Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

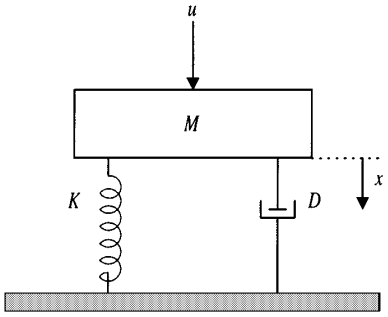
LAB 1: Introduction to MATLAB Toolbox SIMSCAPE along with the modeling and analysis of Translational as well as Rotational Mechanical Systems.

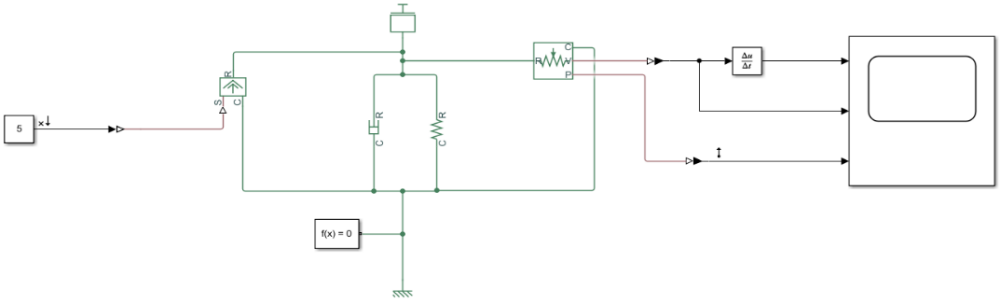
Aim and Objectives:

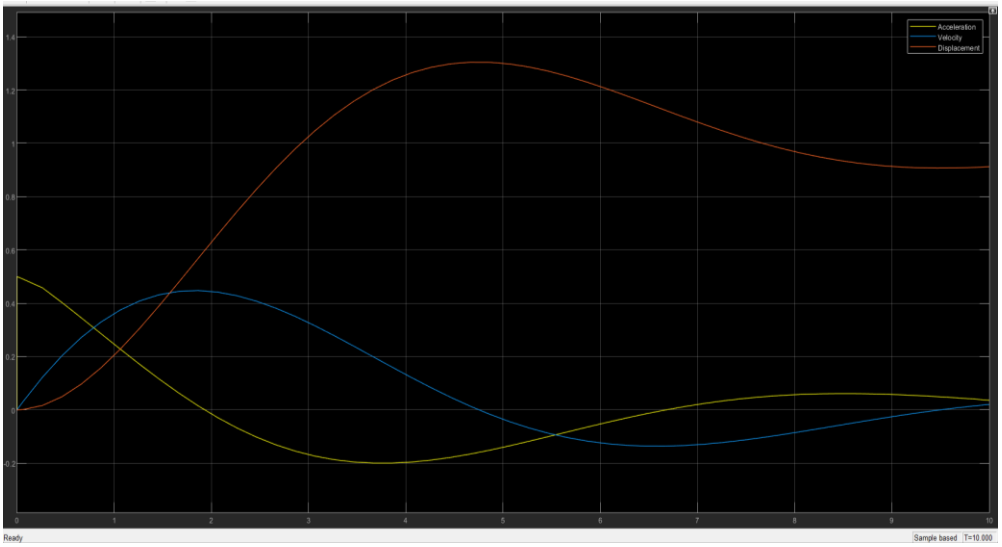
The fundamental aim of this lab is to explore the different libraries of SIMSCAPE that would help us to model and analyze the different categories of dynamic systems. Furthermore, this lab is primarily dedicated to two categories of mechanical systems, and certain tangible objectives are enlightened below:

1. To explore the work environment and built-in libraries of MATLAB Toolbox SIMSCAPE.
2. To model and analyze the response of the mechanical translational as well as rotational systems.
3. To obtain the transfer function from developed SIMSCAPE models.
4. To cross-validate the outcomes of SIMSCAPE Models with the results obtained during modeling and simulation labs (ODE45 and SIMULINK).

Example#01 (Simple Mass Spring Damper System)

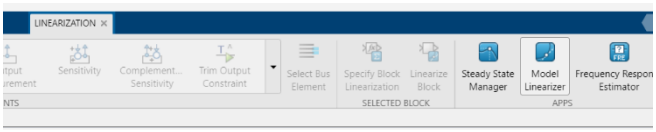




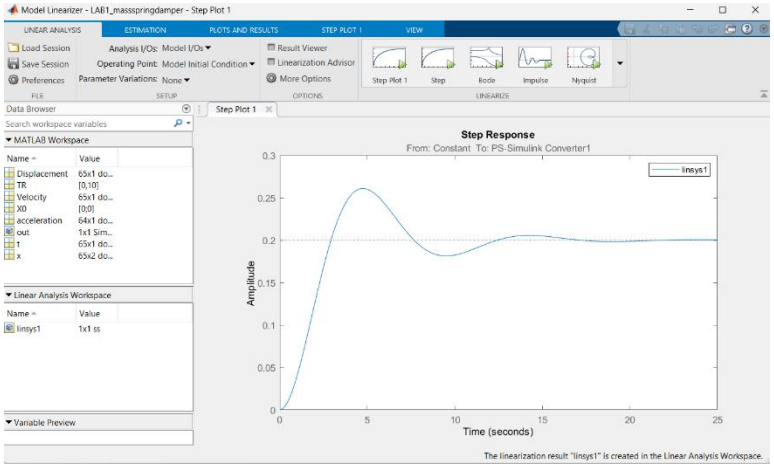


The response of simple mass spring damper system has been analysed by applying force of 5 N under the following parametric values: M=10 Kg, D=5 Ns/m, and K=5 N/m. Under these conditions, the maximum value(s) of (a) Displacement is 1.3 m, (b) Velocity is 0.5 m/s (c) 0.1 m/s^2.

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan



```
From input "Constant" to output "PS-Simulink Converter1":
        0.1
-----------------
s^2 + 0.5 s + 0.5
```

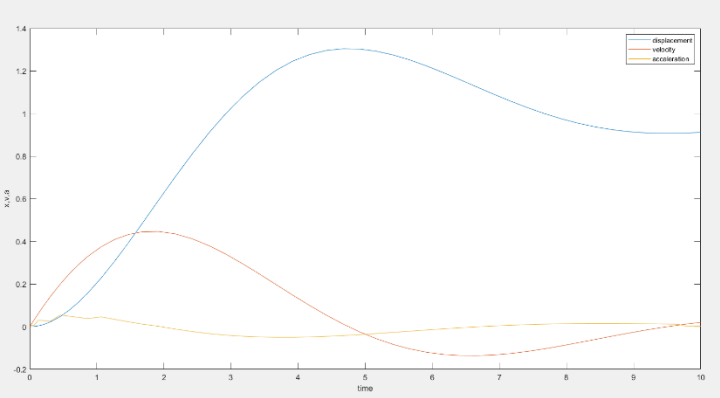


To find the **transfer function** from any developed SIMSCAPE model, select the linear analysis points (In terms of input and output of the transfer function), and apply Control Linearization approach (Model Linearizer: A tool available in MATLAB Updated Versions) as shown in the sequence of Figures.

Cross-Validation through Modeling & Simulation Labs (Using M-file and Simulink)

M-file

```
X0=[0;0];%initial condiitons are zero
TR=[0 10];%time response RANGE
%t=0:0.1:50;
[t,x]=ode45(@func1,TR,X0);
Displacement=x(:,1)
Velocity=x(:,2)
plot(t, Displacement)
hold on
plot(t,Velocity)
acceleration = diff(Velocity);
hold on
plot(t,[0;acceleration])
ylabel('x,v,a')
xlabel('time')
legend("displacement","velocity","acceleration");

%state variables give the following function
function dx = func1(t,x)
   M=10;B=5;K=5;F=5;
   dx(1)=x(2)%for x dot
   dx(2)=(F-B*x(2)-K*x(1))/M    %for x dot dot
   dx = dx';
end
```



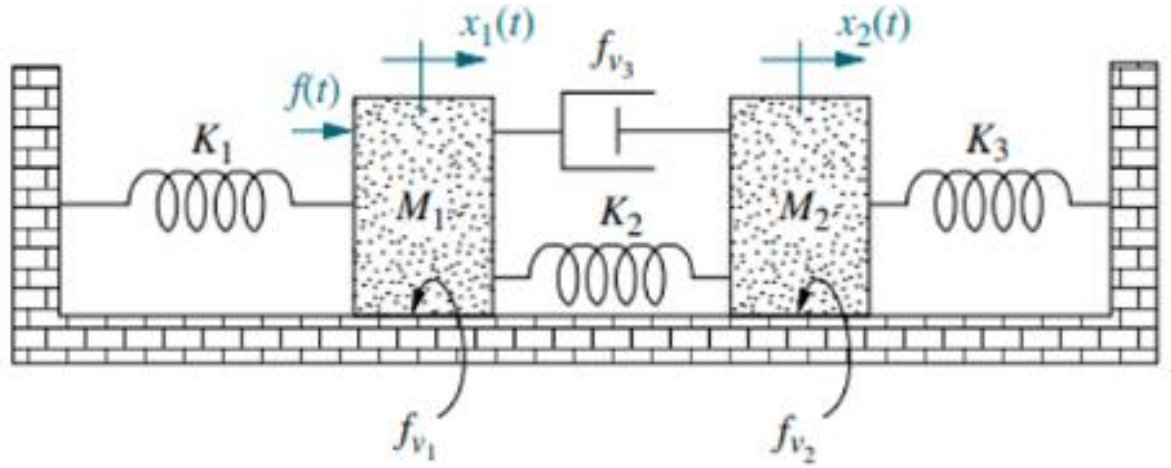
Under the same parametric conditions (M=10 Kg, D=5 Ns/m, and K=5N/m),the maximum value(s) of (a) Displacement is 1.3 m, (b) Velocity is 0.5 m/s , and (c) Acceleration is 0.1 m/s^2. **Explore the shared Simulink file!!!**

Passive System has been **Cross verified**!!!

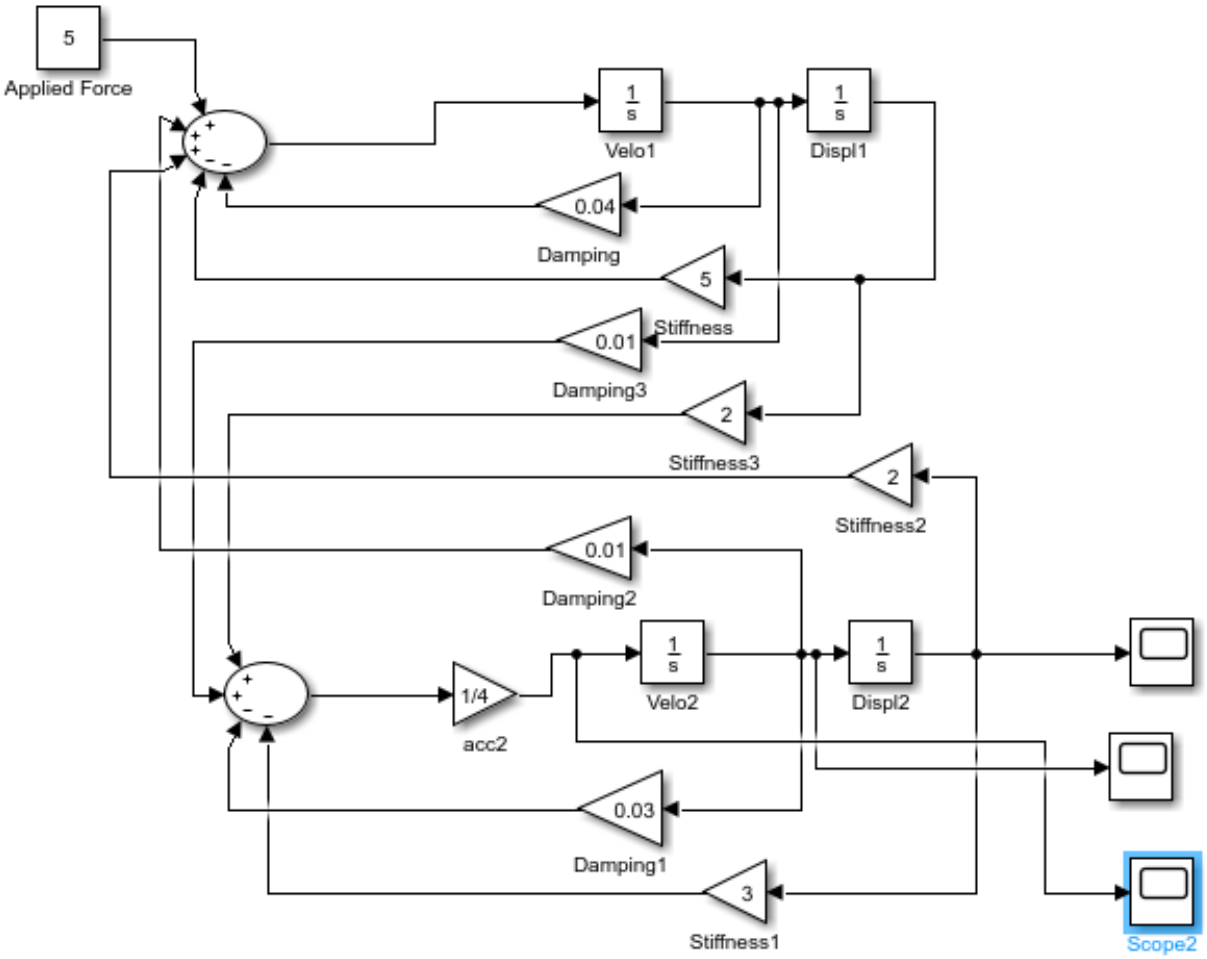
General Remarks after 1st model on SIMSCAPE:

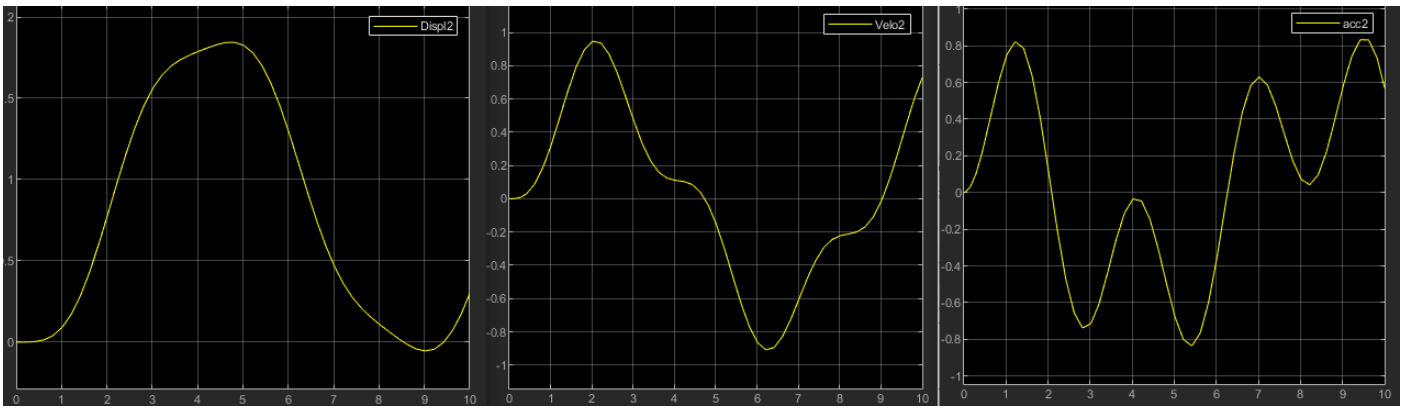
The ideal mass spring damper system has been analyzed under various parametric conditions and subsequently, the results are further cross-validated by considering the **existing methodologies from Modeling &Simulation labs** (M-File and SIMULINK). Based on the tangible findings, it is concluded that **SIMSCAPE will be utilized in the first three to four labs of Control System-1** to model, analyze, and cross-validate all the developed/studied examples of **Modeling and Simulation**. Apart from analyzing the response of the dynamic system(s), students shall be exploring the **novel methodology** of finding **(a) Transfer Function and (b) State Space** from the developed model of SIMSCAPE.

Example#02 (Dual Mass Spring(s) Damper System)



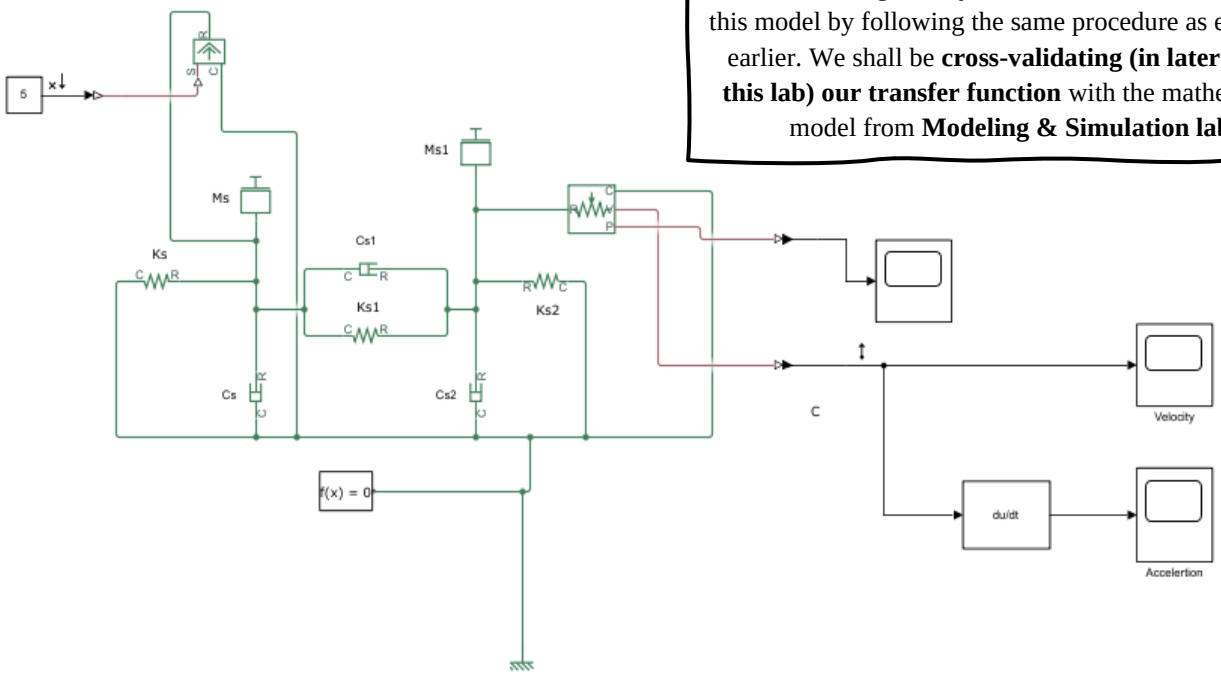
Simulink Results:

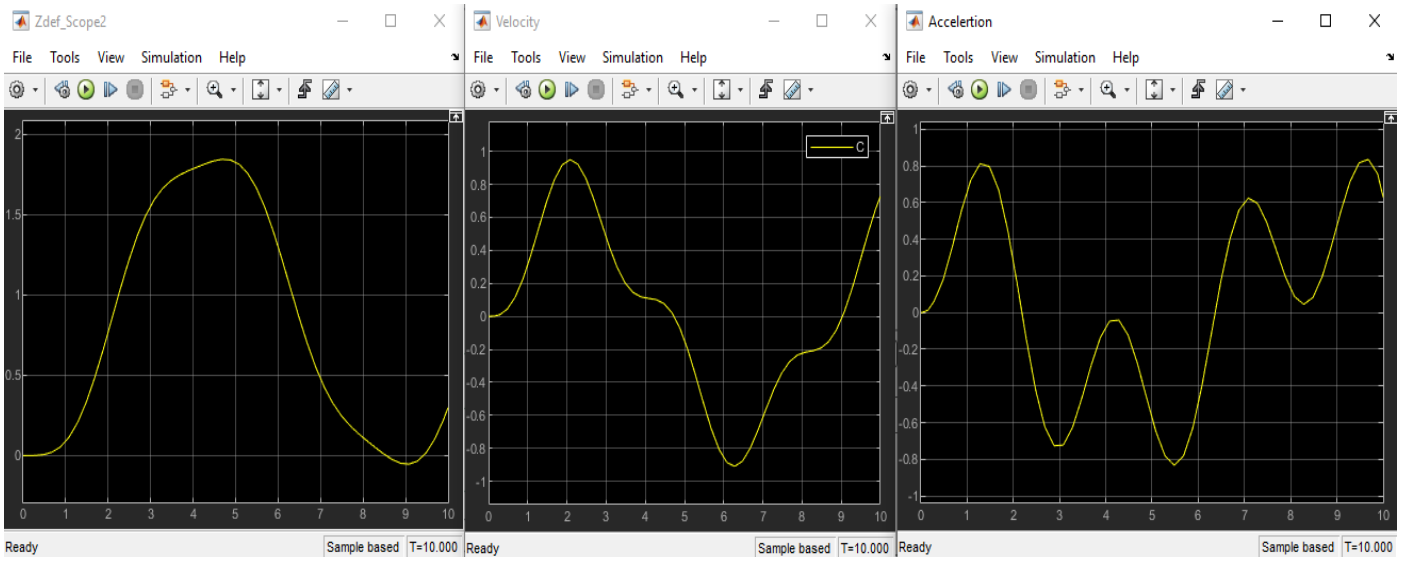




SIMSCAPE Results:

Students are urged to synthesize the transfer function of this model by following the same procedure as explained earlier. We shall be **cross-validating (in later half of this lab) our transfer function** with the mathematical model from **Modeling & Simulation lab**.



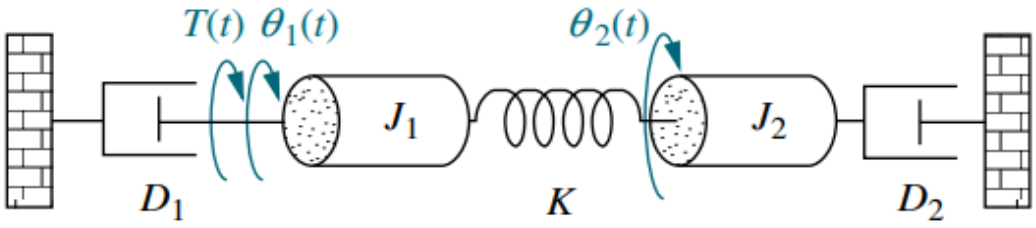


Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

```
                                0.05 s + 0.05
PS-Simulink Converter3:  ------------------------------------
                         s^4 + 2 s^3 + 2.75 s^2 + 1.5 s + 0.75
```

Transfer Function synthesized from SIMSCAPE Moddel.

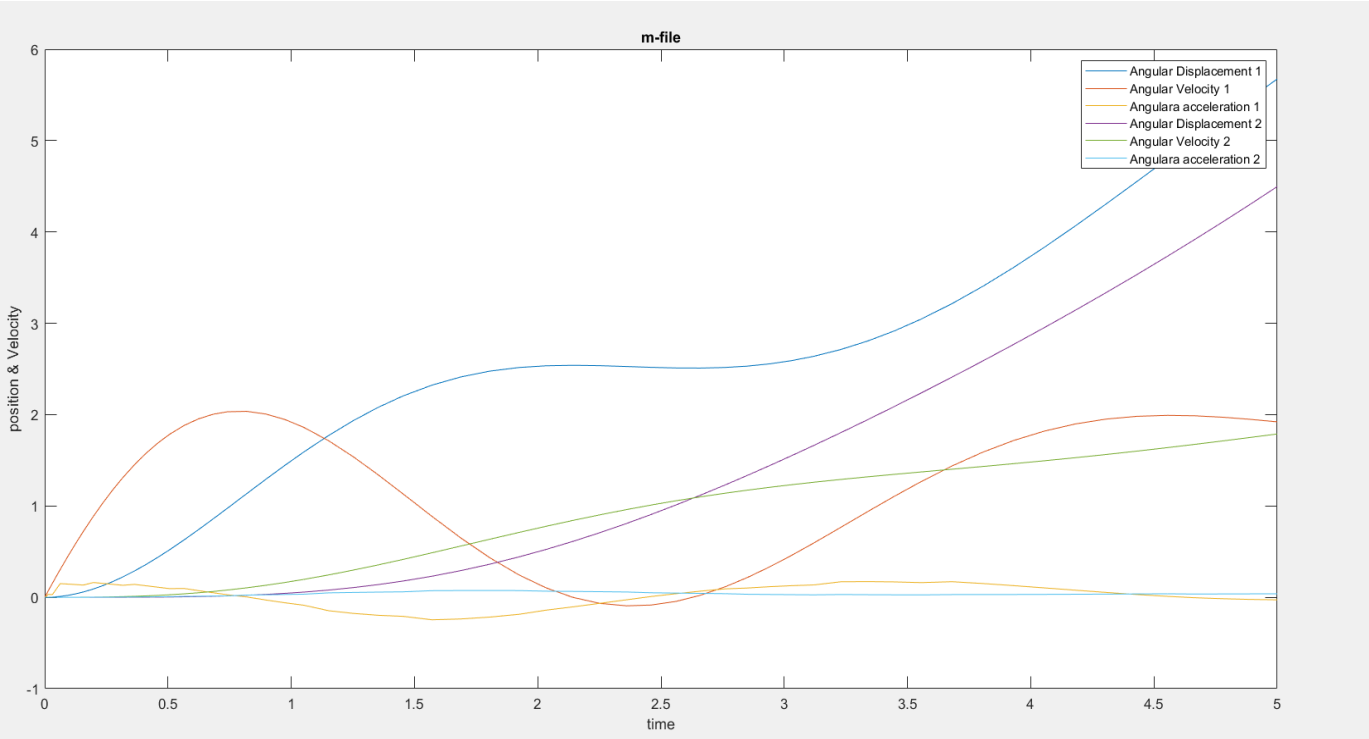
Example#03 (Dual Rotational System)



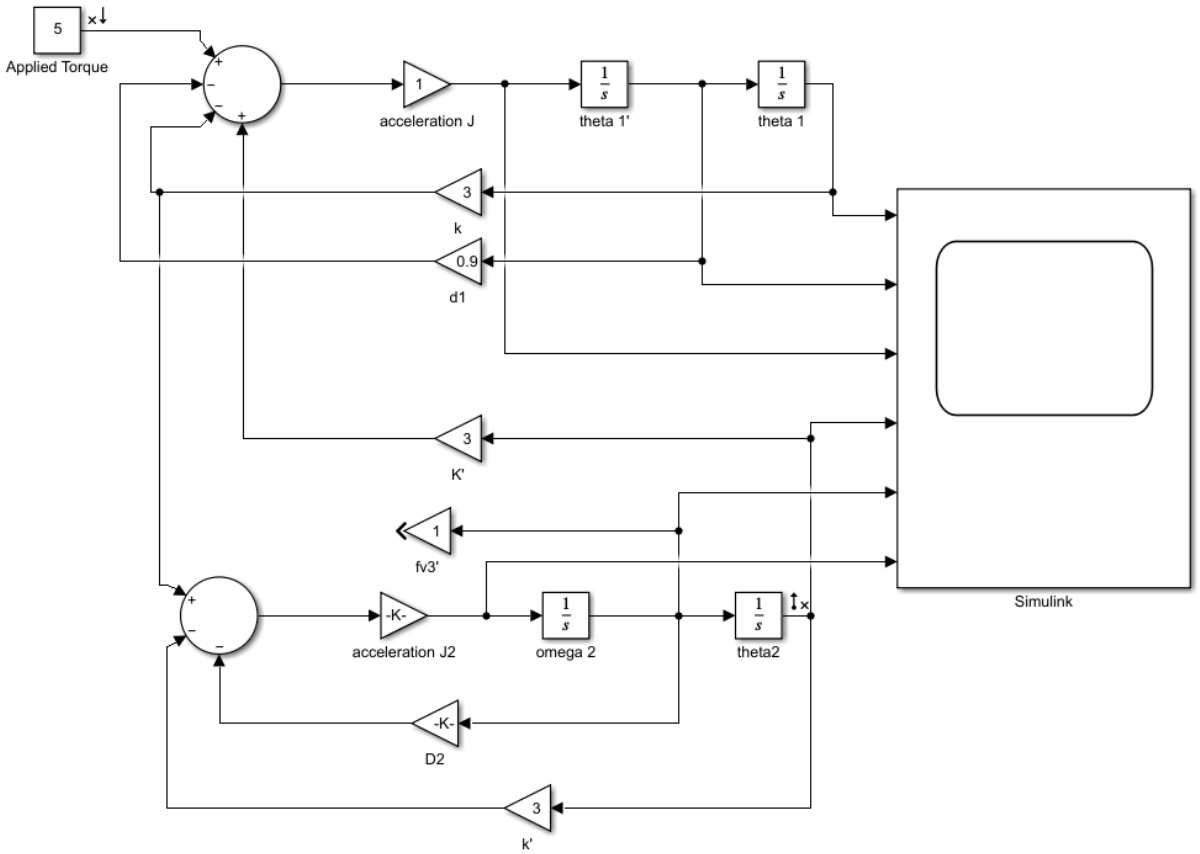
In this example, all the studied concepts of rotational systems will be implied to analyze the response of the dynamic system under the specific conditions. Initially, M-file will be executed to measure and visulaize the variataions in the Angular displacements, velocities, and accelerations of both inertial elements. Afterward, Simulink model will be executed to justify the developed mathematical model of the system. Finally, the most important task for this specific system will be again the modelling, analysis, and finding transfer function using **SIMSCAPE ENVIRONMENT**.

M-file

```
% For constant tourque 5Nm
clc
clear
TR = [0 5]; % time RANGE
X0 = [0;0;0;0];%initial conditions
[t,z] = ode45(@func1, TR, X0);%calling thr ide solver to solve by
function
%storing given array as vectors
theta1 = z(:, 1);
AngVel1 = z(:, 2);
theta2 = z(:, 3);
AngVel2 = z(:, 4);

%plotting the angular displacements and vel
acc1 = diff(AngVel1);
acc2= diff(AngVel2);
plot(t,theta1,t,AngVel1,t,[0;acc1],t,theta2,t,AngVel2,t,[0;acc2]);
xlabel('time')
legend('Angular Displacement 1','Angular Velocity 1','Angulara
acceleration 1','Angular Displacement 2','Angular Velocity
2','Angulara acceleration 2')
ylabel('position & Velocity')
title("m-file")
%function containing the differentialequations
function dx = func1(t, x)
% Values of Coefficients
J1=1; J2=10; D1=0.9; D2=0.02; k=3,T=5;

% State Equations
dx(1) = x(2);
dx(3) = x(4);
dx(2) = (T-D1*x(2)-k*x(1)+k*x(3))/J1;
dx(4) = (-k*x(3)-D2*x(4)+k*x(1))/J2;
dx = dx';
end
```

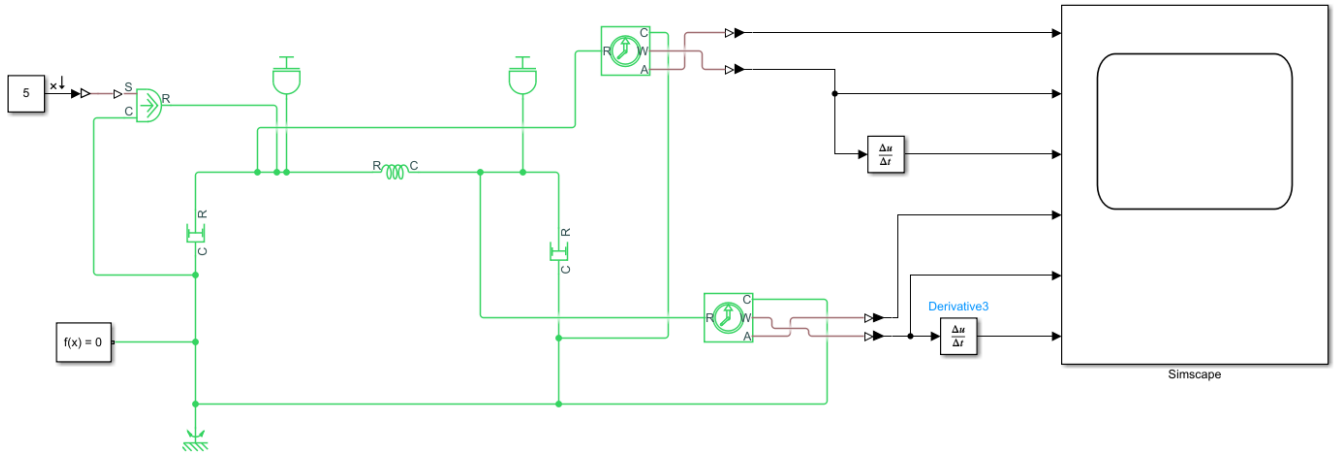


SIMULINK Model

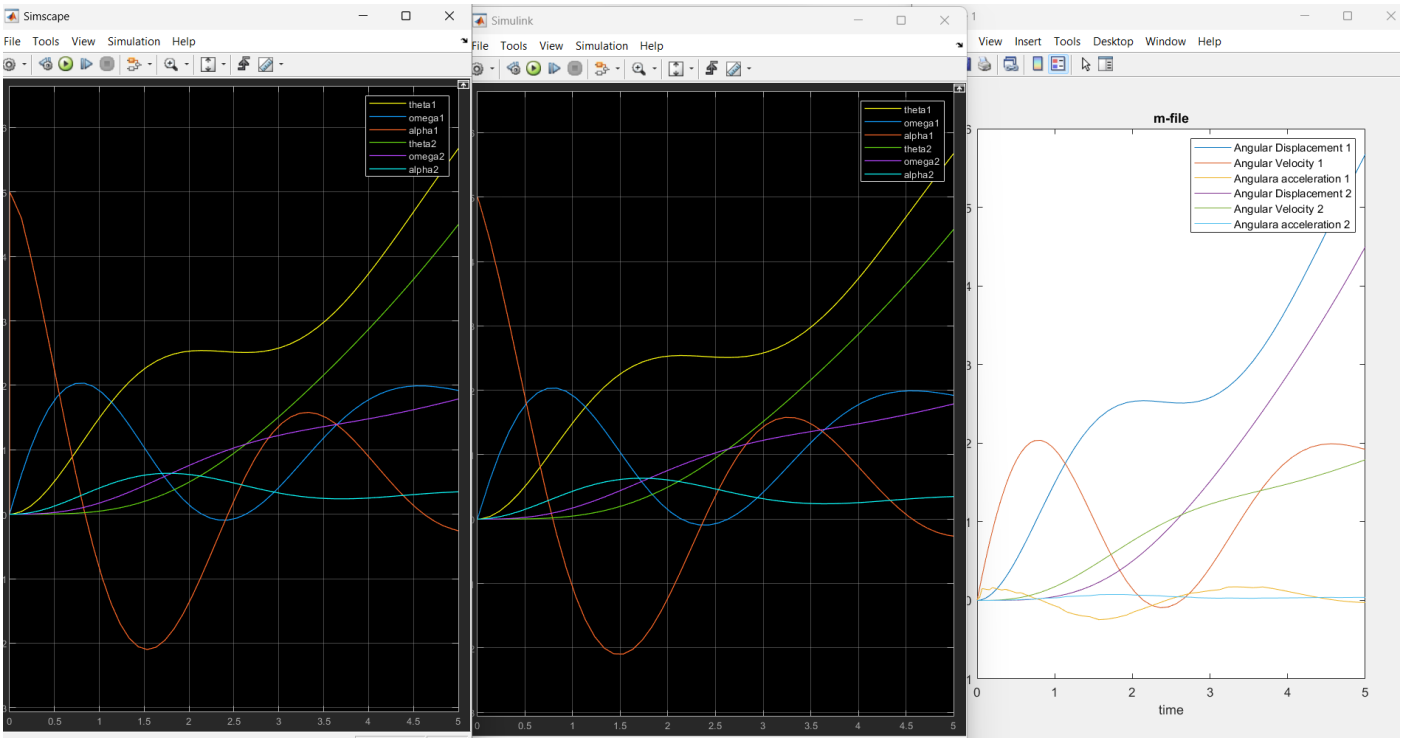


Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

SIMSCAPE MODEL

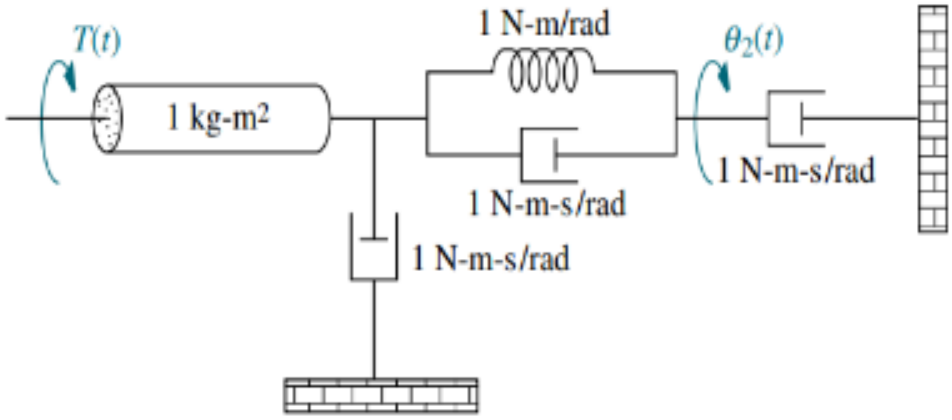


Comparison between three different approaches



Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

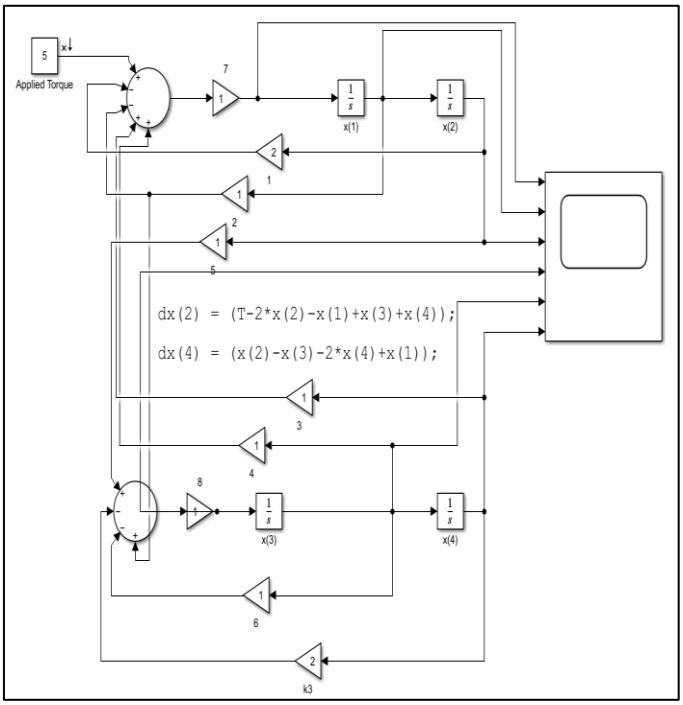
Example#04 (Rotational System With series and Parallel combination)



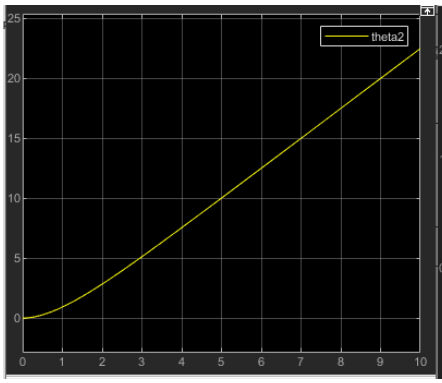
M-File

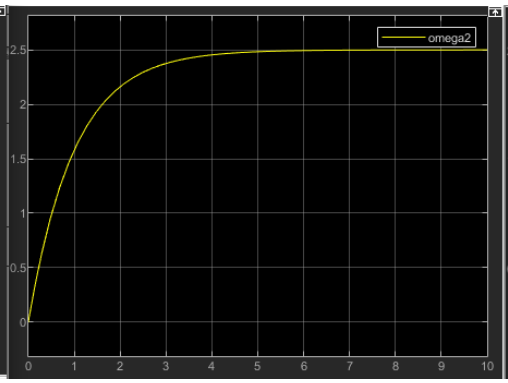
```
clc
clear
TR = [0 5]; % time RANGE
X0 = [0;0;0;0];%initial conditions
[t,z] = ode45(@func1, TR, X0);%calling thr ide solver to solve by
function
%storing given array as vectors
theta1 = z(:, 1);
AngVel1 = z(:, 2);
theta2 = z(:, 3);
AngVel2 = z(:, 4);
%plotting the angular displacements and velocities
acc1 = diff(AngVel1);
acc2= diff(AngVel2);
plot(t,theta1,t,AngVel1,t,[0;acc1],t,theta2,t,AngVel2,t,[0;acc2]);
xlabel('time')
legend('Angular Displacement 1','Angular Velocity 1','Angulara
acceleration 1','Angular Displacement 2','Angular Velocity
2','Angulara acceleration 2')
ylabel('position & Velocity')
title("m-file")
%function containing the differentialequationsa
function dx = func1(t, x)
% Values of Coefficients
% J1=1; J2=10; D1=0.9; D2=0.02; k=3
T=5;
%
% State Equations
dx(1) = x(2);
dx(3) = x(4);
dx(2) = (T-2*x(2)-x(1)+x(3)+x(4));
dx(4) = (x(2)-x(3)-2*x(4)+x(1));
dx = dx';
end
```

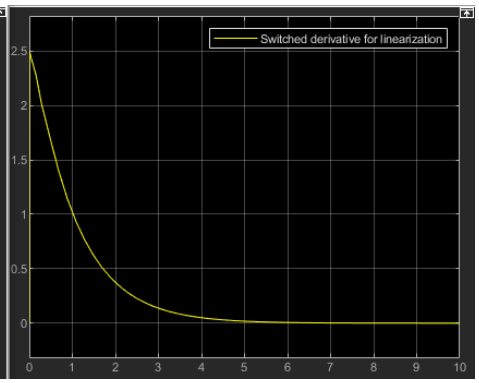
Simulink File



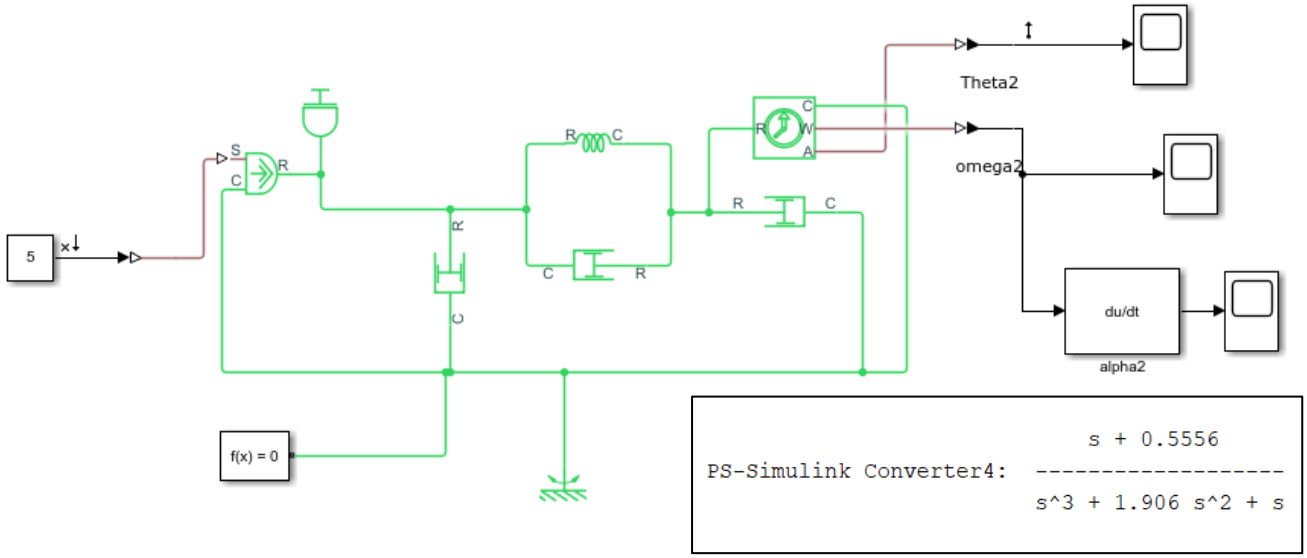
Simulink Results:





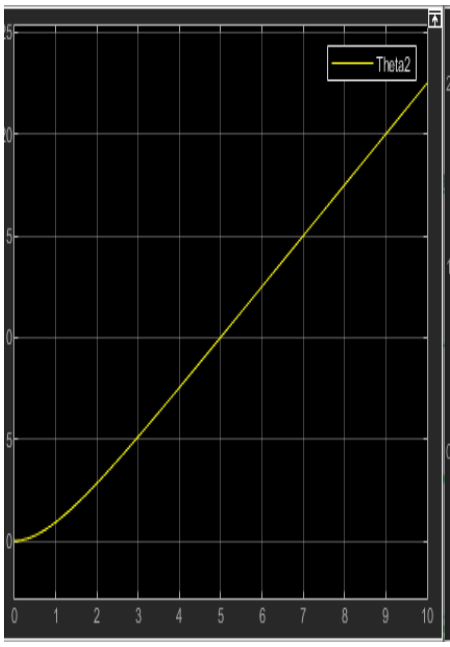


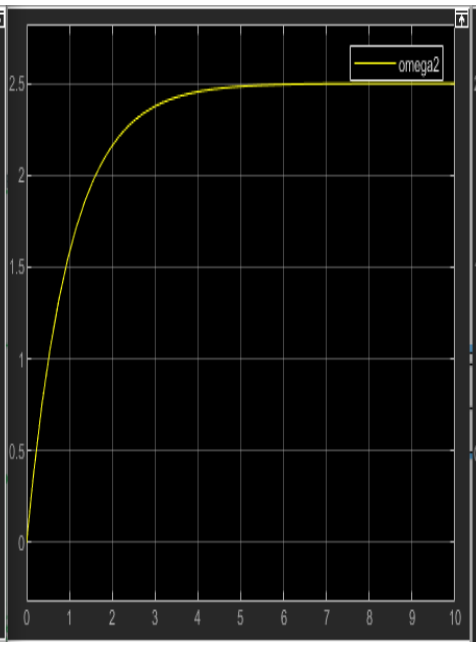
SIMSCAPE MODEL

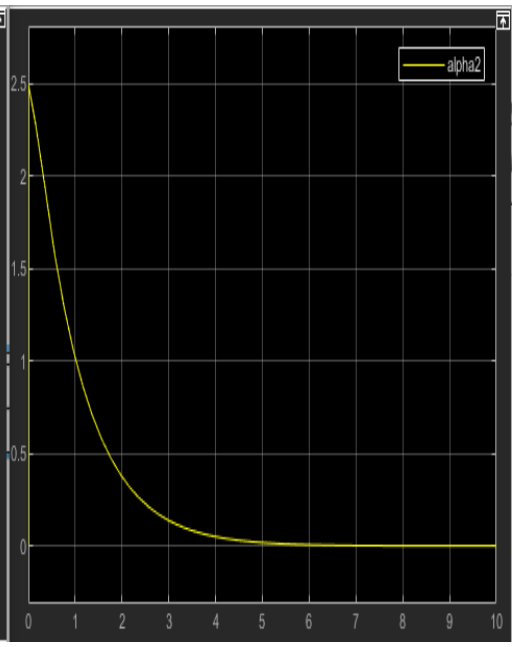


PS-Simulink Converter4:

$$\frac{s + 0.5556}{s^3 + 1.906\,s^2 + s}$$

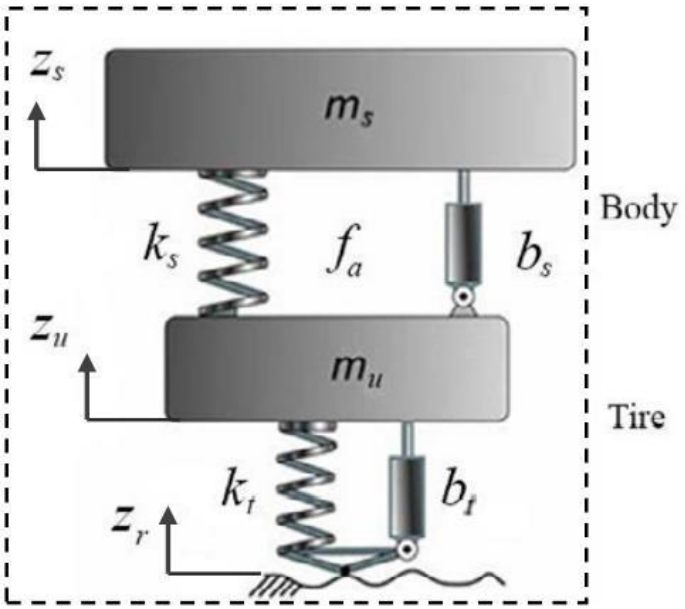




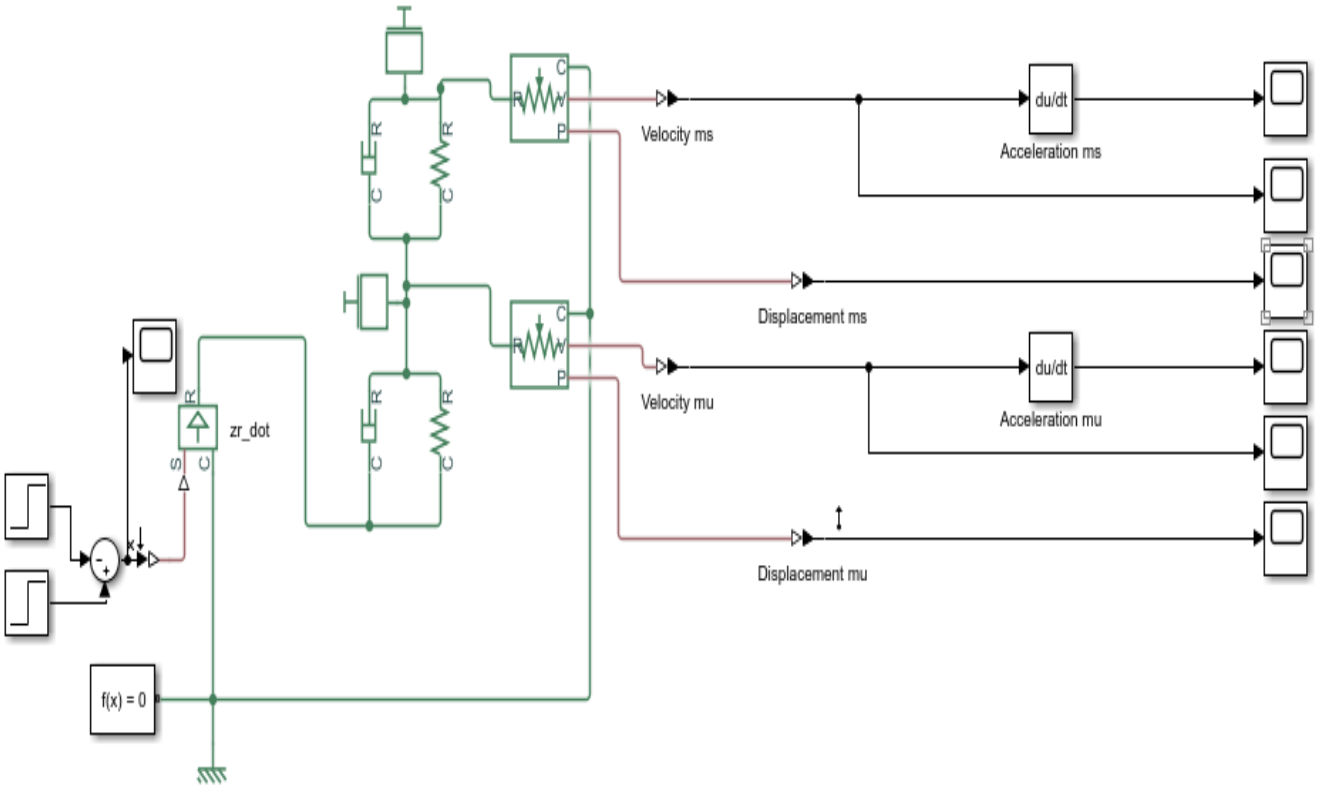


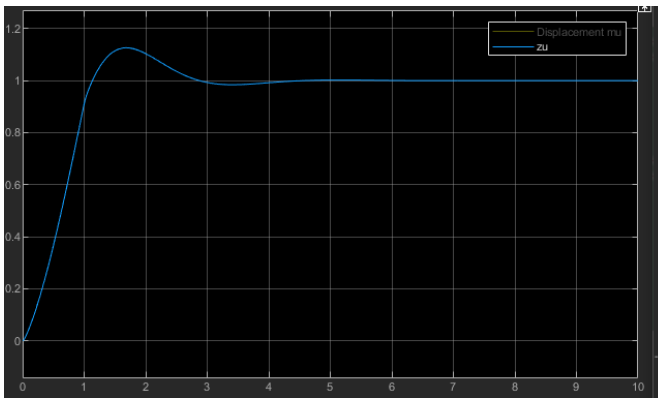
Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

