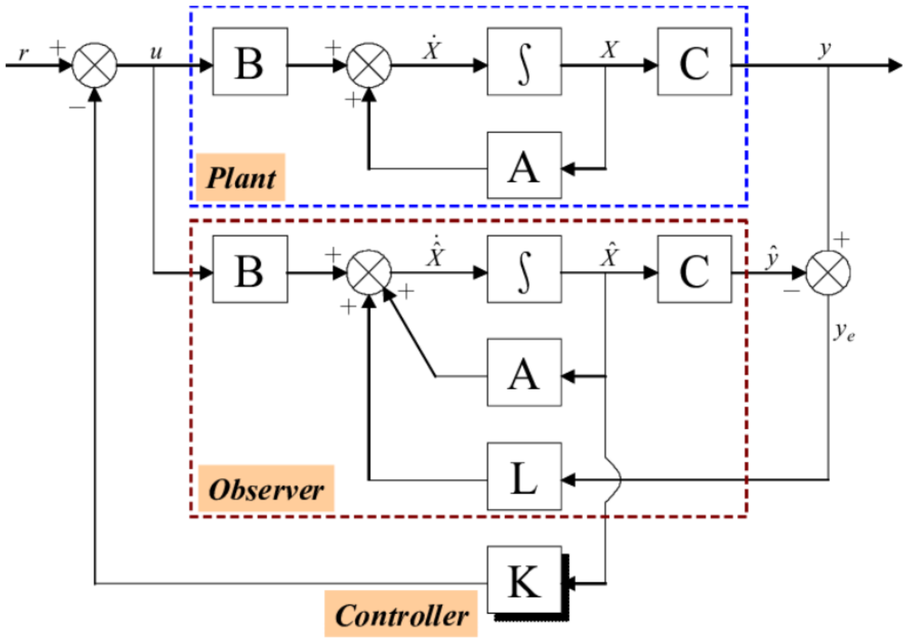


```
%Q14

p5 = [-3 -4 -5];
K8 = acker(A,B,p5)

  K8 = 1×3
         9.5000    56.5000    81.0000

K9 = acker (transpose(A),transpose(C),p5);
L = transpose(K9)

  L = 3×1
       21.9773
       17.4091
        0.6364
```

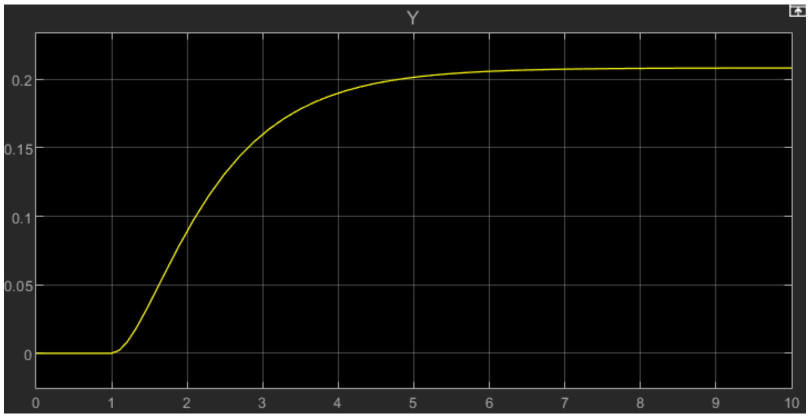
The simulation will be as follows:

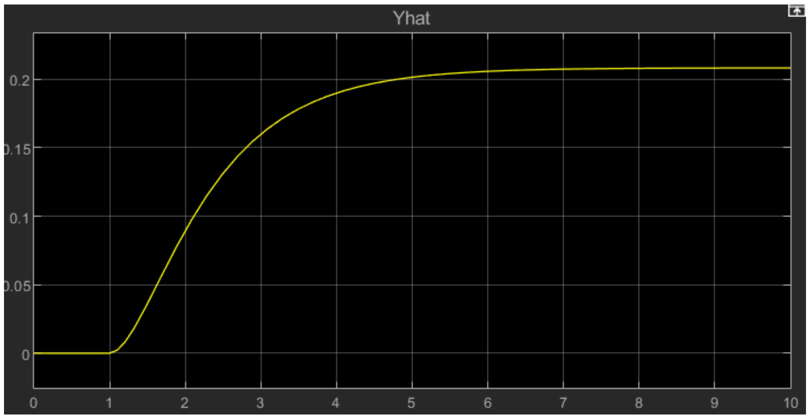


We have used the following block diagram, which we were already familiar with, in the above simulation.



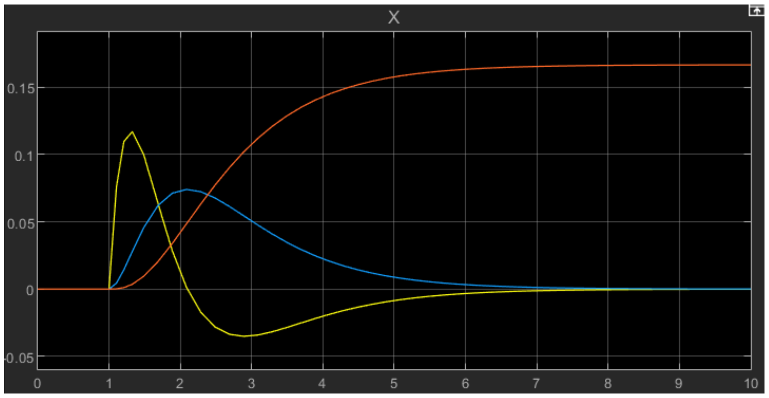
The outputs for y and yhat are shown below.

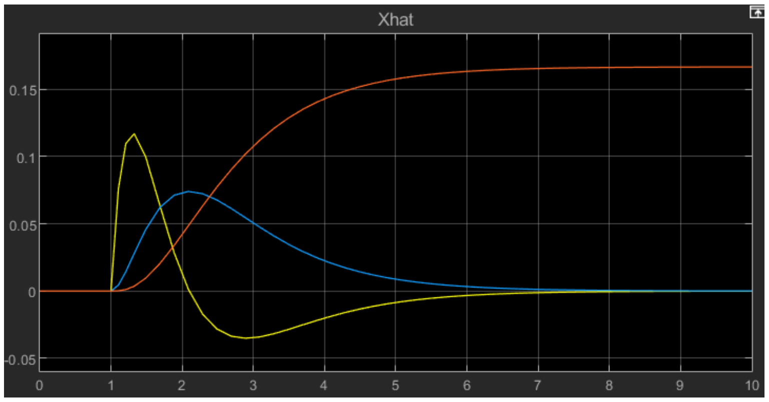




It can be seen that the estimator is working well, with both outputs being similar. Additionally, tracking brings the system from an unstable state to stability.

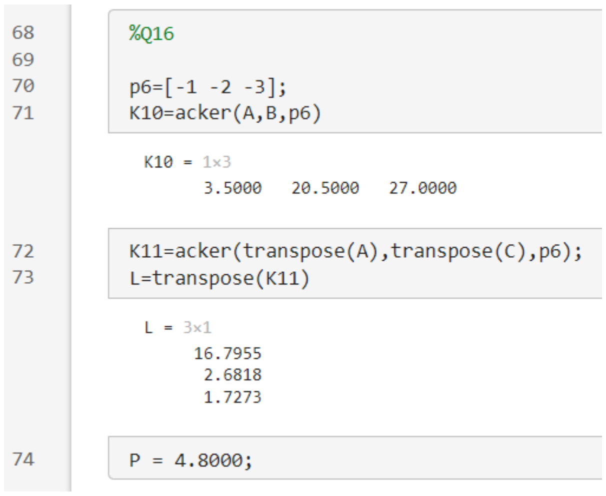
The state variables of the system can also be displayed as follows:





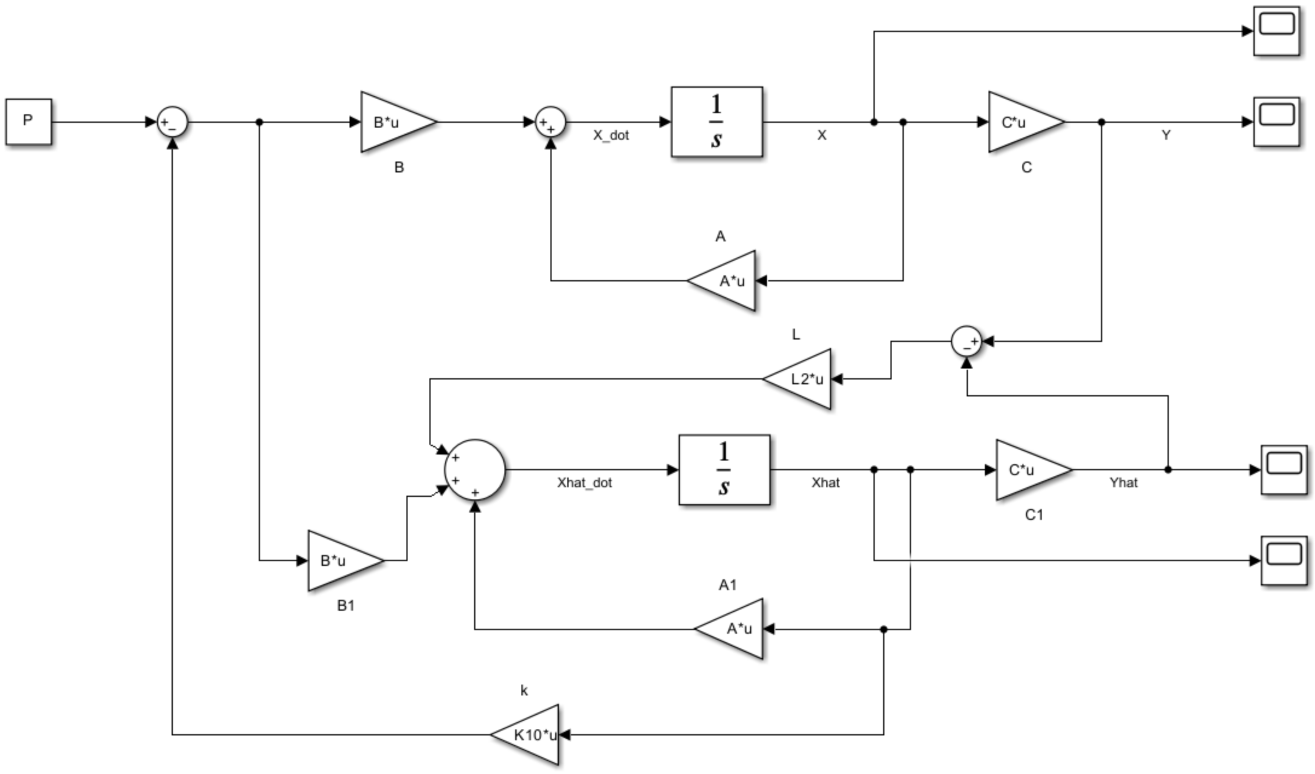
Q14:

First, we calculate K and L as shown below. We know that P in the static observer designed in question 8 was equal to 4.8, so we use the same value for P here. The desired poles are set at -1, -2, and -3.