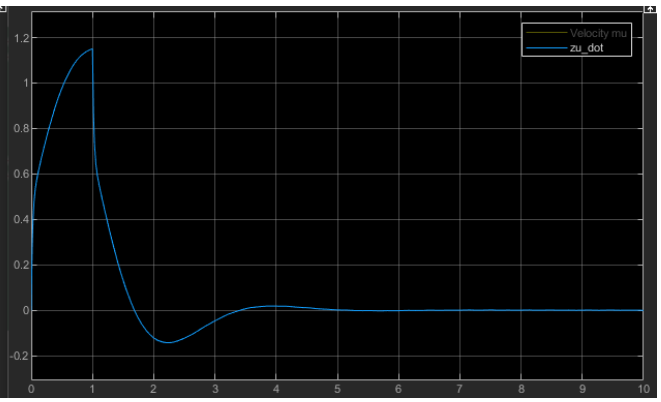
Example#05 (Quarter Car Model)

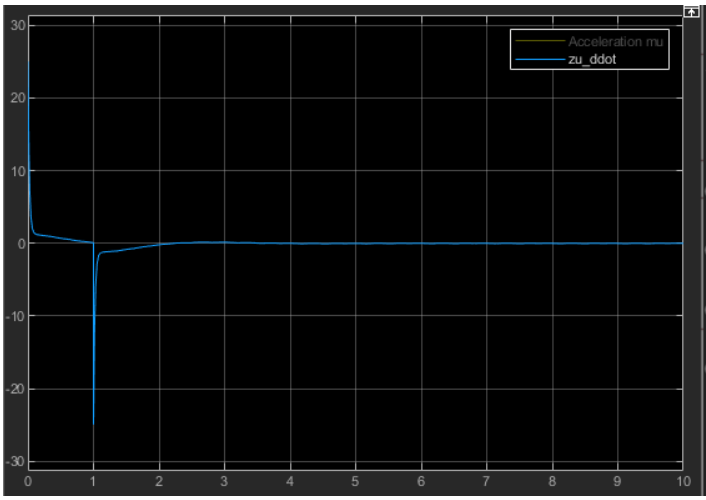


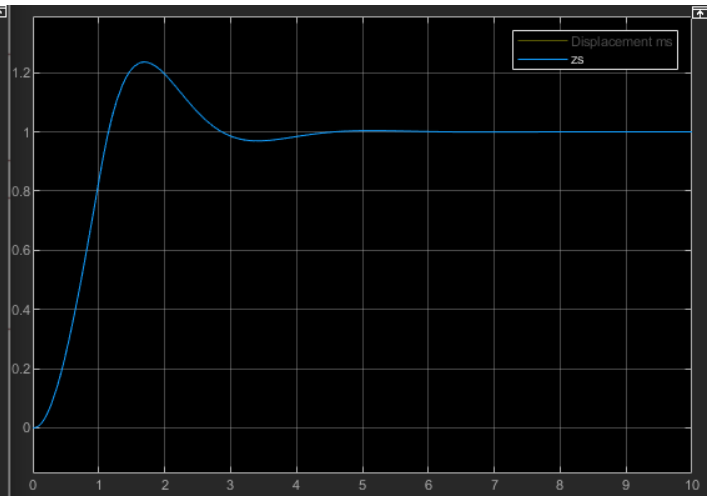
SIMSCAPE MODEL

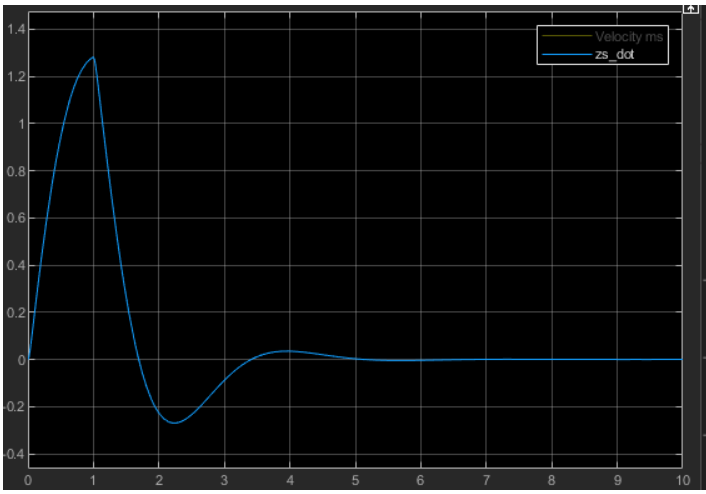


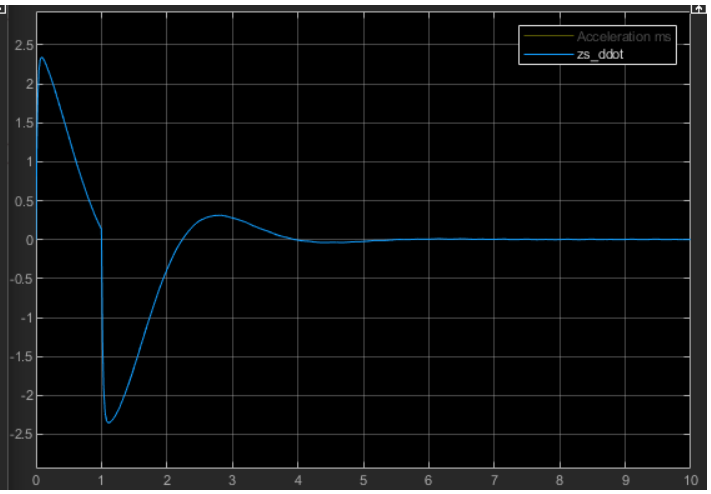

Displacement mu
zu

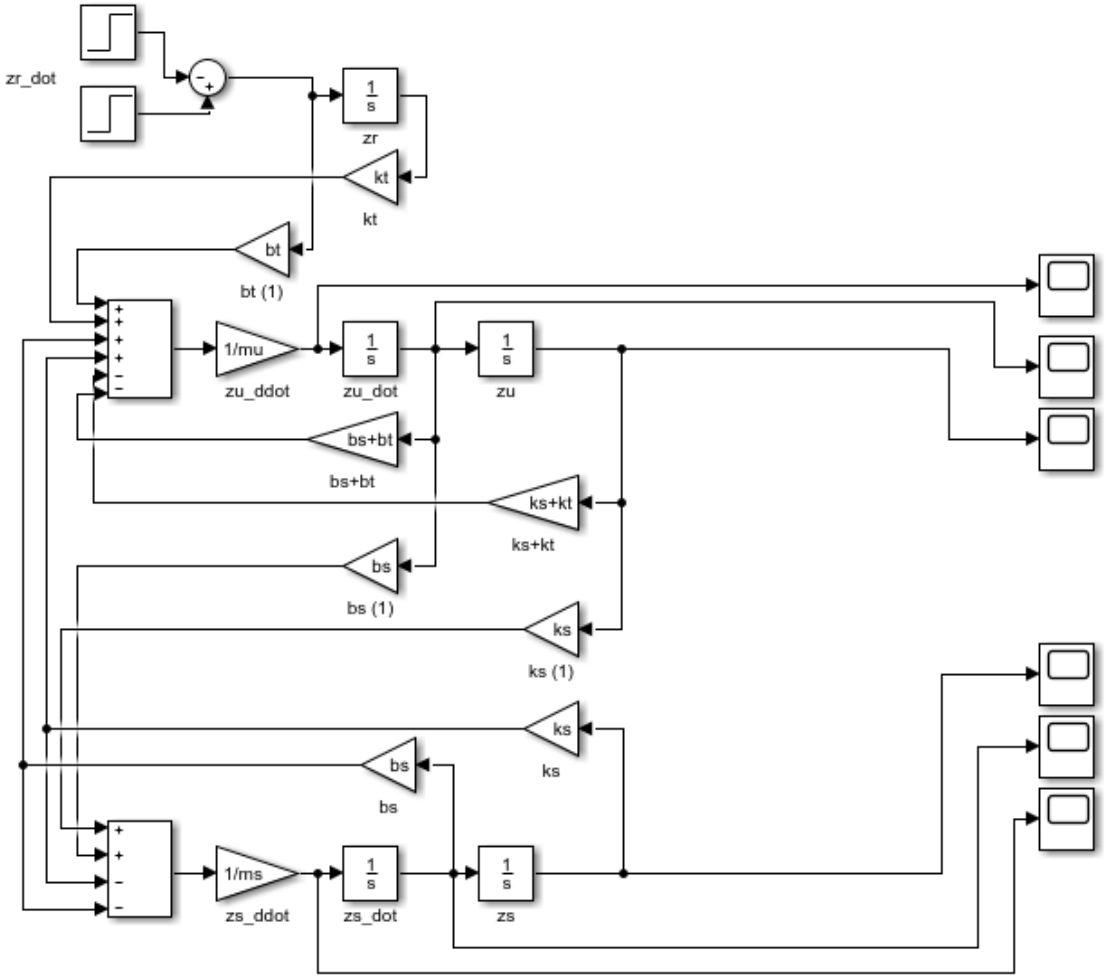

Velocity mu
zu_dot

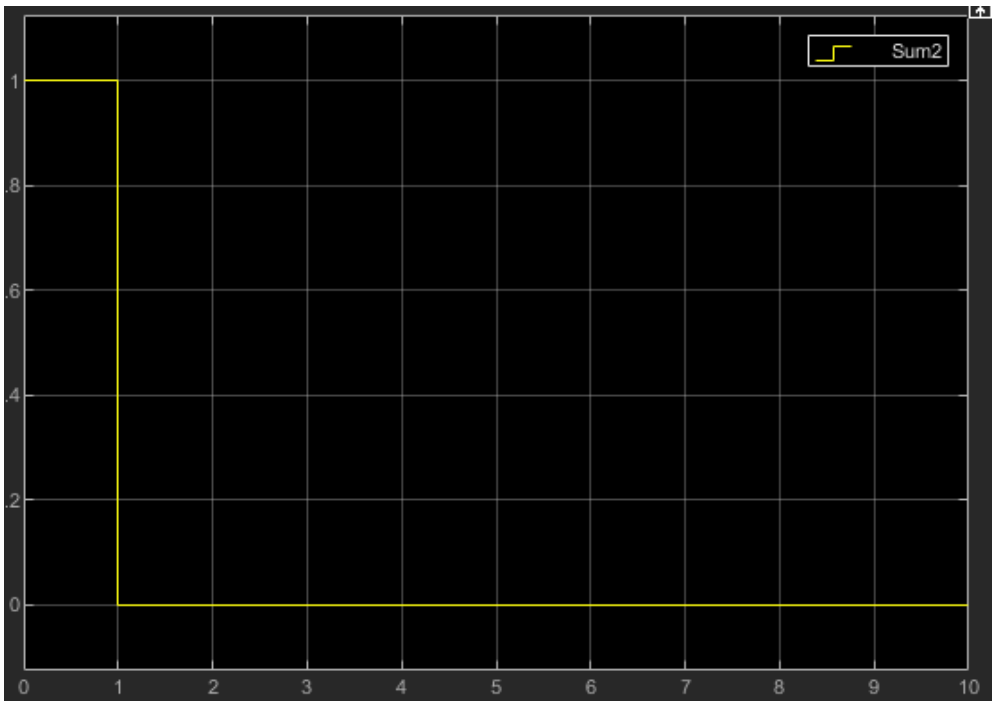

Acceleration mu
zu_ddot

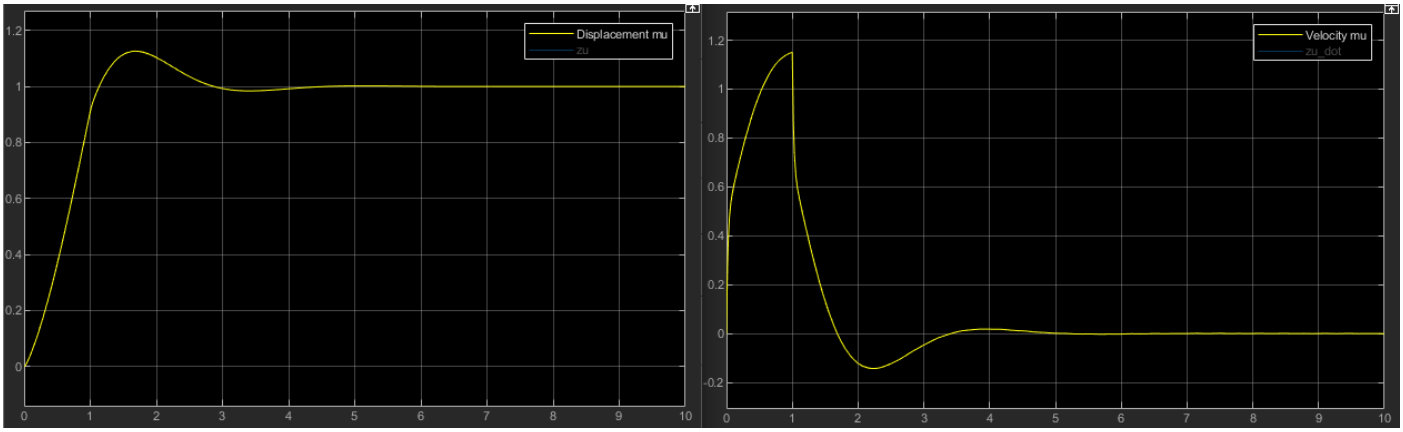

Displacement ms
zs

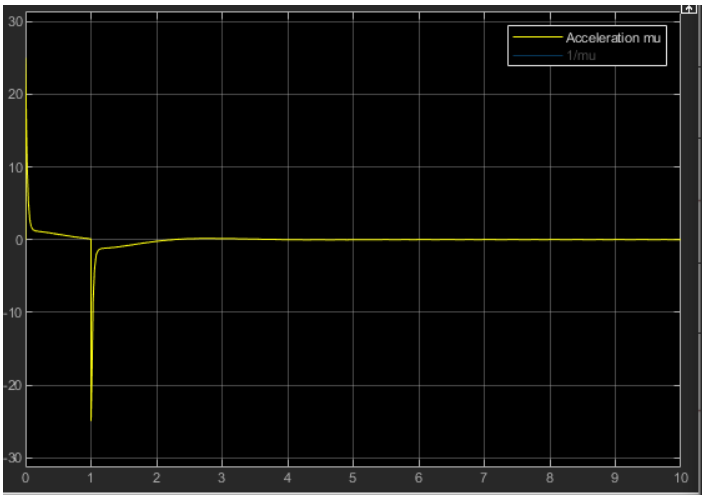

Velocity ms
zs_dot

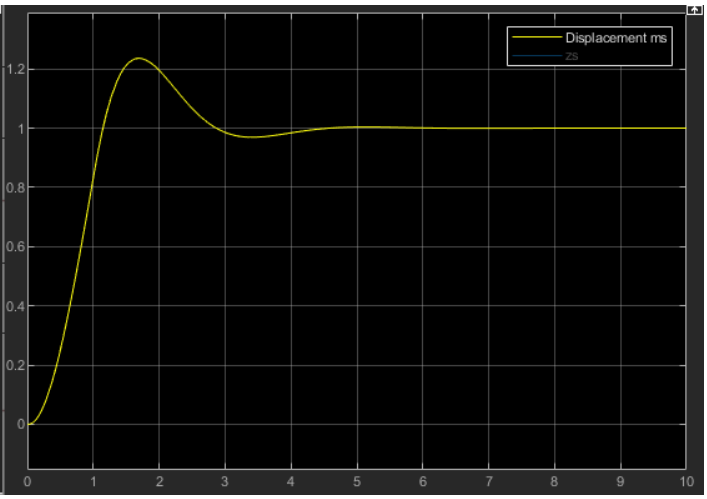

Acceleration ms
zs_ddot

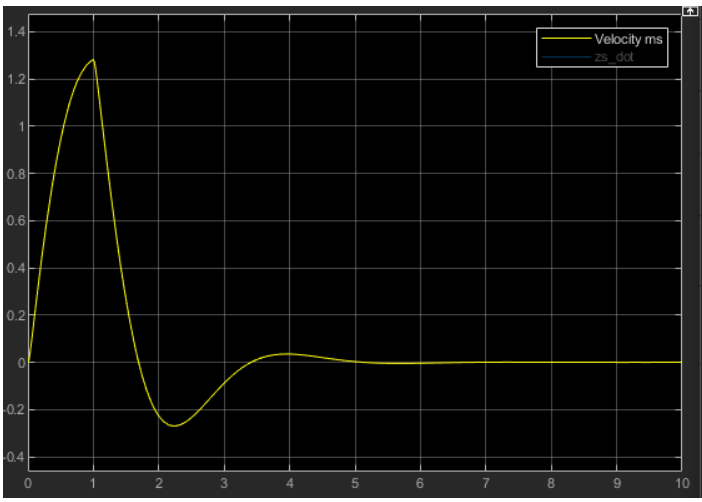

zr_dot
1
s
zr
kt
kt
bt
bt (1)
1/mu
zu_ddot
1
s
zu_dot
1
s
zu
bs+bt
bs+bt
ks+kt
ks+kt
bs
bs (1)
ks
ks (1)
ks
ks
bs
bs
1/ms
zs_ddot
1
s
zs_dot
1
s
zs

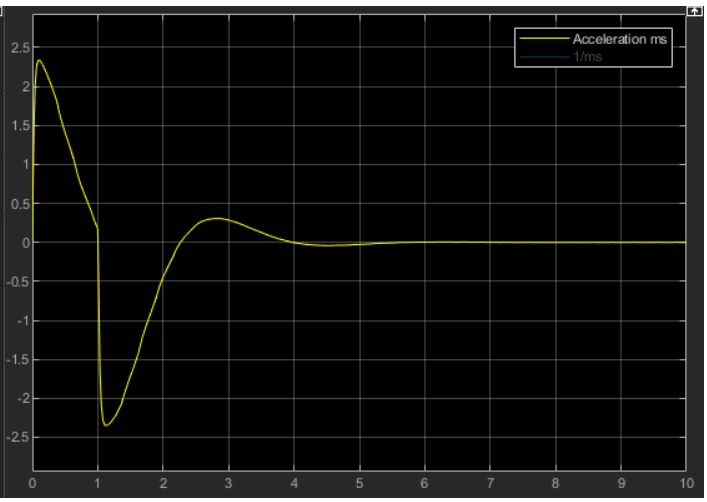

Sum2
1
.8
.6
.4
.2
0
0
1
2
3
4
5
6
7
8
9
10











Transfer Function from SIMSCAPE model

To obtain the transfer function from the SIMSCAPE model, the following steps were opted:

- Right-click on the input side. Select “Linear Analysis Points” and then “open-loop input”. Do the same on the output side but this time select “Output Measurement”.
- Run the file.

- In the options available at the top, select Analysis, Control Design, and then Linear Analysis. From there click on the Step option at the top. This creates a variable named “linsys1” in the linear analysis workspace. Drag the variable and drop it in the workspace.
- Go to the command window and type “tf(linsys1)” to obtain the transfer function.
- Following is the output:

```
>> tf(linsys1)

ans =

  From input "Sum2" to output "Displacement mu":
              25 s^3 + 175 s^2 + 500 s + 500
  ---------------------------------------------------
  s^5 + 55 s^4 + 235 s^3 + 500 s^2 + 500 s + 5.961e-13

Name: Linearization at model initial condition
Continuous-time transfer function.
```

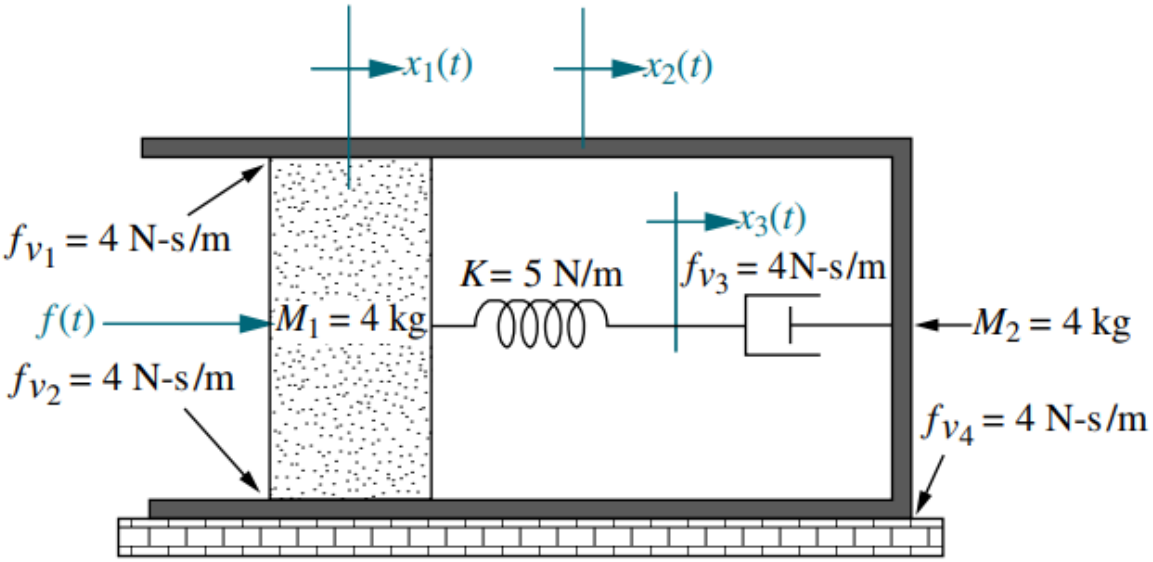
Transfer Function from M-File (Mathematical Model)

```
syms s zrdot
A=[(mu*s^2+(bs+bt)*s+ks+kt) , -(ks+bs*s) ; -(ks+bs*s) , (ms*s^2+ks+bs*s)];
B=[(kt/s+bt)*zrdot;0];
C=A\B;
zs=C(1);
G=zs/zrdot;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1); %dividing by den(1) means dividing by the leading coefficient of
denominator i.e., 20 in this case. Done to match the outputs.
den=den/den(1);
G=tf(num,den)
```

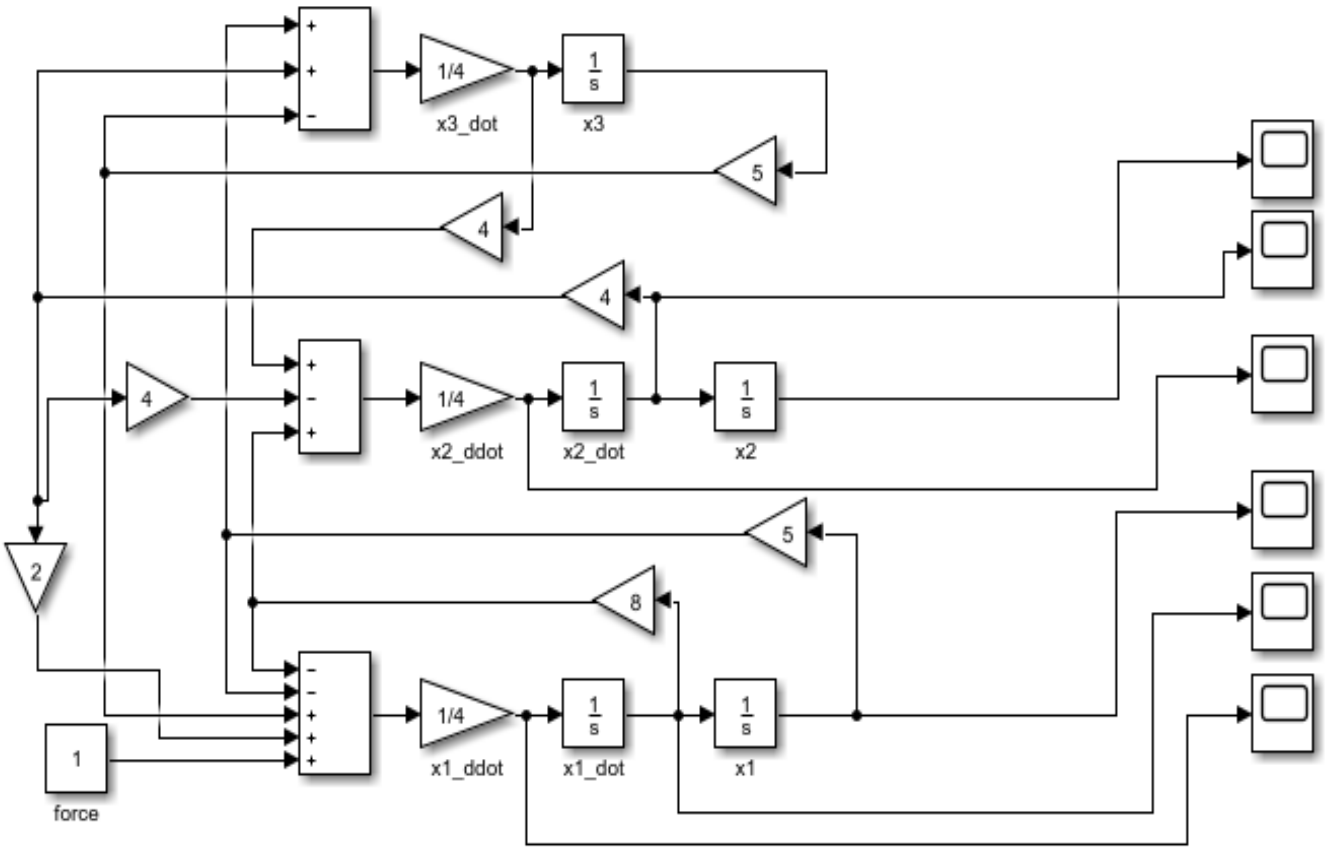
Following is the output:

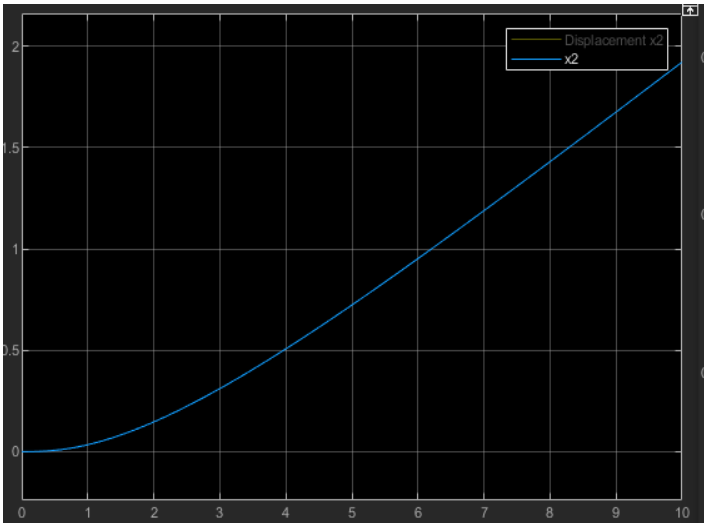
```
>> Task1TransFunc

G =

        25 s^3 + 175 s^2 + 500 s + 500
  ---------------------------------------
  s^5 + 55 s^4 + 235 s^3 + 500 s^2 + 500 s

Continuous-time transfer function.
```

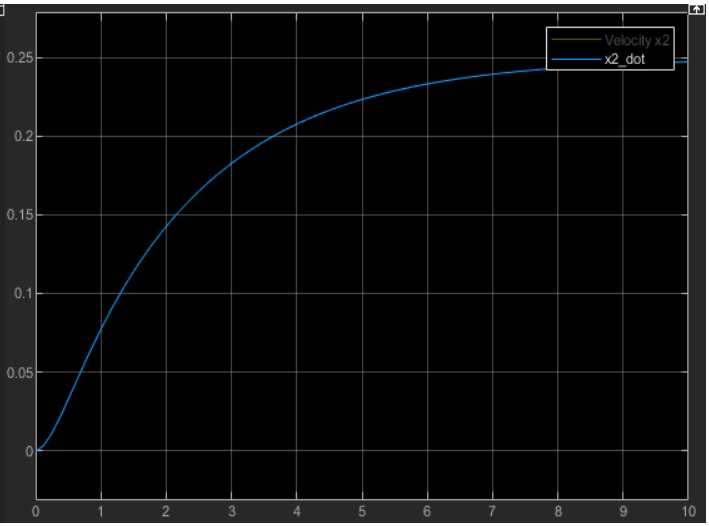
Example#06 (Framed System: Translational Piston Spring System)

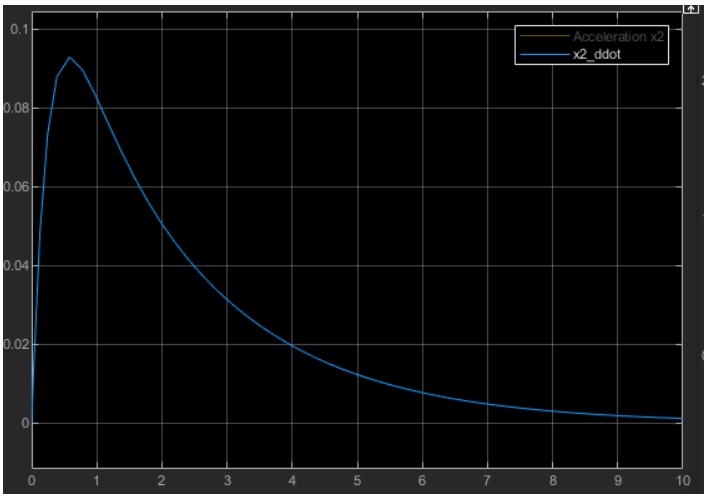


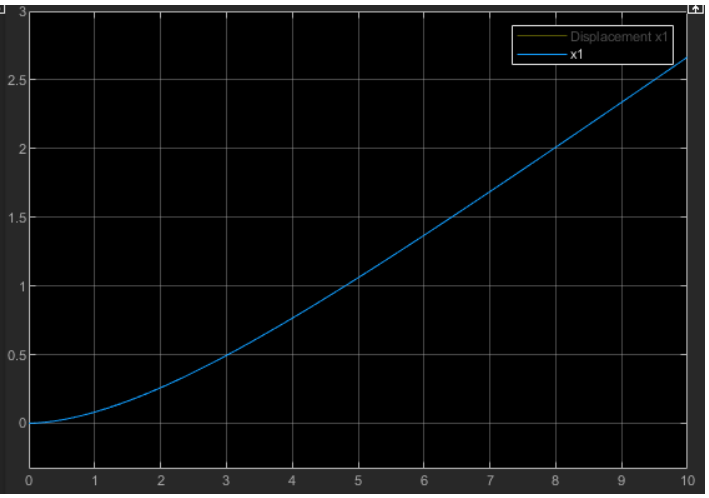
Simulink Model

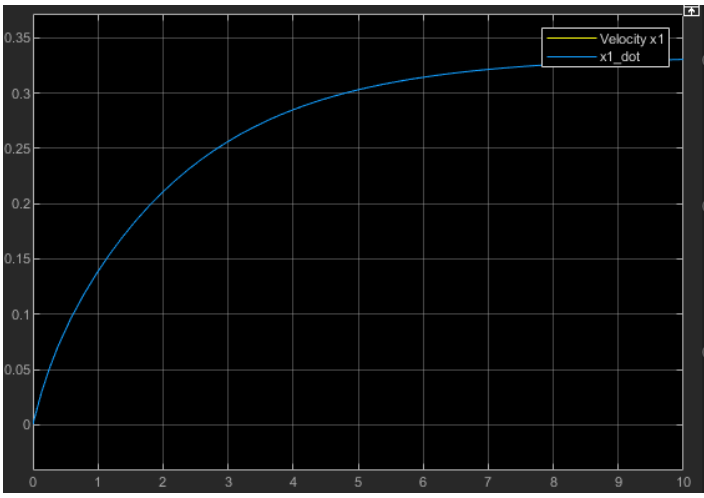


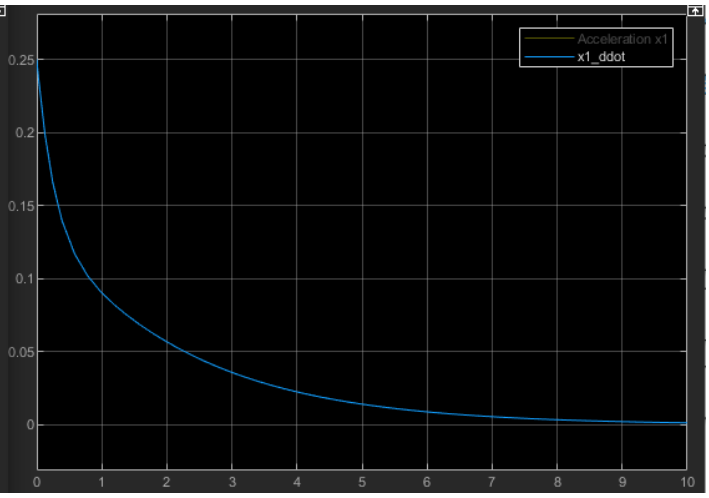

Displacement x2
x2


Velocity x2
x2_dot

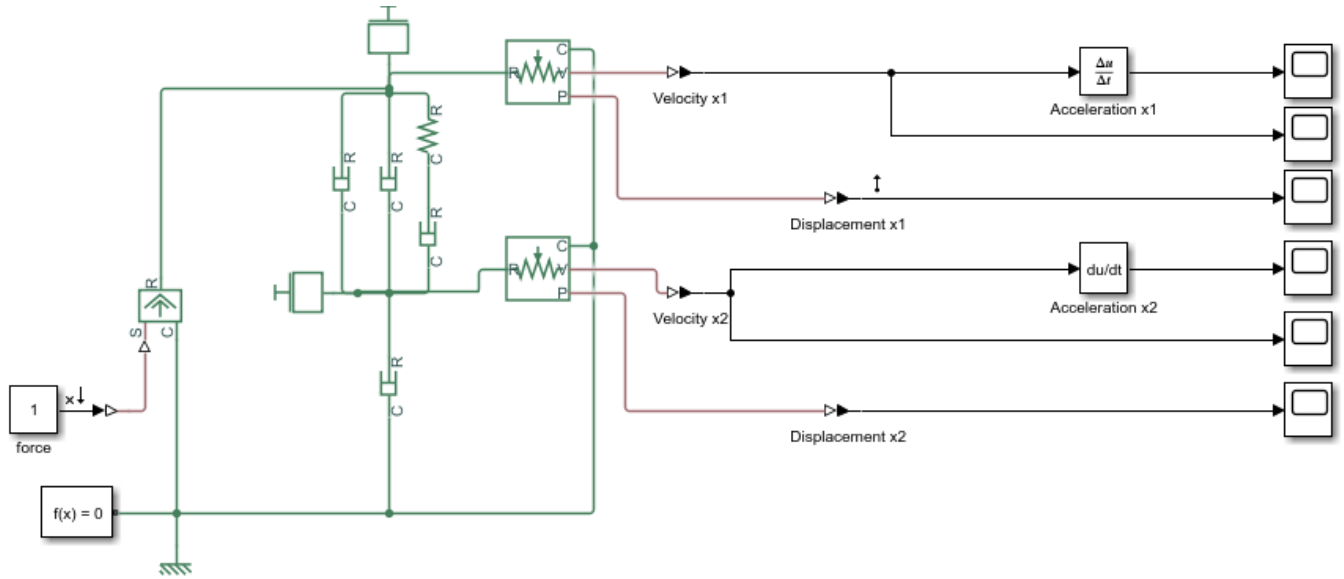

Acceleration x2
x2_ddot

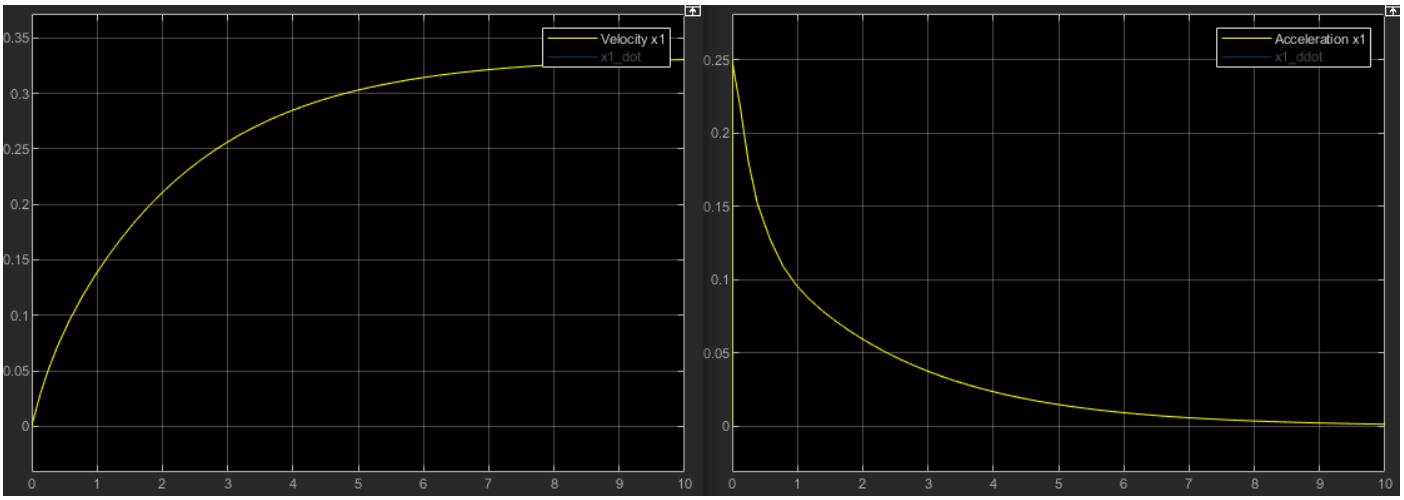

Displacement x1
x1

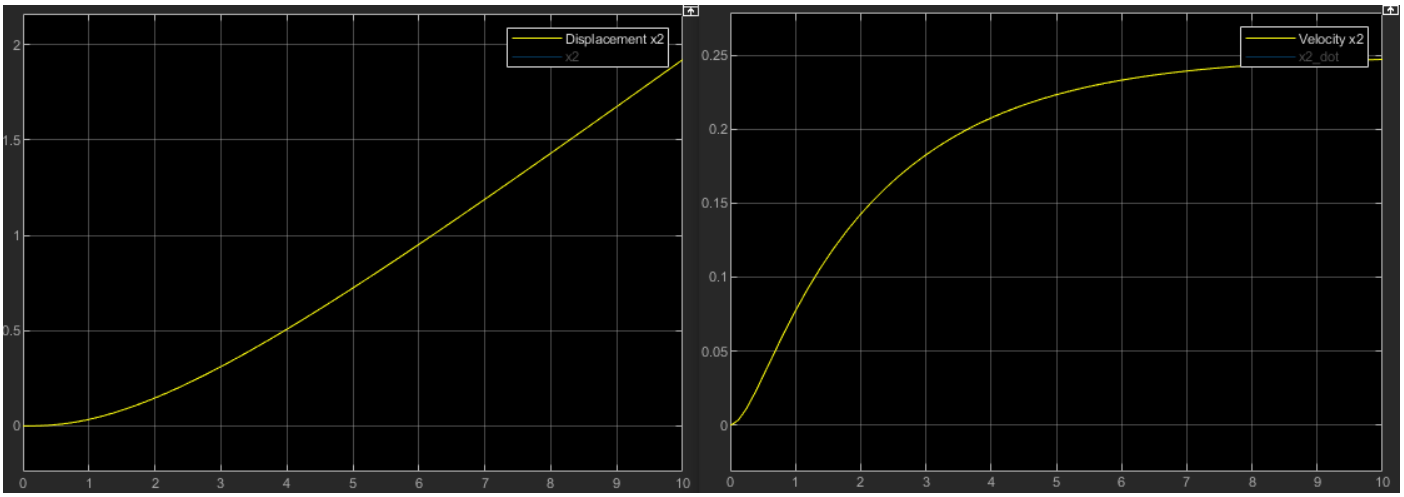

Velocity x1
x1_dot


Acceleration x1
x1_ddot

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan



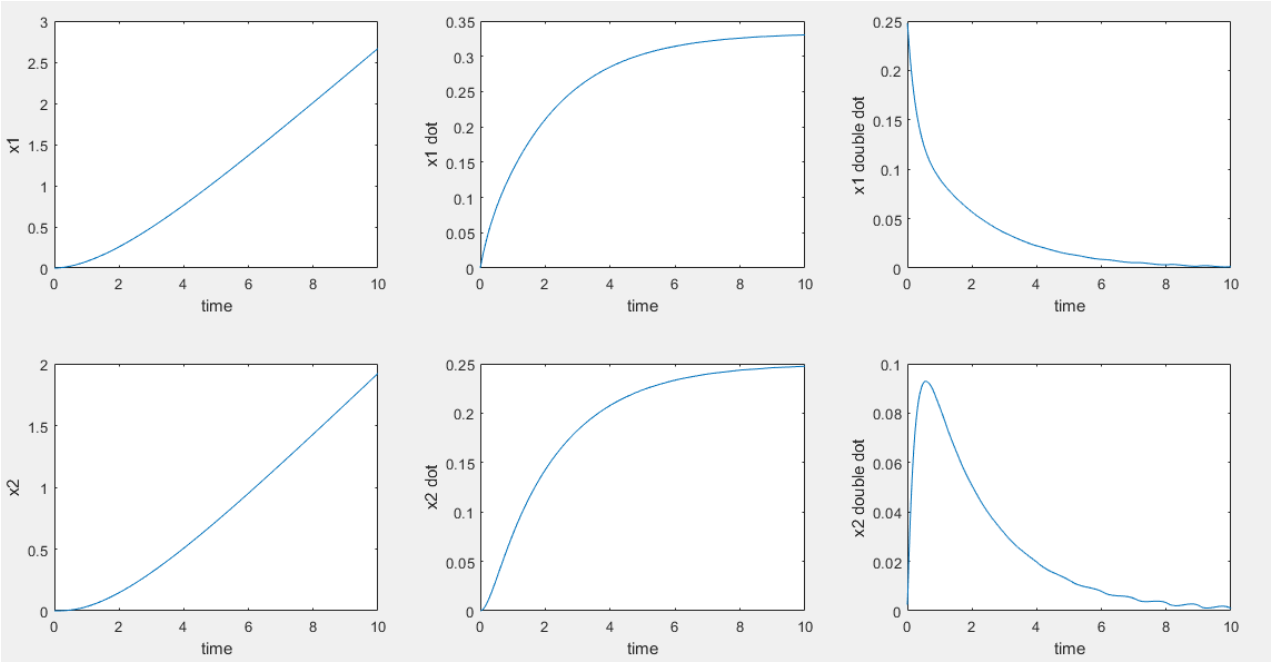




Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

M file

```
clc;
TR=0:0.01:10;
x0=[0;0;0;0;0];
[t,x]=ode45(@Task2Fun,TR,x0);
x1=x(:,1);
x1_dot=x(:,2);
x1_ddot=gradient(x1_dot)./gradient(t);
x2=x(:,3);
x2_dot=x(:,4);
x2_ddot=gradient(x2_dot)./gradient(t);

subplot(2,3,1);
plot(t,x1);xlabel('time');ylabel('x1');
subplot(2,3,2);
plot(t,x1_dot);xlabel('time');ylabel('x1 dot');
subplot(2,3,3);
plot(t,x1_ddot);xlabel('time');ylabel('x1 double dot')
subplot(2,3,4);
plot(t,x2);xlabel('time');ylabel('x2');
subplot(2,3,5);
plot(t,x2_dot);xlabel('time');ylabel('x2 dot');
subplot(2,3,6);
plot(t,x2_ddot);xlabel('time');ylabel('x2 double dot')
%%%%%%%%%%%%%%%%%%%%
function dy=Task2Fun(t,y)
    f=1;
    dy(1)=y(2);
    dy(2)=1/4*(f - 8*y(2) - 5*y(1) + 8*y(4) + 5*y(5));
    dy(3)=y(4);
    dy(5)=1/4*(5*y(1) + 4*y(4) - 5*y(5));
    dy(4)=1/4*(8*y(2) + 4*dy(5) - 16*y(4));
    dy=dy';
end
```



Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

Transfer Function from SIMSCAPE Model

```
>> tf(linsys1)

ans =

  From input "force" to output "Displacement x1":
            0.25 s^2 + 1.063 s + 1.25
  ---------------------------------------------
  s^4 + 6.25 s^3 + 10.75 s^2 + 3.75 s - 1.925e-15

Name: Linearization at model initial condition
Continuous-time transfer function.
```

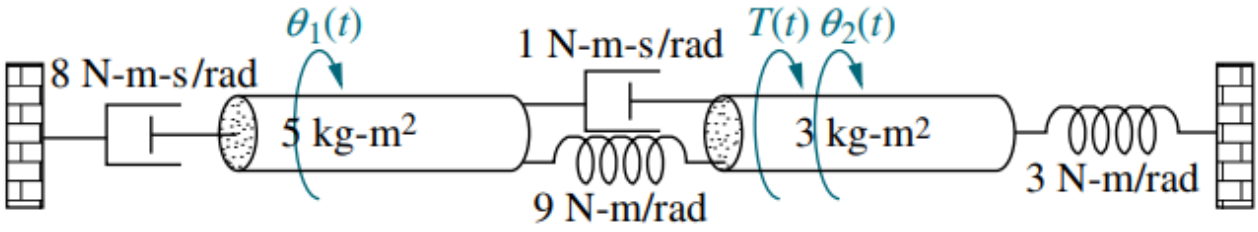
Transfer Function from m File

```
syms s
f=1;
A=[(4*s^2+8*s+5) , -8*s , -5 ; -8*s , (4*s^2+16*s) , -4*s ; -5 , -4*s ,
(4*s+5)];
B=[f;0;0];
C=A\B;
x1=C(1);
G=x1/f;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);     %dividing by den(1) means dividing by the leading
coefficient of denominator i.e., 20 in this case. Done to match the outputs
den=den/den(1);
tf(num,den)
```

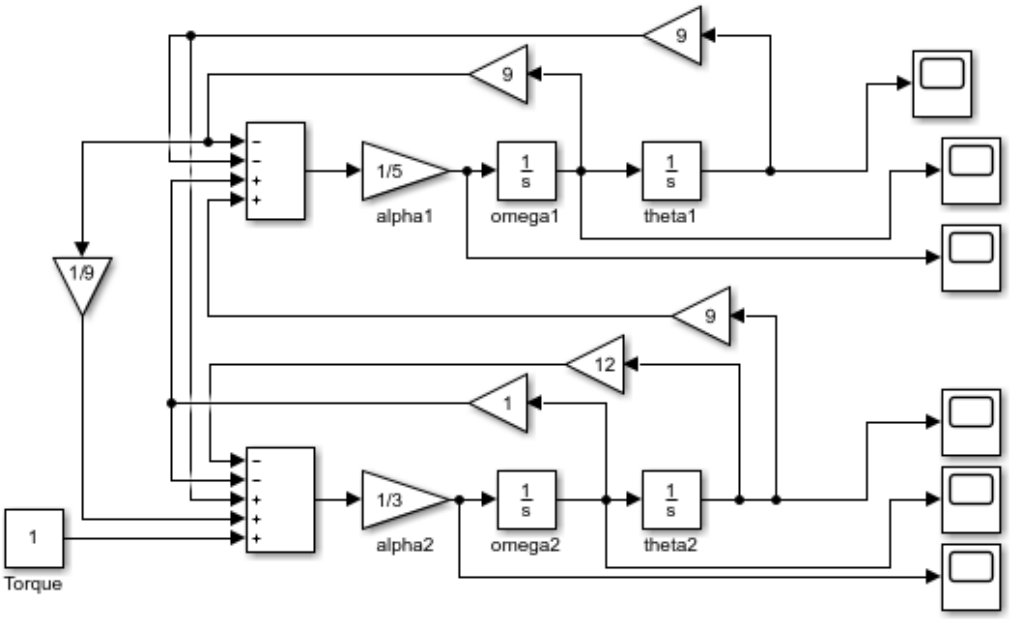
```
>> Task2TransFunc

ans =

        0.25 s^2 + 1.063 s + 1.25
  -----------------------------------
  s^4 + 6.25 s^3 + 10.75 s^2 + 3.75 s

Continuous-time transfer function.
```

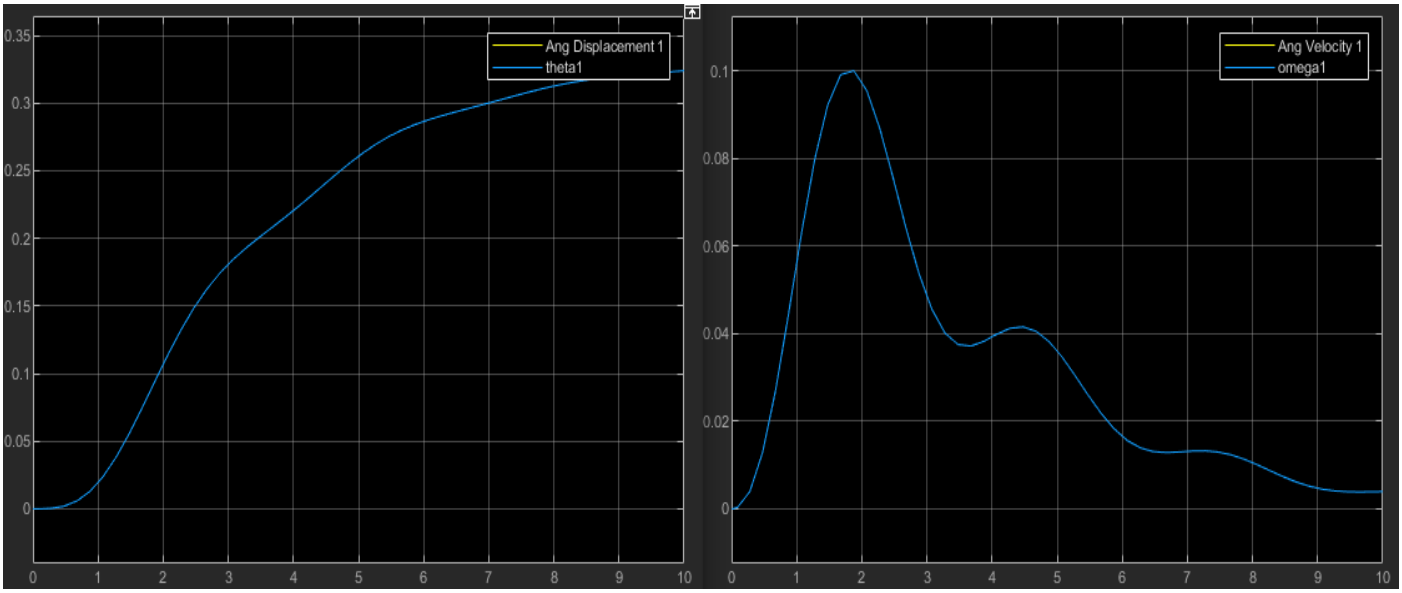
Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

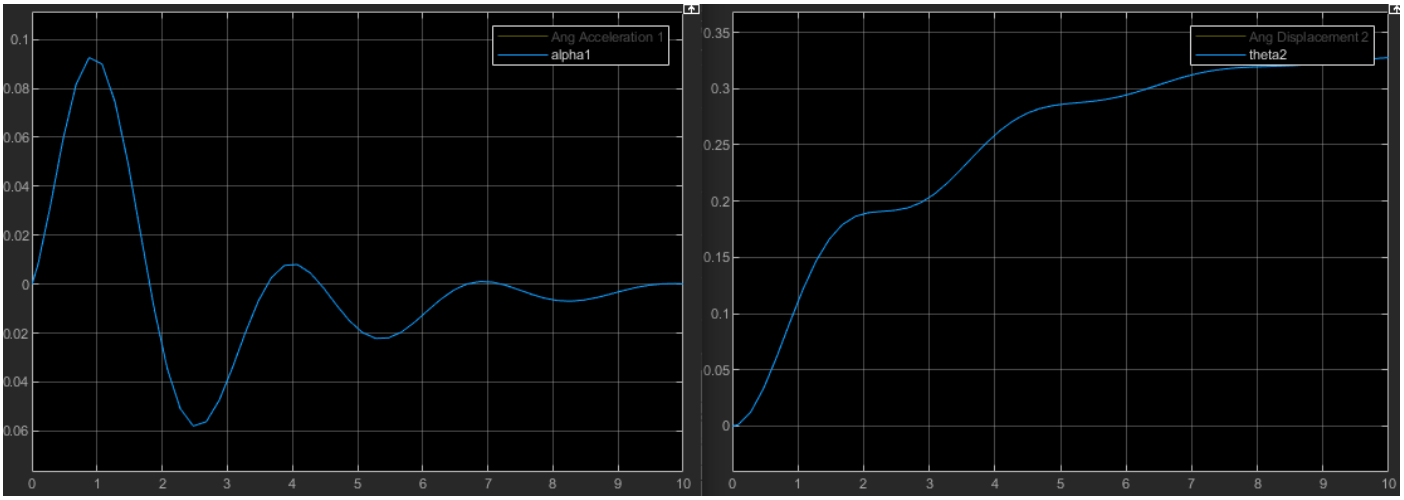
Example#07 (Dual Rotational Mechanical System)

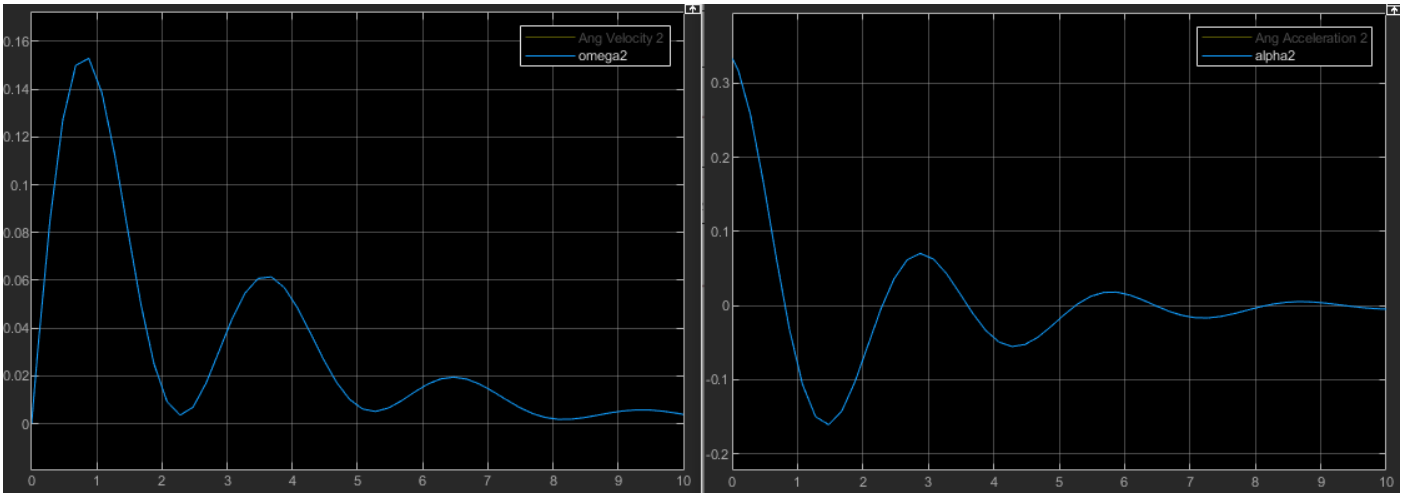


Simulink Model

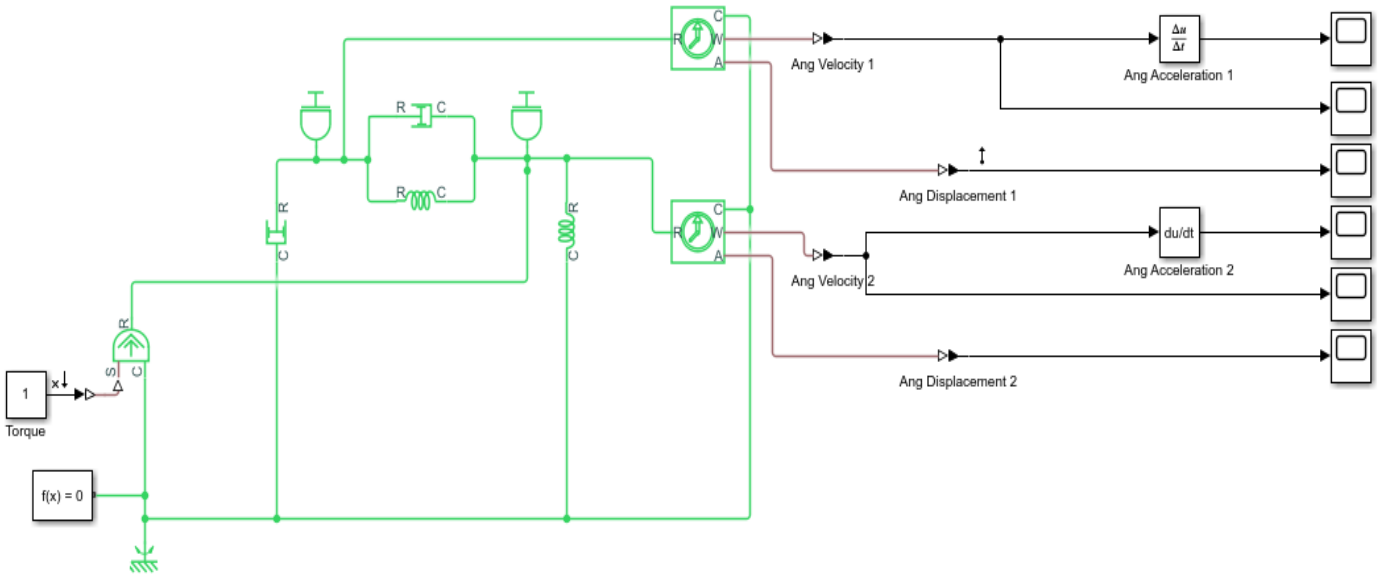


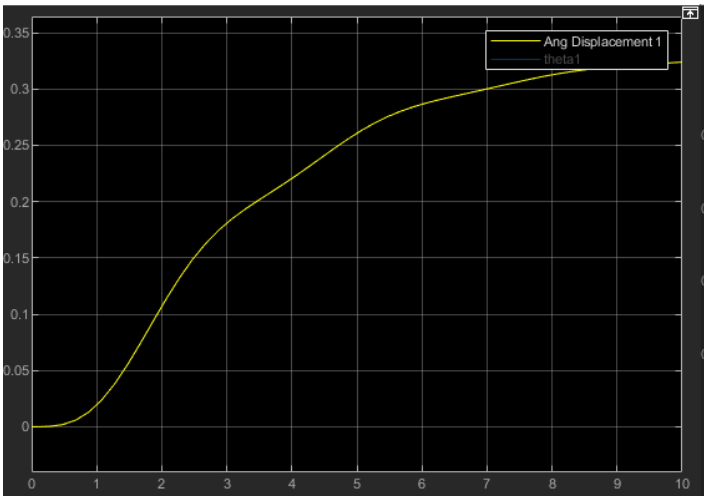


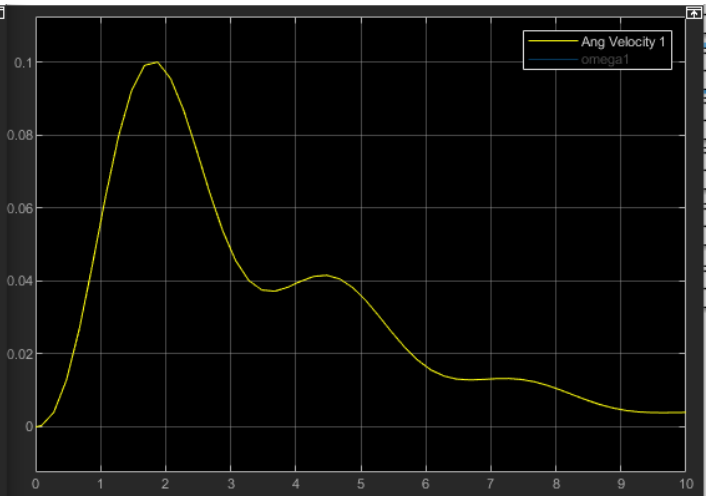


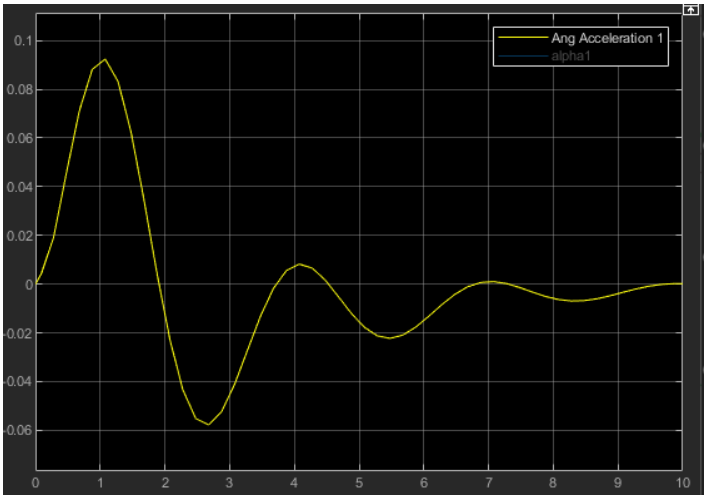


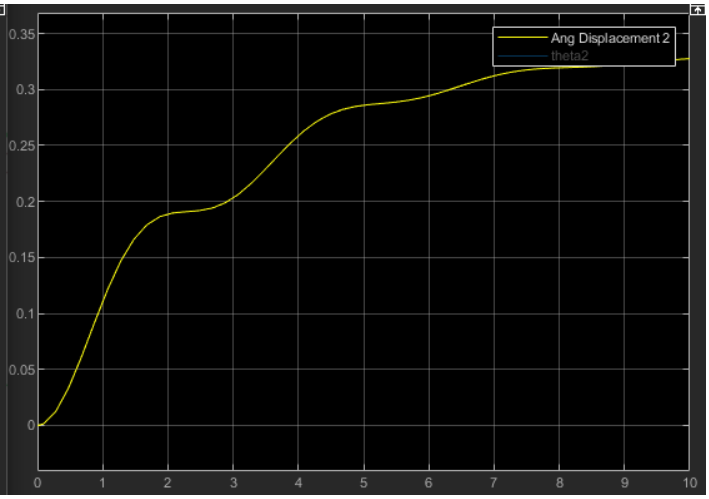
SIMSCAPE Model

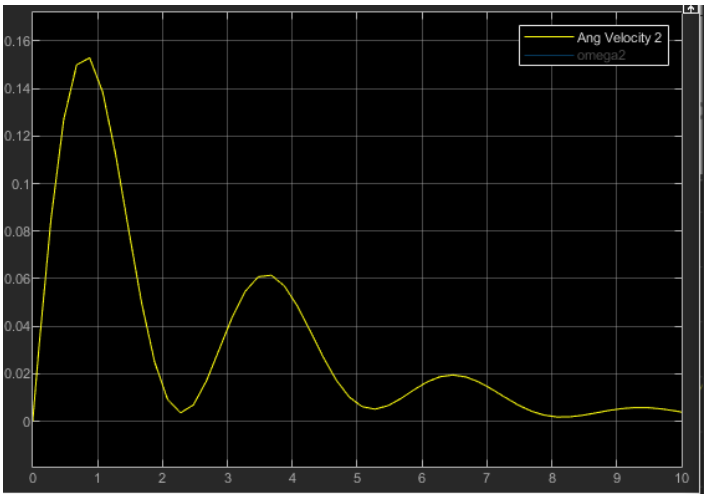


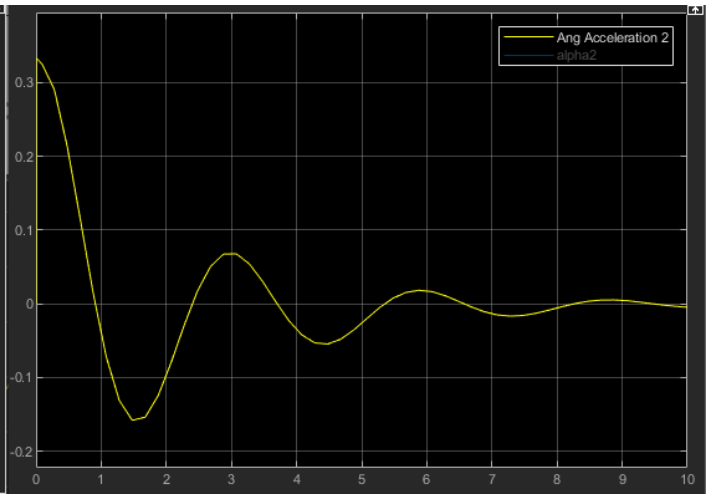

Ang Displacement 1
theta1


Ang Velocity 1
omega1


Ang Acceleration 1
alpha1


Ang Displacement 2
theta2

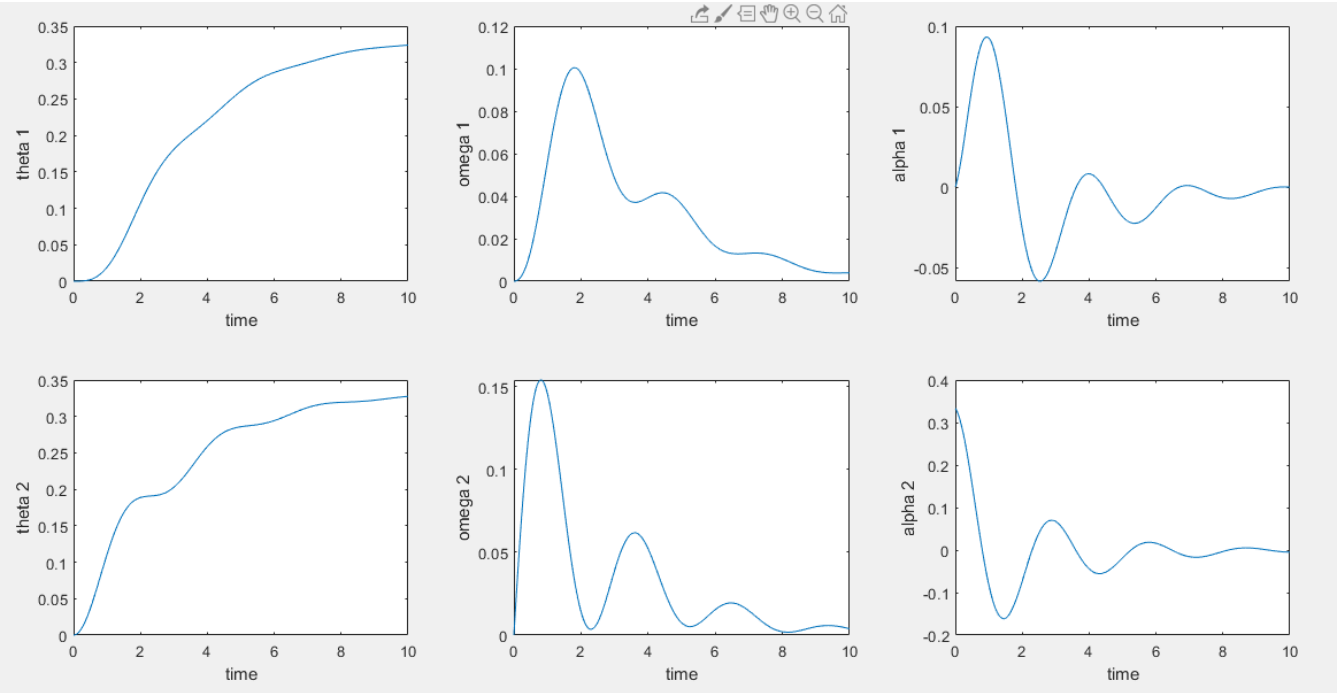

Ang Velocity 2
omega2


Ang Acceleration 2
alpha2

M-File

```
clc;
TR=0:0.01:10;
x0=[0;0;0;0];
[t,x]=ode45(@Task3Fun,TR,x0);
th1=x(:,1);
om1=x(:,2);
alpha1=gradient(om1)./gradient(t);
th2=x(:,3);
om2=x(:,4);
alpha2=gradient(om2)./gradient(t);

subplot(2,3,1);
plot(t,th1);xlabel('time');ylabel('theta 1');
subplot(2,3,2);
plot(t,om1);xlabel('time');ylabel('omega 1');
subplot(2,3,3);
plot(t,alpha1);xlabel('time');ylabel('alpha 1')
subplot(2,3,4);
plot(t,th2);xlabel('time');ylabel('theta 2');
subplot(2,3,5);
plot(t,om2);xlabel('time');ylabel('omega 2');
subplot(2,3,6);
plot(t,alpha2);xlabel('time');ylabel('alpha 2')
%%%%%%%%%%%%%%%%%%%%
function dy=Task3Fun(t,y)
    T=1;
    dy(1)=y(2);
    dy(2)=1/5*(y(4) + 9*y(3) - 9*y(2) - 9*y(1));
    dy(3)=y(4);
    dy(4)=1/3*(T+y(2) + 9*y(1) - y(4) - 12*y(3));
    dy=dy';
end
```

Following is the output



Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

Transfer Function from SIMSCAPE model

```
>> tf(linsys1)

ans =

  From input "Torque" to output "Ang Displacement 1":
                0.06667 s + 0.6
  ----------------------------------------
  s^4 + 2.133 s^3 + 6.333 s^2 + 6.6 s + 1.8

Name: Linearization at model initial condition
Continuous-time transfer function.
```

Transfer Function from m file

```
syms s
T=1;
A=[(5*s^2+9*s+9) , -(s+9) ; -(s+9) , (3*s^2+s+12)];
B=[0;T];
C=A\B;
th1=C(1);
G=th1/f;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of
denominator i.e., 20 in this case. Done to match the outputs
den=den/den(1);
tf(num,den)
```

```
>> Task3TransFunc

ans =

                0.06667 s + 0.6
  ----------------------------------------
  s^4 + 2.133 s^3 + 6.333 s^2 + 6.6 s + 1.8

Continuous-time transfer function.
```

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

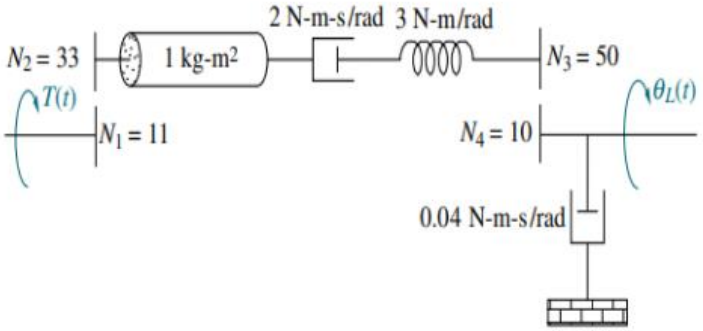
LAB 2: Modeling, Analysis, and Cross-Validation of the Dynamic Responses of Geared Rotational Systems and Combined Mechanical Dynamic Systems.

Aim and Objectives:

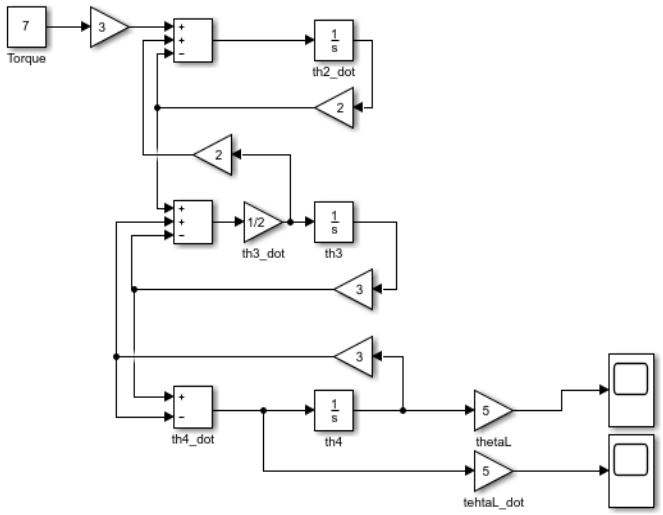
The fundamental aim of this lab is to imply the studied concepts of LAB#01 on modeling and analyzing the third (Geared Rotational Systems) as well as a fourth module (Combined Systems) of the Mechanical Dynamic Systems. Furthermore, the outcomes of the dynamic systems have been validated with the findings of SIMULINK and **ODE solvers**. Therefore, this lab is primarily dedicated to two categories of mechanical systems and certain tangible objectives are enlightened below:

1. To model and analyze the dynamic response of Geared Dynamics as well as combined Mechanical systems (A system with the translational, rotational, and geared elements along with certain rotary to linear converters).
2. To obtain the transfer function from developed SIMSCAPE models.
3. To cross-validate the outcomes of SIMSCAPE Models with the results obtained from ODEs 45 and SIMULINK files.

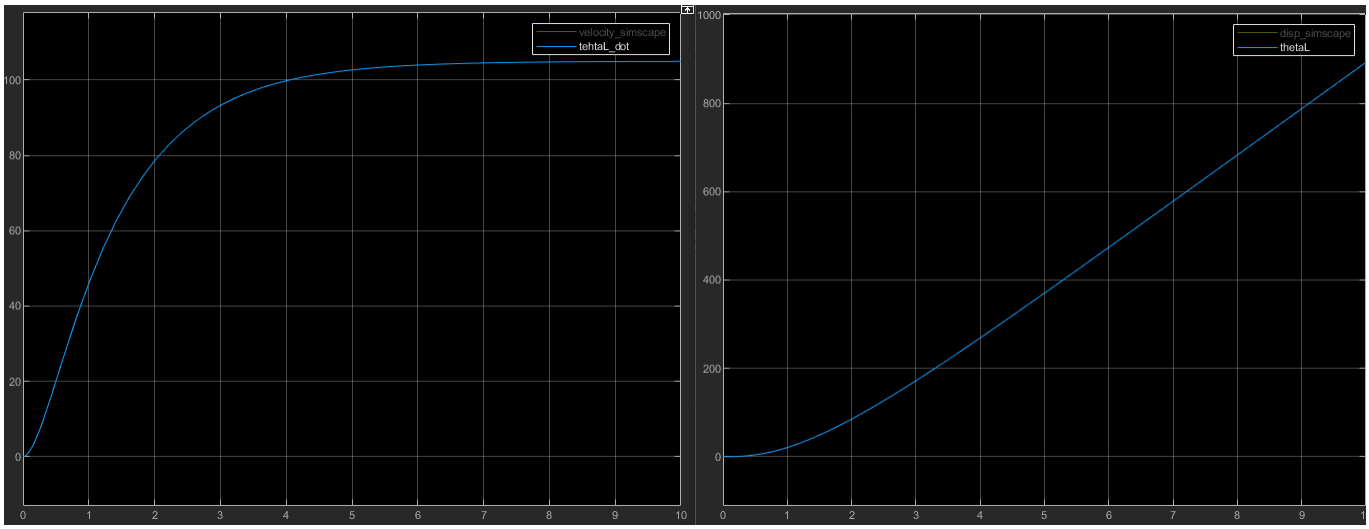
Example#01(Simple Geared Rotational System)



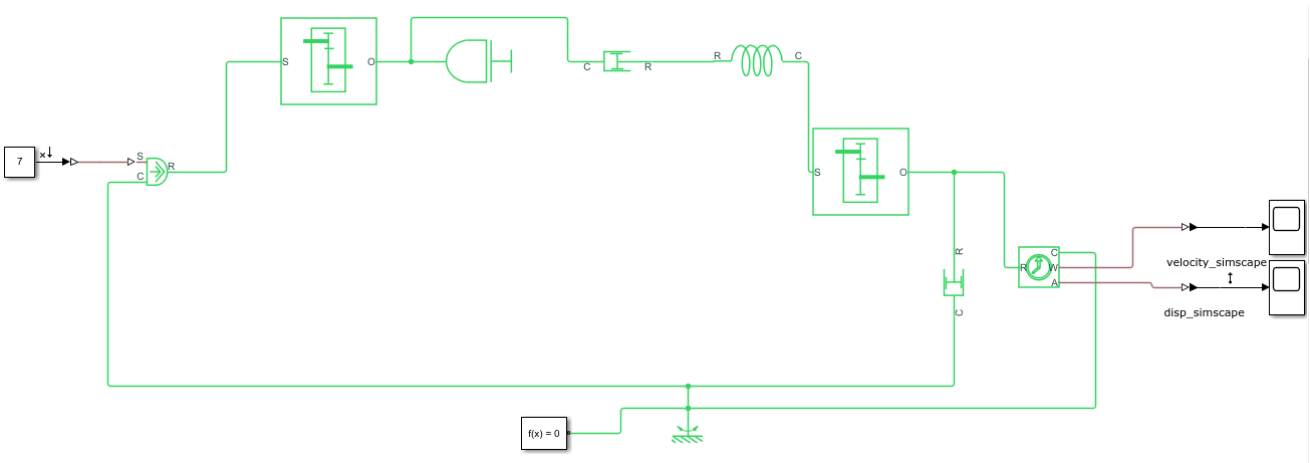
Simulink Model



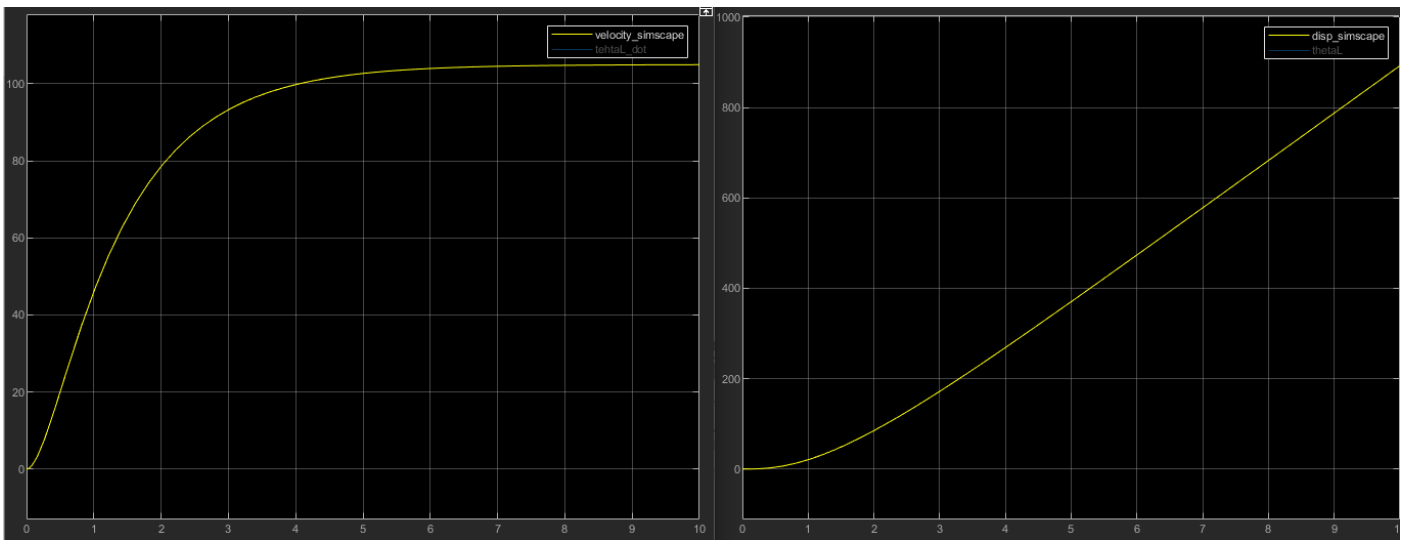
Results



SIMSCAPE Model



Results



Conclusion/Remarks

The results obtained by both Simulink as well as SIMSCAPE are the same. Hence the developed SIMSCAPE model is correct and is following the mathematical model (Simulink model). The graphs obtained are logical as well; the rotational output is measured across the gear which ideally rotates continuously maintaining a constant angular velocity (by the law of gearing). This phenomenon is also observed in the plots above which exhibit a linearly increasing trend for angular displacement and a constant trend for angular velocity.

Transfer Function from SIMSCAPE model

```
>> tf(linsys1)

ans =

  From input "Constant" to output "disp_simscape":
          45
  -------------------
  s^3 + 4.5 s^2 + 3 s

Name: Linearization at model initial condition
Continuous-time transfer function.
```

Transfer function from m file

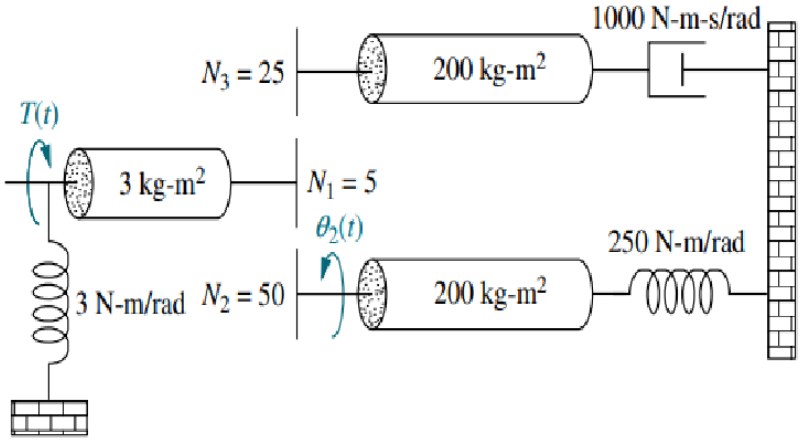
```
syms s T
A=[(s^2+2*s) , -2*s , 0 ; -2*s , (2*s+3) , -3 ; 0 , -3 , (3+s)];
B=[3*T ; 0 ; 0];
C=A\B;
theta4=C(3);
thetaL=5*theta4;
G=thetaL/T;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
```

```
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
fprintf('ThetaL(s)/T(s):')
tf(num,den)
```

Following is the output:

```
ThetaL(s)/T(s):
ans =

          45
  -------------------
  s^3 + 4.5 s^2 + 3 s

Continuous-time transfer function.
```

Example#02 (Double Geared System)

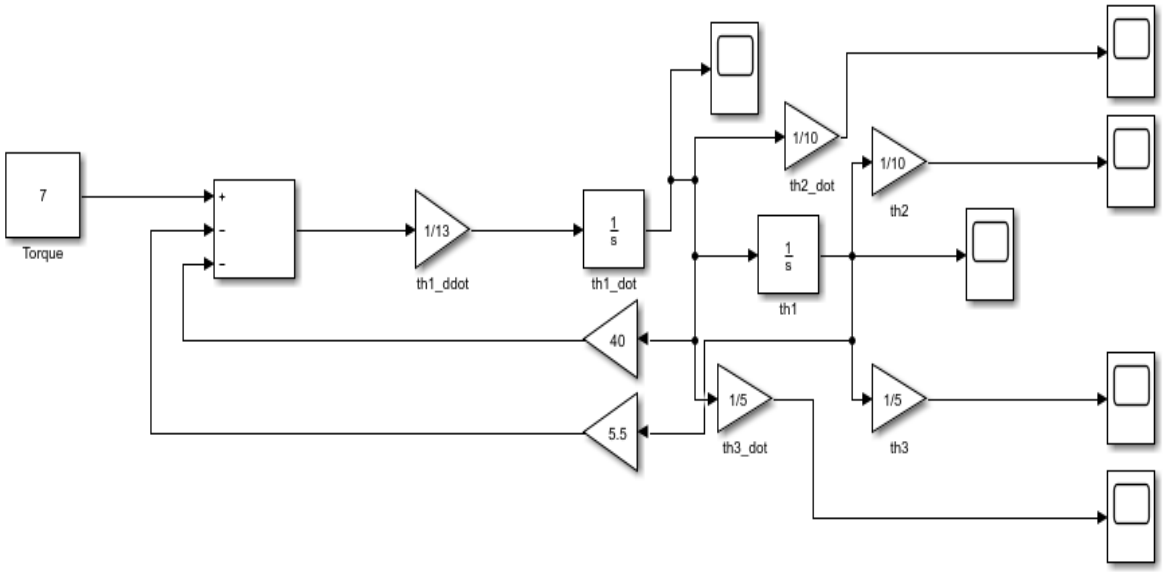


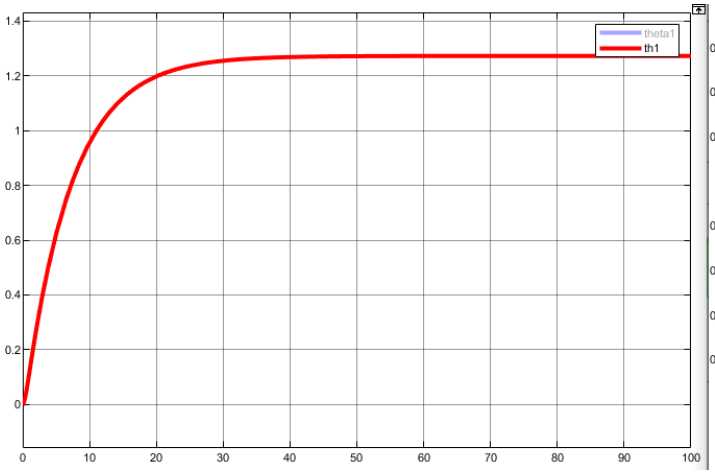
M-File

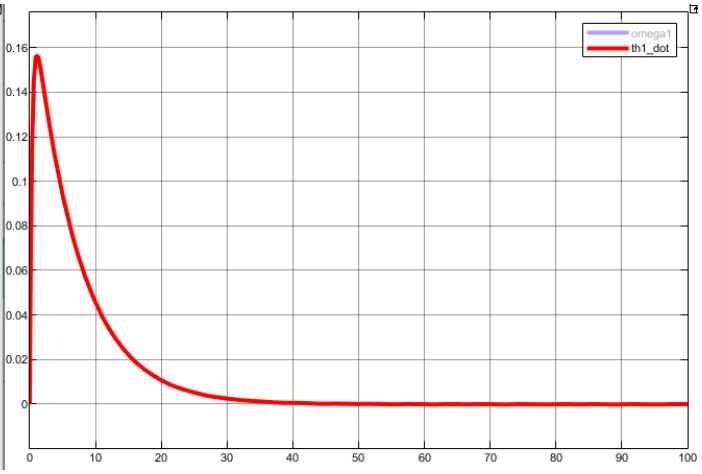
```
TR = [0 2];%time range
X0 = [0;0;0;0;0;0];%initial conditions
[t,z] = ode45(@func_geared, TR, X0);
%placing array as vectors
theta1 = z(:, 1);
AngVel1 = z(:, 2);
theta2 = z(:, 3);
AngVel2 = z(:, 4);
theta3 = z(:, 5);
AngVel3 = z(:, 6);
%plotting all parameters
A1= diff(AngVel1)
A2= diff(AngVel2)
A3= diff(AngVel3)
plot(t,theta1,t,AngVel1,t,[0;A1],t,theta2,t,AngVel2,t,[0;A2],t,theta3,t,AngVel3,t,
[0;A3])
legend('Theta1','AngVel1','AngAcc1','theta2','AngVel2','AngAcc2','theta3','AngV
el3','AngAcc3')
function dx = func_geared(t,x)
TS=7;
dx(1) = x(2);
dx(3) = x(4);
dx(5) = x(6);
dx(2) = (TS-40*x(2)-5.5*x(1))/13;
dx(4) = (TS-55*x(3)-400*x(4))/130;
dx(6) = (2*TS-400*x(6)-55*x(5))/130;
dx=dx';
end
```

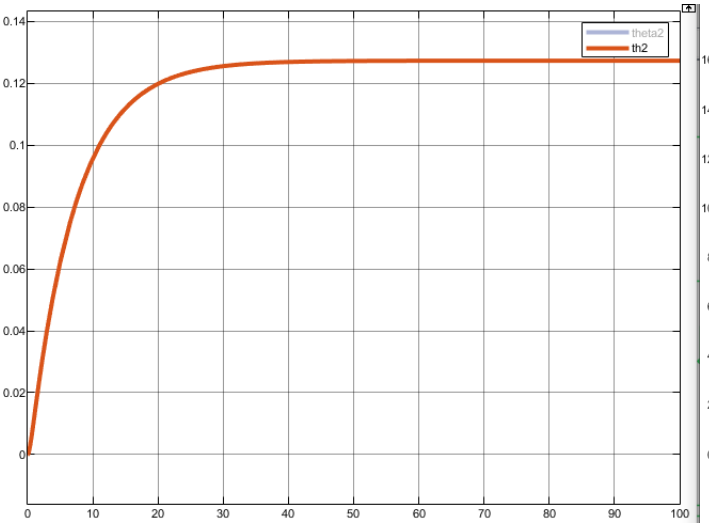
Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

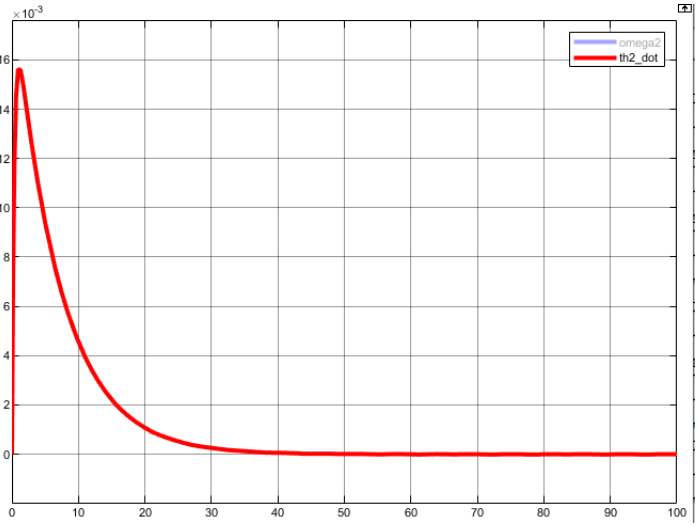
Simulink Model

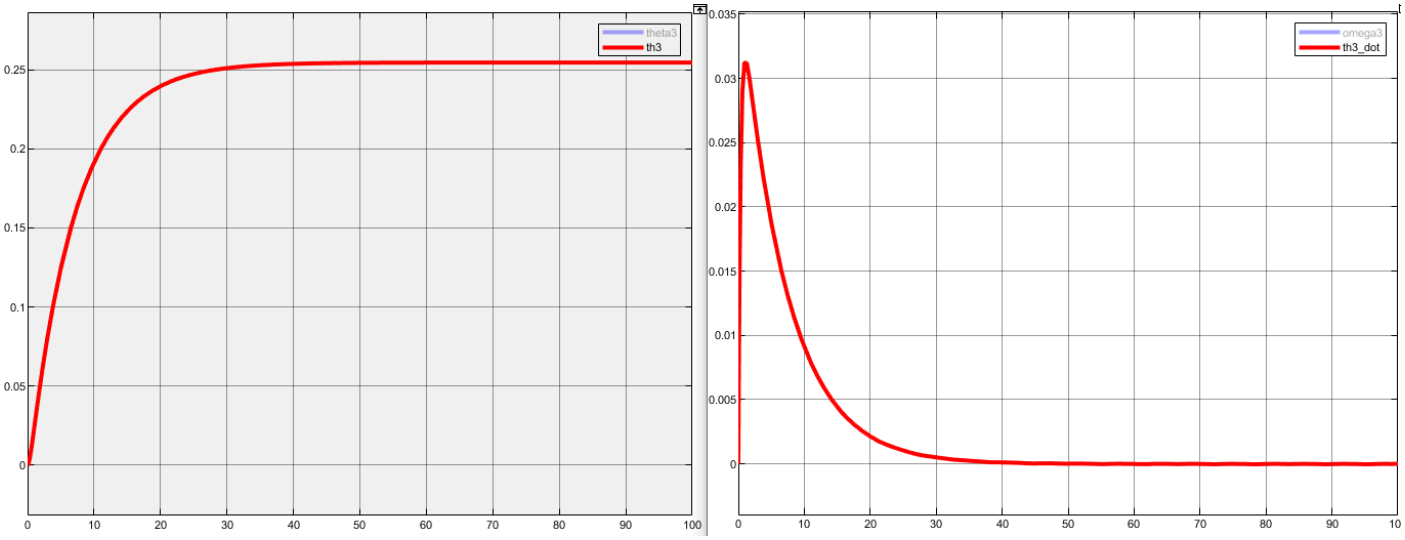




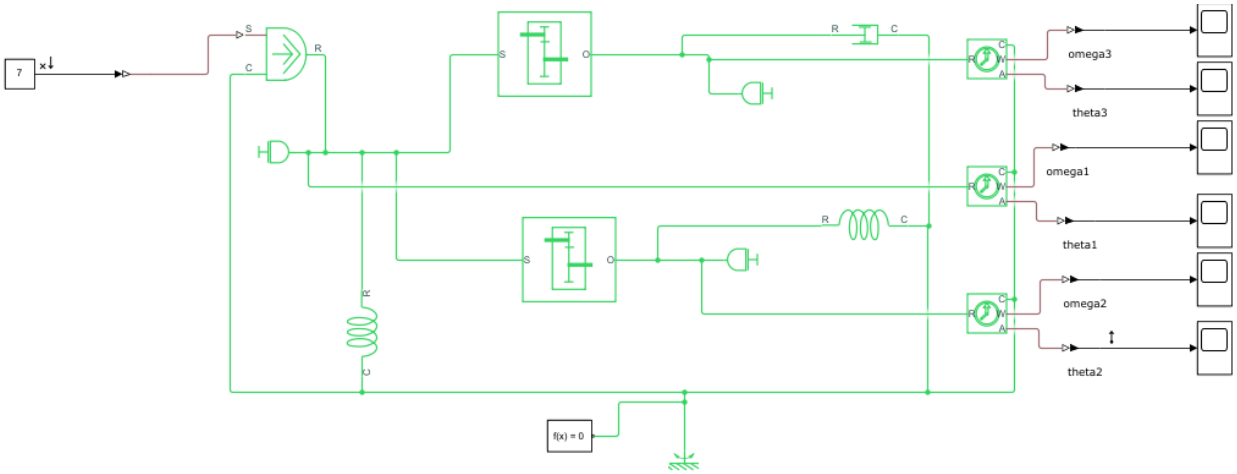




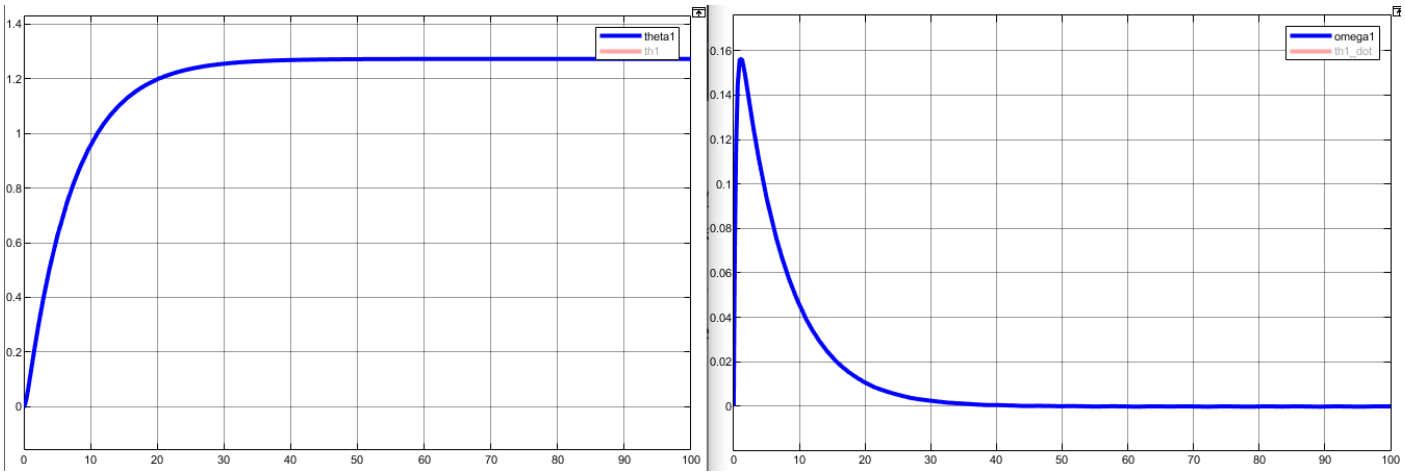


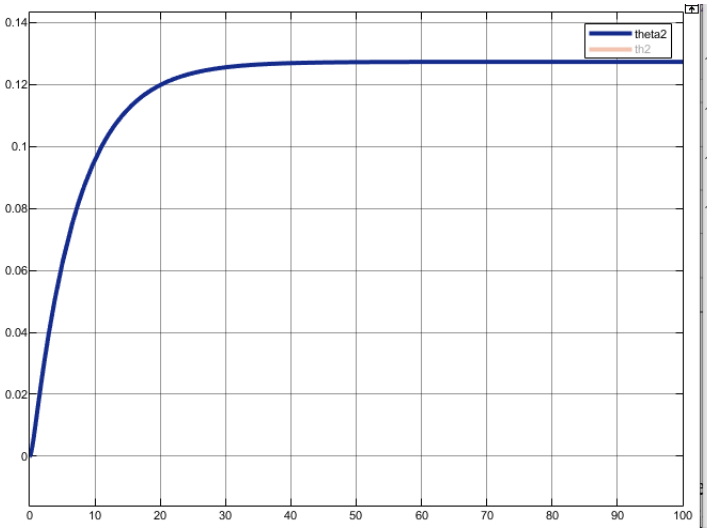


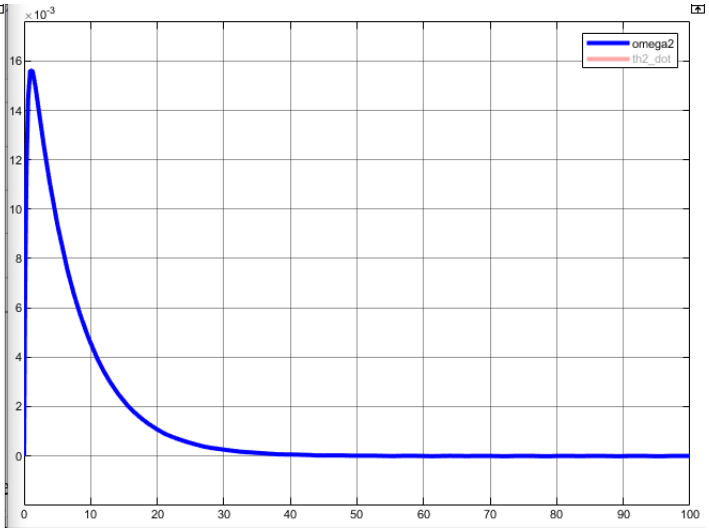
SIMSCAPE model:

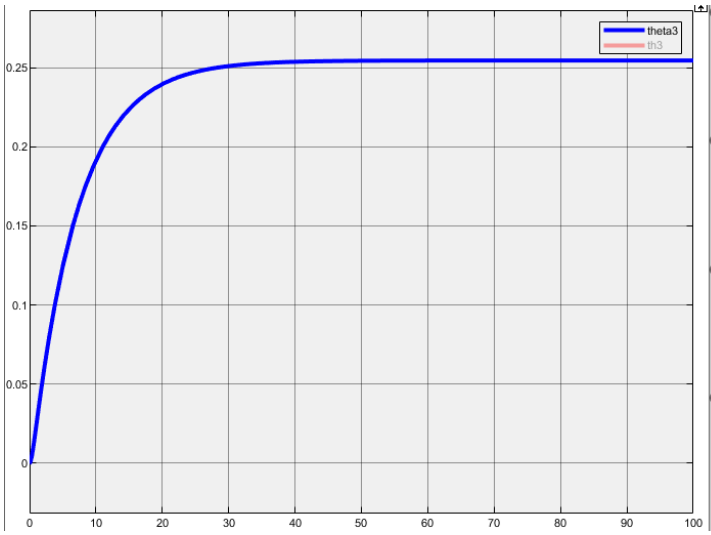


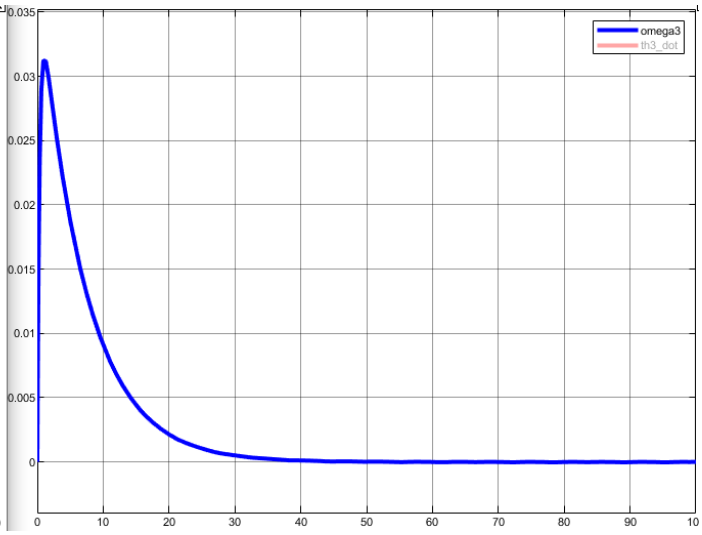
Results:











Conclusion/Remarks:

The results obtained by both Simulink as well as SIMSCAPE are the same. Hence the developed SIMSCAPE model is correct and is following the mathematical model (Simulink model).

The output obtained shows the effect of stiffness and inertia on the geared system. In this case, by adding stiffness in the rod and an inertial element, the gears no longer exhibit a linearly increasing displacement pattern rather the gears rotate for some time, and then the opposing force due to inertia counterbalances the applied torque causing the displacement to become zero (as is evident by the graphs).

Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "Constant" to output "theta2":
         0.007692
  ----------------------
  s^2 + 3.077 s + 0.4231

Name: Linearization at model initial condition
Continuous-time transfer function.
```

Transfer function from m file:

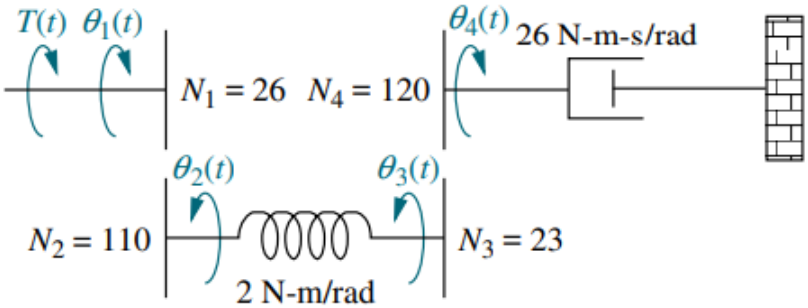
```
syms s T
theta1=T/(13*s^2+40*s+5.5);
theta2=theta1/10;
G=theta2/T;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);      %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
```

```
fprintf('Theta2(s)/T(s):')
tf(num,den)
```

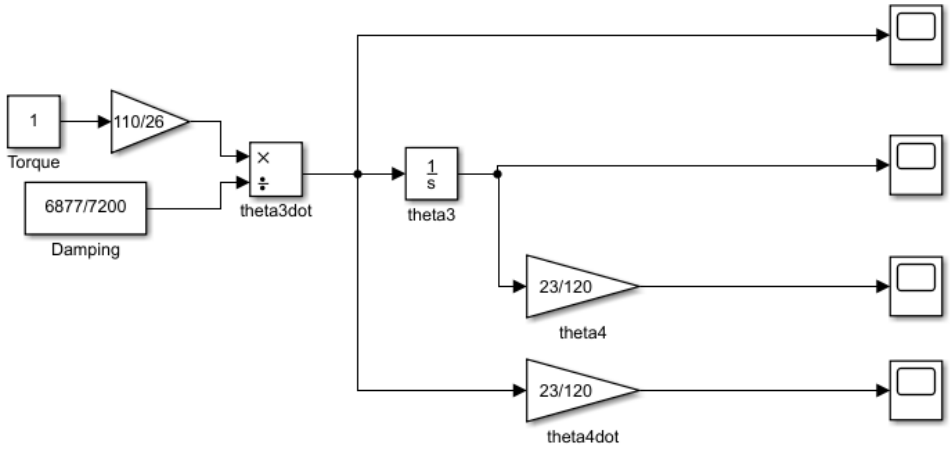
Following is the output:

```
Theta2(s)/T(s):
ans =

          0.007692
  ----------------------
  s^2 + 3.077 s + 0.4231

Continuous-time transfer function.
```

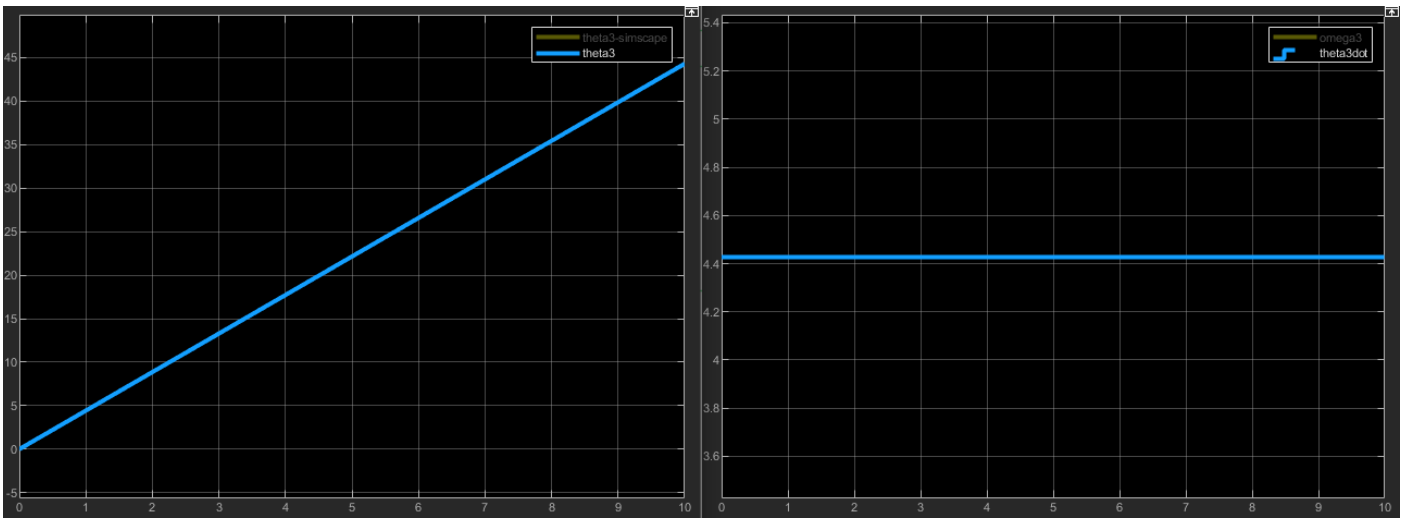
Example#03: Develop a SIMSCAPE model of the given system. The transfer function should be between $\theta 4$ and Applied Torque T(t). You are required to cross-validate the outcomes with the Simulink of this model.

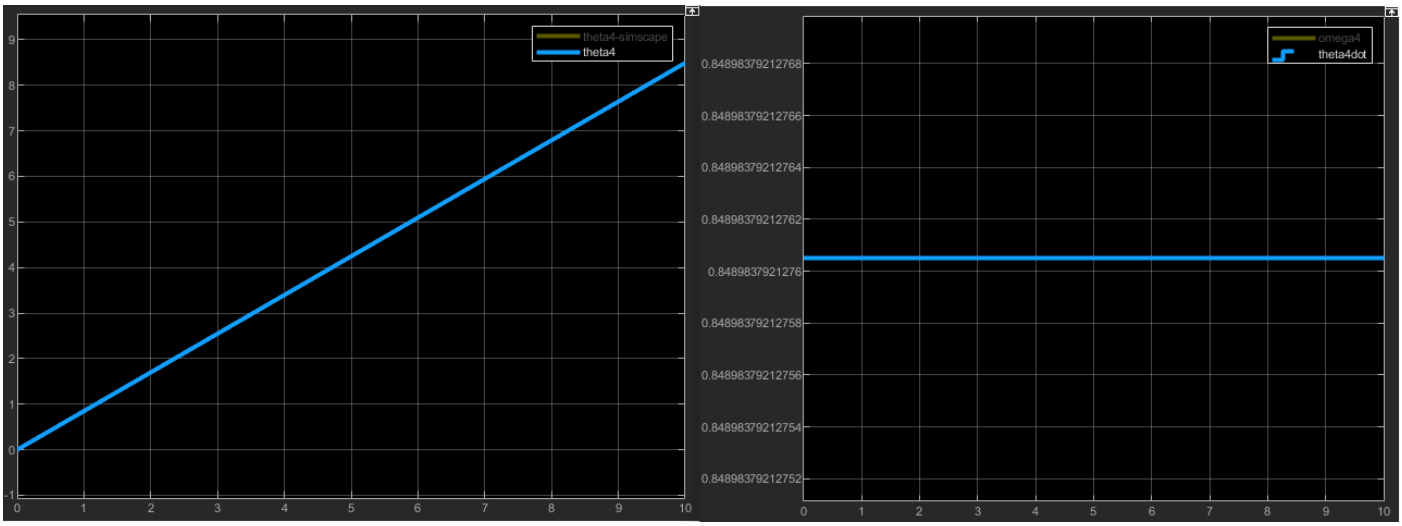


Simulink model:



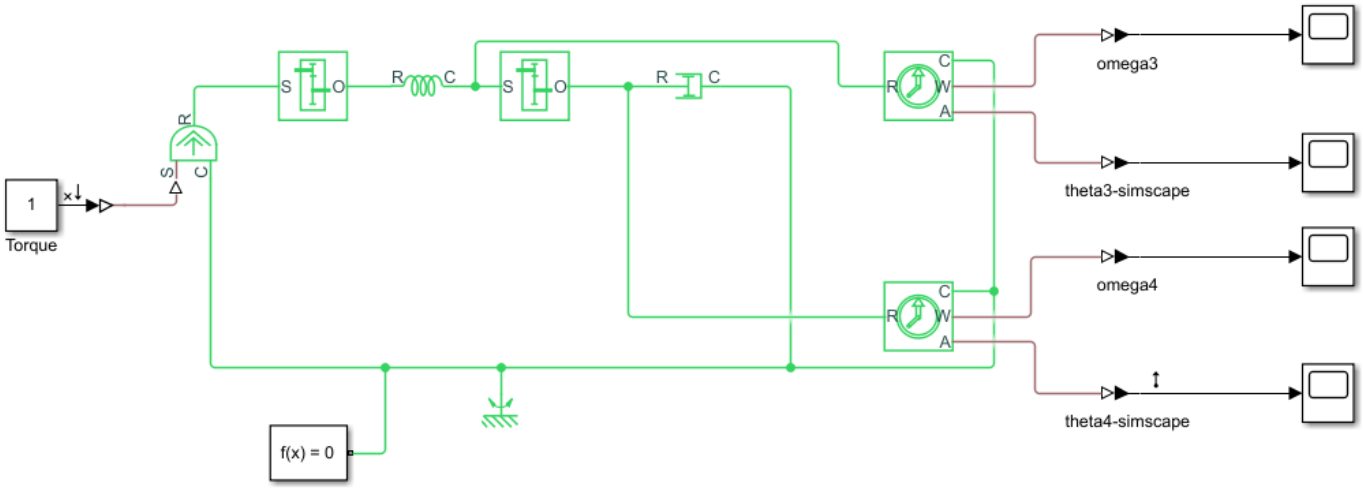
Results:



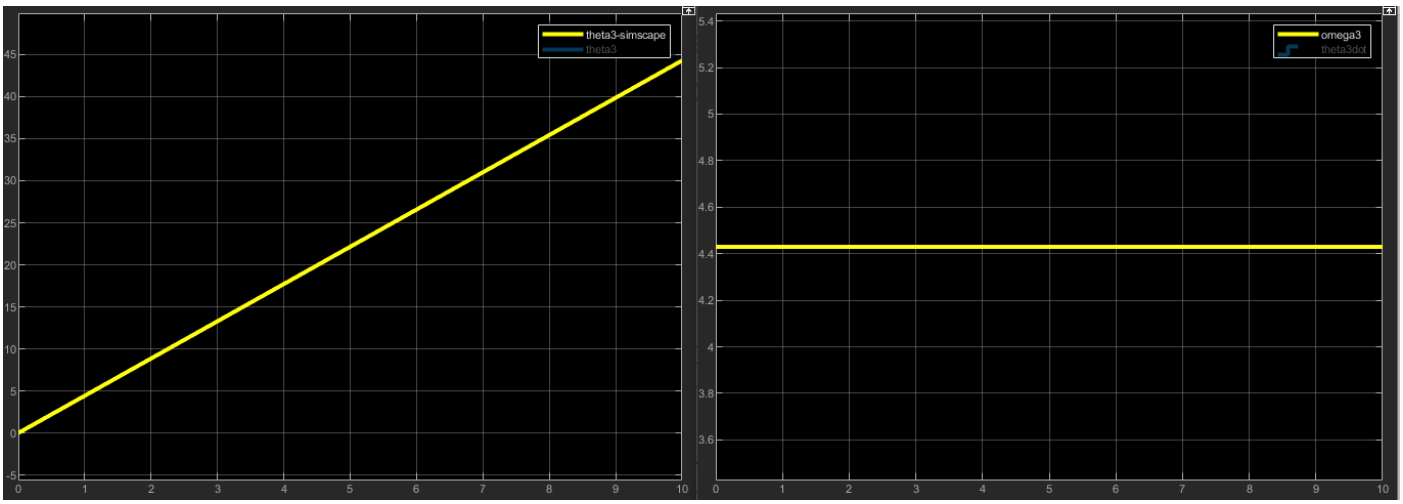


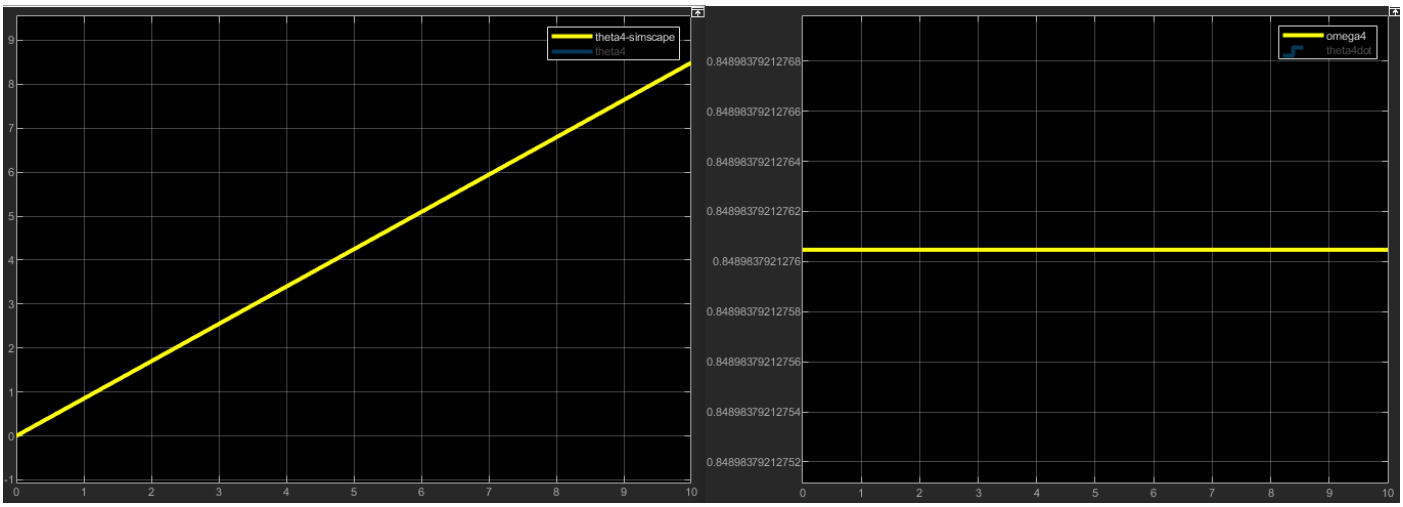
SIMSCAPE model:

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan



Results:





Conclusion/Remarks:

The results obtained by both Simulink as well as SIMSCAPE are the same. Hence the developed SIMSCAPE model is correct and is following the mathematical model (Simulink model). The graphs obtained are logical as well; the rotational output is measured across the gear which ideally rotates continuously maintaining a

constant angular velocity (by the law of gearing). This phenomenon is also observed in the plots above which exhibit a linearly increasing trend for angular displacement and a constant trend for angular velocity.

Transfer Function from SIMSCAPE model:

```
ans =

  From input "Torque" to output "theta4-simscape":
  0.849
  -----
    s

Name: Linearization at model initial condition
Continuous-time transfer function.
```

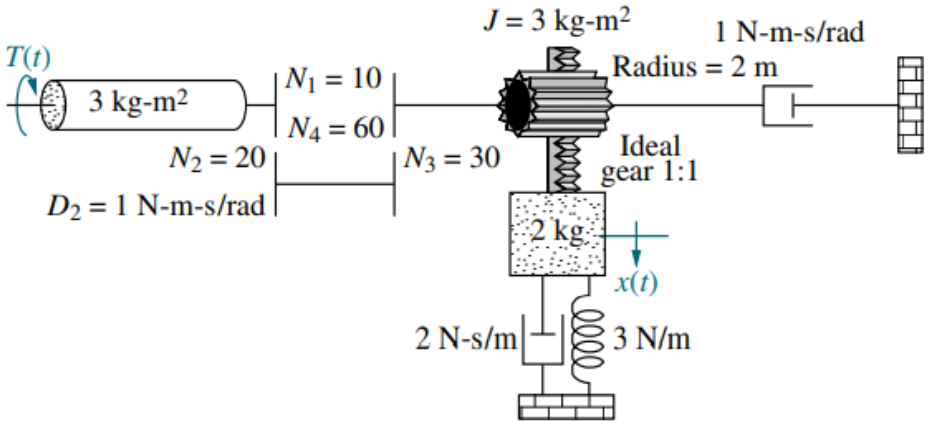
Transfer function from m file:

```
syms s T
k=2;d=6877/7200;
A=[k , -k ; -k , (d*s+k)];
B=[110/26*T;0];
C=A\B;
theta3=C(2);
theta4=theta3*23/120;
G=theta4/T;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of
denominator i.e., 20 in this case. Done to match the outputs
den=den/den(1);
fprintf('theta4(s)/T(s):')
tf(num,den)
```

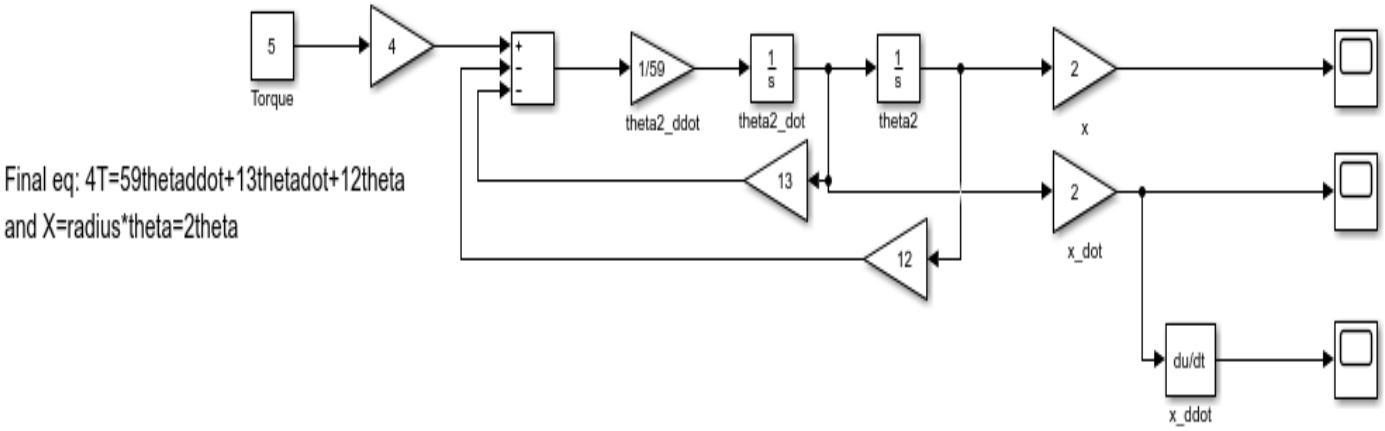
```
ans =

  0.849
  -----
    s

Continuous-time transfer function.
```

Example#04

Develop a SIMSCAPE model of the given system. The transfer function should be between $X(t)$ and Applied Torque T(t). You are required to cross-validate the outcomes with the Simulink as well as the m-files of this model.

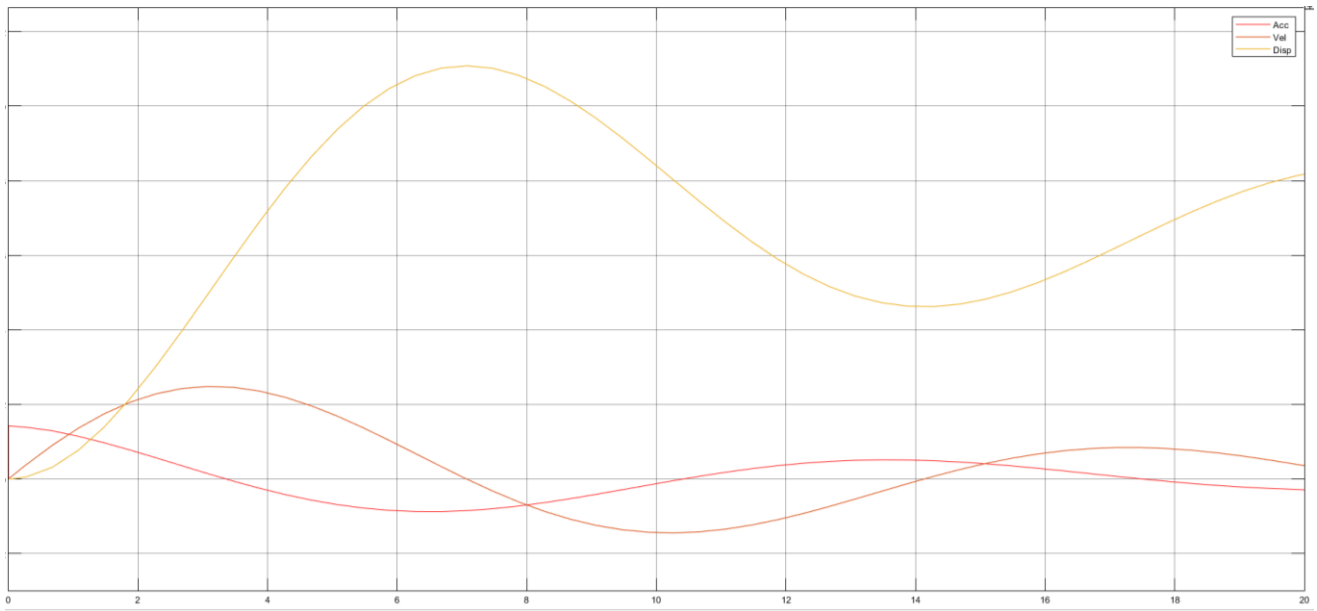


Simulink model:

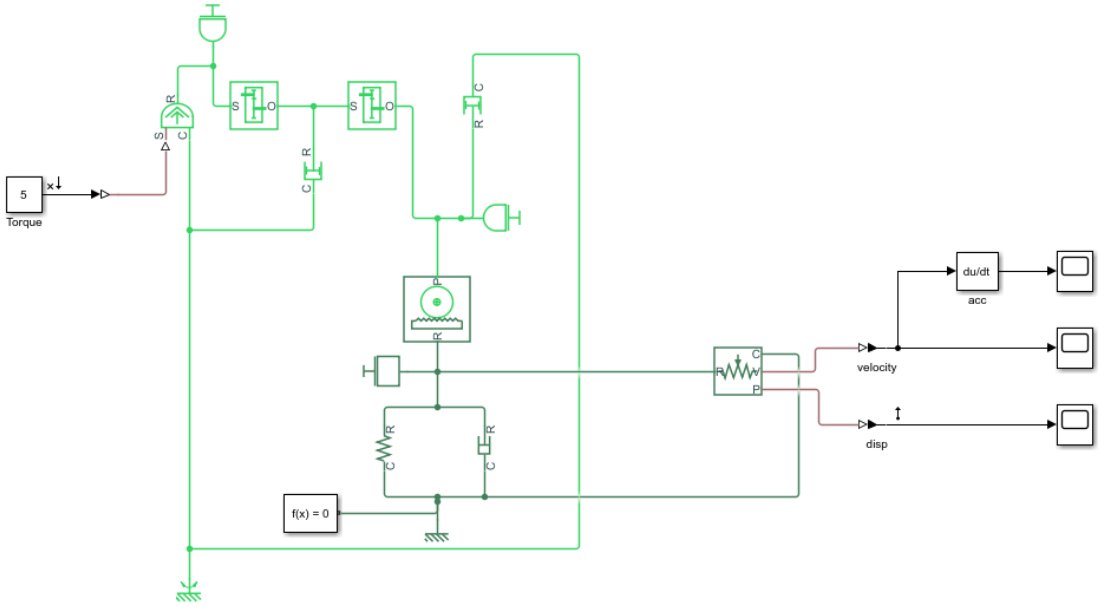


Final eq: 4T=59thetaddot+13thetadot+12theta
and X=radius*theta=2theta

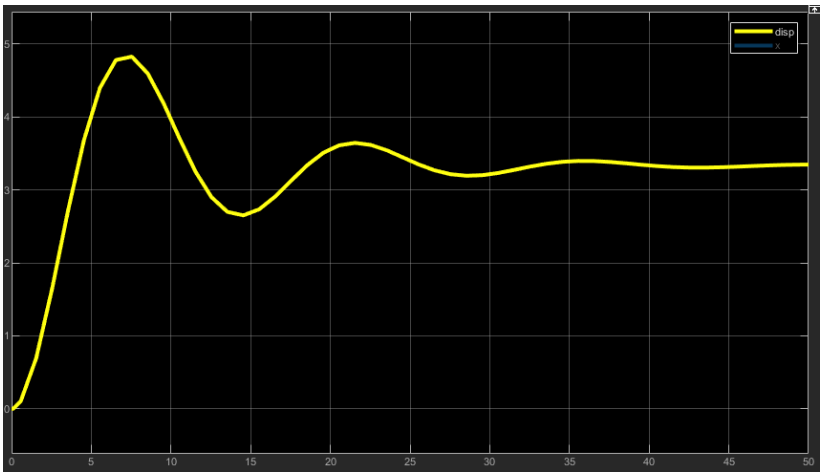
Results:

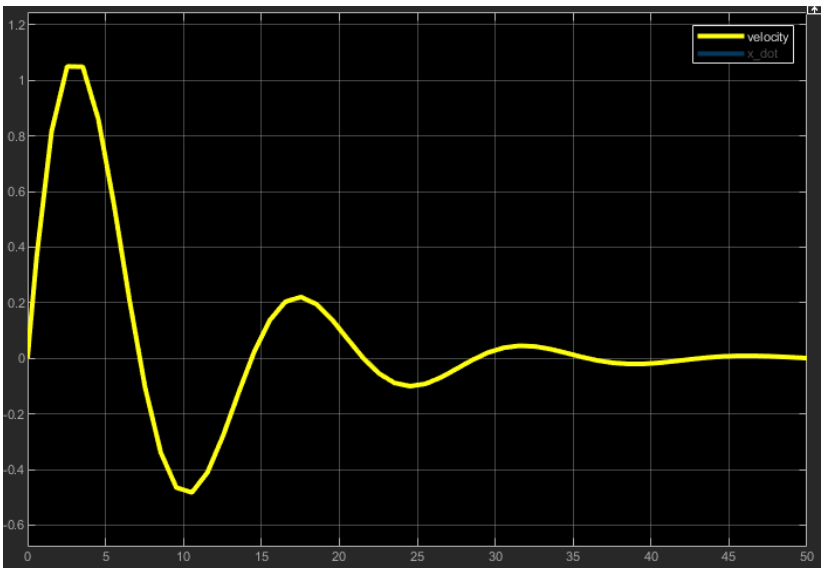


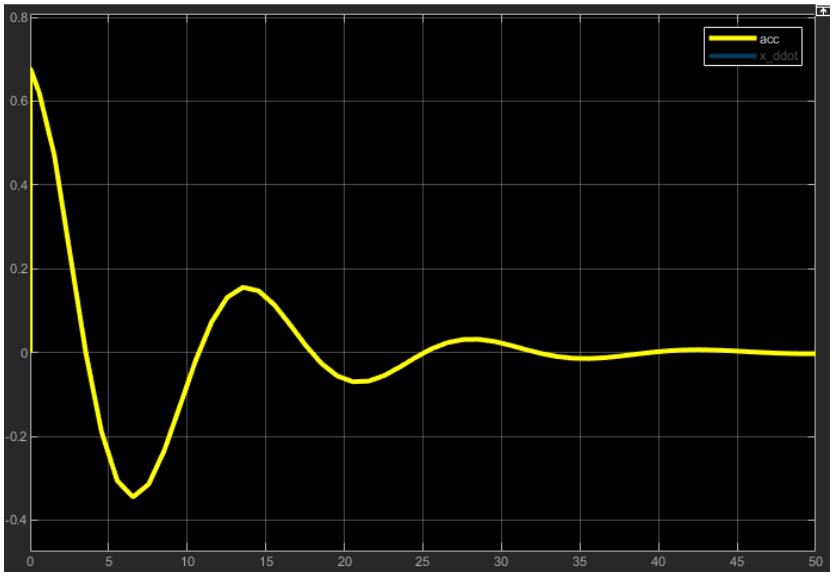
SIMSCAPE model:



Results:

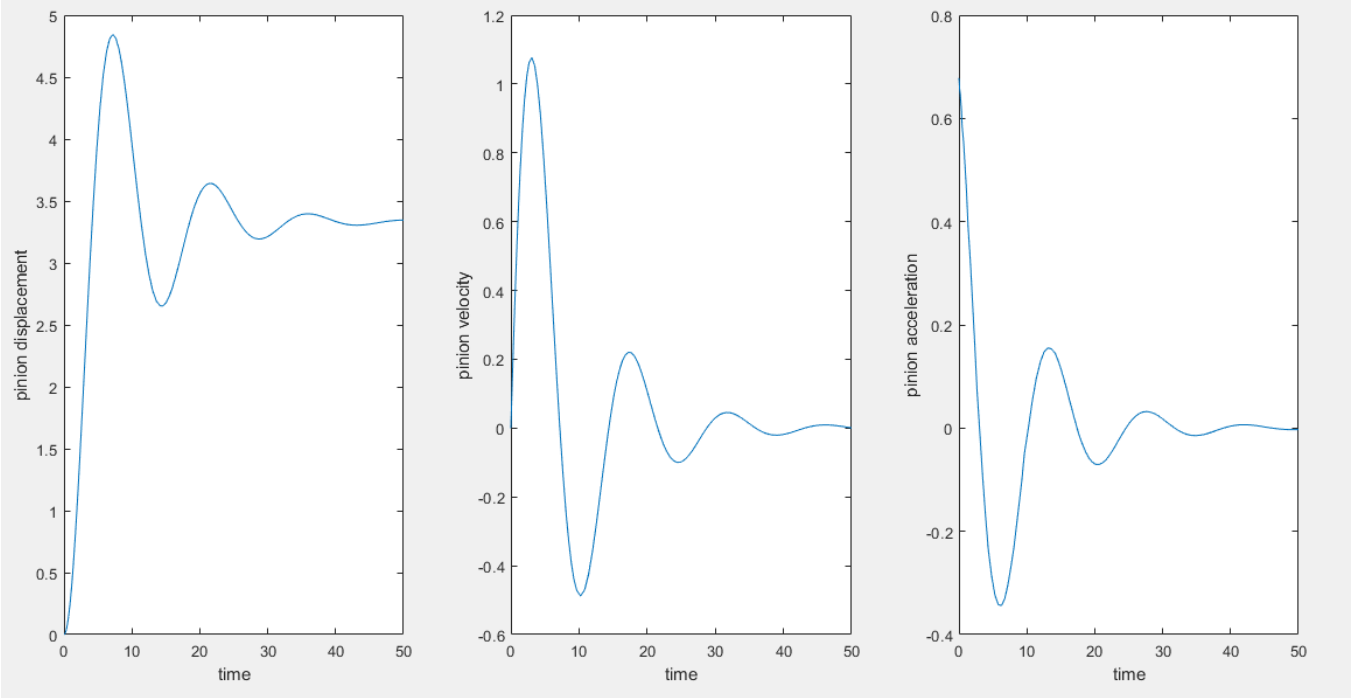






m file result:

```
TR=[0,10];
th0=[0;0];
[t,y]=ode45(@RackNPinion, TR, th0);
theta=y(:,1);
theta_dot=y(:,2);
x=2*theta;
x_dot=2*theta_dot;
x_ddot=gradient(x_dot)./gradient(t);
subplot(1,3,1);plot(t,x);
xlabel('time');ylabel('pinion displacement');
subplot(1,3,2);plot(t,x_dot);
xlabel('time');ylabel('pinion velocity');
subplot(1,3,3);plot(t,x_ddot);
xlabel('time');ylabel('pinion acceleration');
%%%%%%%%%%
function dy=RackNPinion(t,y)
    T=5;
    dy(1)=y(2);
    dy(2)=1/59*(4*T -13*y(2) - 12*y(1));
    dy=dy';
end
```



Conclusion/Remarks:

The results obtained by Simulink, m file as well as the SIMSCAPE are exactly the same. Hence the developed SIMSCAPE model is correct and is in accordance with the mathematical model (Simulink model).

The model has a rack and pinion which converts rotatory input to linear output hence the mass displaces from its initial position. As the displacement and velocity of the mass as well as the pinion increases the resisting forces due to the stiffness and damping also increases and after some time, they are large enough to counteract the input. Hence the system locks itself into that static position. This is evident by the plot which shows that initially the displacement, velocity and acceleration of the mass shoot to a high value and after some time the displacement becomes steady and velocity and acceleration become zero.

Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "Torque" to output "disp":
          0.1356
  -----------------------
  s^2 + 0.2203 s + 0.2034

Name: Linearization at model initial condition
Continuous-time transfer function.
```

Transfer function from m file:

```
syms s T
radius=2;
theta=4*T/(59*s^2+13*s+12);
x=2*theta;
G=x/T;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);      %dividing by den(1) means dividing by the leading coefficient
of denominator i.e., 20 in this case. Done to match the outputs
den=den/den(1);
fprintf('X(s)/T(s):')
tf(num,den)
```

Following is the output:

```
X(s)/T(s):
ans =

          0.1356
  -----------------------
  s^2 + 0.2203 s + 0.2034

Continuous-time transfer function.
```

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

LAB 3: Modeling, Analysis, and Cross-Validation of the Passive and Active Electrical Systems.

Aim and Objectives:

The fundamental aim of this lab is to imply the studied concepts of SIMSCAPE to model and analyze the various electrical Systems. Furthermore, the outcomes of the dynamic systems have been validated with the findings of SIMULINK and ODE SOLVERS. Therefore, this lab is primarily dedicated to both active as well as passive categories of electrical systems, and certain tangible objectives are enlightened below:

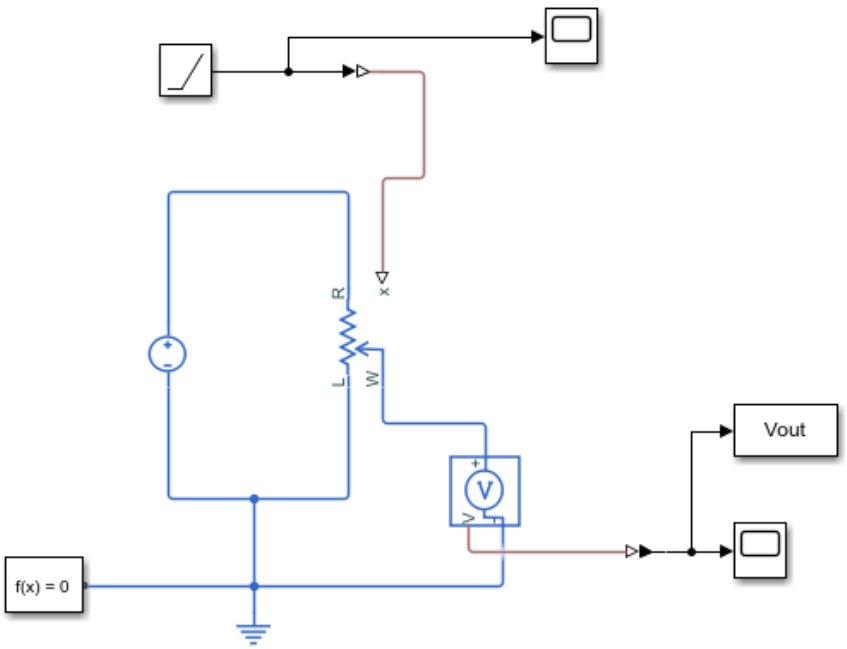
1. To model and analyze the dynamic response of active and passive electrical systems.
2. To obtain the transfer function from developed SIMSCAPE models.
3. To cross-validate the outcomes of SIMSCAPE Models with the results obtained from ODEs 45 and SIMULINK files.
4. To model, analyze, and cross-validate the dynamic response of Buck and Boost Converters.

Assigned Tasks:

Model 1:

Model of Potentiometer. You are required to cross-validate the findings with the Mathematical formula. This cross-validation must be in the graphical form (Theta vs the Output Voltage). Two Lines should be on that graph entitled SIMSCAPE Results and Mathematical Findings.

SIMSCAPE model:



Results:

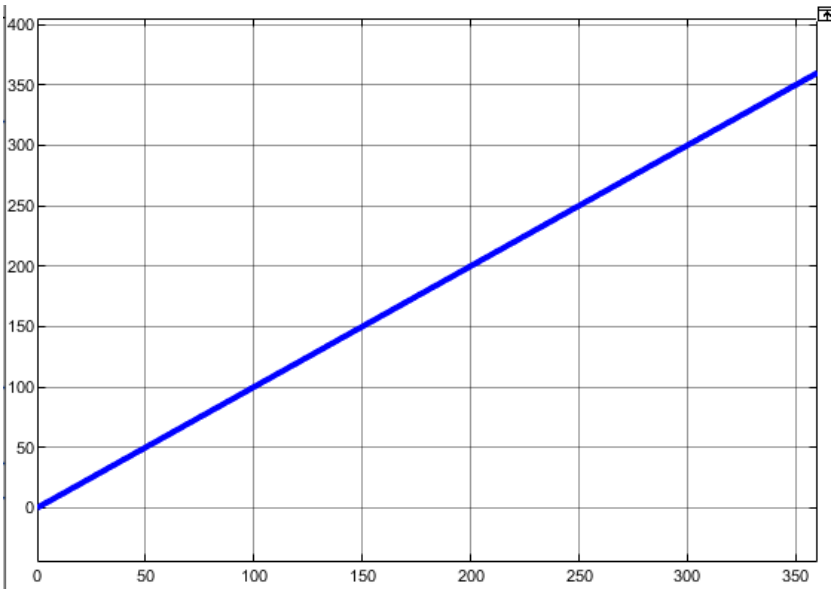


Figure: Potentiometer Theta vs Time

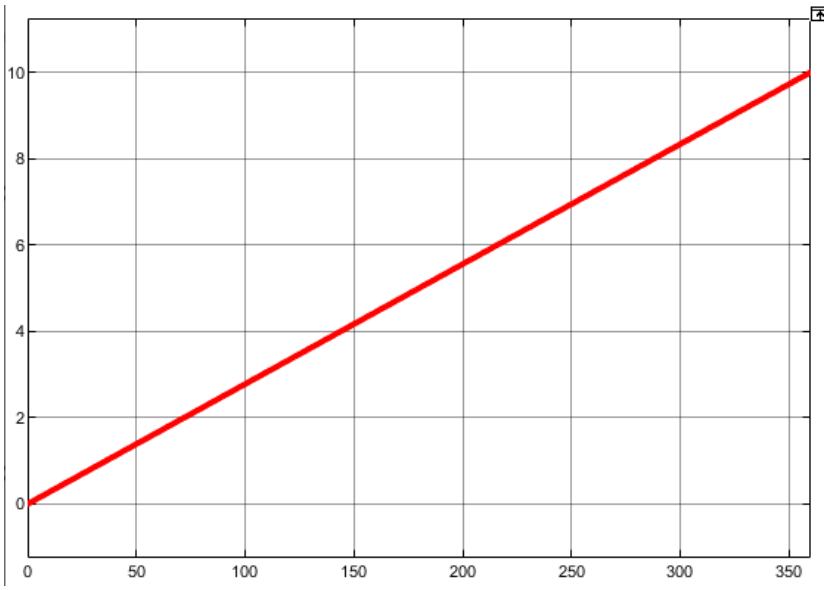


Figure: Output Voltage vs Time

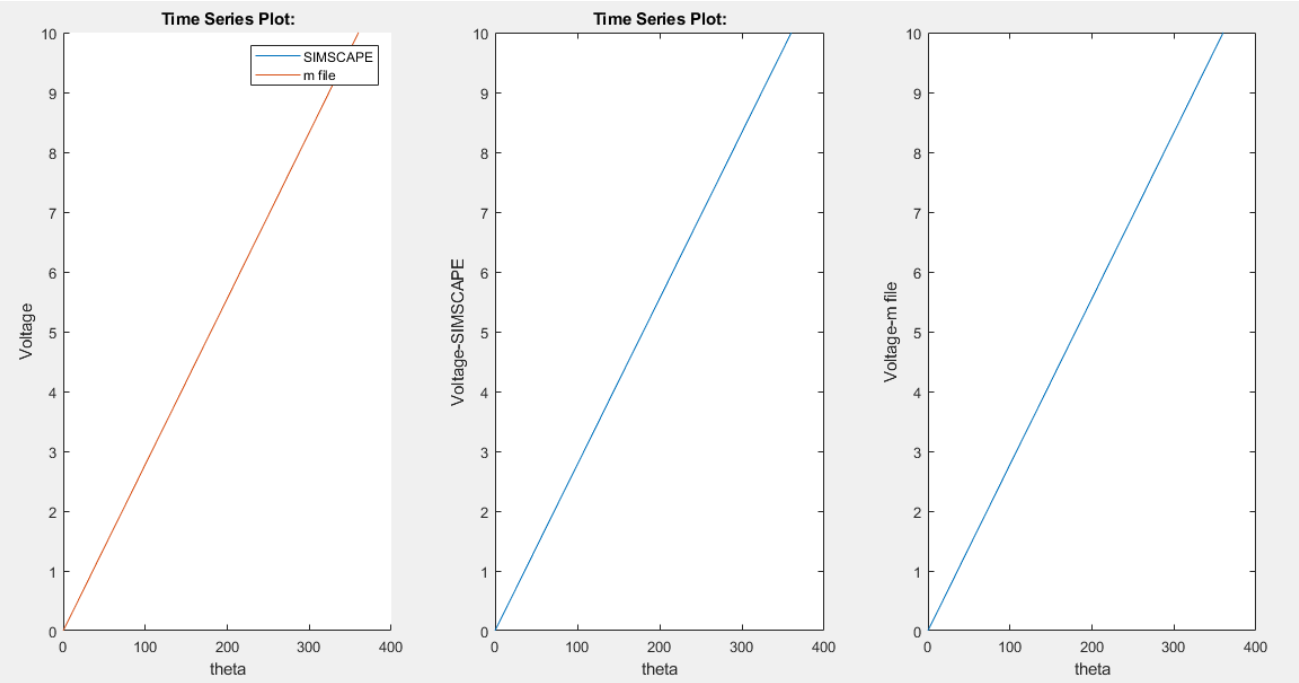
Conclusion/Remarks:

It can be clearly seen that when the potentiometer knob is rotated from 0 to 360 degrees, the output voltage varies from 0 to 10 volts (from 0 to applied voltage). This result is consistent with the actual behavior of rotary potentiometer.

M File results:

```
clc;
subplot(1,3,1);
hold on
plot(out.Vout);
theta=linspace(0,360,length(out.tout));
Vin=10;
V=Vin*theta/360;
plot(theta,V);
xlabel('theta');ylabel('Voltage');
legend('SIMSCAPE', 'm file');
subplot(1,3,2);
plot(out.Vout);xlabel('theta');ylabel('Voltage-SIMSCAPE');
subplot(1,3,3);
plot(theta,V);xlabel('theta');ylabel('Voltage-m file');
```

In this m file, the output is taken from the SIMSCAPE model and compared with the mathematical equation of the potentiometer. The result is then plotted for comparison. Note that in order for this m file to work, the SIMSCAPE model must be executed first. Following are the results.



Conclusion/Remarks:

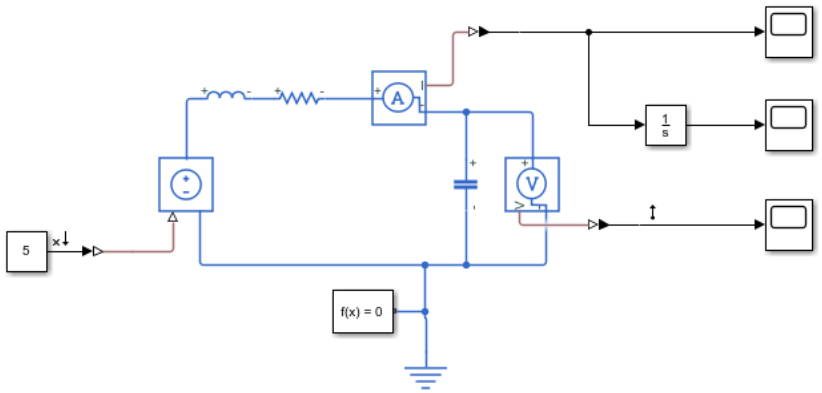
It can be clearly seen that the plots obtained by SIMSCAPE results and by the mathematical model (m-file) are exactly the same and also validate the real life behavior of a potentiometer.

Model 2:

Model, Analyze, and Cross-Validate (Either Through m-file or SIMULINK: An Open Choice) the response of Series and Parallel RLC Circuits.

a) Series RLC circuit:

SIMSCAPE model:



Results:

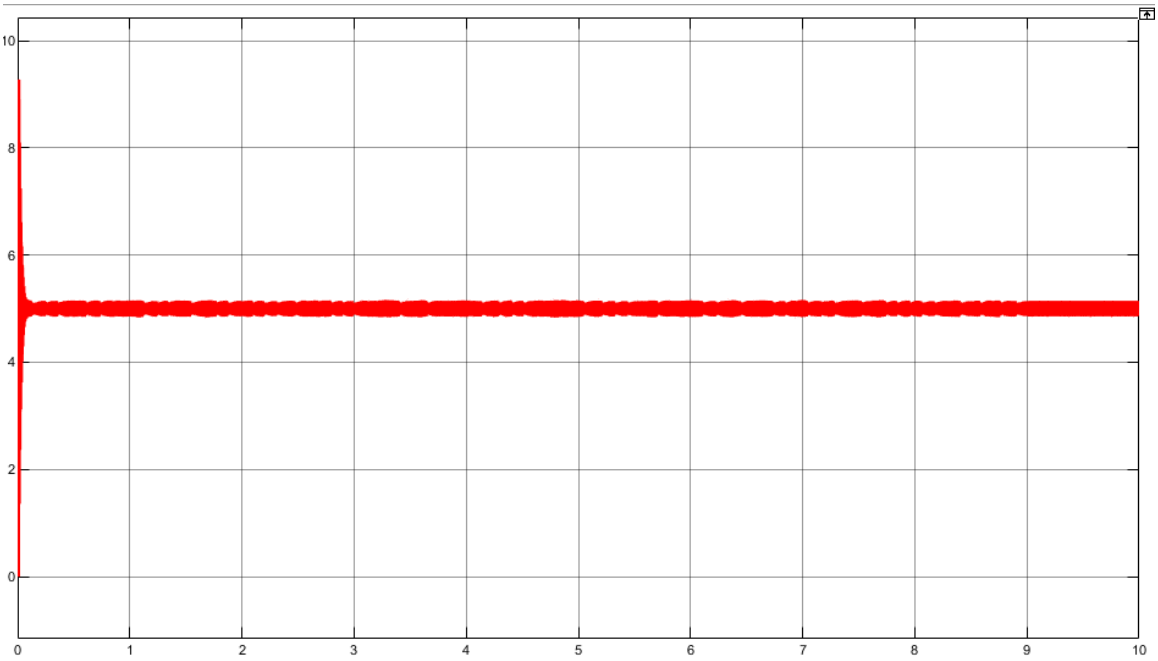


Figure: Capacitor Voltage vs Time

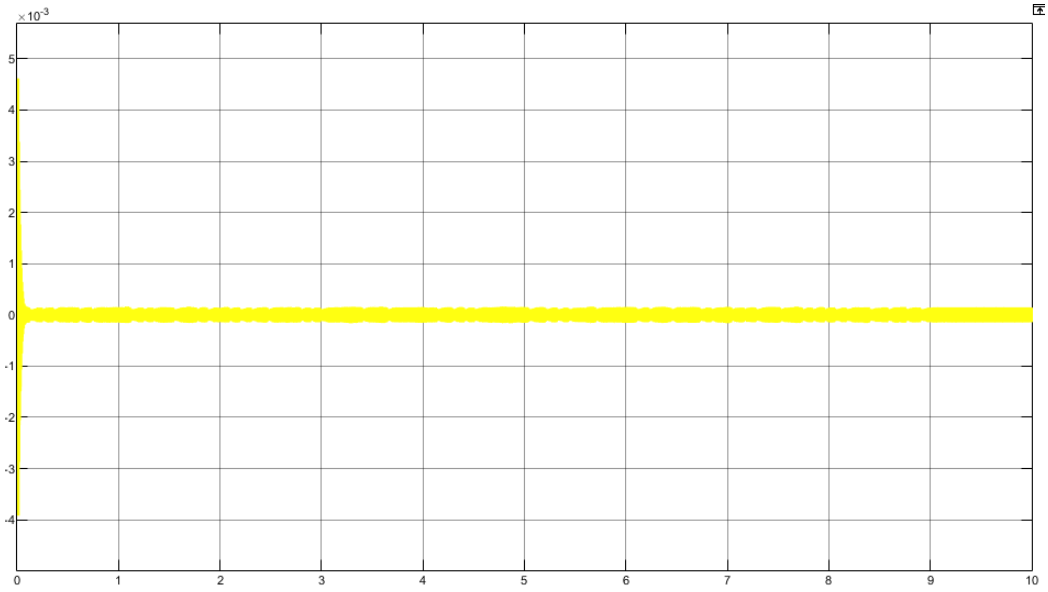


Figure: Current vs Time

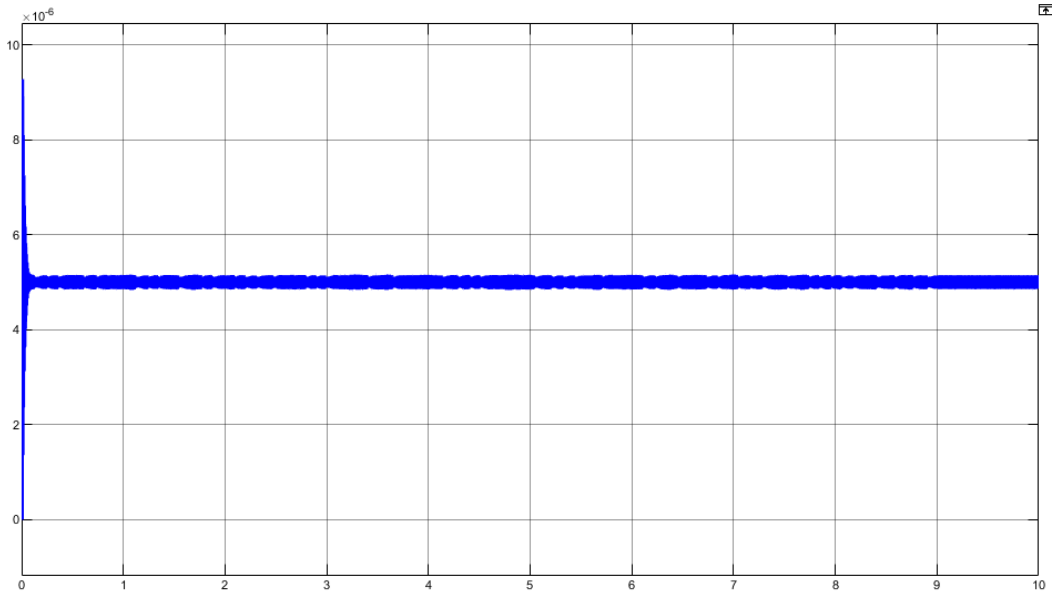
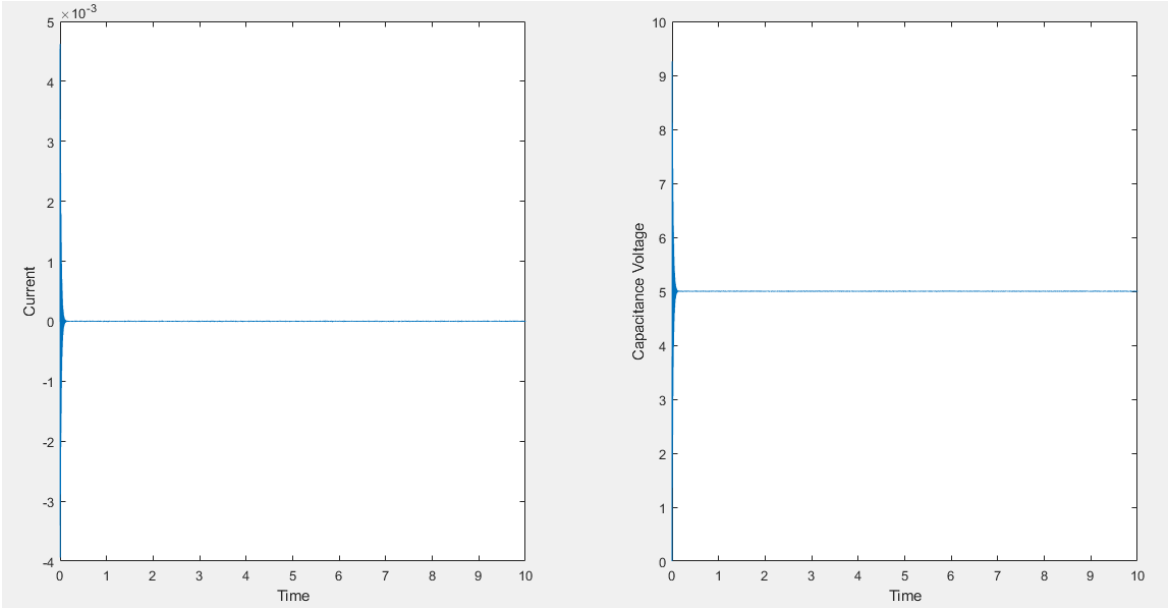


Figure: Charge vs Time

M File results:

```
TR=[0,10];
I0=[0;0];
C=1*10^-6;
[t,i]=ode45(@Series,TR,I0);
I=i(:,2);
q=i(:,1);
V=q/C;
subplot(1,2,1);
plot(t,I);
xlabel('Time');ylabel('Current');
subplot(1,2,2);
plot(t,V);
xlabel('Time');ylabel('Capacitance Voltage');

function di=Series(t,i)
    V=5;R=100;L=1;C=10^-6;
    di(1)=i(2);
    di(2)=1/L*(V - i(1)/C - R*i(2));
    di=di';
end
```



Conclusion/Remarks:

The graphs obtained are quite logical and strengthen the theoretical concepts. Firstly, we know that capacitor blocks DC current once it charges. So, when it charges in 10^-4 secs time (t=R*C=100*10^-6=10^-4 sec), the current becomes zero and circuit becomes open so all the voltage accumulates across capacitor. That's why we get 0 current after 10^-4 secs and approximately constant 5 volts. But then if no current flows, why do we get charge? The reason is again the capacitance. The capacitor accumulates charge across it given by the equation (Q=C*V=10^-6*5=5*10^-6 coulomb).

Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "Voltage" to output "Capacitor Volt":
         1e06
  ------------------
  s^2 + 100 s + 1e06

Name: Linearization at model initial condition
Continuous-time transfer function.
```

Transfer function from m file:

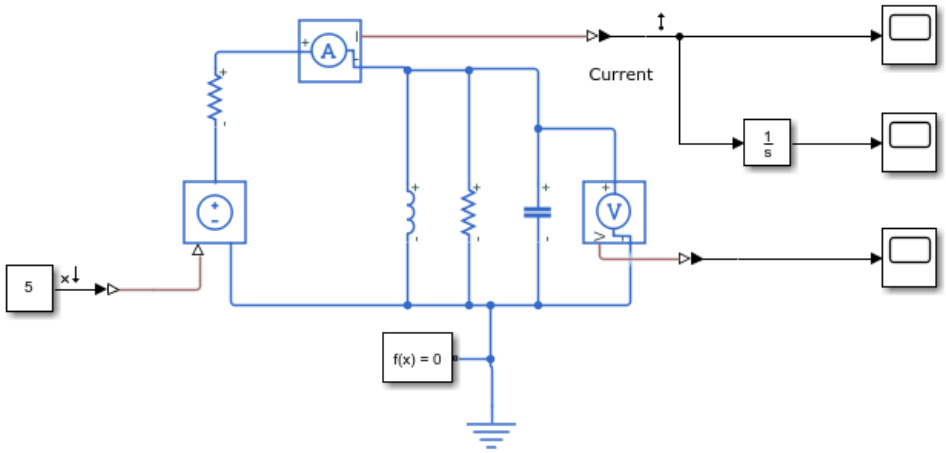
```
R=100;
L=1;
C=1*10^-6;
syms s V
i=V/(R+L*s+1/(C*s));
Vout=1/(C*s)*i;
G=Vout/V;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);      %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
fprintf('Vout(s)/Vin(s):')
tf(num,den)
```

Following is the output:

```
>> SeriesRLCTrans
Vout(s)/Vin(s):
ans =

          1e06
  ------------------
  s^2 + 100 s + 1e06

Continuous-time transfer function.
```

b) <u>Parallel RLC circuit:</u>

SIMSCAPE model:



Results:

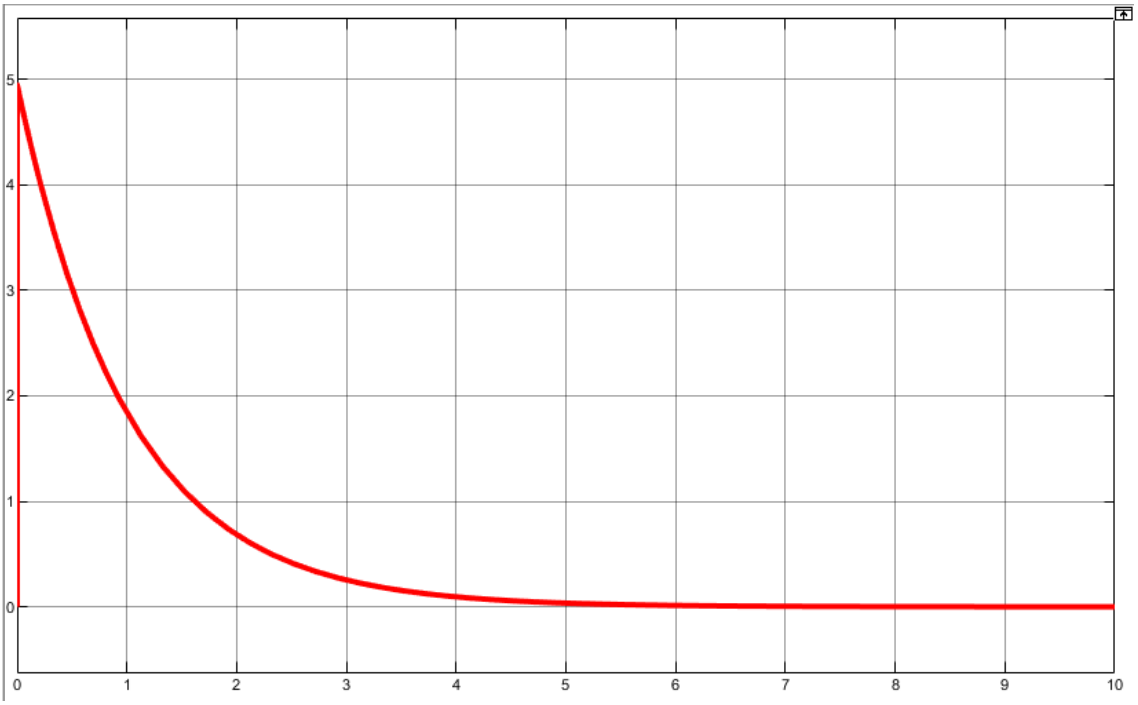


Figure: Capacitor Voltage vs Time

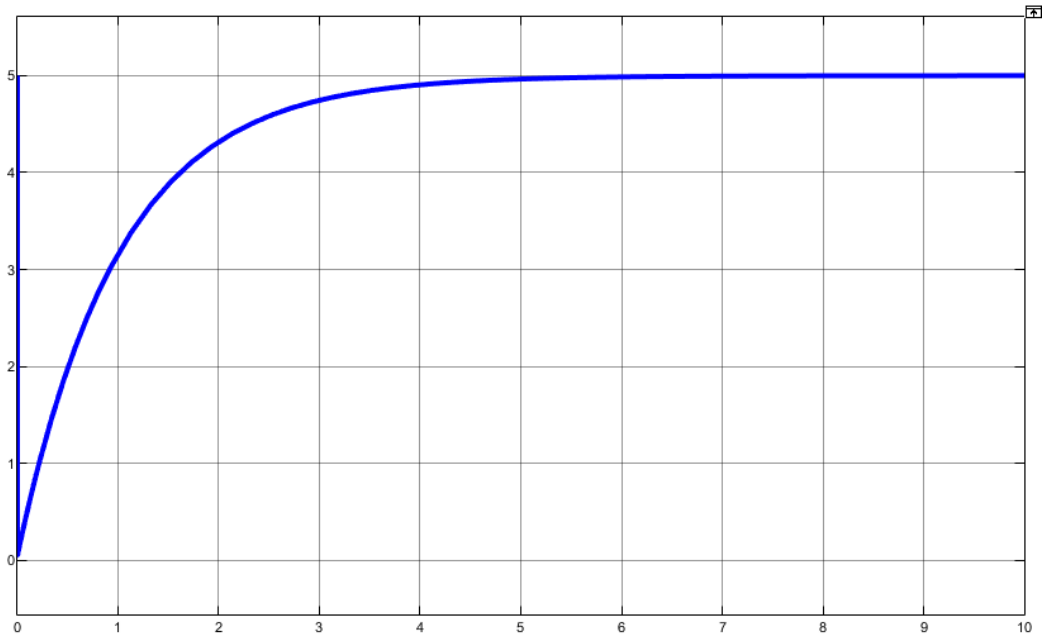


Figure: Current vs Time

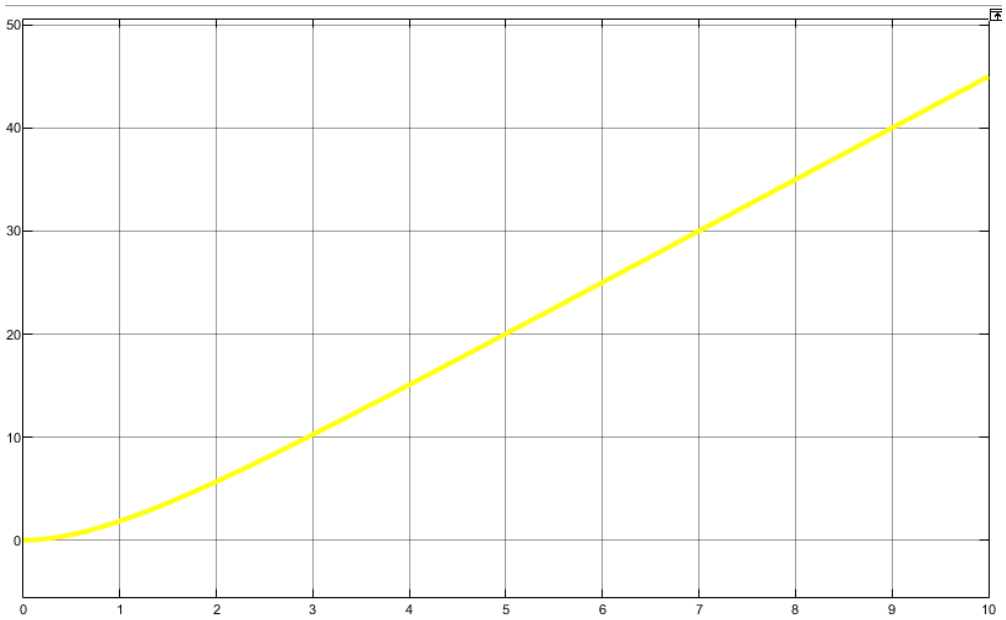


Figure: Charge vs Time

Conclusion/Remarks:

The graphs obtained are quite logical and strengthen the theoretical concepts. Firstly, we know that capacitor acts as an open while inductor acts as a short circuit in DC circuits. So, the circuit reduces to just a resistor of 1 ohm at the start and a resistor of 100 ohm with a short and an open circuit in parallel. The short circuit shorts the entire thing and we are essentially left with a 1-ohm resistor only. That's why we get 0 volts across the capacitor (due to short circuit in the branch) and 5 amps current (I=V/R=5/1). The charge in this case shows a linear trend (integral of constant current curve gives linear charge).

Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "Constant" to output "Current":
   s^2 + 10000 s + 1e06
  ----------------------
  s^2 + 1.01e06 s + 1e06

Name: Linearization at model initial condition
Continuous-time transfer function.
```

Transfer function from m file:

```
R1=100;
R2=1;
L=1;
C=1*10^-6;
syms s V
Induc=L*s;
Cap=1/(C*s);
Z=(R1*Induc)/(R1+Induc);
Z2=(Z*Cap)/(Z+Cap);
Z3=Z2+R2;
i=V/Z3;
G=i/V;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
fprintf('I(s)/Vin(s):')
G=tf(num,den)
t=0:0.01:10;
u=5*ones(1,length(t));
lsim(G,u,t);
```

Following is the output:

```
I(s)/Vin(s):
G =

   s^2 + 10000 s + 1e06
  ----------------------
  s^2 + 1.01e06 s + 1e06

Continuous-time transfer function.
```

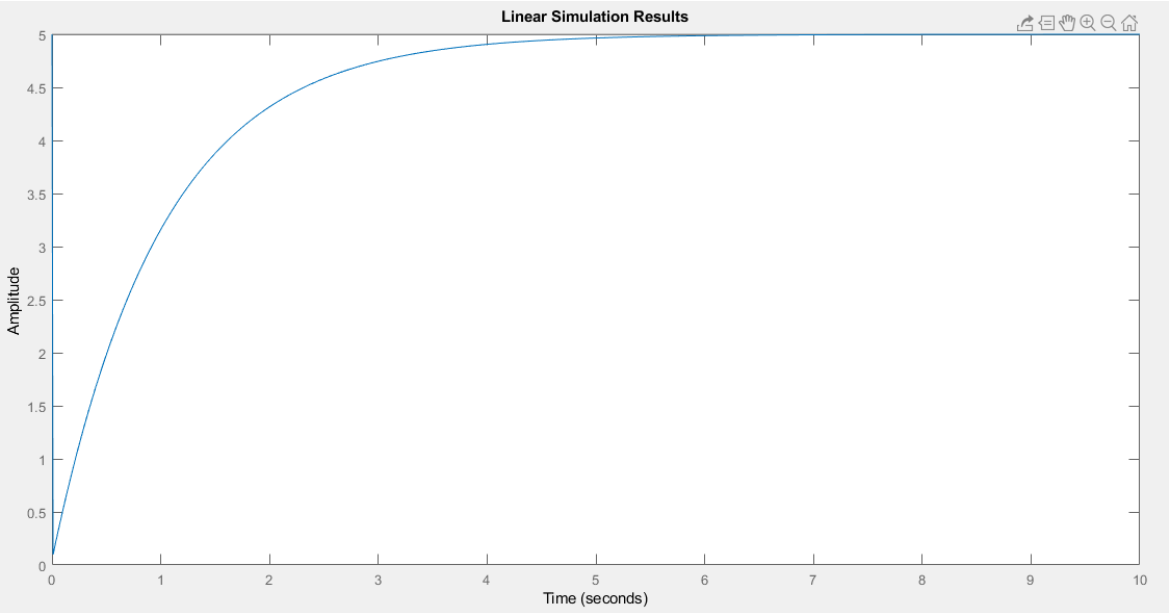


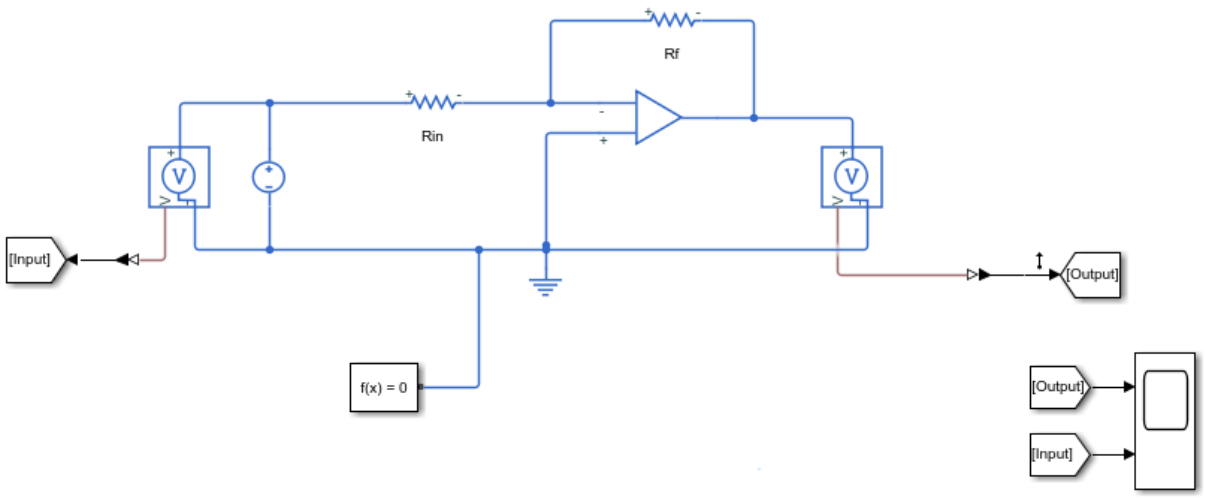
Figure: Current vs time obtained by m file

Model 3:

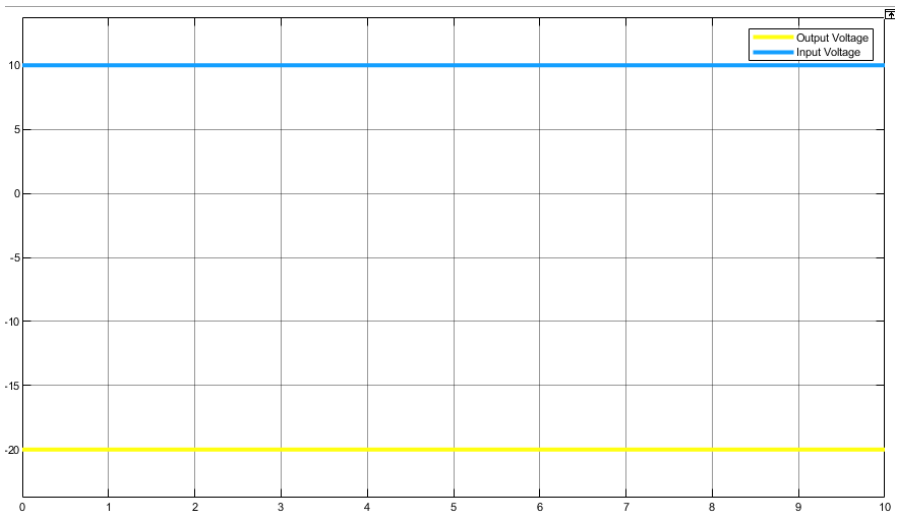
Model, Analyze, and Cross-Validate (Either Through m-file or SIMULINK: An Open Choice) the response of Inverting and Non-Inverting Operational Amplifiers.

SIMSCAPE model:

a) Inverting Op-Amp:



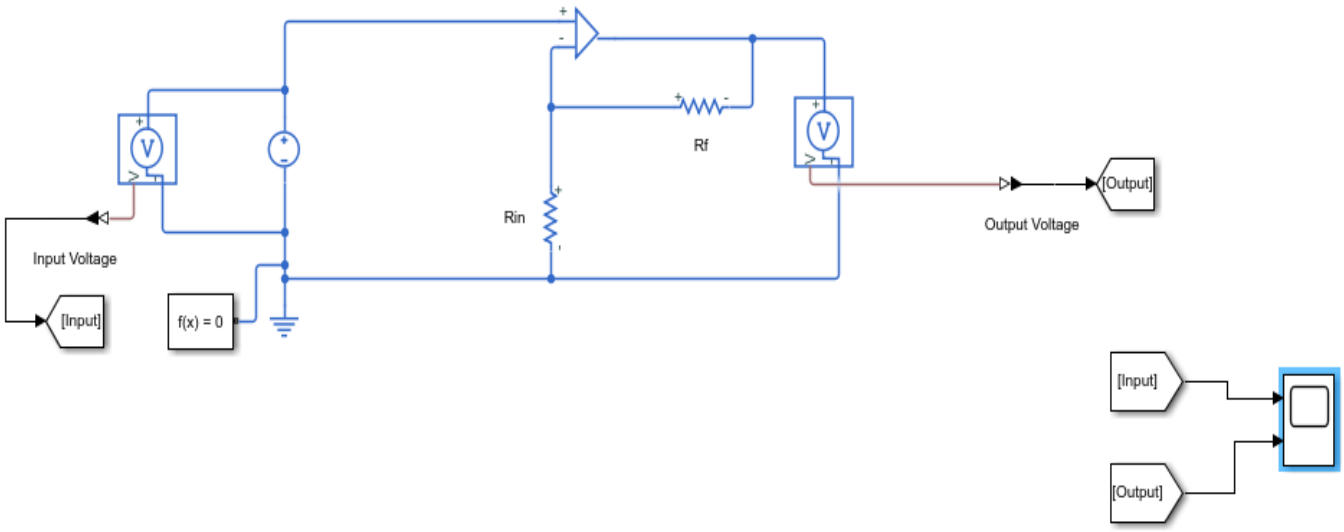
Results:



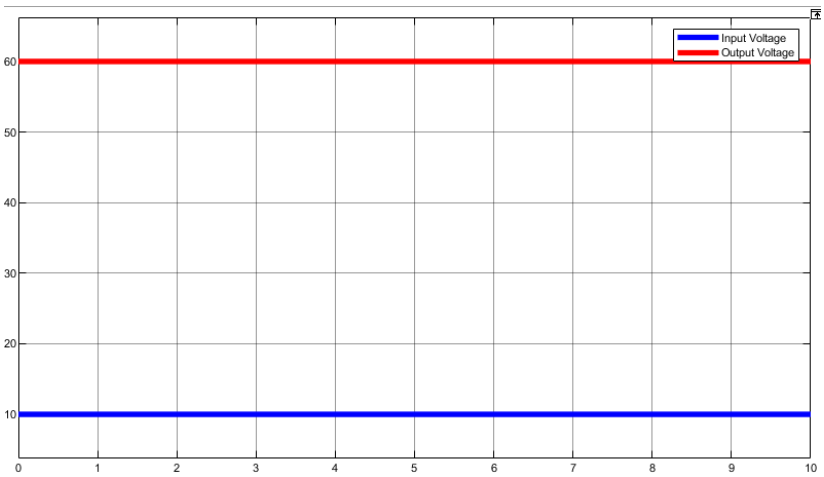
Conclusion/Remarks:

The feedback resistance (2 ohm) has twice the value of input resistance (1 ohm). The gain of the op amp is 2 (Rf/Ri=2/1=2). So, an input of 10 volts gets converted to -20 volts (twice the magnitude with inverted sense).

b) <u>Non-Inverting Op-Amp:</u>



Results:



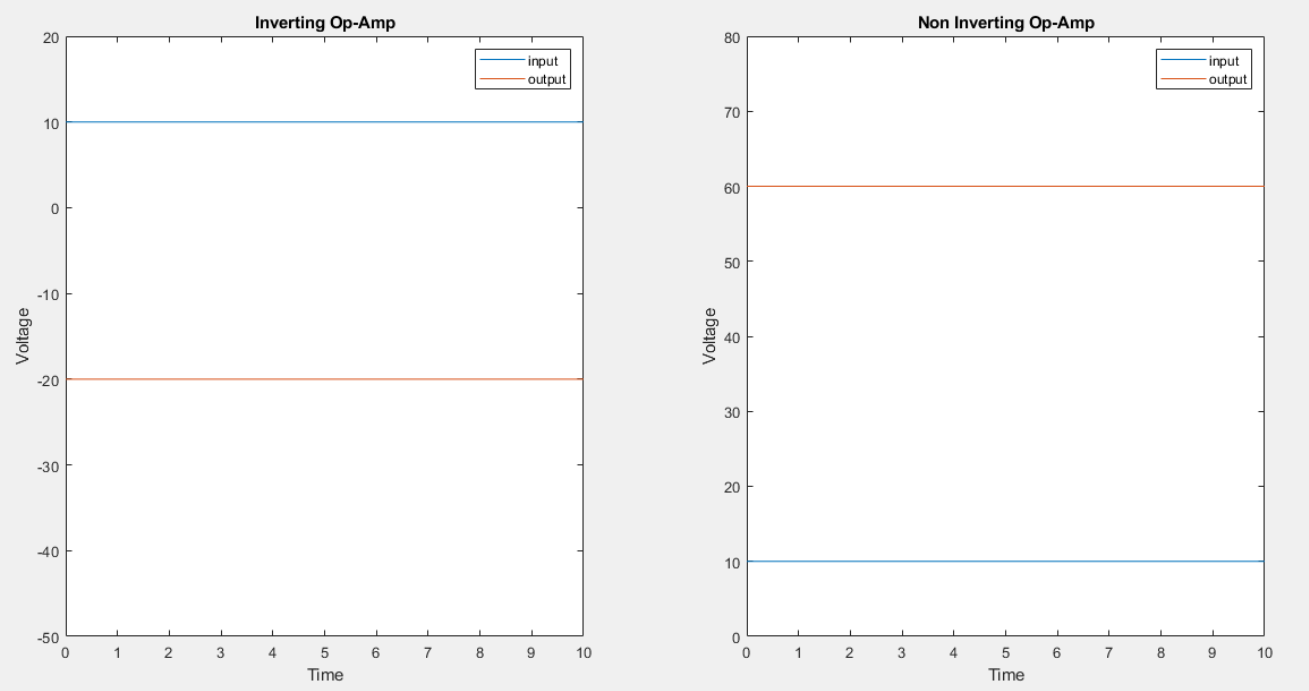
Conclusion/Remarks:

The feedback resistance (5 ohm) has five times the value of input resistance (1 ohm). The gain of the op amp is therefore 6 (1+Rf/Ri=1+5/1=6). So, an input of 10 volts gets converted to 60 volts (six times the magnitude).

M file result:

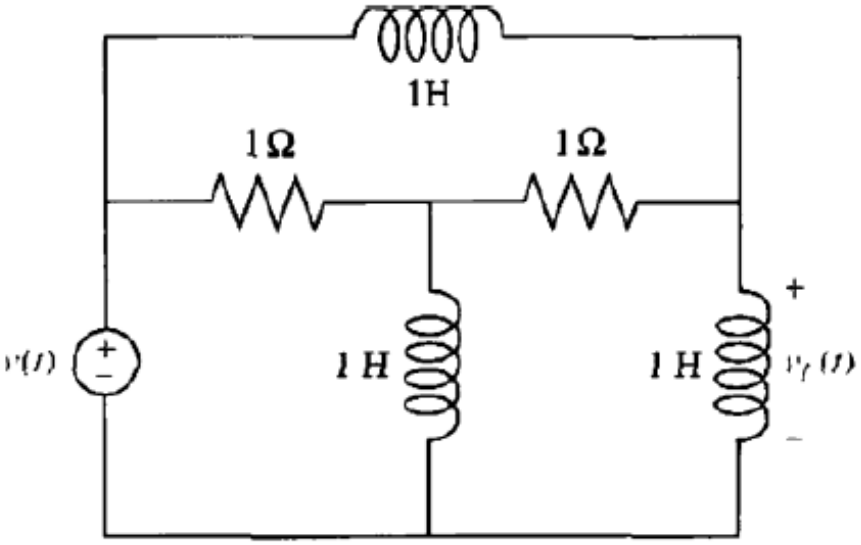
```
Rin_1=1;
Rf_1=2;
Rin_2=1;
Rf_2=5;
t=0:0.1:10;
Vin=10*ones(1,length(t));
Vout_Inv=-(Rf_1/Rin_1)*Vin;
Vout_NonInv=(1+Rf_2/Rin_2)*Vin;
subplot(1,2,1);
plot(t,Vin,t,Vout_Inv);
axis([0,10,-50,20]);
title('Inverting Op-Amp');
xlabel('Time');ylabel('Voltage');
subplot(1,2,2);
plot(t,Vin,t,Vout_NonInv);
axis([0,10,0,80]);
title('Non Inverting Op-Amp');
xlabel('Time');ylabel('Voltage');
```

Following is the output

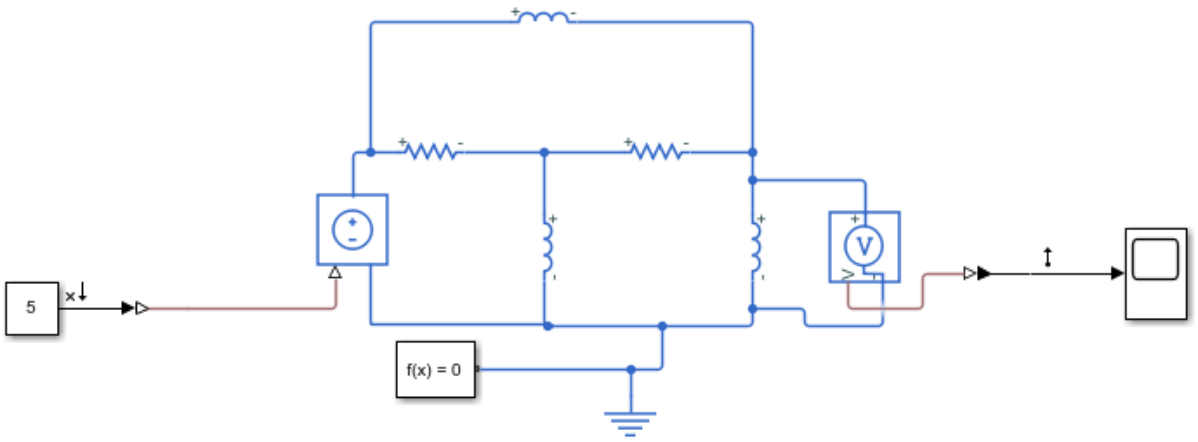


Model 4:

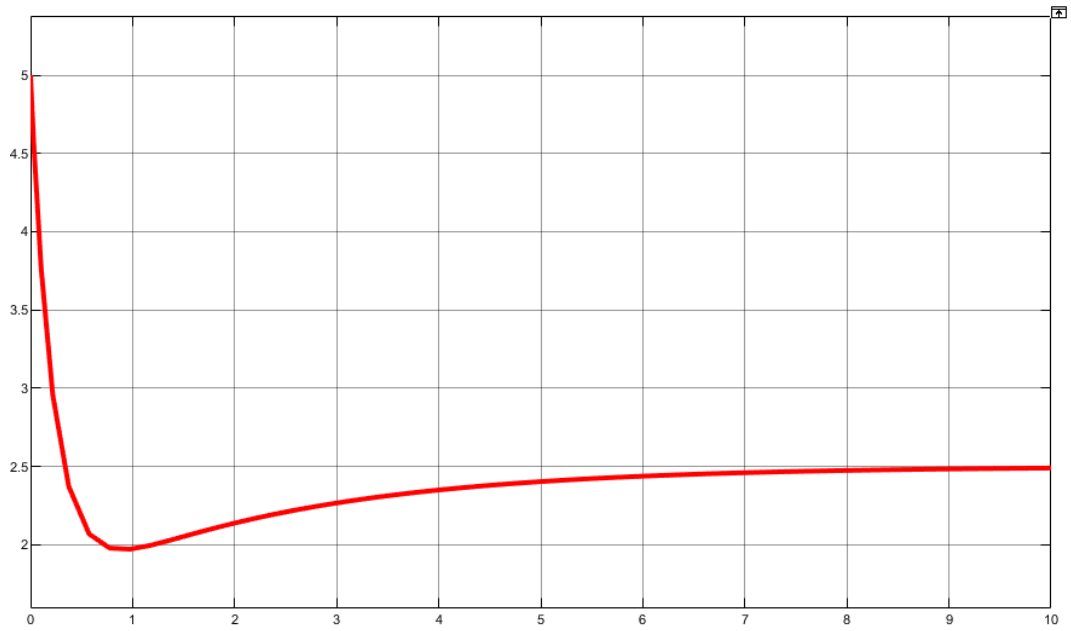
Model, Analyze, and Cross-Validate (Either Through m-file or SIMULINK: An Open Choice) the response and Transfer functions of the following Electrical Systems.



SIMSCAPE model:



Results:



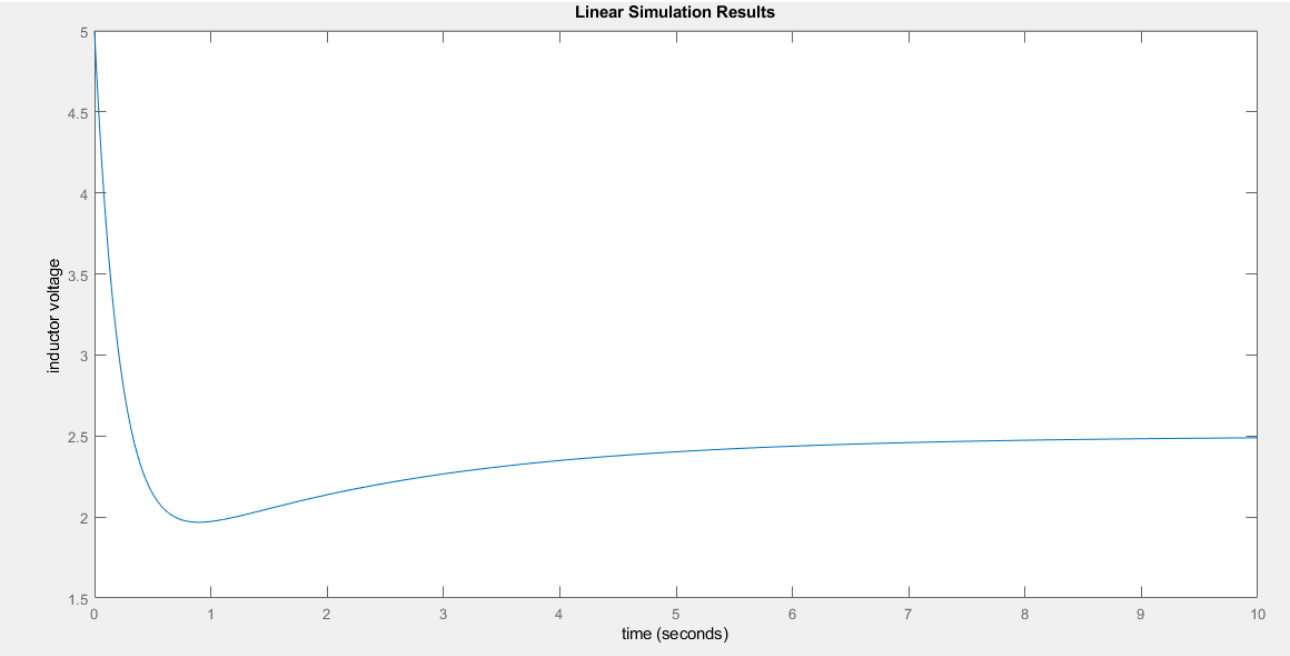
Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "Constant" to output "Inductor Voltage":
          s^3 + 2 s^2 + s
  ----------------------------
  s^3 + 5 s^2 + 2 s - 2.265e-16

Name: Linearization at model initial condition
Continuous-time transfer function.
```

M file result:

```
syms Vin s
A=[(2+1/s) , -1 ; -1 , (1+2/s)];
B=[Vin ; Vin/s];
C=A\B;
VL=C(2);
G=VL/Vin;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of denominator. Done to match the outputs
den=den/den(1);
fprintf('VL(s)/Vin(s):')
tf(num,den)
t=0:0.01:10;
u=5*ones(1,length(t));
lsim(num,den,u,t);
xlabel('time');ylabel('inductor voltage');
```

Following is the output



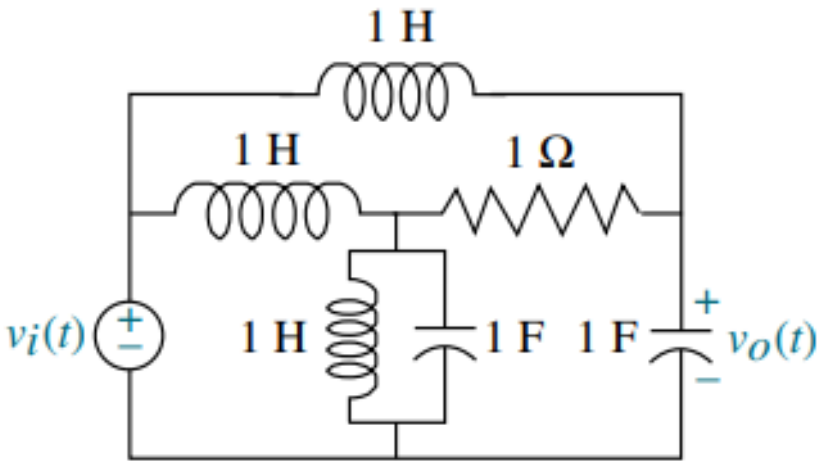

```
VL(s)/Vin(s):
ans =

  s^2 + 2 s + 1
  -------------
  s^2 + 5 s + 2

Continuous-time transfer function.
```

Conclusion/Remarks:

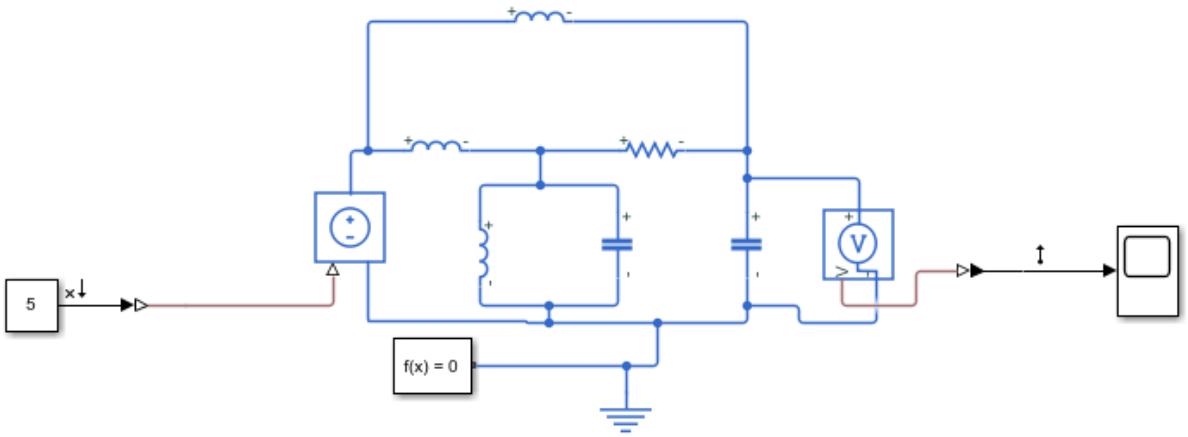
The results obtained by m file as well as the SIMSCAPE are exactly the same. Hence the developed SIMSCAPE model is correct and is in accordance with the mathematical model (m file).

Model 5:

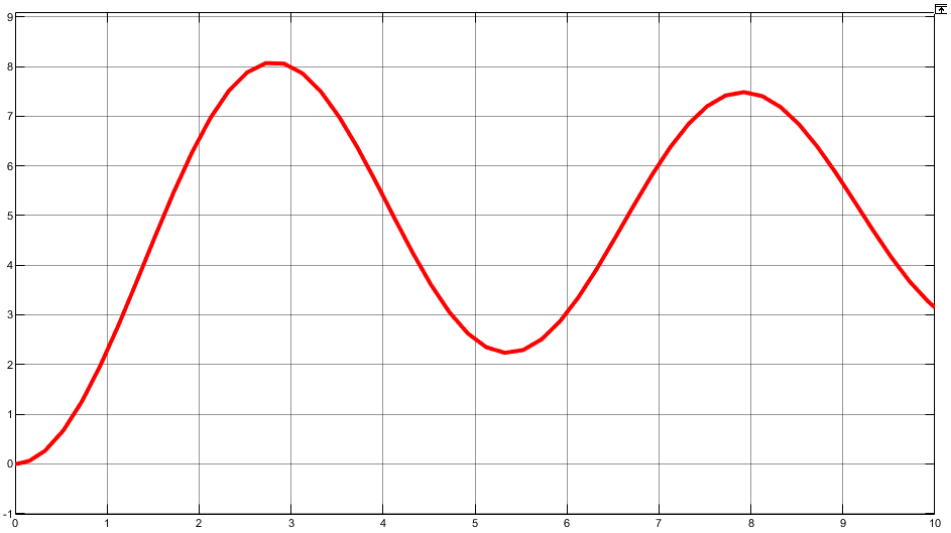
Model, Analyze, and Cross-Validate (Either Through m-file or SIMULINK: An Open Choice) the response and Transfer functions of the following Electrical Systems.



SIMSCAPE model:



Results:



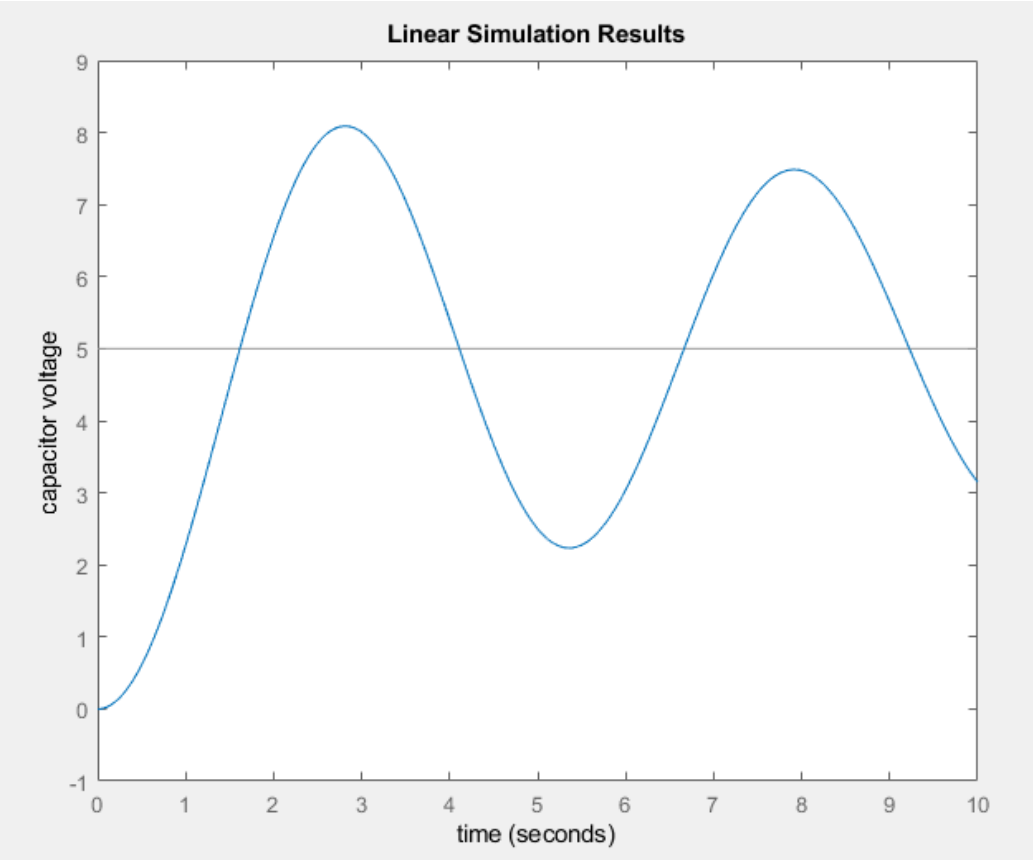
Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "Input Voltage" to output "Capacitor Voltage":
                  s^3 + 2 s^2 + 2 s
  ---------------------------------------------
  s^5 + 2 s^4 + 3 s^3 + 3 s^2 + 2 s + 3.549e-17

Name: Linearization at model initial condition
Continuous-time transfer function.
```

M file result:

```
syms Vin s
A=[(s+s/(1+s^2)) , -s/(1+s^2) , -s ; -s/(1+s^2) , (1+s/(1+s^2)+(1/s)) , -1 ; -s , -1
, (2*s+1)];
B=[Vin ; 0 ; 0];
C=A\B;
I2=C(2);
Vc=I2/s;
G=Vc/Vin;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);      %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
fprintf('Vc(s)/Vin(s):')
tf(num,den)
t=0:0.01:10;
u=5*ones(1,length(t));
lsim(num,den,u,t);
xlabel('time');ylabel('capacitor voltage');
```

Following is the output




```
Vc(s)/Vin(s):
ans =

          s^2 + 2 s + 2
  ----------------------------
  s^4 + 2 s^3 + 3 s^2 + 3 s + 2

Continuous-time transfer function.
```

Conclusion/Remarks:

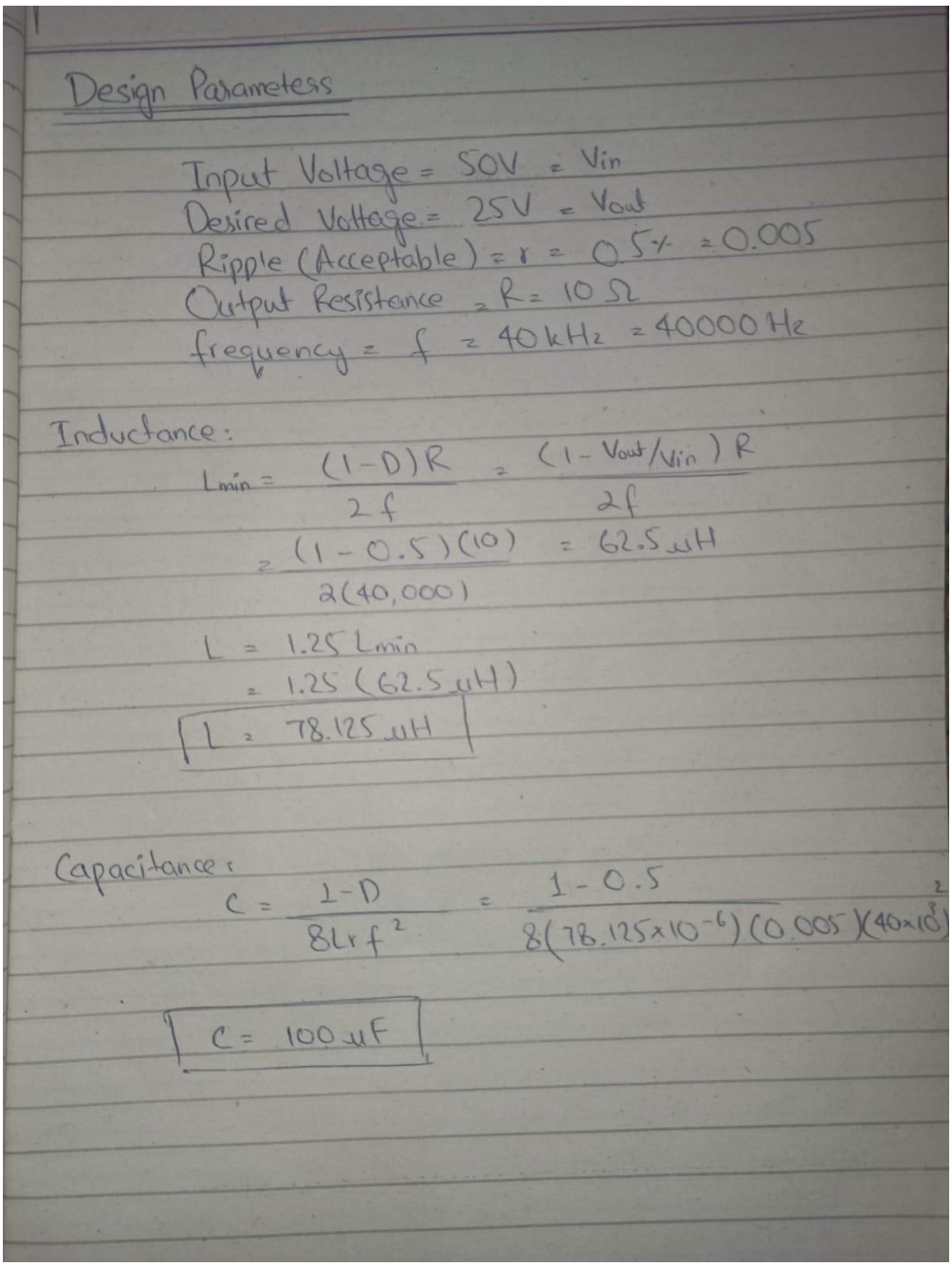
The results obtained by m file as well as the SIMSCAPE are exactly the same. Hence the developed SIMSCAPE model is correct and is in accordance with the mathematical model (m file).

Model 6:

By following the conceptual framework from the referred video, develop the model of the BUCK Converter. Try to understand and cross-validate the results with the Mathematical findings (Also explained in the video).

The following model has been designed for stepping down a **50-volt DC** input to a **25-V DC** output. The mathematical calculations and the corresponding SIMSCAPE model are given below.