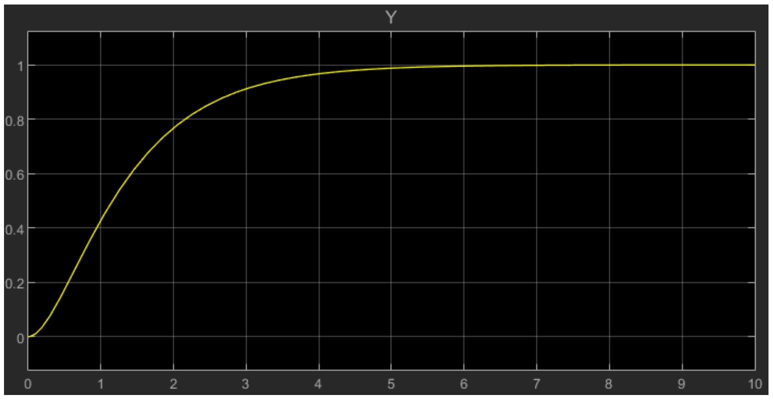


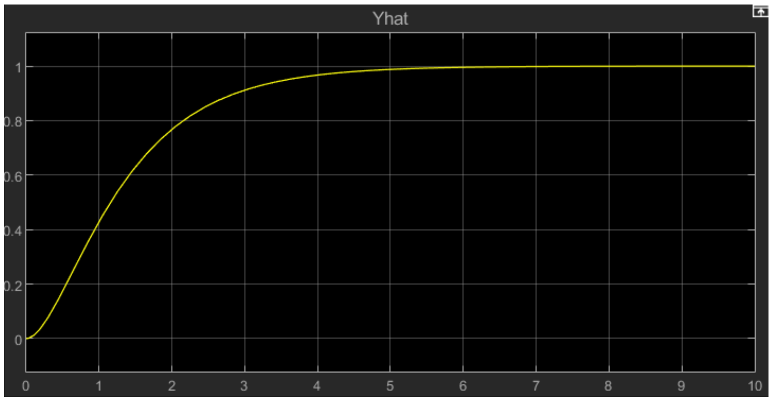
```
%Q16

p6=[-1 -2 -3];
K10=acker(A,B,p6)

  K10 = 1×3
         3.5000    20.5000    27.0000

K11=acker(transpose(A),transpose(C),p6);
L=transpose(K11)

  L = 3×1
        16.7955
         2.6818
         1.7273

P = 4.8000;
```

The simulation for this question is as follows:

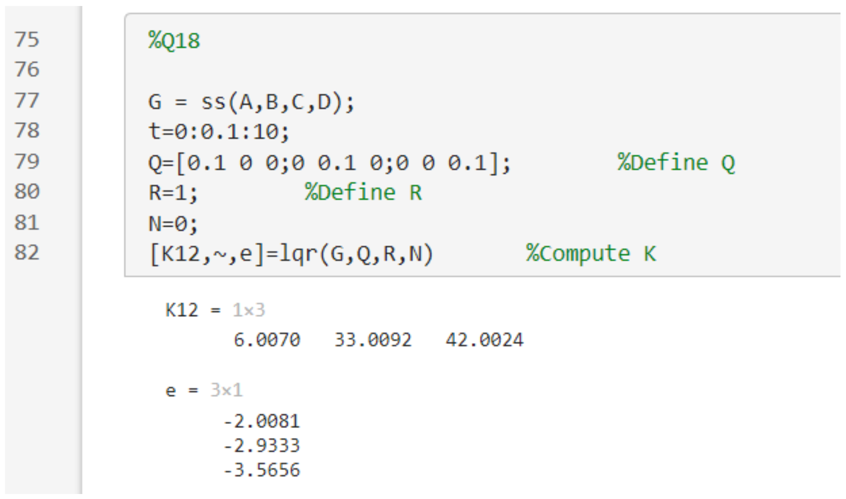


The outputs of this system are shown below. It can be seen that both the estimator is functioning correctly and the observer we designed is effectively tracking the unit step input.