Mathematical Model:



Design Parameters

Input Voltage = 50V = Vin
Desired Voltage = 25V = Vout
Ripple (Acceptable) = r = 0.5% = 0.005
Output Resistance = R = 10 Ω
frequency = f = 40 kHz = 40000 Hz

Inductance:

$$L_{min} = \frac{(1-D)R}{2f} = \frac{(1 - V_{out}/V_{in})R}{2f}$$

$$= \frac{(1-0.5)(10)}{2(40{,}000)} = 62.5\,\mu H$$

$$L = 1.25\,L_{min}$$

$$= 1.25\,(62.5\,\mu H)$$

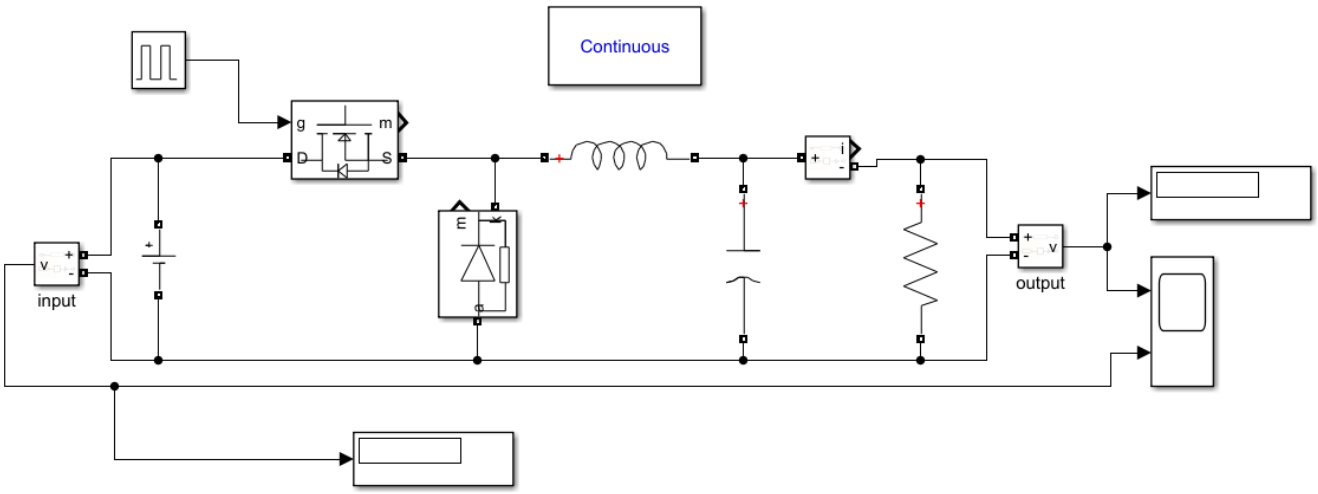
$$\boxed{L = 78.125\,\mu H}$$

Capacitance:

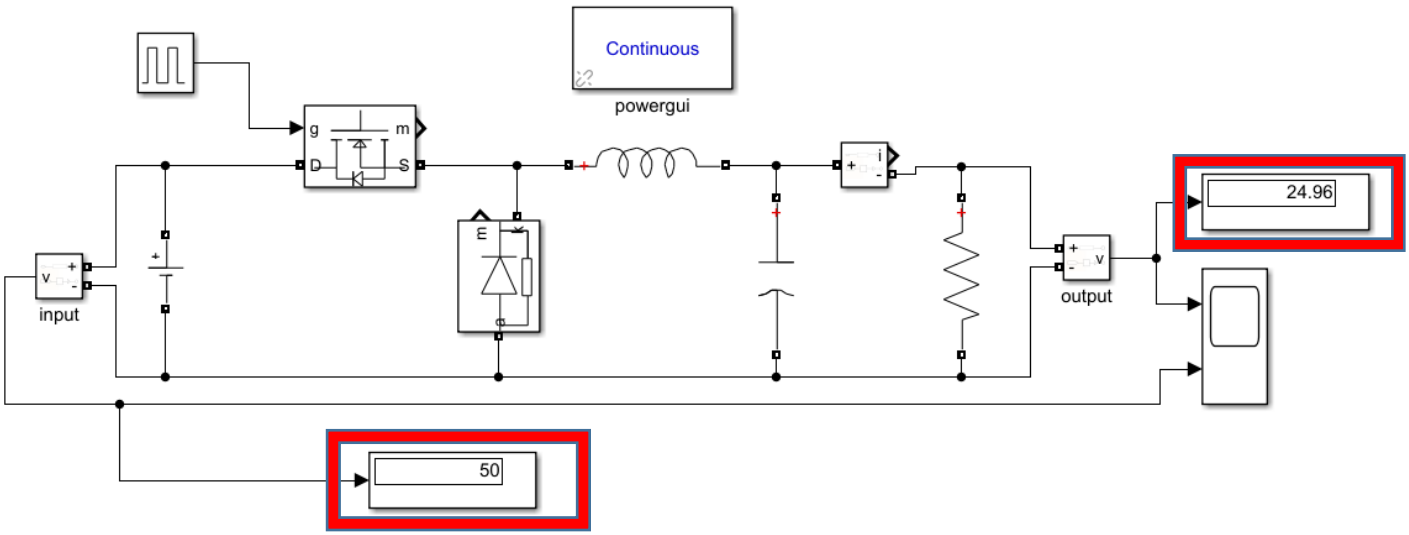
$$C = \frac{1-D}{8Lrf^2} = \frac{1-0.5}{8(78.125\times10^{-6})(0.005)(40\times10^{3})^2}$$

$$\boxed{C = 100\,\mu F}$$

SIMSCAPE Model:

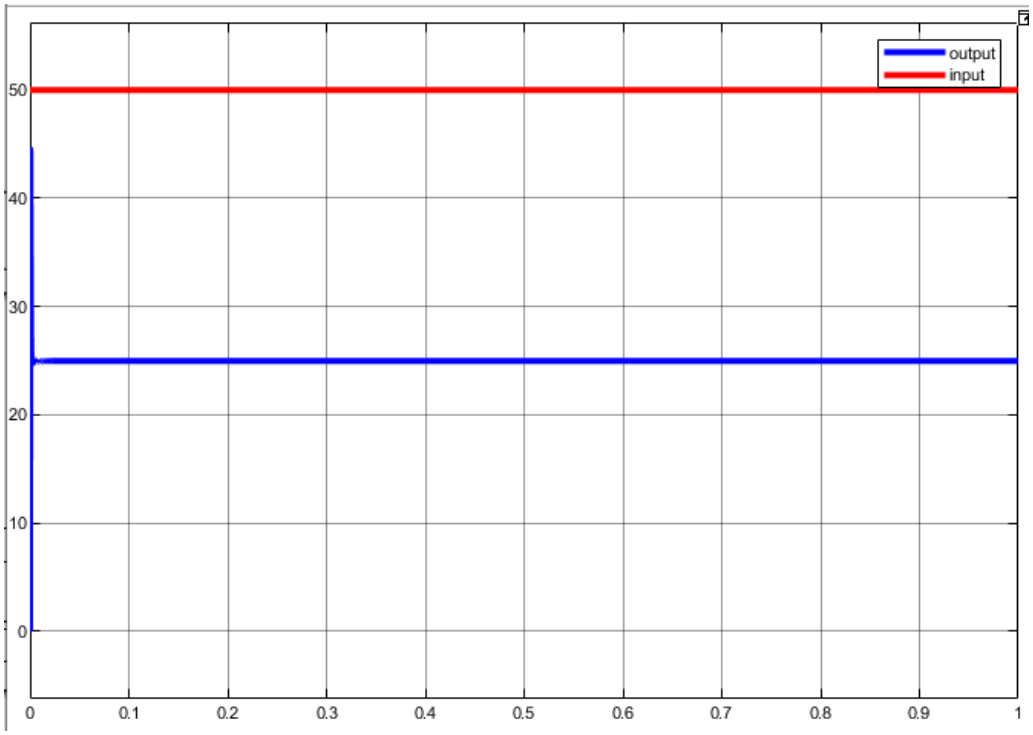


Results:



Conclusion/Remarks:

The results obtained by SIMSCAPE are exactly as anticipated by the mathematical model. This is evident by both the display block as well as the plot.


output
input
50
40
30
20
10
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

LAB 4: Modeling and Analysis of the Electromechanical Actuator (i.e., DC Motor along with the manipulation in the gear trains).

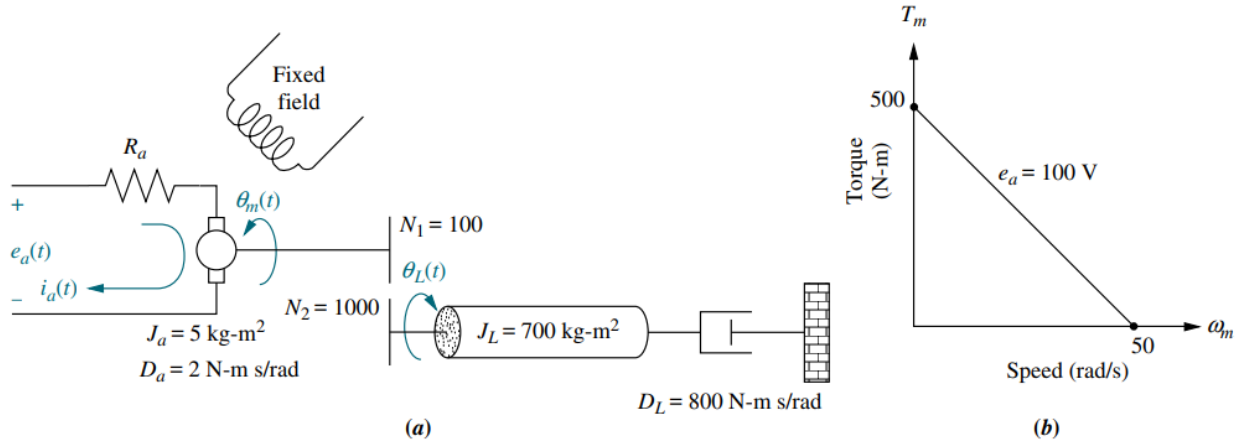
Aim and Objectives:

The fundamental aim of this lab is to explore the built-in libraries of SIMSCAPE and by following the progression, the developed concepts will be implied to analyze the dynamic responses of various electromechanical actuators. Most importantly, certain examples from the Modelling and Simulation have been considered to consolidate the theoretical hypothesis (Of by Cross-Validating the Transfer Function and State Space Representation) of the emphasized systems. Additionally, the outcomes of the dynamic systems have been validated with the findings of ODE SOLVERS. Subsequently, certain tangible objectives are enlightened below;

1. To model and analyze the dynamic response of a DC Motor whose Torque Speed Characteristics are given.
2. To model and analyze the dynamic response of DC Geared Motors whose Torque Speed Characteristics are provided either in graphical form or in the final equation.
3. To model and analyze the dynamic response of the Electromechanical Actuator (Along with the Rotary to the Linear Converter) whose Torque Speed Characteristics are provided in graphical form.
4. To obtain the transfer function from developed and above-mentioned SIMSCAPE models.
5. To cross-validate the transfer function and dynamic response of SIMSCAPE Models with the results obtained from Mathematical Model and ODEs 45, respectively.

Assigned Tasks:

Model 1:



(a) Find the transfer function between Motor Displacement and Applied Voltage.

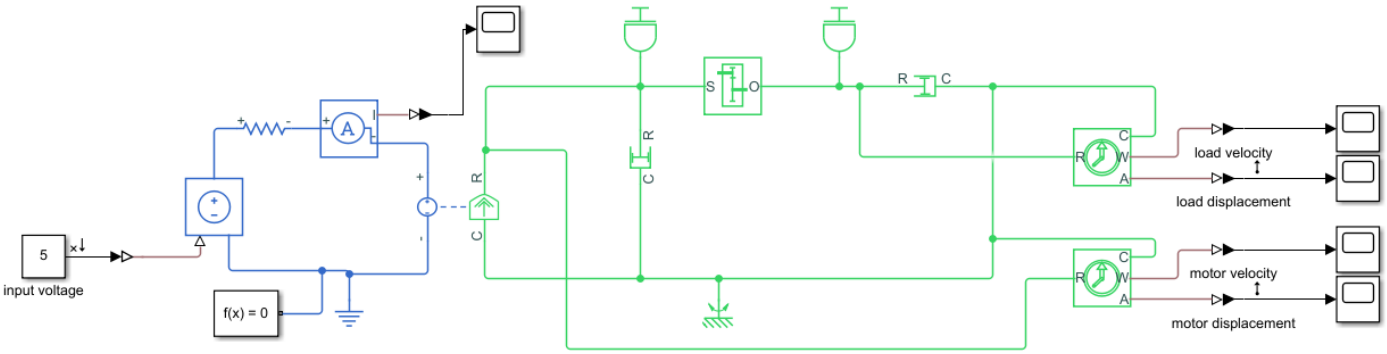
(b) Find the transfer function between Load Displacement and Applied Voltage.

(c) Analyze the dynamic response (Load Displacement, Velocity, and Acceleration) under the given values of the system.

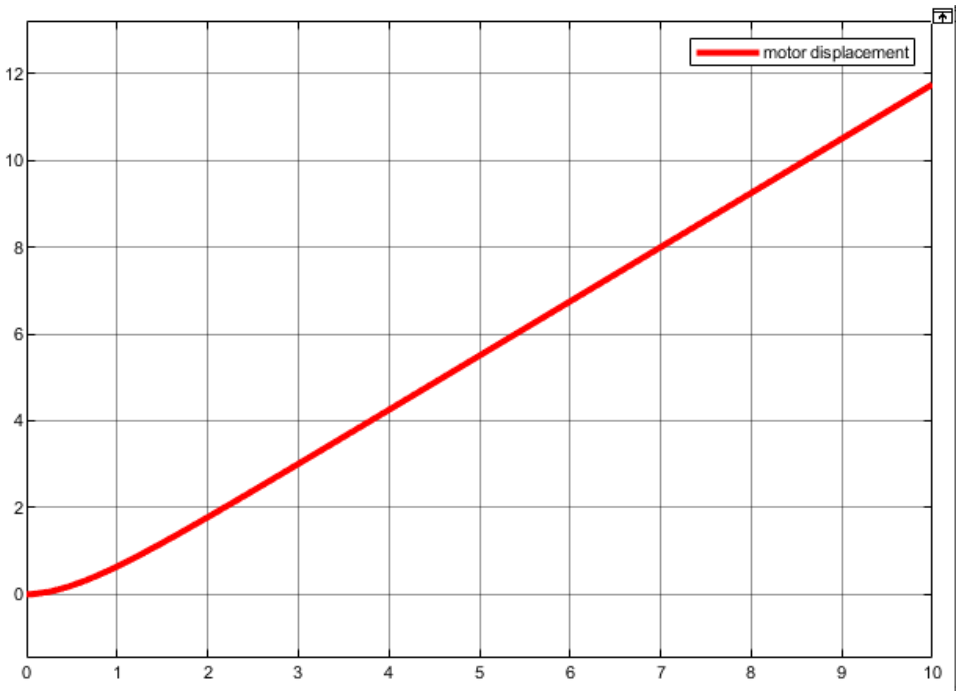
(d) Cross-validate the transfer functions with your Mathematical model (You may utilize MATLAB to find the transfer function as we did by considering the crammer rule).

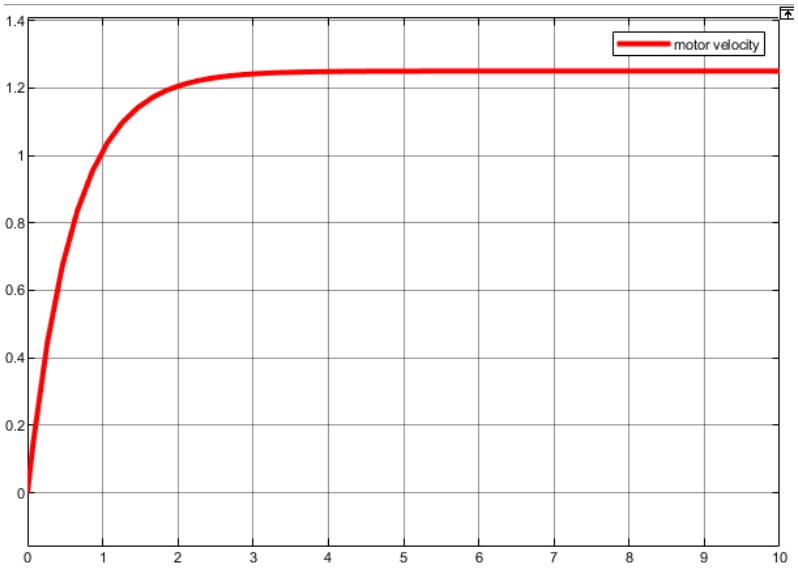
(e) Cross-validate the dynamic response (Load Displacement, Velocity, and Acceleration) with your m-files.

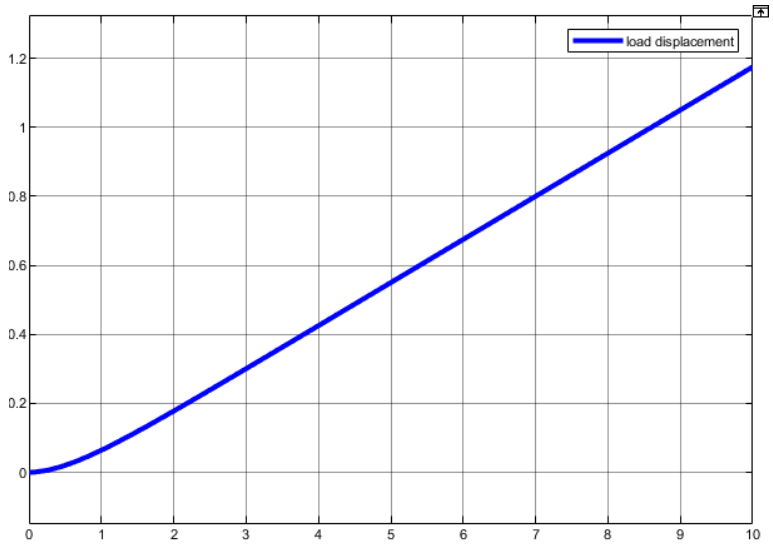
SIMSCAPE model:

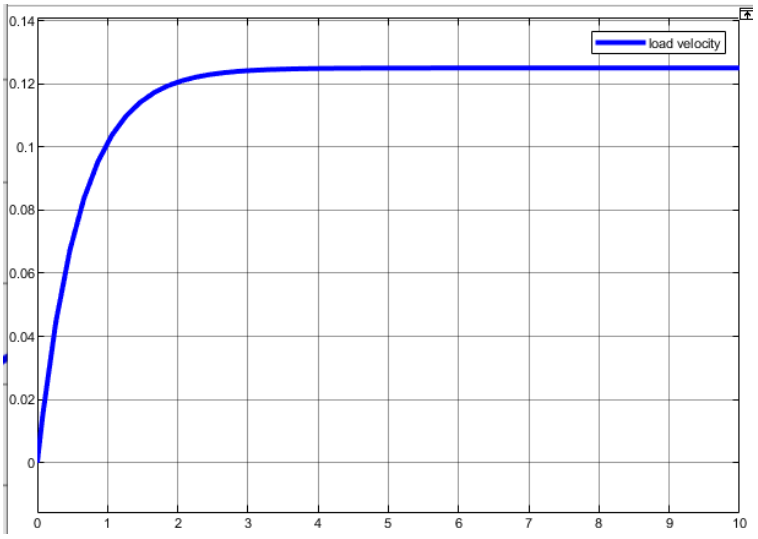


Results:




motor velocity
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9 10


load displacement
1.2
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9 10


load velocity
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0 1 2 3 4 5 6 7 8 9 10

Conclusion/Remarks:

Both graphs validate the real-life behavior of a DC motor i.e., the displacement of the rotor increases linearly with time whereas the angular velocity of the motor reaches to a constant value. This is clearly evident in the graphs. Also due to the ideal gears, there is no mechanical loss in the system and the linear velocity remains constant (by the law of gearing). This basically means that the angular velocity should decrease by the factor of gear ratio (N2/N1, 10 in this case).

Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "input voltage" to output...
                          0.04167
   load displacement:  -------------
                       s^2 + 1.667 s

                            0.4167
   motor displacement:  -------------
                        s^2 + 1.667 s

Name: Linearization at model initial condition
Continuous-time transfer function.
```

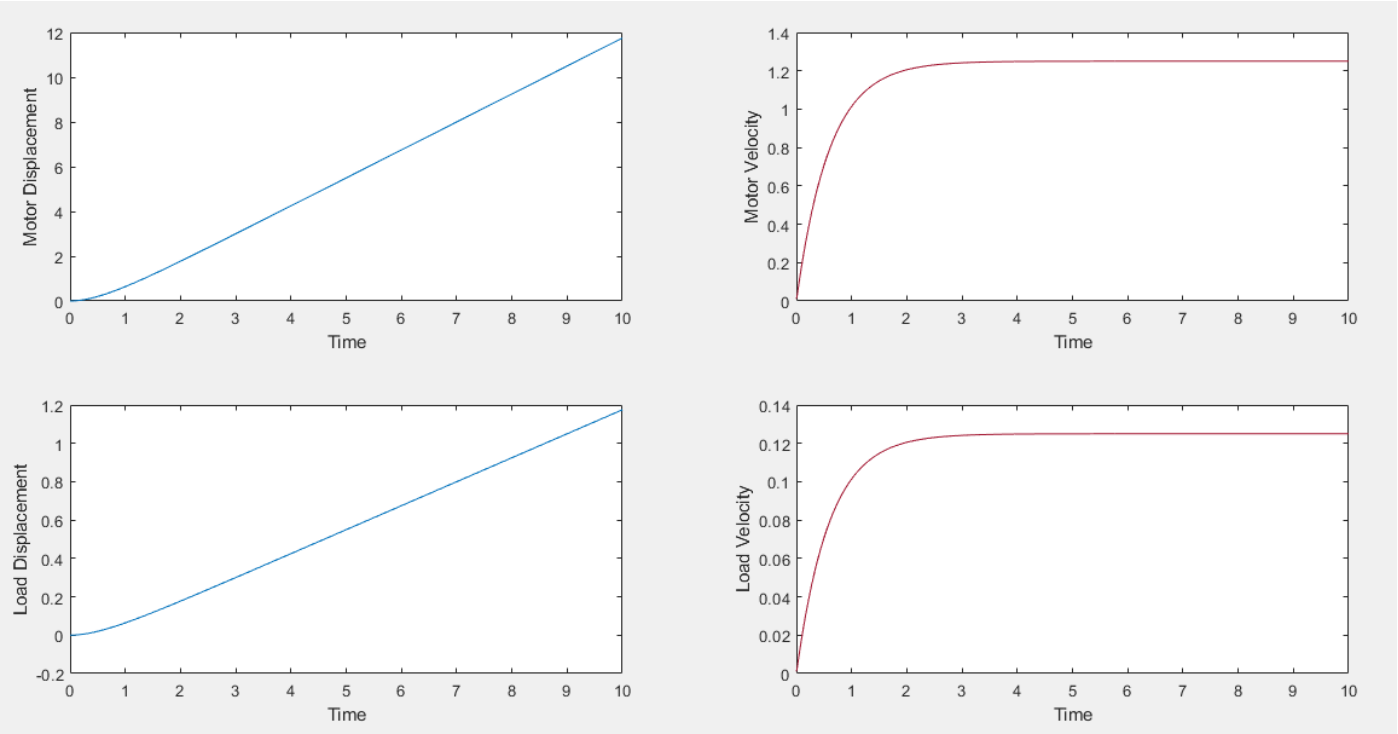
M File results:

```
syms s V
Ra=0.4;kt=2;kb=2;Jm=12;Dm=10;
A=[Ra/kt , kb*s ; 1 , -(Jm*s*s+Dm*s)];
B=[V;0];
C=A\B;
thetaM=C(2);
thetaL=thetaM/10;
G=thetaM/V;G2=thetaL/V;
G=collect(G,s);G2=collect(G2,s);
[num,den]=numden(G);[num2,den2]=numden(G2);
num=sym2poly(num);den=sym2poly(den);
num2=sym2poly(num2);den2=sym2poly(den2);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
num2=num2/den2(1);     %dividing by den(1) means dividing by the leading coefficient
of denominator. Done to match the outputs
den2=den2/den2(1);
fprintf('ThetaM(s)/Vin(s):');
tf(num,den)
fprintf('ThetaL(s)/Vin(s):');
tf(num2,den2)
%%%%%%%%%% plotting variables
t=0:0.01:10;
u=5*ones(1,length(t));
```

```
y=lsim(num,den,u,t);
y2=lsim(num2,den2,u,t);
subplot(2,2,1);plot(t,y);
xlabel('Time');ylabel('Motor Displacement')
subplot(2,2,2);plot(t,gradient(y)./gradient(t));
xlabel('Time');ylabel('Motor Velocity')
subplot(2,2,3);plot(t,y2);
xlabel('Time');ylabel('Load Displacement')
subplot(2,2,4);plot(t,gradient(y2)./gradient(t));
xlabel('Time');ylabel('Load Velocity')
```

Following are the results.

```
>> Task1MFile
ThetaM(s)/Vin(s):
ans =

     0.4167
  -------------
  s^2 + 1.667 s

Continuous-time transfer function.

ThetaL(s)/Vin(s):
ans =

     0.04167
  -------------
  s^2 + 1.667 s

Continuous-time transfer function.
```

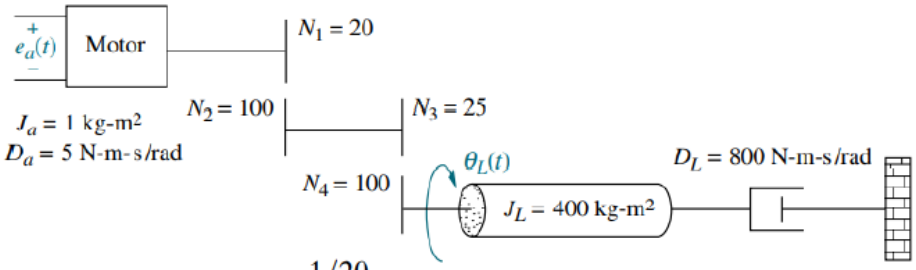


Conclusion/Remarks:

It can be clearly seen that the plots obtained by SIMSCAPE results and by the mathematical model (m-file) are exactly the same and also validate the real-life behavior of a DC motor.

Model 2:

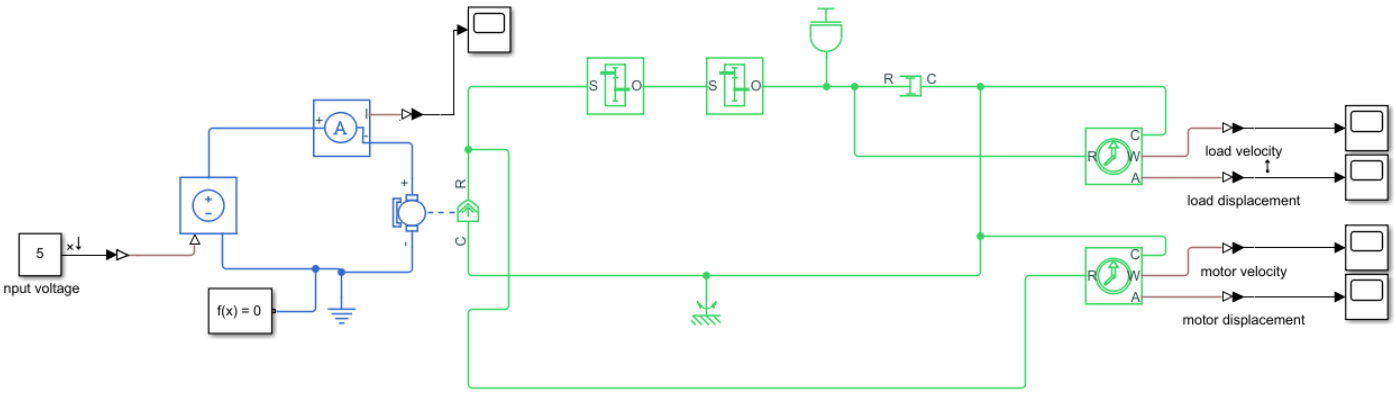
PROBLEM: Find the transfer function, $G(s) = \theta_L(s)/E_a(s)$, for the motor and load shown in Figure 2.40. The torque-speed curve is given by $T_m = -8\omega_m + 200$ when the input voltage is 100 volts.



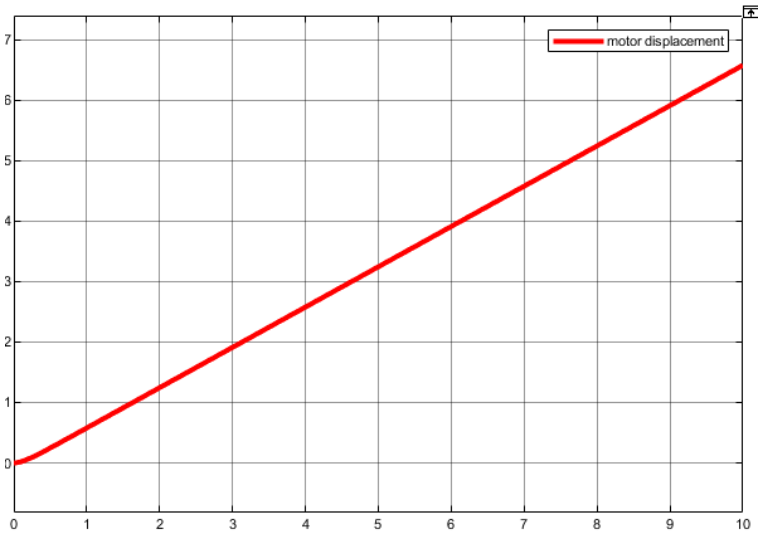
ANSWER: $G(s) = \dfrac{1/20}{s[s + (15/2)]}$

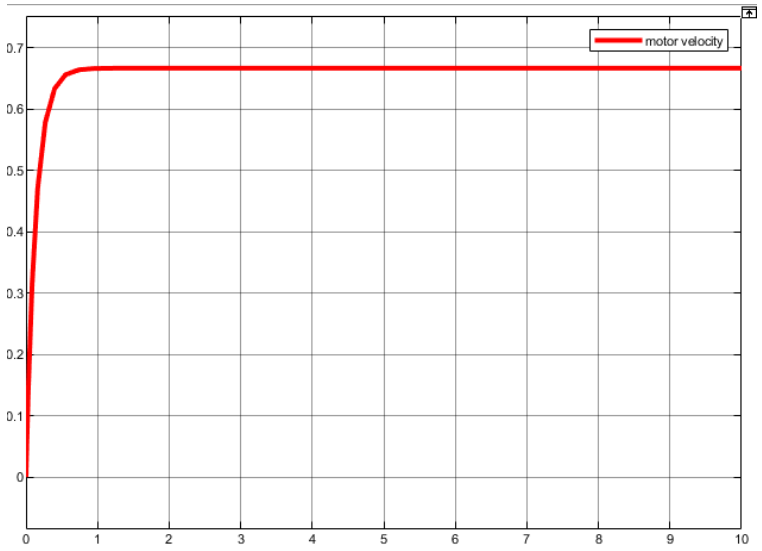
In this example, you just need to cross-verify the transfer function and dynamic response with your m-files. **Note:** You have to solve this example using the **DC Motor model of SIMSCAPE** rather than the **Rotational Electromechanical Converter.**

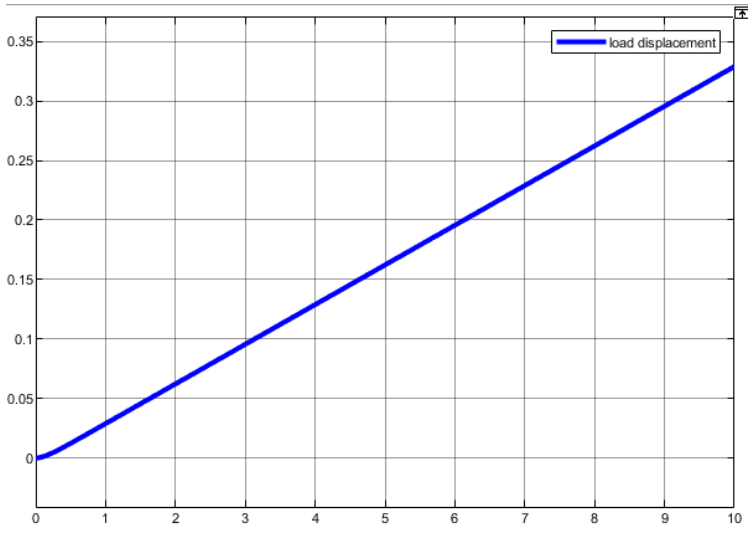
SIMSCAPE model:

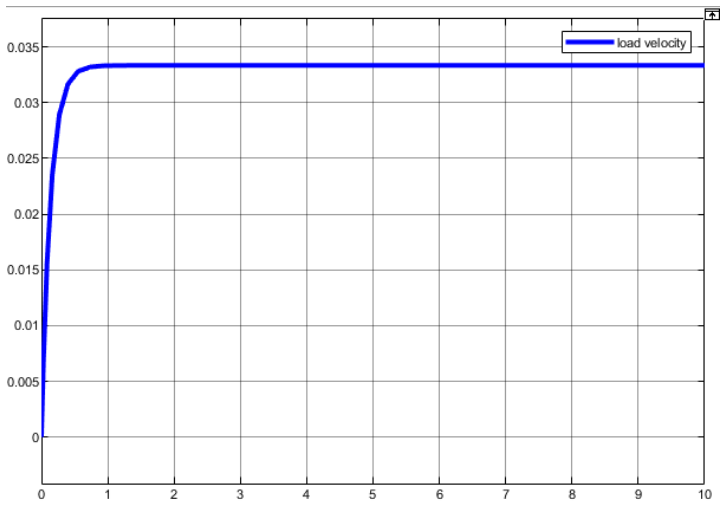


Results:


motor displacement
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
8
9
10


motor velocity
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1
2
3
4
5
6
7
8
9
10


load displacement
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0
1
2
3
4
5
6
7
8
9
10



Conclusion/Remarks:

Both graphs validate the real-life behavior of a DC motor i.e., the displacement of the rotor increases linearly with time whereas the angular velocity of the motor reaches to a constant value. This is clearly evident in the graphs. Also due to the ideal gears, there is no mechanical loss in the system and the linear velocity remains constant (by the law of gearing). This basically means that the angular velocity should decrease by the factor of gear ratio (N2/N1, 20 in this case).

Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "input voltage" to output "load displacement":
     0.05
  -----------
  s^2 + 7.5 s

Name: Linearization at model initial condition
Continuous-time transfer function.
```

```
syms s V
Ra=2;kt=4;kb=4;Jm=2;Dm=7;
A=[Ra/kt , kb*s ; 1 , -(Jm*s*s+Dm*s)];
B=[V;0];
C=A\B;
thetaM=C(2);
thetaL=thetaM/20;
G=thetaL/V;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of denominator. Done to match the outputs
den=den/den(1);
fprintf('ThetaL(s)/Vin(s):');
tf(num,den)
```

```
>> Task2MFile
ThetaL(s)/Vin(s):
ans =

      0.05
  -----------
  s^2 + 7.5 s

Continuous-time transfer function.
```

Model 3:

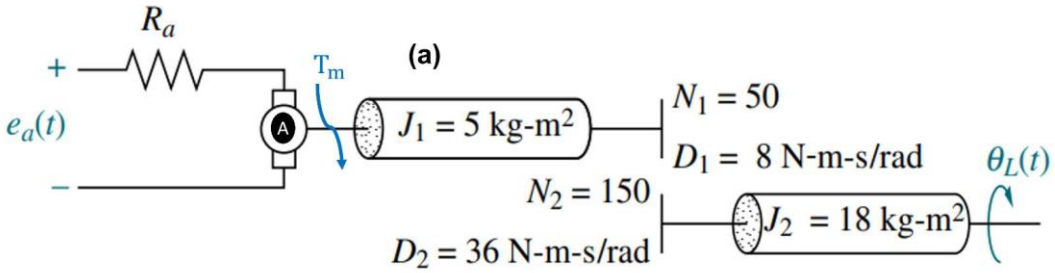
Consider the electromechanical system is utilized to drive the joint of a particular robotic arm in which the different discrete inertias, dampers, and gear ratios are included to model the dynamic system. After investigating the dynamic response, the modeler further enlightens the torque-speed characteristics of the DC-Geared motor as shown in Figure (b). You are required to utilize the above-mentioned information for the following action items;

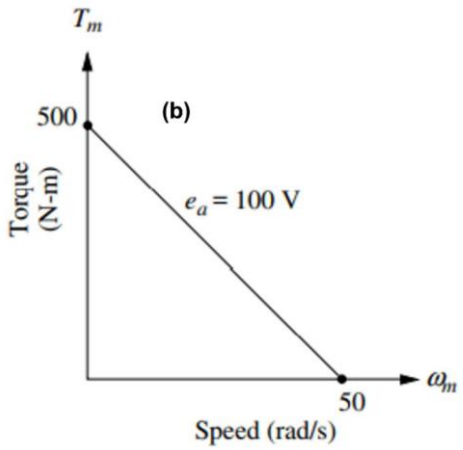
a. Find the transfer function between Armature Current (*Ia*) and Applied Voltage (*ea*).

b. Find the transfer function between Motor Torque (*Tm*) and Applied Voltage (*ea*).

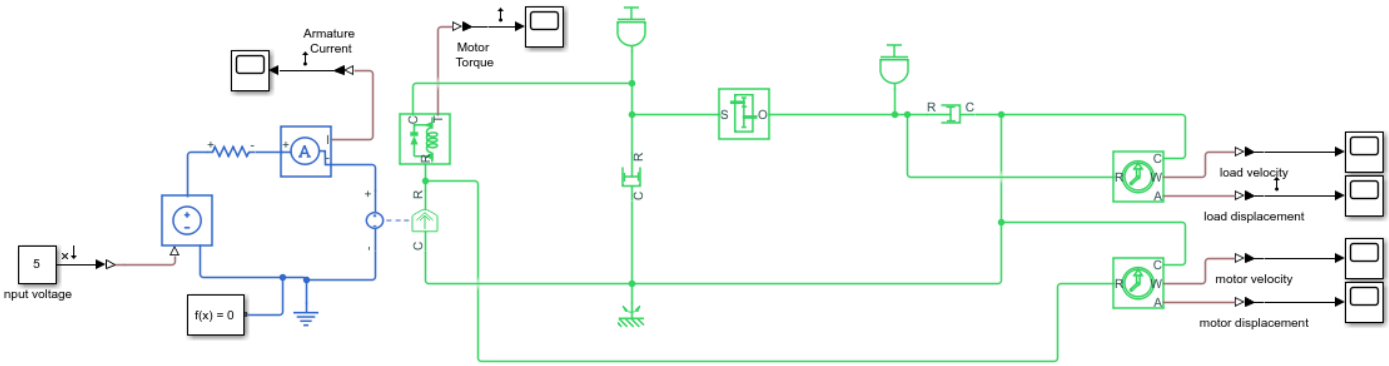
c. Find the transfer function between Load Displacement (*θL*) and Applied Voltage (*ea*).

d. Analyze and Cross-validate the dynamic response (Load Displacement of the given model).

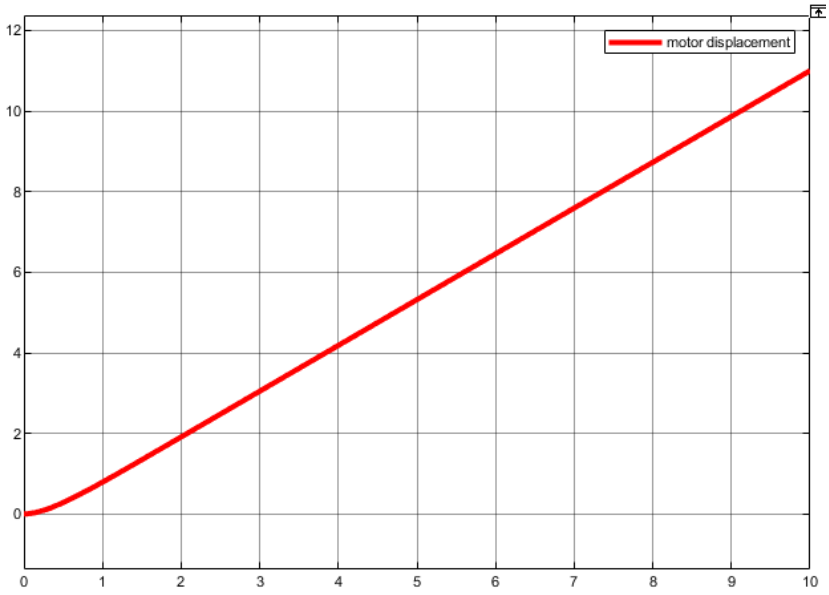


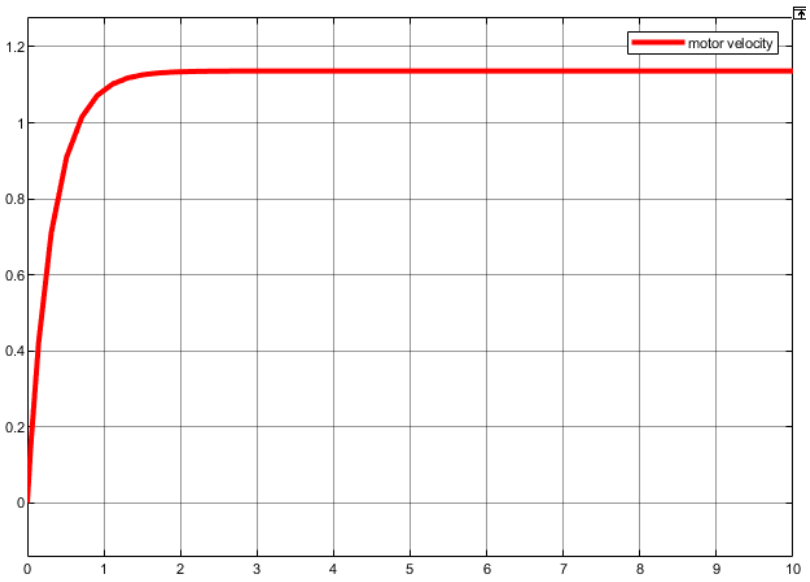


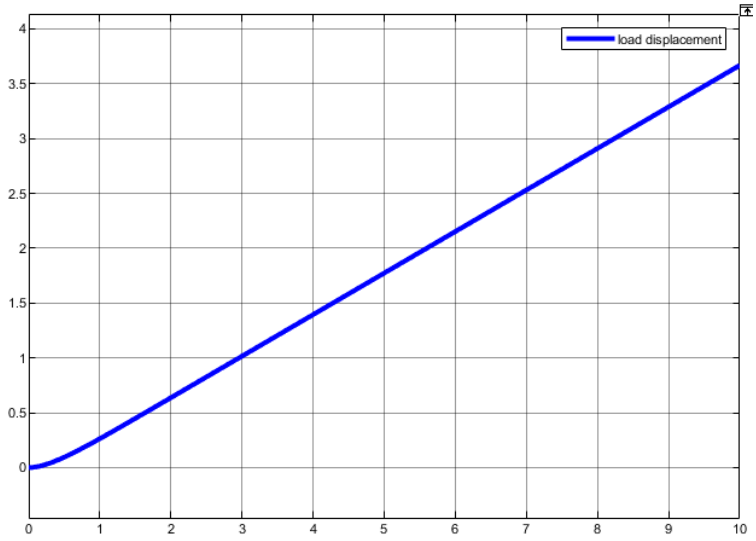
SIMSCAPE model:

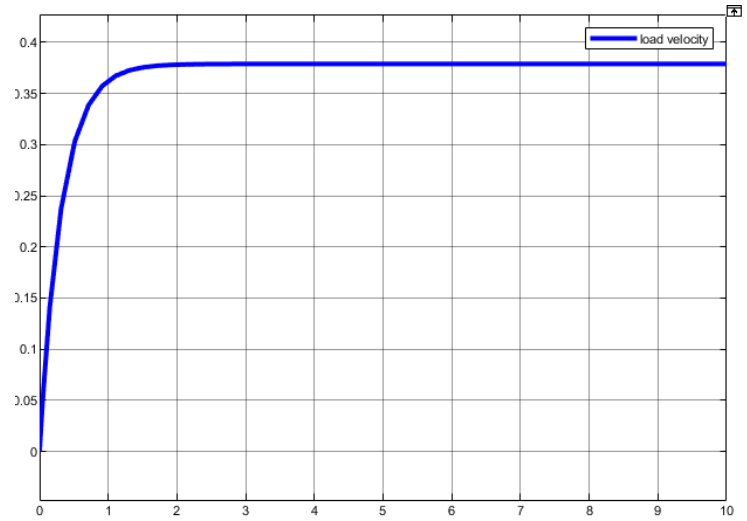


Results:









Conclusion/Remarks:

Both graphs validate the real-life behavior of a DC motor i.e., the displacement of the rotor increases linearly with time whereas the angular velocity of the motor reaches to a constant value. This is clearly evident in the graphs. Also due to the ideal gears, there is no mechanical loss in the system and the linear velocity remains constant (by the law of gearing). This basically means that the angular velocity should decrease by the factor of gear ratio (N2/N1, 3 in this case).

Transfer Function from SIMSCAPE model:

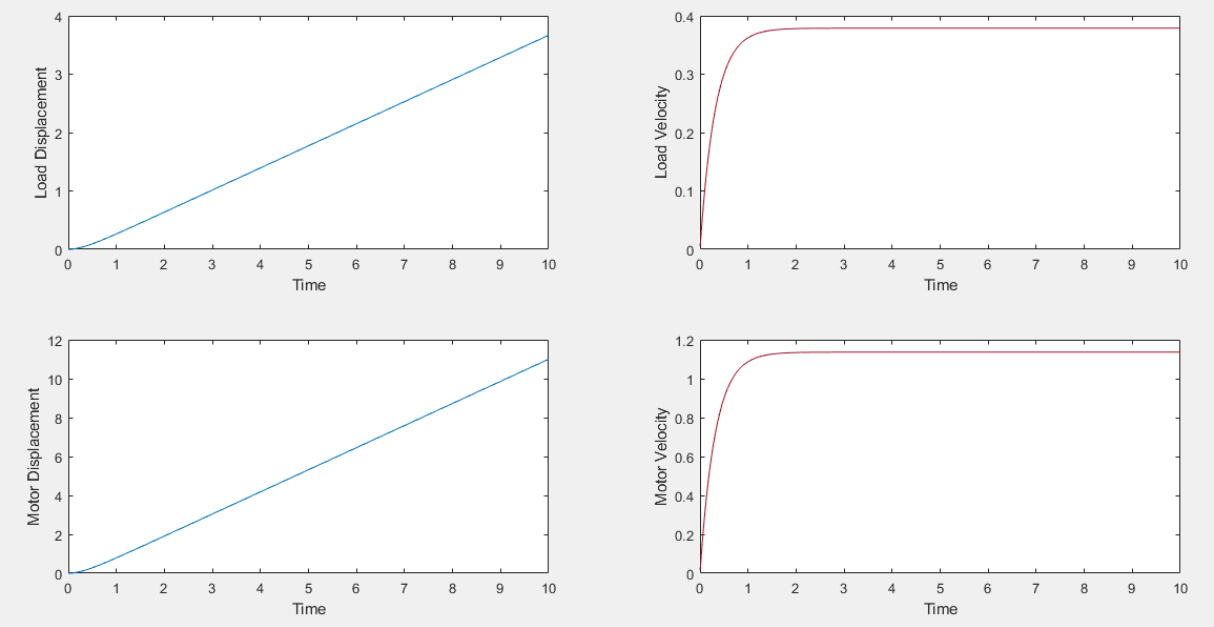
```
>> tf(linsys1)

ans =

  From input "input voltage" to output...
                          2.5 s + 4.286
   Armature  Current:  -------------
                            s + 3.143

                       5 s + 8.571
   Motor  Torque:  -----------
                        s + 3.143

                            0.2381
   load displacement:  -------------
                       s^2 + 3.143 s

Name: Linearization at model initial condition
Continuous-time transfer function.
```

M File results:

```
syms s V
Ra=0.4;kt=2;kb=2;Jm=7;Dm=12;
A=[Ra/kt , kb*s ; 1 , -(Jm*s*s+Dm*s)];
B=[V;0];
C=A\B;
thetaM=C(2);
torque=C(1);
thetaL=thetaM/3;
current=torque/kt;
% transfer functions
G1=thetaL/V;G2=torque/V;G3=current/V;
G1=collect(G1,s);G2=collect(G2,s);G3=collect(G3,s);
[num,den]=numden(G1);[num2,den2]=numden(G2);[num3,den3]=numden(G3);
num=sym2poly(num);den=sym2poly(den);
num2=sym2poly(num2);den2=sym2poly(den2);
num3=sym2poly(num3);den3=sym2poly(den3);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
num2=num2/den2(1);     %dividing by den(1) means dividing by the leading coefficient
of denominator. Done to match the outputs
den2=den2/den2(1);
num3=num3/den3(1);     %dividing by den(1) means dividing by the leading coefficient
of denominator. Done to match the outputs
den3=den3/den3(1);
fprintf('ThetaL(s)/Vin(s):');
tf(num,den)
fprintf('T(s)/Vin(s):');
tf(num2,den2)
fprintf('Ia(s)/Vin(s):');
tf(num3,den3)
```

```
%%%%%%%%%% plotting variables
t=0:0.01:10;
u=5*ones(1,length(t));
y=lsim(num,den,u,t);
y2=lsim((3*num),den,u,t); % since thetaM=3*thetaL
subplot(2,2,1);plot(t,y);
xlabel('Time');ylabel('Load Displacement')
subplot(2,2,2);plot(t,gradient(y)./gradient(t));
xlabel('Time');ylabel('Load Velocity')
subplot(2,2,3);plot(t,y2);
xlabel('Time');ylabel('Motor Displacement')
subplot(2,2,4);plot(t,gradient(y2)./gradient(t));
xlabel('Time');ylabel('Motor Velocity')
```

```
>> Task3MFile
ThetaL(s)/Vin(s):
ans =

     0.2381
  -------------
  s^2 + 3.143 s

Continuous-time transfer function.

T(s)/Vin(s):
ans =

  5 s + 8.571
  -----------
   s + 3.143

Continuous-time transfer function.

Ia(s)/Vin(s):
ans =

  2.5 s + 4.286
  -------------
    s + 3.143

Continuous-time transfer function.
```



Conclusion/Remarks:

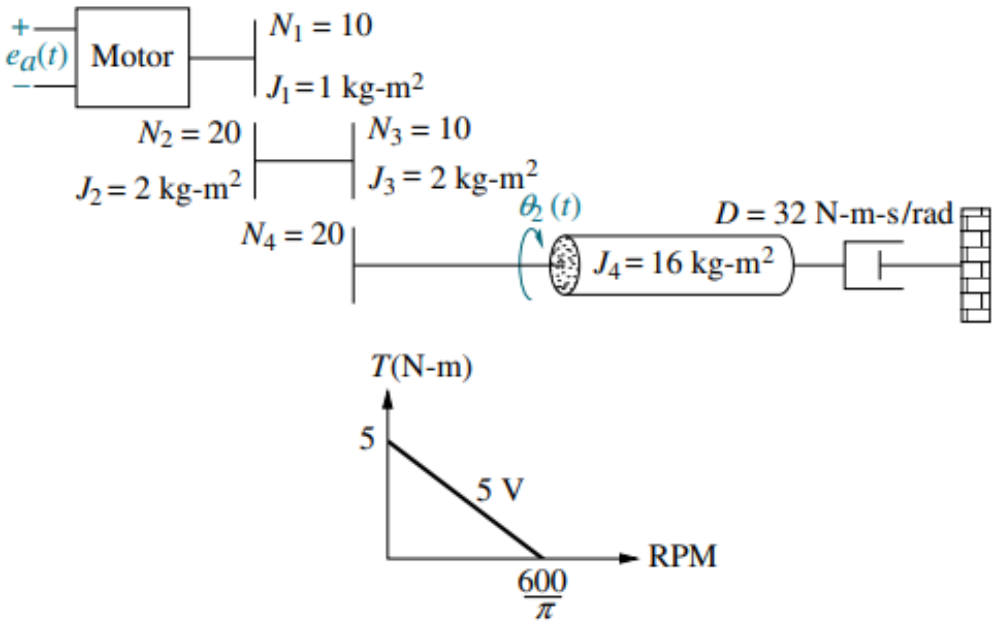
It can be clearly seen that the plots obtained by SIMSCAPE results and by the mathematical model (m-file) are exactly the same and also validate the real-life behavior of a DC motor.

Model 4 (Additional Model):

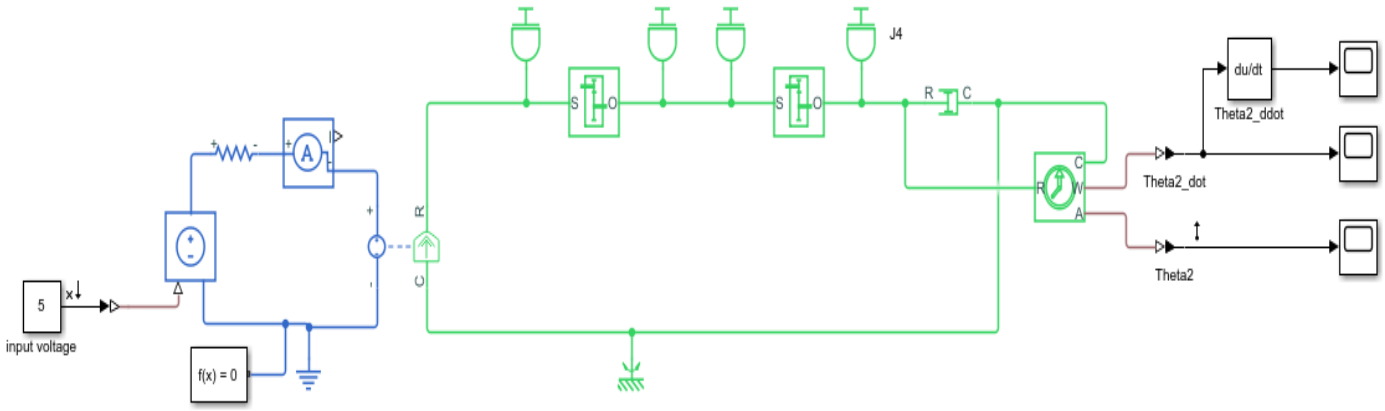
In this example, you should be considering the same constant value (torque Constant and back emf constant) throughout the dynamic analysis, and the following action items are required:

1. **Find the Transfer Function between Θ2 and applied voltage. Afterward, cross-validate the obtained transfer function with the m-file.**

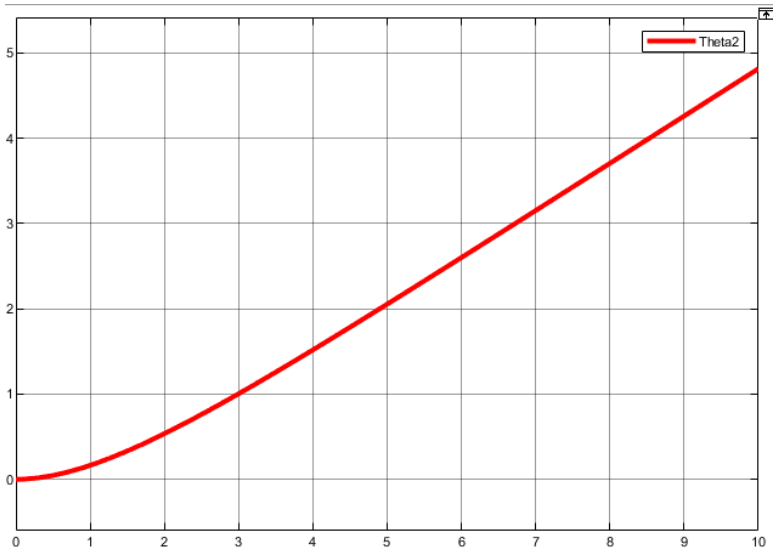
2. **Analyze the displacement, velocity, as well as acceleration of the fourth inertial element (J4), and afterward, cross-validate with the m-files!**

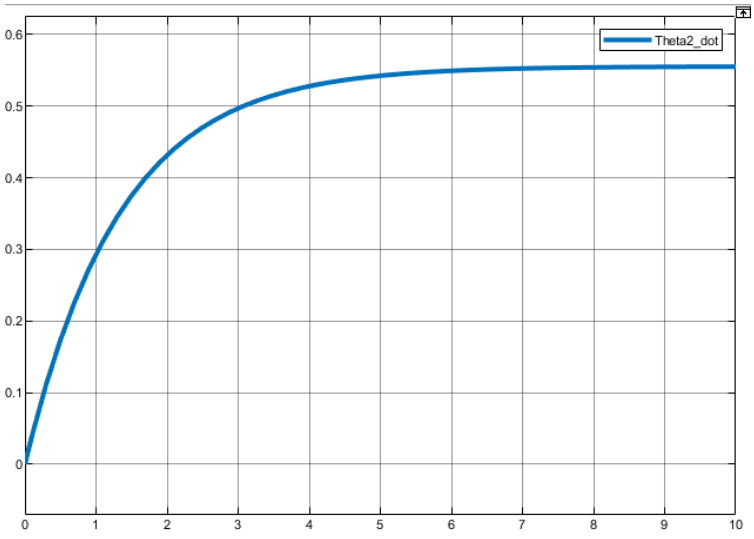


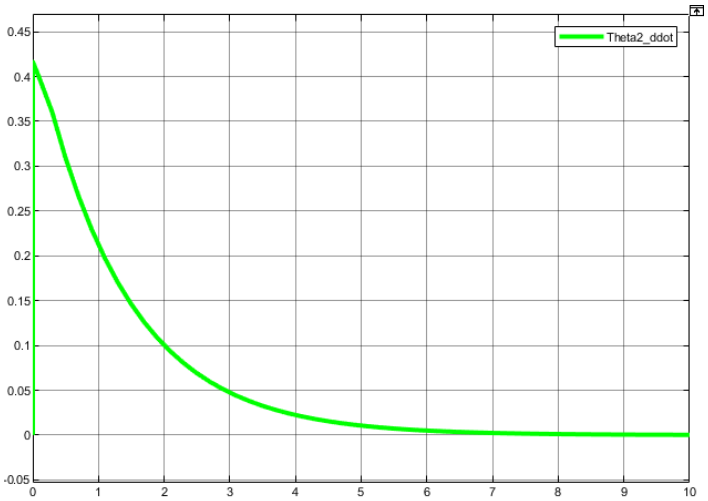
SIMSCAPE model:



Results:


Theta2
0
1
2
3
4
5
0
1
2
3
4
5
6
7
8
9
10


Theta2_dot
0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
3
4
5
6
7
8
9
10


Theta2_ddot
-0.05
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0
1
2
3
4
5
6
7
8
9
10

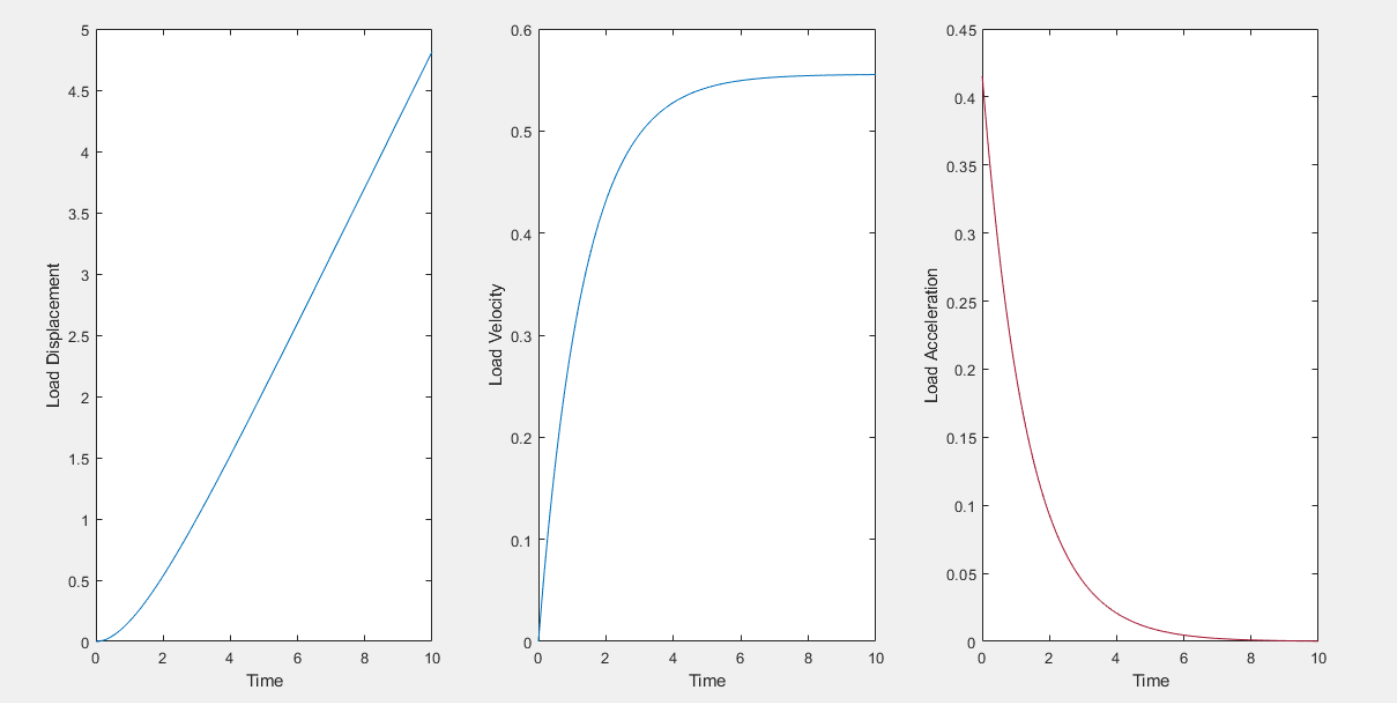
Transfer Function from SIMSCAPE model:

```
>> tf(linsys1)

ans =

  From input "input voltage" to output "Theta2":
    0.08333
  ------------
  s^2 + 0.75 s

Name: Linearization at model initial condition
Continuous-time transfer function.
```

M file result:

```
syms s V
Ra=0.25;kt=0.25;kb=0.25;Jm=1+2*(2/4)+16*(1/4)^2;Dm=32*(1/4)^2;
A=[Ra/kt , kb*s ; 1 , -(Jm*s*s+Dm*s)];
B=[V;0];
C=A\B;
thetaM=C(2);
theta2=thetaM*(1/2)*(1/2);
% transfer functions
G=theta2/V;
G=collect(G,s);
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
num=num/den(1);     %dividing by den(1) means dividing by the leading coefficient of
denominator. Done to match the outputs
den=den/den(1);
fprintf('Theta2(s)/Vin(s):');
tf(num,den)
%%%%%%%%%% plotting variables
t=0:0.01:10;
u=5*ones(1,length(t));
th2=lsim(num,den,u,t);
num2=[num(1:end) , 0]; %shifting each element to right, equivalent of multplication
by s (for obtaining derivative)
th2dot=lsim(num2,den,u,t);
th2ddot=gradient(th2dot)./gradient(t);
subplot(1,3,1);plot(t,th2);
xlabel('Time');ylabel('Load Displacement')
subplot(1,3,2);plot(t,th2dot);
xlabel('Time');ylabel('Load Velocity')
subplot(1,3,3);plot(t,th2ddot);
xlabel('Time');ylabel('Load Acceleration')
```

Following is the output




```
>> Task4MFile_Additional
Theta2(s)/Vin(s):
ans =

    0.08333
  ------------
  s^2 + 0.75 s

Continuous-time transfer function.
```

Conclusion/Remarks:

The results obtained by m file as well as the SIMSCAPE are exactly the same. Hence the developed SIMSCAPE model is correct and is in accordance with the mathematical model (m file).

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

LAB 5: Implying the block diagram reduction rules and exploring the built-in libraries of SIMSCAPE for designing solar modules.

Aim and Objectives:

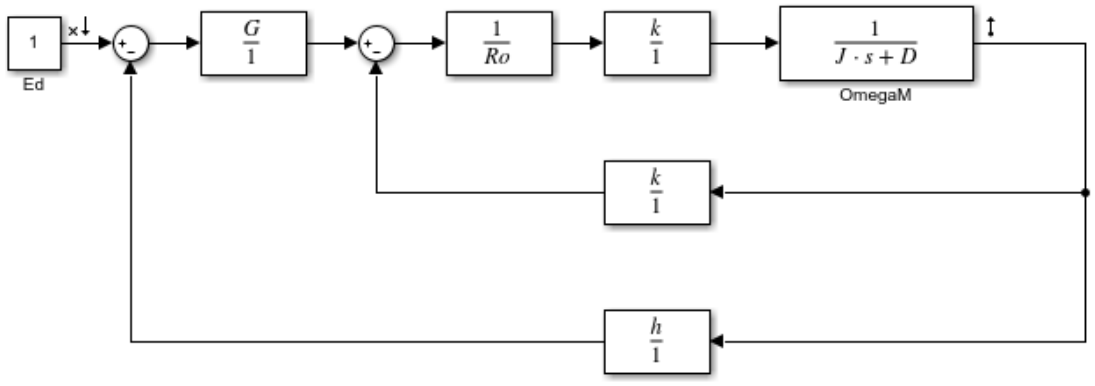
The fundamental aim of this lab is to imply the tangible block diagram reduction rules for obtaining the simplified form of the dynamic system. Different built-in Simulink blocks are incorporated to cross-validate the transfer functions with the m-files (With the deduced form of the model). Afterward, built-in libraries of SIMSCAPE are explored to design and analyze the PV and VI characteristics of the solar module(s).

1. To imply block diagram reduction rules on m-files for finding the simplified form of the dynamic system (Transfer Function).
2. To model the dynamic system in SIMULINK Environment for finding and cross-validating the transfer function.
3. To develop the block diagram of DC Motor, imply the block diagram reduction rules for finding the simplified transfer function and cross-validating the findings with the m-files.
4. To explore the different series-parallel combinations of the solar cells for the desired power rating.
5. To cross-validate the transfer function and dynamic response of SIMSCAPE Models with the results obtained from Mathematical Model and ODEs 45, respectively.

Assigned Tasks:

Block Diagram reduction of closed loop velocity control of DC motor.

SIMSCAPE model:



Mathematical Model:

$\because E_m = Ge - I_A R_o$

$\Rightarrow I_A R_o = Ge - E_m$

$\Rightarrow I_A = \frac{1}{R_o}[Ge - E_m]$

Here,

$e = E_d - E_f$

E_d = input

E_f = feedback voltage = $h\omega_m$

$E_m = k\omega_m$

Mechanical side:

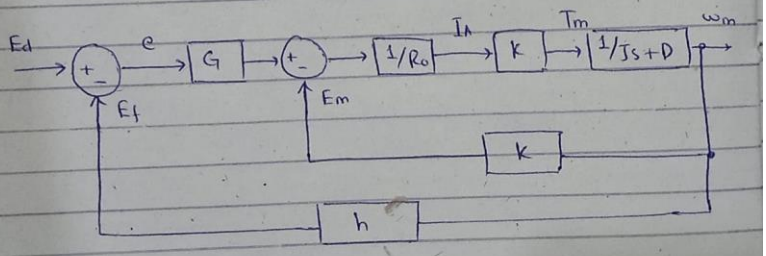
$T_m = (Js^2 + Ds)\theta_m$

$\Rightarrow T_m = (Js + D)s\theta_m$

$T_m = (Js + D)\omega_m$

and,

$T_m = K_T I_A = k I_A$



Results:

```
>> tf(linsys1)

ans =

  From input "Ed" to output "OmegaM":
    0.1
  -------
  s + 1.2

Name: Linearization at model initial condition
Continuous-time transfer function.
```

M File results:

```
G=1;h=1;k=1;Ro=10;J=1;D=1;
g1=tf(G,1);
g2=tf(1,Ro);
g3=tf(k,1);
g4=tf(1,[J,D]);
feed1=tf(k,1);
feed2=tf(h,1);
sys1=series(g2,g3);
sys2=series(sys1,g4);
sys3=feedback(sys2,feed1,-1);
sys4=series(sys3,g1);
sys5=feedback(sys4,feed2);
[num,den]=tfdata(sys5);
num=cell2mat(num);den=cell2mat(den);
num=num/den(1);den=den/den(1);
fprintf("Omega(s)/Ed(s):");tf(num,den)
```

Following are the results.

```
Omega(s)/Ed(s):
ans =

    0.1
  -------
  s + 1.2

Continuous-time transfer function.
```

Conclusion/Remarks:

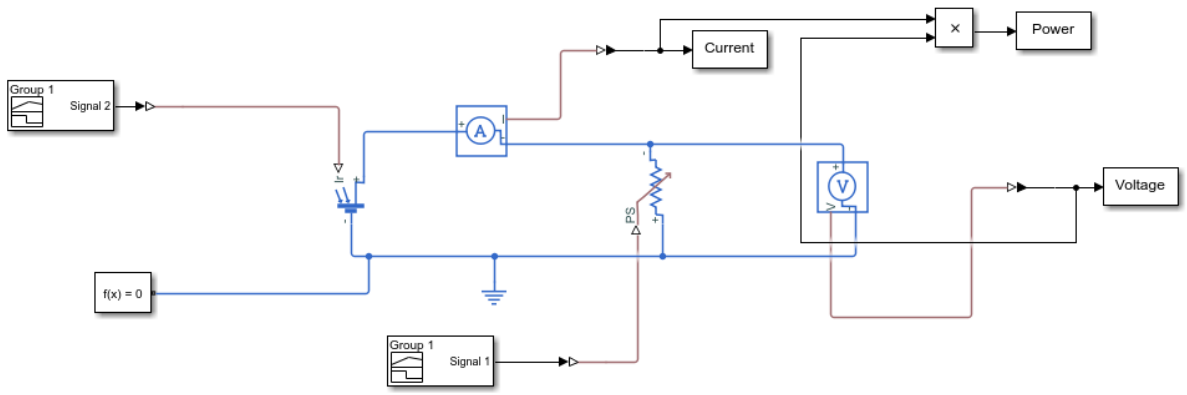
It can clearly be seen that the transfer function obtained by both the SIMSCAPE model as well as by the m file is exactly the same. Another thing to add is that without the feedback (tachometer) the transfer function comes out to be 0.1/(s+1.1). This corresponds to a time period of 0.909s (1/1.1) whereas with the feedback the time period comes out to be 0.833s (1/1.2). This indicates the advantage of using closed loop control system i.e., the steady state value is achieved relatively faster in closed loop system.

Model 2:

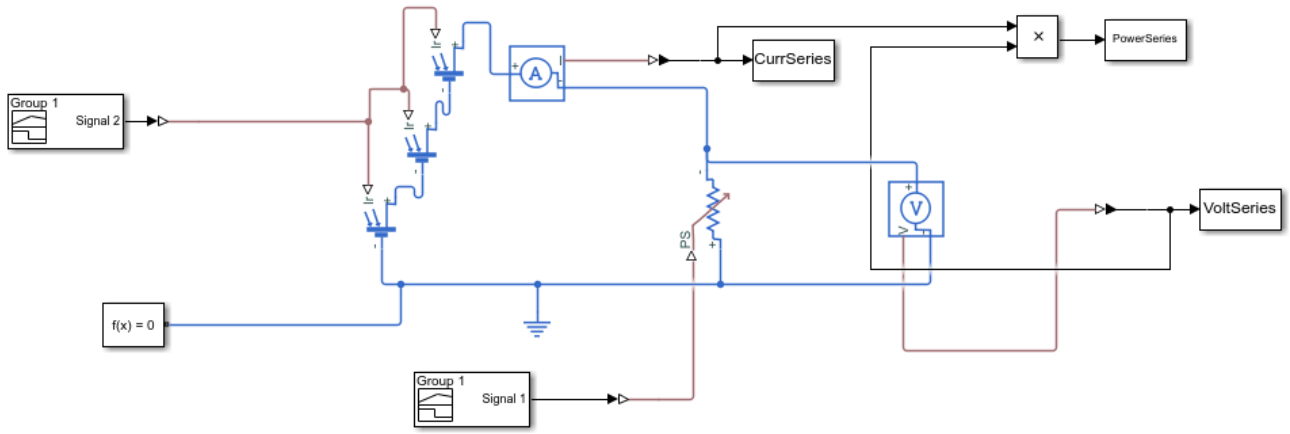
Develop a user-friendly way/model to illustrate the PV and VI characteristics of Solar Cells under five different values of Irradiance (600-1000 W/m^2). Afterwards, develop the three series and parallel connected solar cells to illustrate the PV and VI characteristics under the influence of different values of Irradiance (600-1000 W/m^2).

SIMSCAPE model:

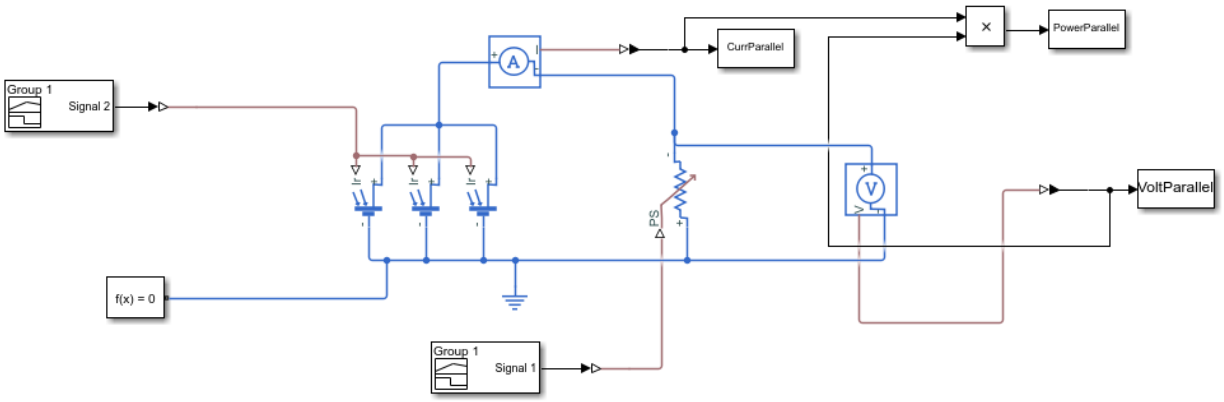
a) Single Solar Cell:



b) Solar Cells in Series:



c) Solar Cells in Parallel:



Working/Methodology:

The SIMSCAPE model is quite straightforward except maybe the signal builder part. In this model, two signal builders have been incorporated; one for setting the irradiance value whereas second for varying the potentiometer (to observe different values of current). The way it works is that 5 different irradiances have been provided in a step like fashion for equal intervals (1 second in this case). Corresponding to each step, there's a ramp signal of equal interval provided to the potentiometer, hence we have 5 ramp signals. So, for each value of irradiance the resistance varies from 0 to 1 ohm giving us the required VI/PV characteristics.

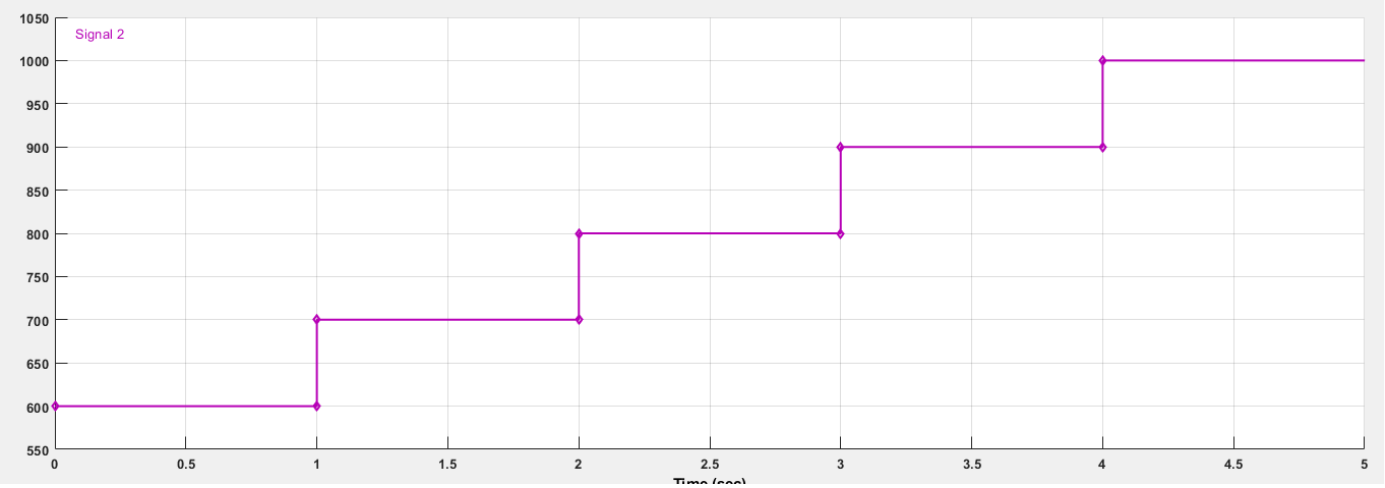


Figure: Signal Builder for Irradiance

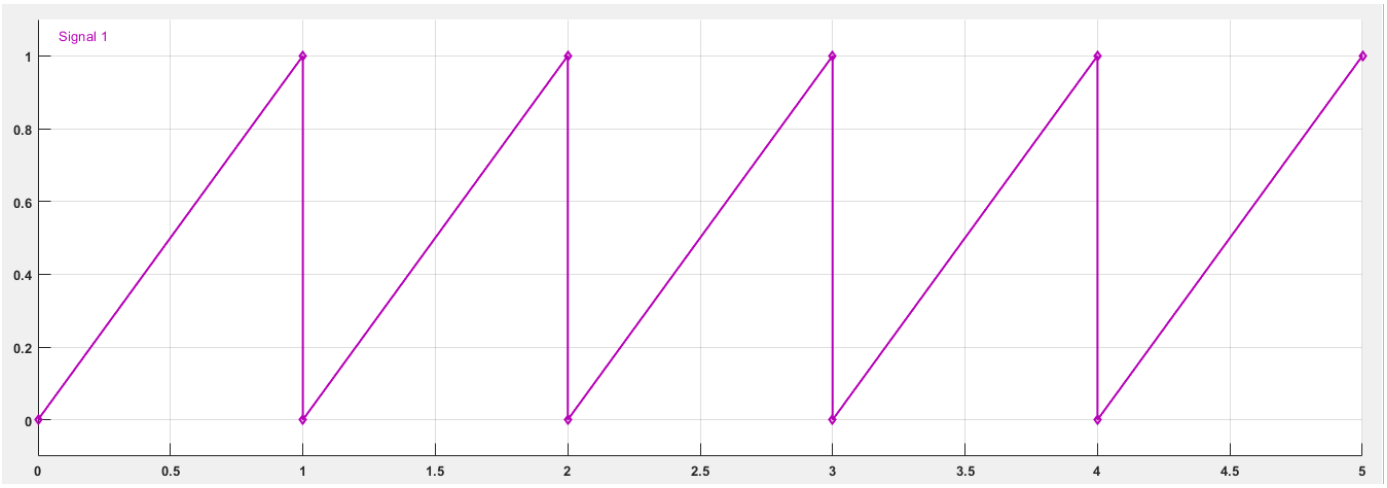
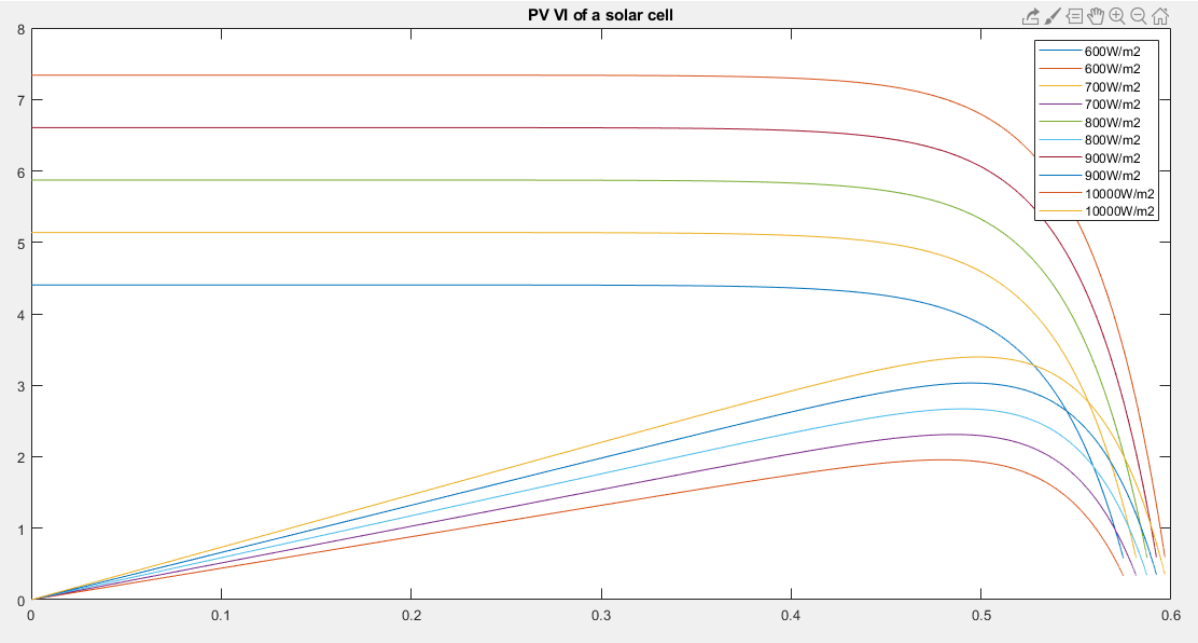


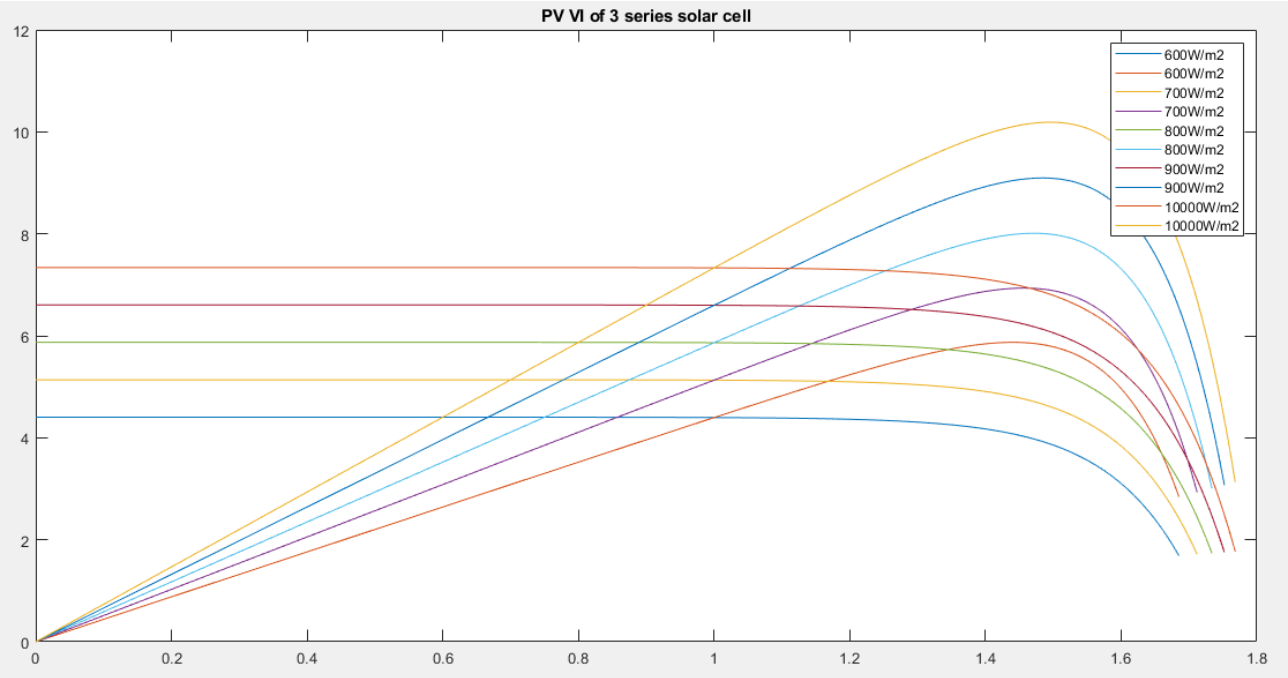
Figure: Signal Builder for Potentiometer

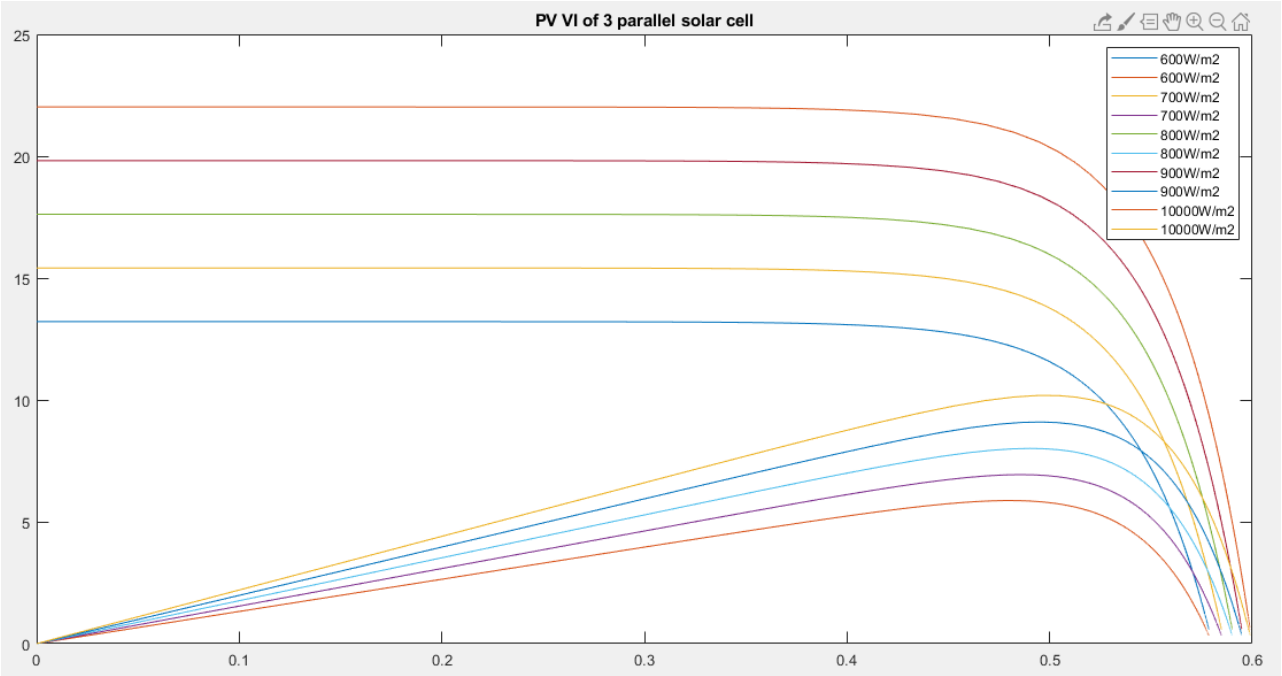
M File:

```
figure;
for i=0:4

plot(out.Voltage(i*1000+1:((i+1)*1000)),out.Current(i*1000+1:((i+1)*1000)),out.Volta
ge(i*1000+1:((i+1)*1000)),out.Power(i*1000+1:((i+1)*1000)));
    hold on
end
title('PV VI of a solar cell');
figure;
for i=0:4

plot(out.VoltSeries(i*1000+1:((i+1)*1000)),out.CurrSeries(i*1000+1:((i+1)*1000)),out
.VoltSeries(i*1000+1:((i+1)*1000)),out.PowerSeries(i*1000+1:((i+1)*1000)));
    hold on
end
title('PV VI of 3 series solar cell');
figure;
for i=0:4

plot(out.VoltParallel(i*1000+1:((i+1)*1000)),out.CurrParallel(i*1000+1:((i+1)*1000))
,out.VoltParallel(i*1000+1:((i+1)*1000)),out.PowerParallel(i*1000+1:((i+1)*1000)));
    hold on
end
title('PV VI of 3 parallel solar cell');
for i=1:3
    figure(i);

legend('600W/m2','600W/m2','700W/m2','700W/m2','800W/m2','800W/m2','900W/m2','900W/m
2','10000W/m2','10000W/m2');
end
```

Results:







Conclusion/Remarks:

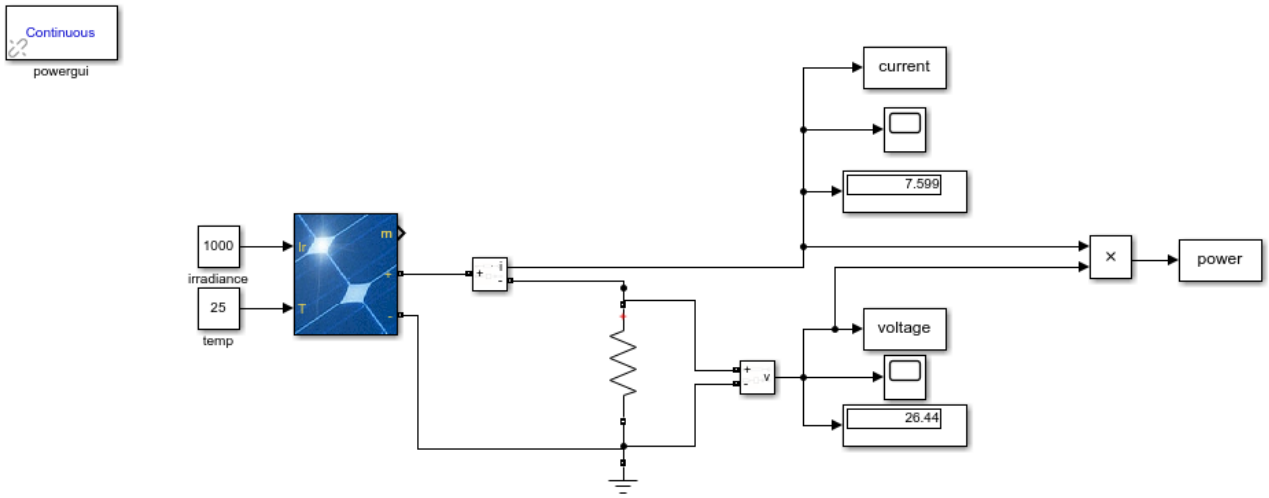
The PV/IV results obtained match the ideal results and are logical as well. For example, the short circuit current is 7.36 A whereas the open circuit voltage is 0.6 V for a solar cell. When three cells are attached in series, the current remains the same whereas voltage comes out to be three times (~1.8 V as evident in the graph). Similarly, when three cells are connected in parallel, the voltage remains the same whereas the current becomes thrice of its original value (~22 A as evident in the graph).

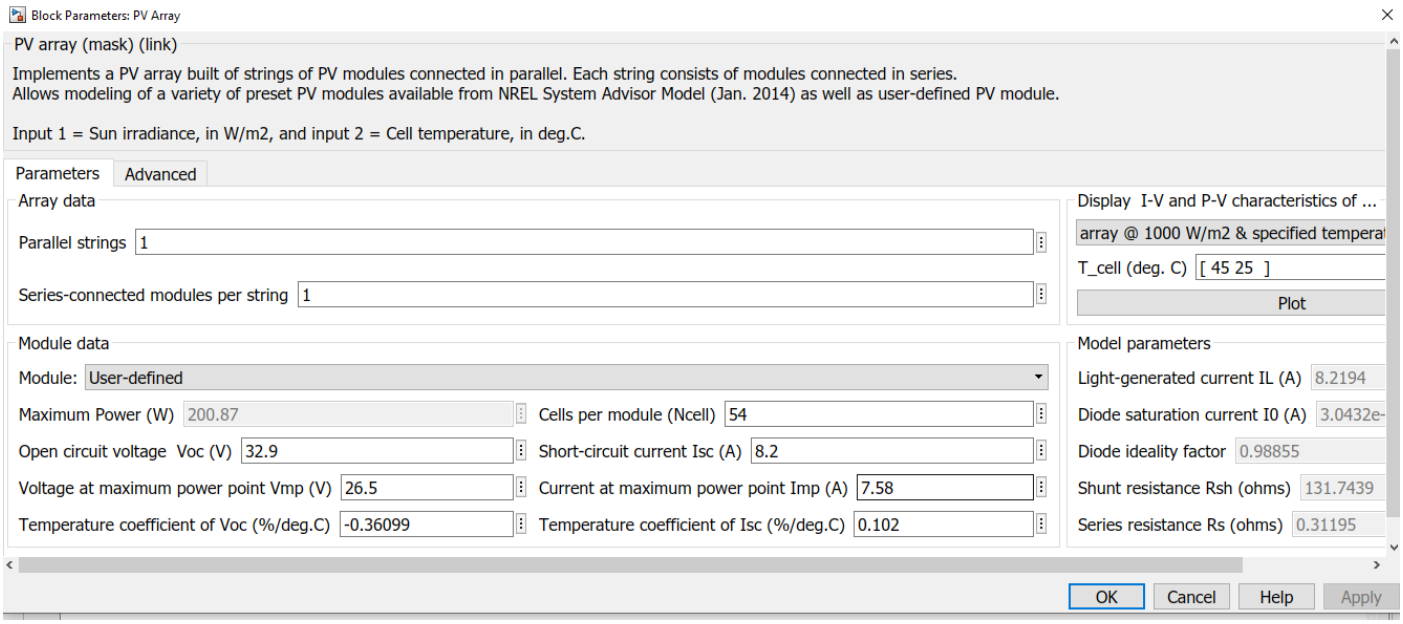
Model 3:

Design a Solar/Photo-voltaic array having the following specifications using MATLAB SIMSCAPE and analyze the results.

Sr#	Individual Parameters	Rated Value
1	Rated Power	~200 W
2	Voltage at maximum power	26.5 V
3	Current at maximum power	7.58 A
4	Open Circuit Voltage	32.9 V
5	Short Circuit Current	8.2 A
6	Number of Cells in series	54
7	Number of Cells in Parallel	1

SIMSCAPE model:

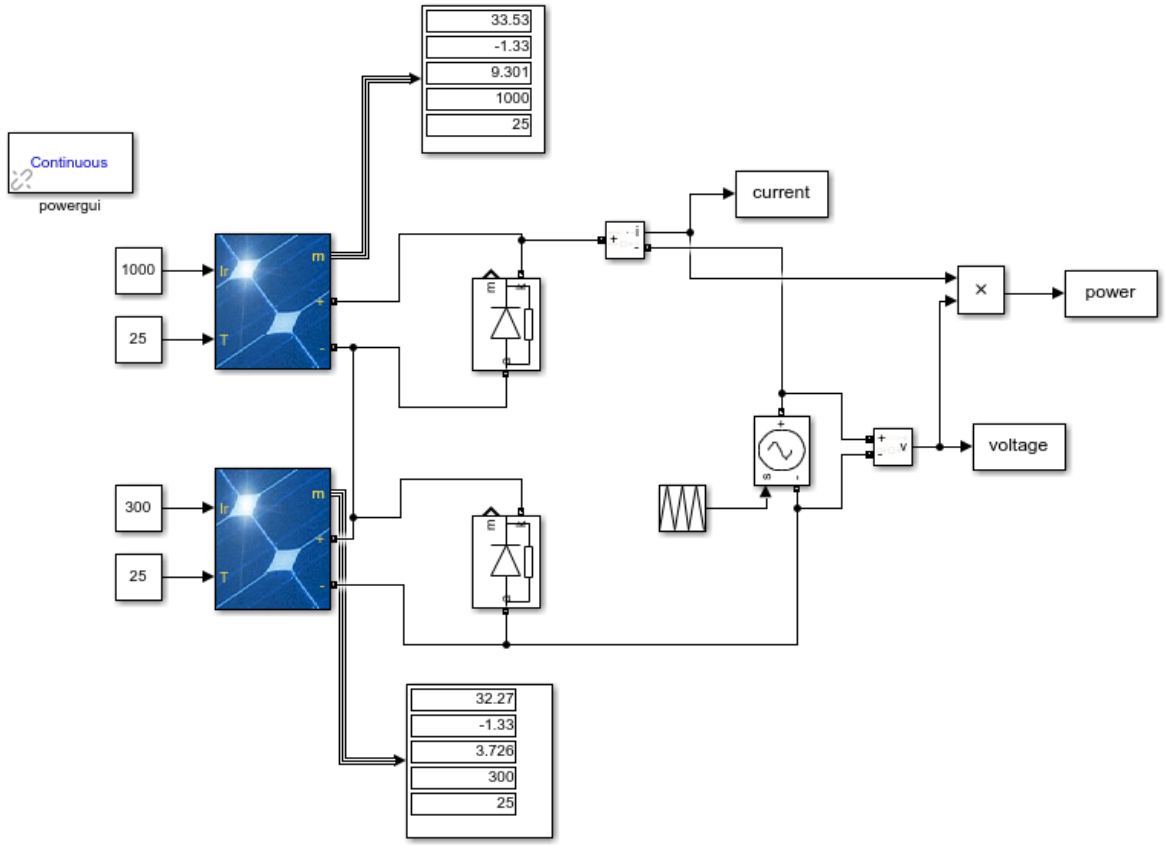




Model 4:

Design a Solar/Photo-voltaic array under partial shading condition using MATLAB SIMSCAPE and analyze PV and VI characteristics of the model.

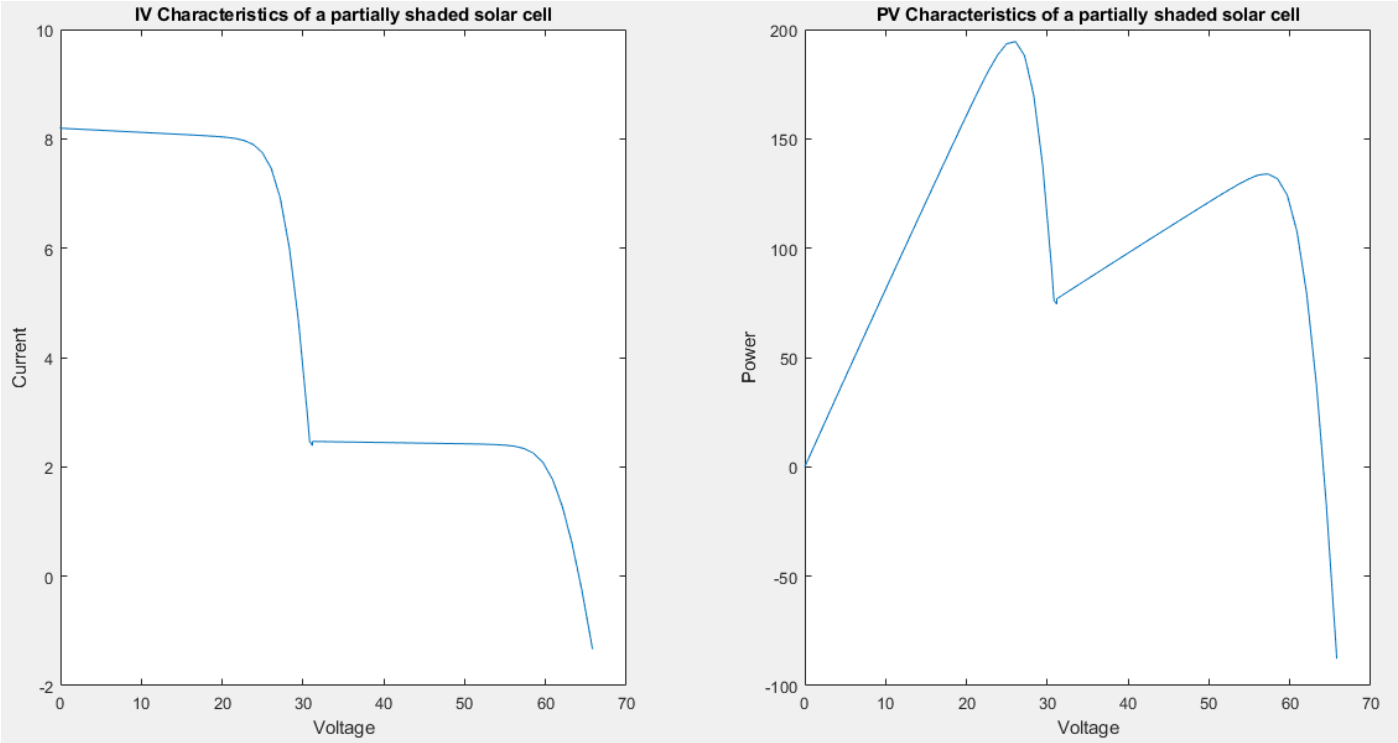
SIMSCAPE model:



M file:

```
subplot(1,2,1);
plot(out.voltage,out.current);
xlabel('Voltage');ylabel('Current');
title('IV Characteristics of a partially shaded solar cell');
subplot(1,2,2);
plot(out.voltage,out.power);
xlabel('Voltage');ylabel('Power');
title('PV Characteristics of a partially shaded solar cell');
```

Results:



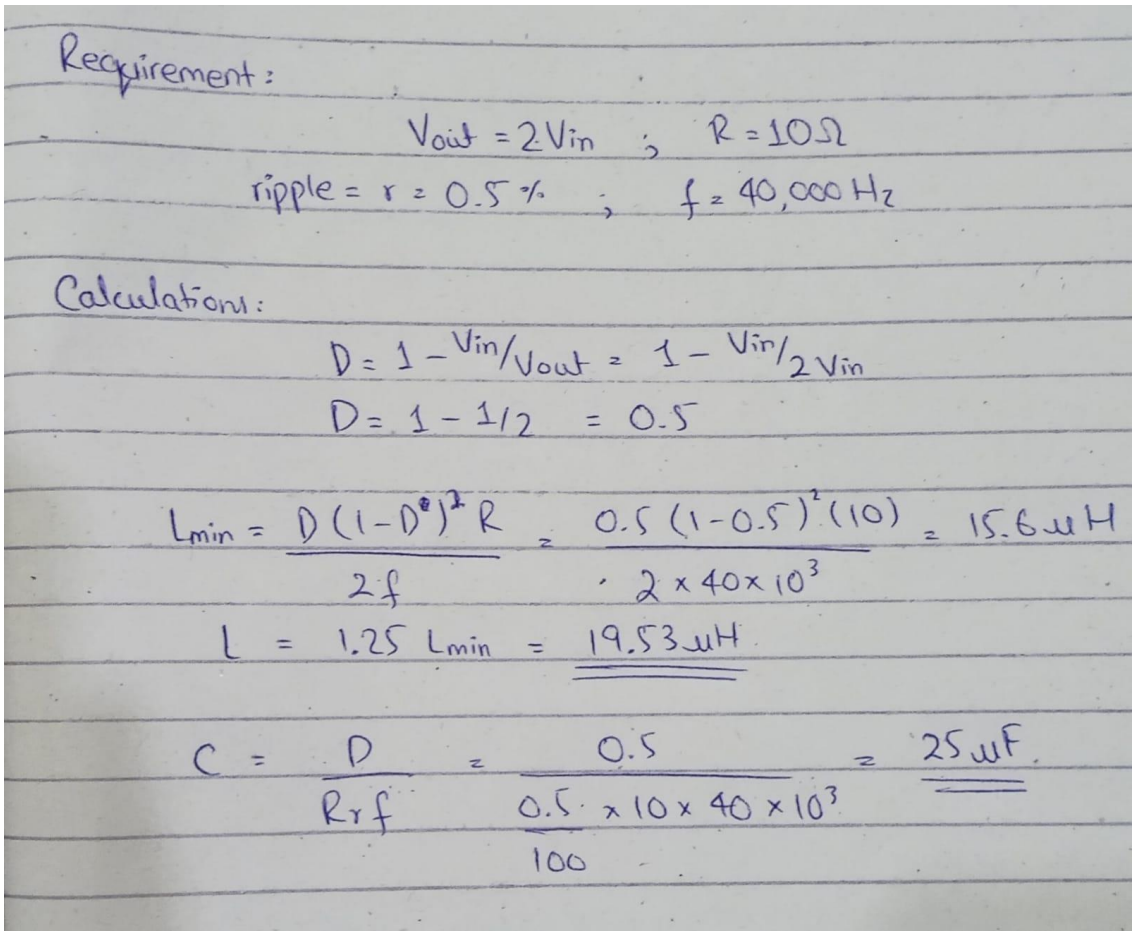
Conclusion/Remarks:

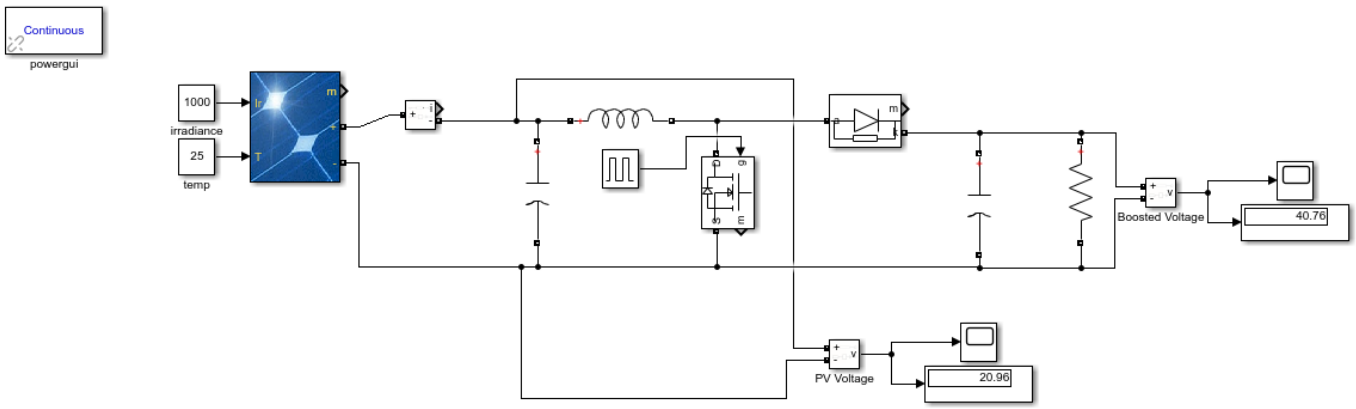
Under partial shading conditions, photovoltaic (PV) arrays are subjected to different irradiance levels caused by nonuniform shading. As a result, a mismatch between the modules, a reduction in the power generated, increase in operating temperature and the hotspot phenomenon will be observed. This is also evident in the graph which shows that both power and current decrease and have different peaks at a higher voltage. Also, the individual electric parameters provided by the cells are quite different (as shown in the display block).