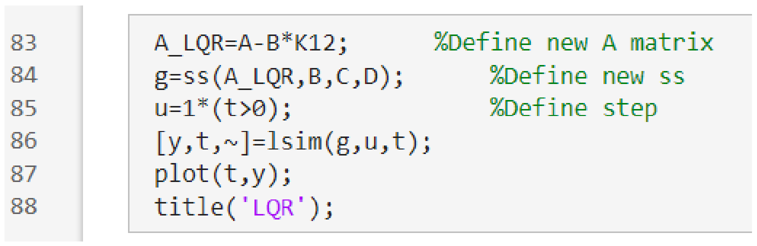
Q15:



```
%Q18

G = ss(A,B,C,D);
t=0:0.1:10;
Q=[0.1 0 0;0 0.1 0;0 0 0.1];        %Define Q
R=1;          %Define R
N=0;
[K12,~,e]=lqr(G,Q,R,N)        %Compute K
```

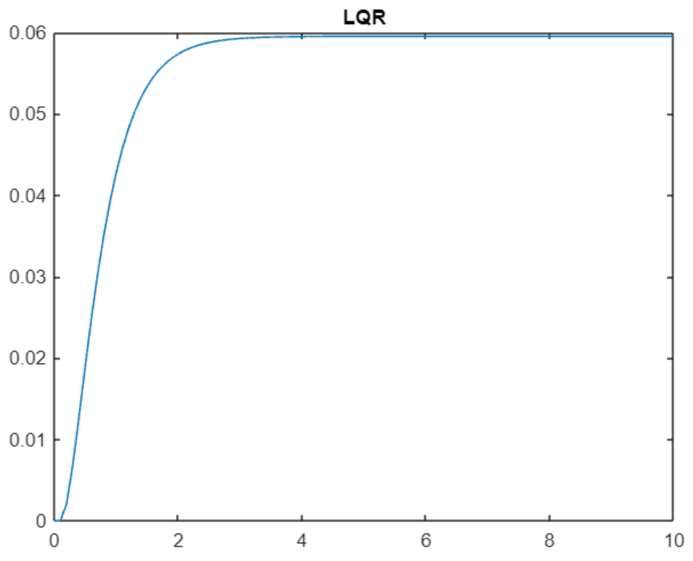
```
K12 = 1×3
      6.0070    33.0092    42.0024

e = 3×1
     -2.0081
     -2.9333
     -3.5656
```

First, we define Q and R and calculate K based on these values.
Next, we define a new matrix A and, using this new A, redefine the state-space representation, applying a step input to observe the output.

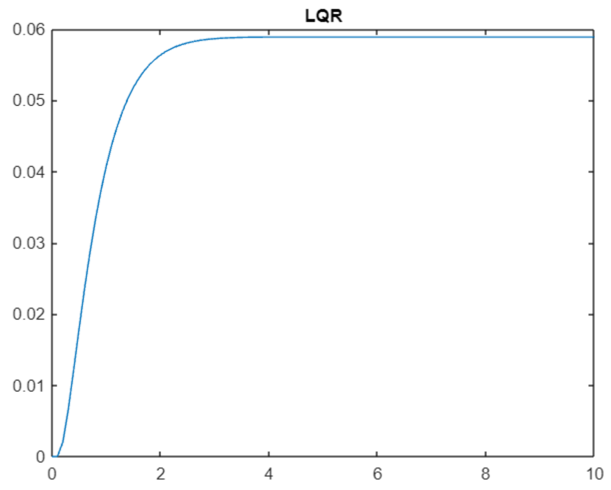


```
A_LQR=A-B*K12;      %Define new A matrix
g=ss(A_LQR,B,C,D);      %Define new ss
u=1*(t>0);              %Define step
[y,t,~]=lsim(g,u,t);
plot(t,y);
title('LQR');
```

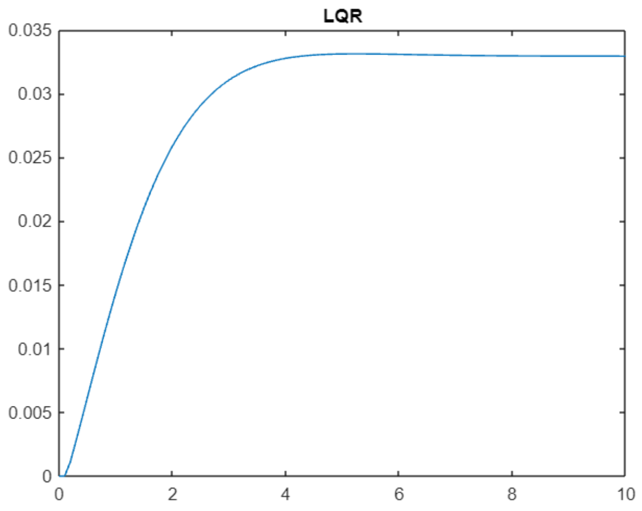
The LQR output is as follows:



We see that the system responds more quickly.
In the next step, we keep Q constant and examine the effect of changing R on the output.
Setting R to 0.01:

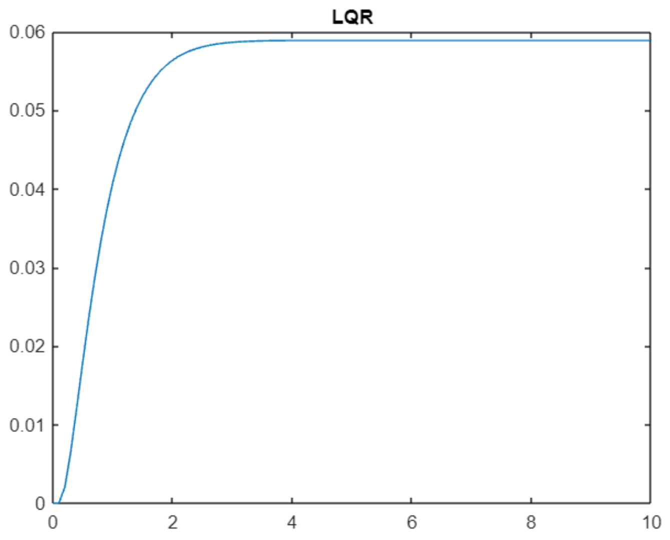


It is observed that changes in R have minimal impact on the output, except when R is very small (e.g., 0.0001), in which case the output is as follows:

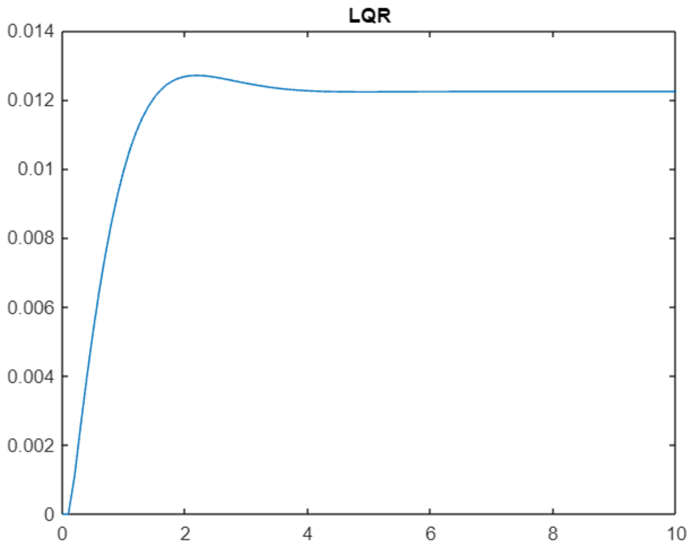


We see that the output appears to be scaled down by a gain of less than one, and there is a slight overshoot in this case.

Now, we set R back to 1 and vary Q.
We increase Q by a factor of 100 and observe the output:



Increasing Q further (1000 * I)



We see that as Q grows excessively large, the system starts to exhibit overshoot.
In general, increasing Q also results in a slightly faster system response.