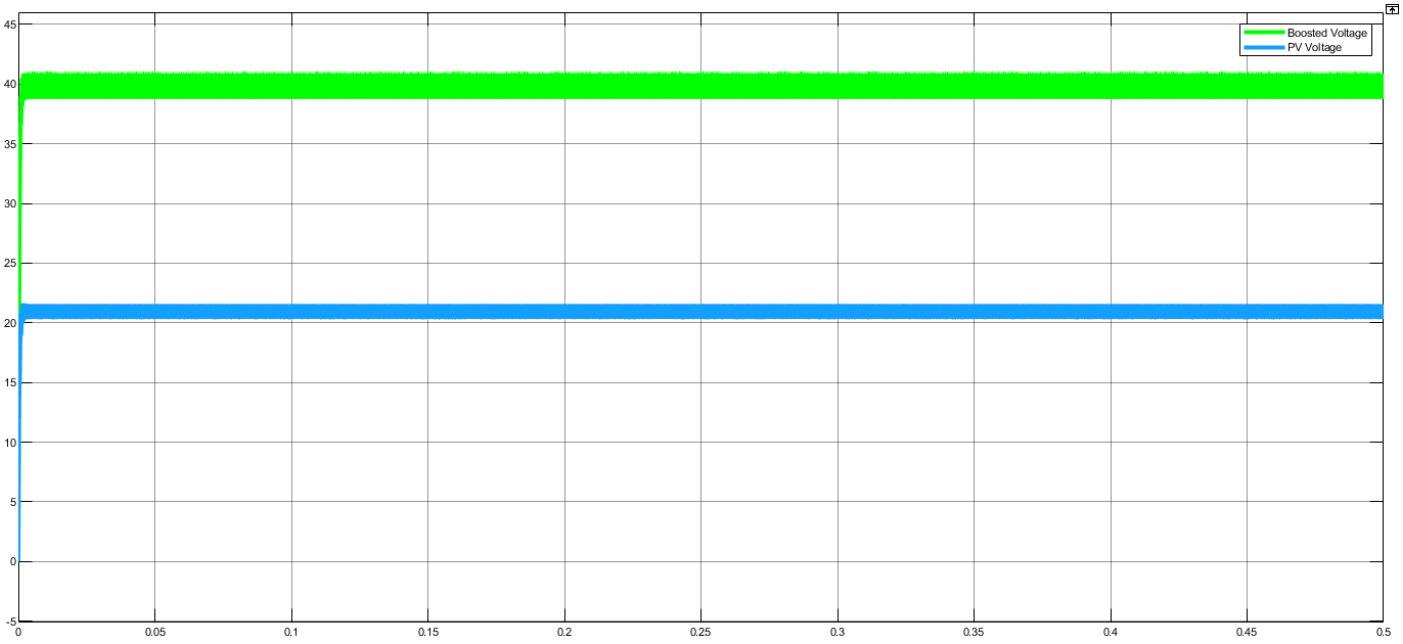
Model 5:

Integrate a boost converter with the developed model of the PV array to see the increased electrical parameters of the system.

Mathematical Calculation:





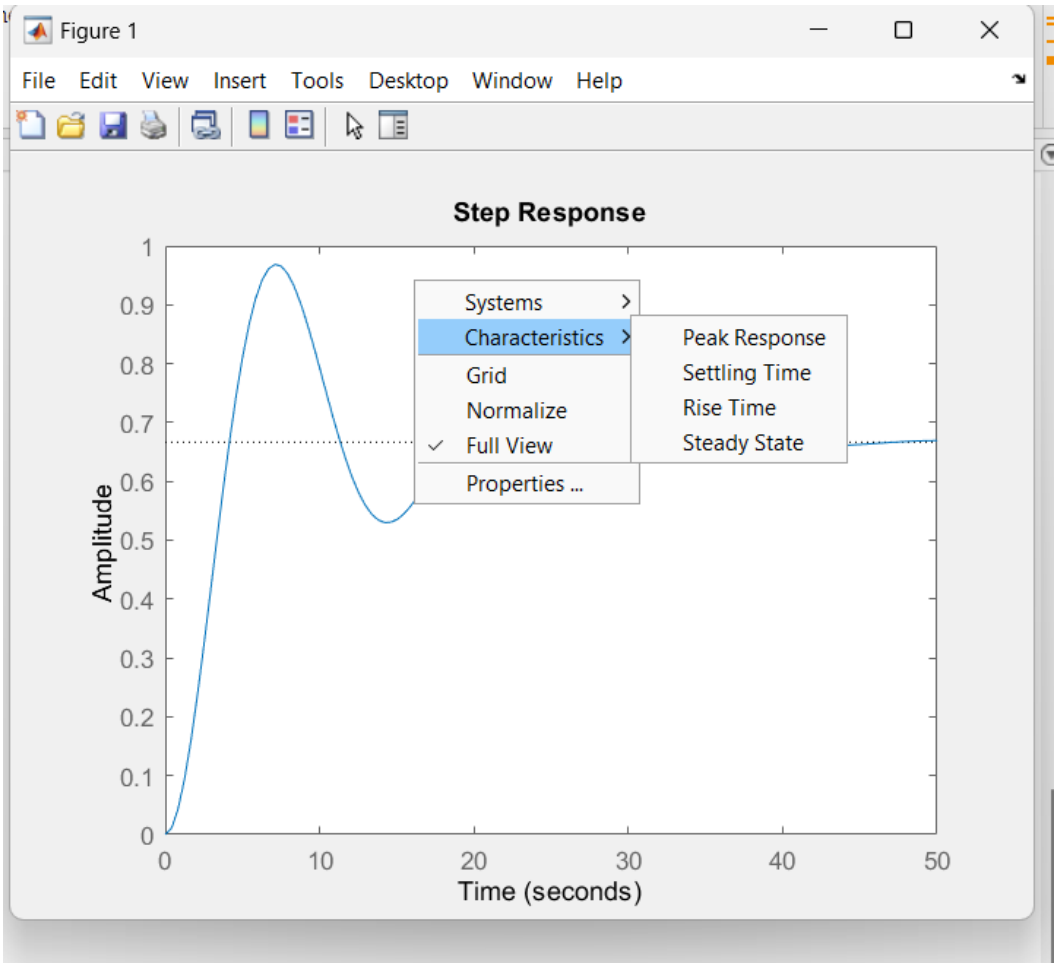

Boosted Voltage
PV Voltage
45
40
35
30
25
20
15
10
5
0
-5
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

LAB#06: Analyze the time-domain characteristics of dynamic system: Influence of natural frequency and damping ratio on the output response. Find different static error constants and sensitivity of the system for predefined discrete values.

Objective: The fundamental aim of this lab is the implication of studied basic concepts of control system on the realistic dynamic systems. The certain tangible objectives of this lab are underlined below;

1. To evaluate the performance of the dynamic systems by considering time domain performance metrices.
2. By implying the studied characteristics, develop the MATLAB code for accessing the characteristics of the system under the predefined discrete values.
3. To depict the influence of natural frequency and damping ratio on the response of the dynamic system.
4. Calculate the steady state error, static error constant(s), and sensitivity of the given systems.

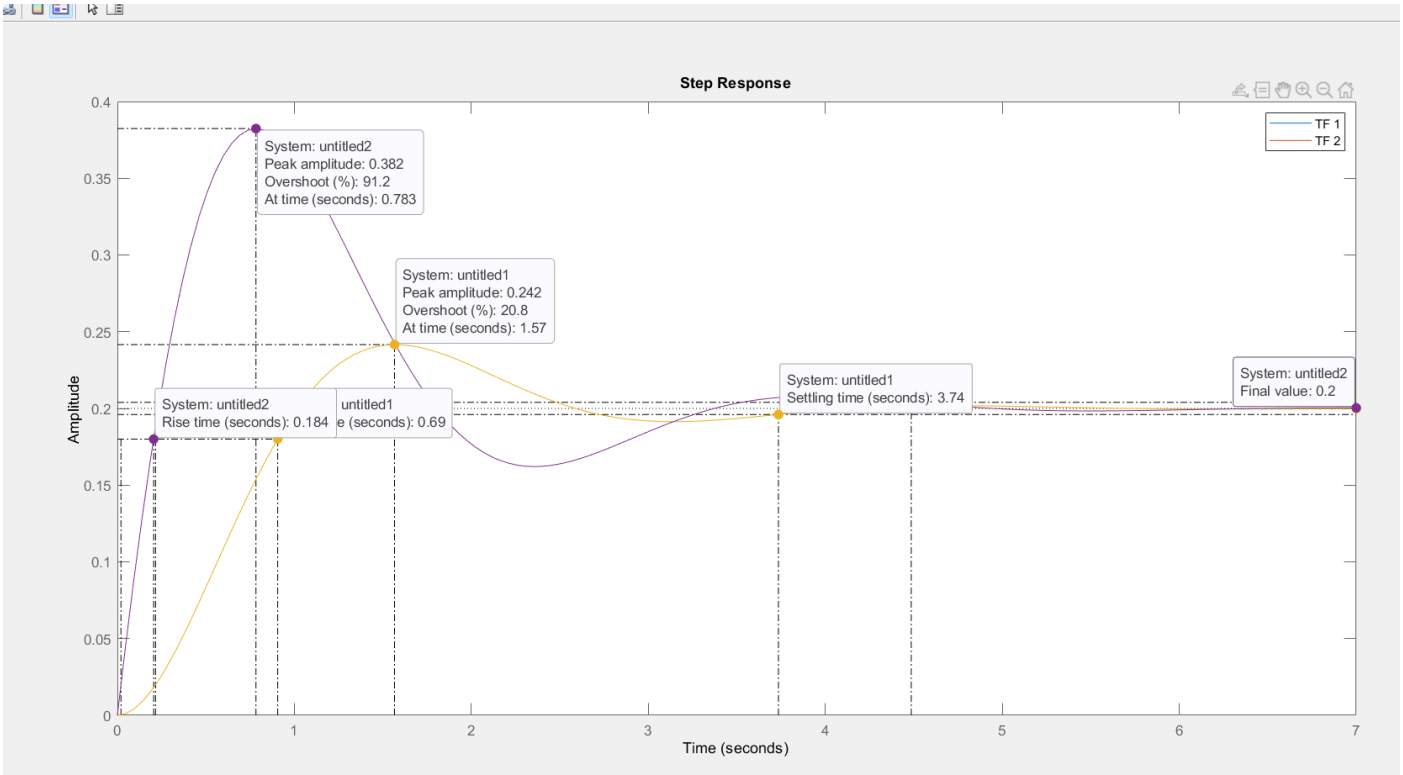
```
%% Finding time domain characteristics of Rotational Geared
System
numg=0.1356;
deng=[1 0.2203 0.2034];
step(tf([numg],[deng]))
omegan=sqrt(deng(3)/deng(1))
zeta=(deng(2)/deng(1))/(2*omegan)
Ts=4/(zeta*omegan)%settling time
omegad=omegan*sqrt(1-zeta^2)
pos=100*exp(-zeta*pi/(sqrt(1-zeta^2)))
Tr=(1.768*zeta^3 -0.417*zeta^2+1.039*zeta+1)/omegan;
```



$TS = 1.2/W_N$

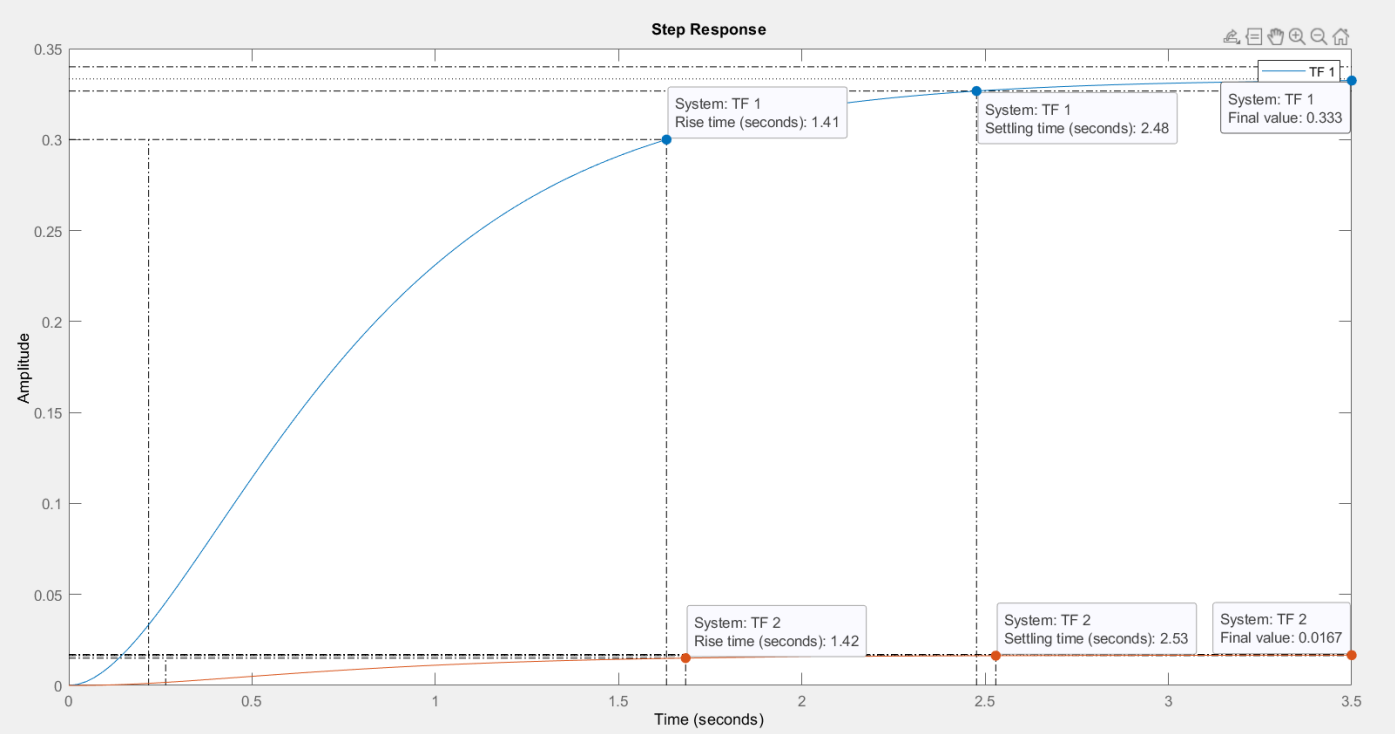
```
%% Finding time domain characteristics of Rotational Geared
System
numg=1;
deng=[1 2 5];
step(tf([numg],[deng]))
omegan=sqrt(deng(3)/deng(1))
zeta=(deng(2)/deng(1))/(2*omegan)
Ts=4/(zeta*omegan)%settling time
omegad=omegan*sqrt(1-zeta^2)
pos=100*exp(-zeta*pi/(sqrt(1-zeta^2)))
Tr=(1.768*zeta^3 -0.417*zeta^2+1.039*zeta+1)/omegan;

hold on
%%
%% Finding time domain characteristics of Rotational Geared
System
numg=[1 1];
deng=[1 2 5];
step(tf([numg],[deng]))
omegan=sqrt(deng(3)/deng(1))
zeta=(deng(2)/deng(1))/(2*omegan)
```

```
Ts=4/(zeta*omegan)%settling time
omegad=omegan*sqrt(1-zeta^2)
pos=100*exp(-zeta*pi/(sqrt(1-zeta^2)))
Tr=(1.768*zeta^3 -0.417*zeta^2+1.039*zeta+1)/omegan;

legend("TF 1","TF 2")
%influence of purelt zero
%influence of non minimal basis? atomic energy.
```



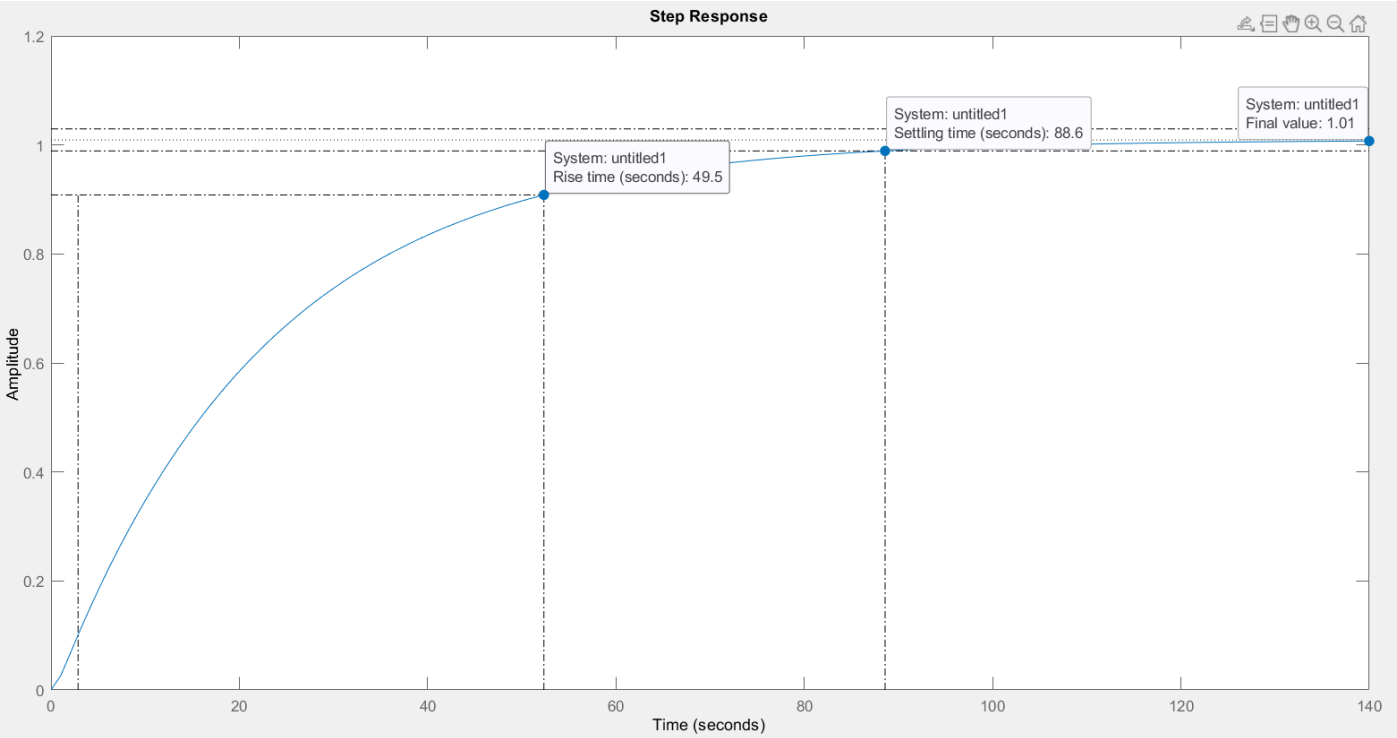

```
%% Finding time domain characteristics of Rotational Geared
System
numg=2;
deng=[1 5 6];
step(tf([numg],[deng]))
omegan=sqrt(deng(3)/deng(1))
zeta=(deng(2)/deng(1))/(2*omegan)
Ts=4/(zeta*omegan)%settling time
omegad=omegan*sqrt(1-zeta^2)
pos=100*exp(-zeta*pi/(sqrt(1-zeta^2)))
Tr=(1.768*zeta^3 -0.417*zeta^2+1.039*zeta+1)/omegan;

hold on
%%
%% Finding time domain characteristics of Rotational Geared
System s+20 here for third pole non dominant
```

```
numg=[2];
deng=[1 25 106 120];
step(tf([numg],[deng]))
omegan=sqrt(deng(3)/deng(1))
zeta=(deng(2)/deng(1))/(2*omegan)
Ts=4/(zeta*omegan)%settling time
omegad=omegan*sqrt(1-zeta^2)
pos=100*exp(-zeta*pi/(sqrt(1-zeta^2)))
Tr=(1.768*zeta^3 -0.417*zeta^2+1.039*zeta+1)/omegan;

legend("TF 1","TF 2")
```



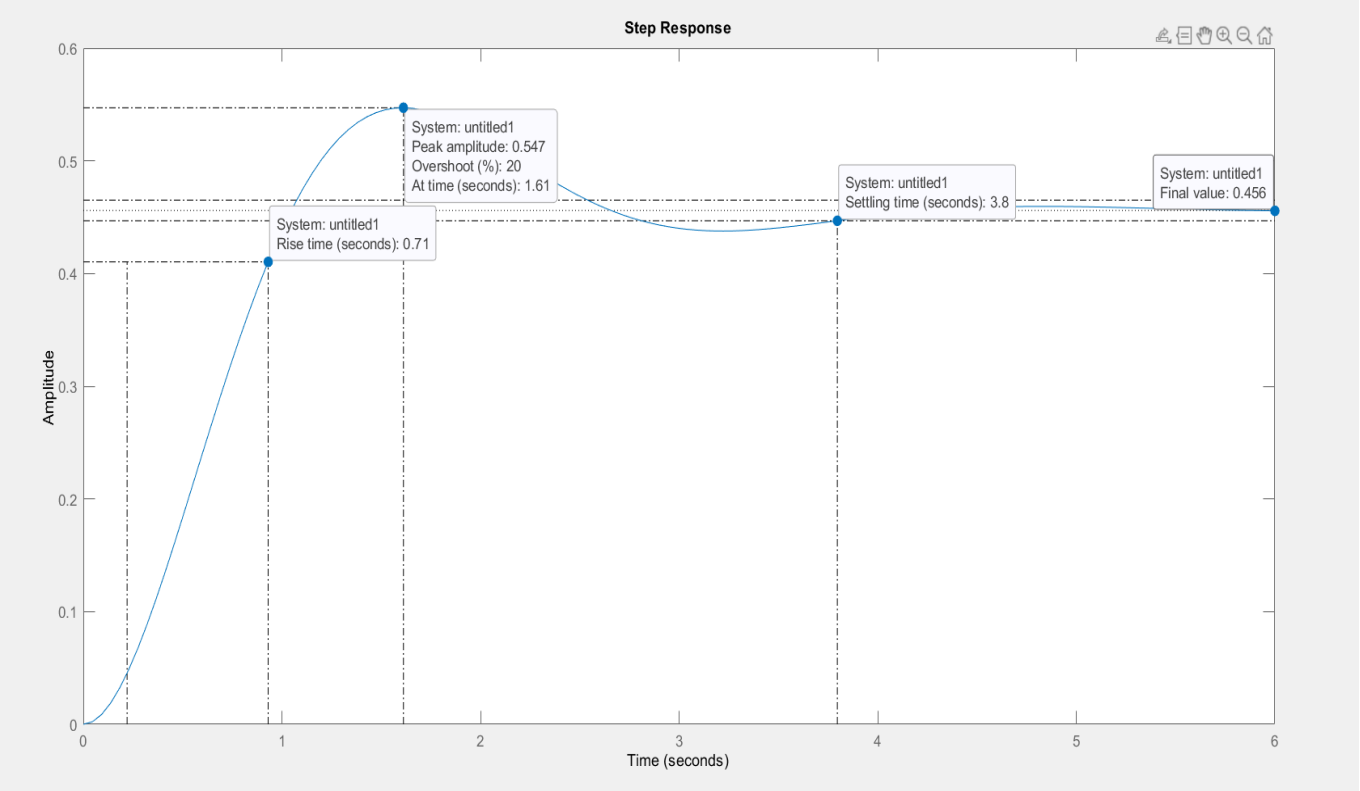
Here is an example of a critically damped system

```
%% Finding time domain characteristics of Rotational Geared System
%N1=20;N2=40;Je=5;De=2;Ke=2;
numg=[0.08982];
deng=[1 2.048 0.089];
step(tf([numg],[deng]))
omegan=sqrt(deng(3)/deng(1))
zeta=(deng(2)/deng(1))/(2*omegan)
Ts=4/(zeta*omegan)%settling time
omegad=omegan*sqrt(1-zeta^2)
pos=100*exp(-zeta*pi/(sqrt(1-zeta^2)))
Tr=(1.768*zeta^3 -0.417*zeta^2+1.039*zeta+1)/omegan;

hold on
```



What if we need to compute transfer function from characteristics?

```
%%% code for calculating the TF from % 20% percentage overshoot
and 4
%%% second settling time

pos=0.2;
pos = (log(pos))^2;
zeta = sqrt(pos/(pi^2+pos));
Ts = 4;
omegan = 4/(zeta*Ts);

deng = [0 0 0];
deng(1) = 1
deng(2) = 2*zeta*omegan
deng(3) = omegan^2

numg = omegan

step(tf([numg],[deng]))

tf(numg/deng)
```



Example#04: From nowonwards, different examples of the dynamic systems (In terms of transfer function) will be considered to obtain and analyze the influence of each individual performance metrices. In this example, the students will observe the following underlined action items;

- ✓ **Influence of natural frequency and damping ratio.**
- ✓ **Different static error constants.**

$$G(s) = \frac{1000(s+8)}{(s+7)(s+9)}$$

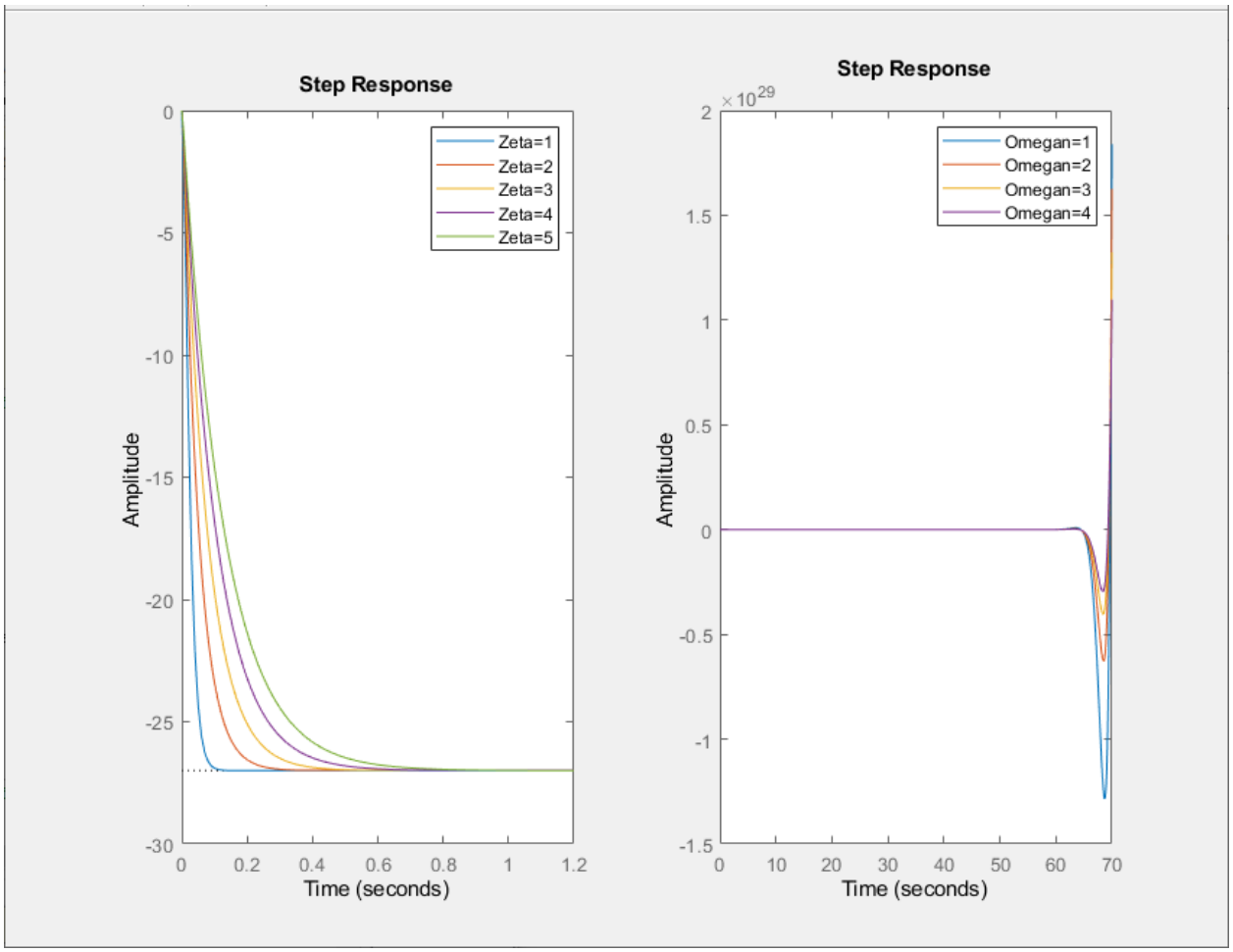
```
num1 = 1000*[1 8];
den1 = poly([-7 -9]);
G1 = tf(num1, den1);
damp(G1);
[wn, zeta, k] = damp(G1)
%% Generic Representation to illustrate the influence the Damping Ratio and Natural frequency!
zeta = 1:5;
k = -27;
omegan = 78;
figure;
% Plot for varying values of zeta
subplot(1, 2, 1)
hold on;
for i = 1:length(zeta)
    b = tf([k * omegan^2], [1 2 * zeta(i) * omegan omegan^2]);
    step(b);
end
hold off;
```

```
legend('Zeta=1', 'Zeta=2', 'Zeta=3', 'Zeta=4', 'Zeta=5');
%% Generic Representation to illustrate the influence of Omega!
% Plot for varying values of omegan
subplot(1, 2, 2)
omegan_vals = 1:5;
a = tf(k * omegan_vals.^2, [1 2 * zeta(1) * omegan_vals zeta(1) * omegan_vals.^2]);
hold on;
for i = 2:length(zeta)
    b = tf(k * omegan_vals.^2, [1 2 * zeta(i) * omegan_vals zeta(i) * omegan_vals.^2]);
    step(b);
end
hold off;
legend('Omegan=1', 'Omegan=2', 'Omegan=3', 'Omegan=4', 'Omegan=5');

%% Determine the type of the control system
% Calculate the number of poles at zeroes in the transfer function
poles_at_origin = sum(den1 == 0);
if poles_at_origin == 0
    fprintf('The system is of type 0 .\n');
elseif poles_at_origin == 1
    fprintf('The system is of type 1.\n');
elseif poles_at_origin == 2
    fprintf('The system is of type 2.\n');
else
    fprintf('The system is of type 3 or higher.\n');
end

%% Finding the steady-state errors
%Finding error constants: Kp
G1=tf(num1,den1)
Kp = dcgain(G1)
%Finding error constants: Kv
z2=[0, -8];
p=[-7, -9];
k=[1000];
H2=zpk(z2, p, k);
G2=tf(H2)
Kv = dcgain(G2)
%Finding error constants: Ka
z3=[0,0,-8];
p=[-7, -9];
k=[1000];
H3=zpk(z3, p, k);
G3=tf(H3)
Ka = dcgain(G3)
%% Finding Test Signals
% Display the steady-state errors
step_error= 1 / (1 + Kp);
ramp_error= 1 /  Kv;
parabolic_error= 1 /  Ka ;
% Step Signal Implications
fprintf('Steady-state error for standard step input: %.4f\n', step_error);
% Ramp Signal Implications
fprintf('Steady-state error for ramp input: %.4f\n', ramp_error);
% Parabola Signal
fprintf('Steady-state error for parabolic input: %.4f\n', parabolic_error);
```

Output:

```
>> Example_1

The system is of type 0 .

G1 =


   1000 s + 8000

  ---------------

  s^2 + 16 s + 63
```

```
Continuous-time transfer function.

Kp =

  126.9841

G2 =

  1000 s^2 + 8000 s

  -----------------

   s^2 + 16 s + 63

Continuous-time transfer function

Kv =

     0

G3 =

  1000 s^3 + 8000 s^2

  -------------------

    s^2 + 16 s + 63

Continuous-time transfer function.

Ka =

     0

Steady-state error for standard step input: 0.0078
Steady-state error for ramp input: Inf
Steady-state error for parabolic input: Inf
```



Example#05: In this example, the students will observe the following underlined action items;

- ✓ **Influence of natural frequency and damping ratio.**
- ✓ **Different static error constants.**

$$G(s) = \frac{K(s+12)}{(s+14)(s+18)}$$

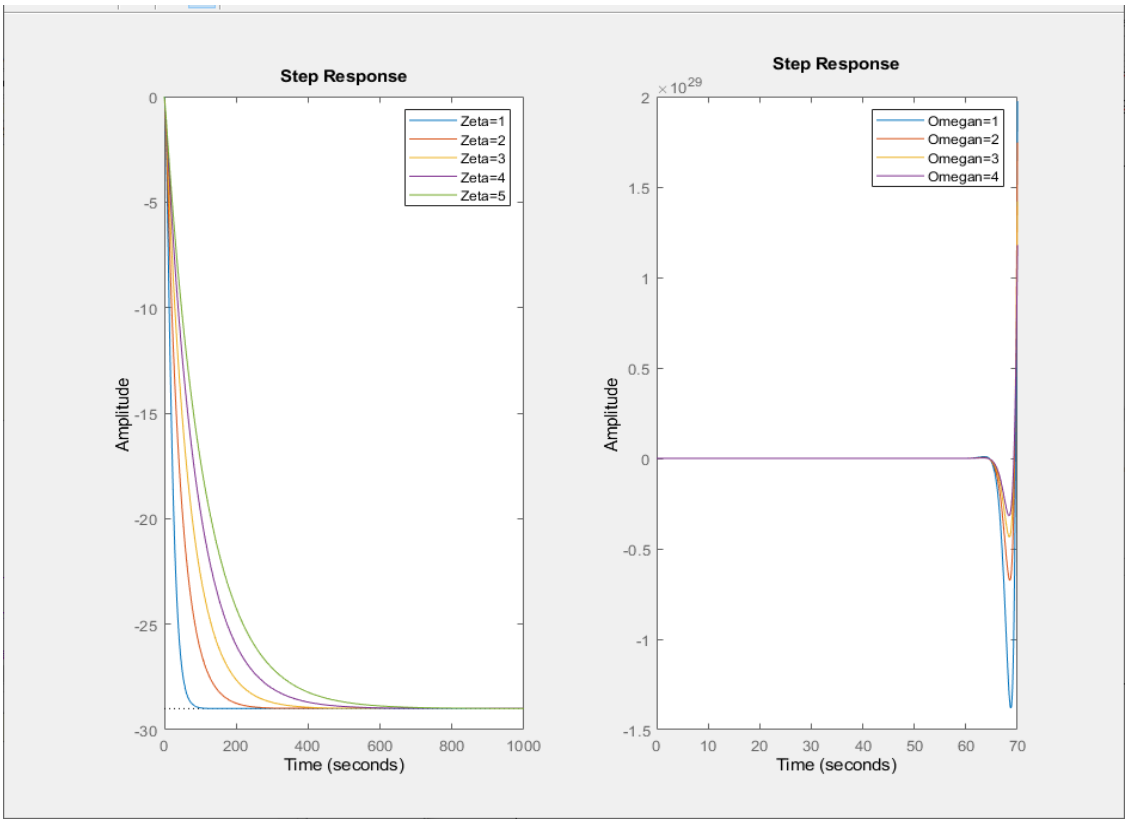
```
num4=10*[1 12];
den4=poly([-14 -18]);
G4 = tf(num4, den4);
damp(G4);
[wn, zeta, k] = damp(G4)

%% Generic Representation to illustrate the influence the Damping Ratio and Natural frequency!
zeta = 1:5;
k = -29;
omegan = 0.09;
figure;
% Plot for varying values of zeta
subplot(1, 2, 1)
hold on;
for i = 1:length(zeta)
    b = tf([k * omegan^2], [1 2 * zeta(i) * omegan omegan^2]);
    step(b);
end
hold off;
legend('Zeta=1', 'Zeta=2', 'Zeta=3', 'Zeta=4', 'Zeta=5');
```

```
%% Generic Representation to illustrate the influence of Omega!
% Plot for varying values of omegan
subplot(1, 2, 2)
omegan_vals = 1:5;
a = tf(k * omegan_vals.^2, [1 2 * zeta(1) * omegan_vals zeta(1) * omegan_vals.^2]);
hold on;
for i = 2:length(zeta)
    b = tf(k * omegan_vals.^2, [1 2 * zeta(i) * omegan_vals zeta(i) * omegan_vals.^2]);
    step(b);
end
hold off;
legend('Omegan=1', 'Omegan=2', 'Omegan=3', 'Omegan=4', 'Omegan=5');

%% Determine the type of the control system
% Calculate the number of poles at zeroes in the transfer function
poles_at_origin = sum(den4 == 0);
if poles_at_origin == 0
    fprintf('The system is of type 0 .\n');
elseif poles_at_origin == 1
    fprintf('The system is of type 1.\n');
elseif poles_at_origin == 2
    fprintf('The system is of type 2.\n');
else
    fprintf('The system is of type 3 or higher.\n');
end

%% Finding the steady-state errors
%Finding error constants: Kp
G4=tf(num4,den4)
Kp = dcgain(G4)

%Finding error constants: Kv
z5=[0, -12];
p=[-14, -18];
k=[10];
H5=zpk(z5, p, k);
G5=tf(H5)
Kv = dcgain(G5)

%Finding error constants: Ka
z6=[0,0,-12];
p=[-14, -18];
k=[10];
H6=zpk(z6, p, k);
G6=tf(H6)
Ka = dcgain(G6)


%% Finding Test Signals
% Display the steady-state errors
step_error= 1 / (1 + Kp);
ramp_error= 1 /  Kv;
parabolic_error= 1 /  Ka ;

% Step Signal Implications
fprintf('Steady-state error for standard step input: %.4f\n', step_error);
% Ramp Signal Implications
fprintf('Steady-state error for ramp input: %.4f\n', ramp_error);
% Parabola Signal
fprintf('Steady-state error for parabolic input: %.4f\n', parabolic_error);
```

Output:

```
>> Example_2
The system is of type 0 .
G4 =

    10 s + 120
  ----------------
  s^2 + 32 s + 252

Continuous-time transfer function.

Kp =
```

```
   0.4762

G5 =

  10 s^2 + 120 s
 ----------------
 s^2 + 32 s + 252

Continuous-time transfer function.

Kv =

     0

G6 =

  10 s^3 + 120 s^2
 ----------------
 s^2 + 32 s + 252

Continuous-time transfer function.

Ka =

     0
Steady-state error for standard step input: 0.6774
Steady-state error for ramp input: Inf
Steady-state error for parabolic input: Inf
```



Example#06: In this example, the students will observe the following underlined action items;

- ✓ **Influence of natural frequency and damping ratio.**
- ✓ **Different static error constants.**

$$\frac{s^3 + 7s^2 + 24s + 24}{(s+1)(s+2)(s+3)(s+4)}$$

```
num13=[1 7 24 24];
den13=poly([-1, -2, -3, -4]);
G13 = tf(num13, den13);
damp(G13);
[wn, zeta, k] = damp(G13)
%% Generic Representation to illustrate the influence the Damping Ratio and Natural frequency!
zeta = 1:5;
k = -4;
omegan = 4;
figure;
% Plot for varying values of zeta
subplot(1, 2, 1)
hold on;
for i = 1:length(zeta)
    b = tf([k * omegan^2], [1 2 * zeta(i) * omegan omegan^2]);
    step(b);
end
hold off;
legend('Zeta=1', 'Zeta=2', 'Zeta=3', 'Zeta=4', 'Zeta=5');
%% Generic Representation to illustrate the influence of Omega!
% Plot for varying values of omegan
subplot(1, 2, 2)
omegan_vals = 1:5;
a = tf(k * omegan_vals.^2, [1 2 * zeta(1) * omegan_vals zeta(1) * omegan_vals.^2]);
hold on;
for i = 2:length(zeta)
    b = tf(k * omegan_vals.^2, [1 2 * zeta(i) * omegan_vals zeta(i) * omegan_vals.^2]);
    step(b);
end
hold off;
legend('Omegan=1', 'Omegan=2', 'Omegan=3', 'Omegan=4', 'Omegan=5');
%% Determine the type of the control system
% Calculate the number of poles at zeroes in the transfer function
poles_at_origin = sum(den13 == 0);
if poles_at_origin == 0
    fprintf('The system is of type 0 .\n');
elseif poles_at_origin == 1
    fprintf('The system is of type 1.\n');
elseif poles_at_origin == 2
    fprintf('The system is of type 2.\n');
else
    fprintf('The system is of type 3 or higher.\n');
end

%% Finding the steady-state errors
%Finding error constants: Kp
G13=tf(num13,den13)
Kp = dcgain(G13)
%Finding error constants: Kv
z14=[0, -1.539, -2.731+2.85i, -2.731-2.85i];
p14=[-1, -2, -3, -4];
k=[1];
H14=zpk(z14, p14, k);
G14=tf(H14)
Kv = dcgain(G14)
%Finding error constants: Ka
z15=[0, 0, -1.539, -2.731+2.85i, -2.731-2.85i];
p15=[-1, -2, -3, -4];
k=[1];
H15=zpk(z15, p15, k);
G15=tf(H15)
Ka = dcgain(G15)
```

```
%% Finding Test Signals
% Display the steady-state errors
step_error= 1 / (1 + Kp);
ramp_error= 1 /  Kv;
parabolic_error= 1 /  Ka ;
% Step Signal Implications
fprintf('Steady-state error for standard step input: %.4f\n', step_error);
% Ramp Signal Implications
fprintf('Steady-state error for ramp input: %.4f\n', ramp_error);
% Parabola Signal
fprintf('Steady-state error for parabolic input: %.4f\n', parabolic_error);
```

```
The system is of type 0 .

G13 =

        s^3 + 7 s^2 + 24 s + 24

  ---------------------------------

  s^4 + 10 s^3 + 35 s^2 + 50 s + 24

Continuous-time transfer function.

Kp =

     1

G14 =

  s^4 + 7.001 s^3 + 23.99 s^2 + 23.98 s

  -------------------------------------

    s^4 + 10 s^3 + 35 s^2 + 50 s + 24

G15 =

  s^5 + 7.001 s^4 + 23.99 s^3 + 23.98 s^2

  ---------------------------------------

     s^4 + 10 s^3 + 35 s^2 + 50 s + 24

Continuous-time transfer function.

Ka =

     0

Steady-state error for standard step input: 0.5000

Steady-state error for ramp input: Inf

Steady-state error for parabolic input: Inf
```

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

Lab#07: Accessing the stability of the system using Routh-Hurwitz Criterion and depicting the shared Root Locus GUI to observe the gain values for different asymptotes/pole-zero conditions.

The fundamental aim of this lab is to explore the stability of dynamic systems under different discrete parametric values of the systems. Different theoretical interpretations of Routh-Hurwitz have been implied in M-file for user-friendly code and afterward, Root Locus GUI has been explored to access the studied rules of Root Locus.

General Developed Code of Routh-Hurwitz for Accessing the stability of the dynamic system(s)

```
to count sign changes in RH table and to find values of roots in right half of plane
%%%Routh Hurwitz Automatically
% Let's make a script file for the generic RH Table
% to check the stability of any dynamic system
%%%%%%INPUT COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL HERE%%%%%
syms K
% Coefficient=[1,6,12,12,5];
Coefficient=[1,-2,5,-4,2,5,-7];
%%%%Create the First Row of the Routh Table
rh = [];
first_row = [];
for i = 1:2:length(Coefficient)
    first_row = [first_row,Coefficient(i)]
end
while length(first_row) <= 1
    first_row = [first_row,0]
end
%disp('First Row')
rh = [rh;first_row]
%%%%Create the second row of the Routh Table
second_row = [];
for i = 2:2:length(Coefficient)
    second_row = [second_row,Coefficient(i)];
end
while length(second_row) < length(first_row)
    second_row = [second_row,0];
end
disp('Second Row')
rh = [rh;second_row]
routh_table_width = length(first_row);
%%%Now create the next rows
numberofrows = length(Coefficient)-2;
%%%Cross Check whether the system is at least
second order
if numberofrows < 0
    disp('Not Possible')
    return
end
for iter = 1:numberofrows
    row = [];
    %disp(['Computing Row ',num2str(loop_row+2)])
    %disp('Divide All Determinants by this var = ')
    divisor = rh(iter+1,1)
    left_side = rh(iter:iter+1,1);
    for col = 1:routh_table_width
        if col == routh_table_width
            right_side = [0;0];
```

```
Second Row

rh =

     1     5     2    -7
    -2    -4     5     0


rh =

    1.0000    5.0000    2.0000   -7.0000
   -2.0000   -4.0000    5.0000         0
    3.0000    4.5000   -7.0000         0


rh =

    1.0000    5.0000    2.0000   -7.0000
   -2.0000   -4.0000    5.0000         0
    3.0000    4.5000   -7.0000         0
   -1.0000    0.3333         0         0
```

```
        else
            right_side = rh(iter:iter+1,col+1);
        end
        %disp('Determinant to be computed')
        both_det = [left_side,right_side]
        value = -det(both_det)/divisor;
        row = [row,value];
    end
   %disp('Next Row of Routh Table')
    rh = [rh;row]
end
%%%Check for stability
%%%Grab the first column
rh
first_column = rh(:,1)
s = sign(first_column(1))
unstable = 0;
P=[];

prev_sign = sign(first_column(1));
sign_changes = 0;

for i = 2:length(first_column)
    value = first_column(i);
    curr_sign = sign(value);
    if curr_sign ~= prev_sign
        sign_changes = sign_changes + 1;
        disp('system unstable');
    end
    prev_sign = curr_sign;
end

disp(['Number of sign changes: ', num2str(sign_changes)]);

%%%%%%Check);
r_roots = [];%%%%%
r=roots(Coefficient)
for i = 1:length(r)
    if real(r(i)) > 0
        r_roots = [r_roots, r(i)];
    end
end
disp('Roots in the right half plane:');
disp(r_roots);
% roots(Coefficient
```

```
rh =

    1.0000    5.0000    2.0000   -7.0000
   -2.0000   -4.0000    5.0000         0
    3.0000    4.5000   -7.0000         0
   -1.0000    0.3333         0         0
    5.5000   -7.0000         0         0
   -0.9394         0         0         0
   -7.0000         0         0         0


first_column =

    1.0000
   -2.0000
    3.0000
   -1.0000
    5.5000
   -0.9394
   -7.0000
```

```
Number of sign changes: 5

r =

   0.1271 + 1.7025i
   0.1271 - 1.7025i
   0.8499 + 1.3397i
   0.8499 - 1.3397i
  -0.9540 + 0.0000i
   1.0000 + 0.0000i

Roots in the right half plane:
   0.1271 + 1.7025i   0.1271 - 1.7025i   0.8499 + 1.3397i   0.8499 - 1.3397i   1.0000 + 0.0000i
```

```
% Case 2 code (Zero Condition in the first row)
%%%Routh Hurwitz Automatically
% Let's make a script file for the generic RH Table
% to check the stability of any dynamic system
clear
clc
close all
check = 2;
%%%%%%INPUT COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL HERE%%%%%
% Coefficient=[1,2,1];
syms K
% Coefficient=[1,5,K-3,K];
Coefficient=[1,6,12,12,5,3];
%Coefficient=[1/1.295 25.15/1.295 0.2034/1.295];
%Coefficient=[1/0.1356 0.4237/0.1356 0.2034/0.1356];


%%%%Create the First Row of the Routh Table
rh = [];
first_row = [];
for i = 1:2:length(Coefficient)
    first_row = [first_row,Coefficient(i)];
end
while length(first_row) <= 1
    first_row = [first_row,0];
end
%disp('First Row')
rh = [rh;first_row];
%%%%Create the second row of the Routh Table
second_row = [];
for i = 2:2:length(Coefficient)
    second_row = [second_row,Coefficient(i)];
end
while length(second_row) < length(first_row)
    second_row = [second_row,0];
end
% disp('Second Row')
rh = [rh;second_row];
routh_table_width = length(first_row);
%%%Now create the next rows
numberofrows = length(Coefficient)-2;
%%%Cross Check whether the system is at least second order
if numberofrows < 0
    disp('Not Possible')
    return
end
for iter = 1:numberofrows
    row = [];
    %disp(['Computing Row ',num2str(loop_row+2)])
    %disp('Divide All Determinants by this var = ')
    divisor = rh(iter+1,1);
    left_side = rh(iter:iter+1,1);
    for col = 1:routh_table_width
        if col == routh_table_width
            right_side = [0;0];
        else
            right_side = rh(iter:iter+1,col+1);
        end
        %disp('Determinant to be computed')
        both_det = [left_side,right_side];
        value = -det(both_det)/divisor;
        row = [row,value];
    end
   %disp('Next Row of Routh Table')
    rh = [rh;row];
end
%%%Check for stability
```

```
%%%Grab the first column
rh
first_column = rh(:,1);


%% Case 2

for i = 1:length(first_column)
    if first_column(i) == 0
        for j = 2:(routh_table_width - i +2)
            if rh(i,j) == 0
                disp('Case 3')
                check = 1;
                break;
            else
                check = 0;
            end
        end
        if check
            break;
        end
    end
end


if check == 0

    Coefficient = fliplr(Coefficient)

    rh = [];
    first_row = [];
            for i = 1:2:length(Coefficient)
                first_row = [first_row,Coefficient(i)];
            end
            while length(first_row) <= 1
                first_row = [first_row,0];
            end
            %disp('First Row')
            rh = [rh;first_row];
            %%%%Create the second row of the Routh Table
            second_row = [];
            for i = 2:2:length(Coefficient)
                second_row = [second_row,Coefficient(i)];
            end
            while length(second_row) < length(first_row)
                second_row = [second_row,0];
            end
            % disp('Second Row')
            rh = [rh;second_row];
            routh_table_width = length(first_row);
            %%%Now create the next rows
            numberofrows = length(Coefficient)-2;
            %%%Cross Check whether the system is at least second order
            if numberofrows < 0
                disp('Not Possible')
                return
            end
            for iter = 1:numberofrows
                row = [];
                %disp(['Computing Row ',num2str(loop_row+2)])
                %disp('Divide All Determinants by this var = ')
```

```
                divisor = rh(iter+1,1);
                left_side = rh(iter:iter+1,1);
                for col = 1:routh_table_width
                    if col == routh_table_width
                        right_side = [0;0];
                    else
                        right_side =
rh(iter:iter+1,col+1);
                    end
                    %disp('Determinant to be computed')
                    both_det = [left_side,right_side];
                    value = -det(both_det)/divisor;
                    row = [row,value];
                end
               %disp('Next Row of Routh Table')
                rh = [rh;row];
            end
            %%%Check for stability
            %%%Grab the first column
            rh
 end
%%
first_column = rh(:,1);

s = sign(first_column(1));
unstable = 0;
for i = 2:length(first_column)
    value = first_column(i);
    if s ~= sign(value)
        disp('Unstable ')
        unstable = 1;
        break
    end
end
if ~unstable
    disp('System is Stable')
end
```

```
rh =

    0.7722    0.1571
   19.4208         0
    0.1571         0

System is Stable
>> CASE_1

rh =

    0.7722   19.4208
         0    0.1571
      -Inf       NaN
       NaN       NaN


Coefficient =

    0.1571   19.4208         0    0.7722


rh =

    0.1571         0
   19.4208    0.7722
```

%Case-3 + determination of range of an unknown parameter (k)

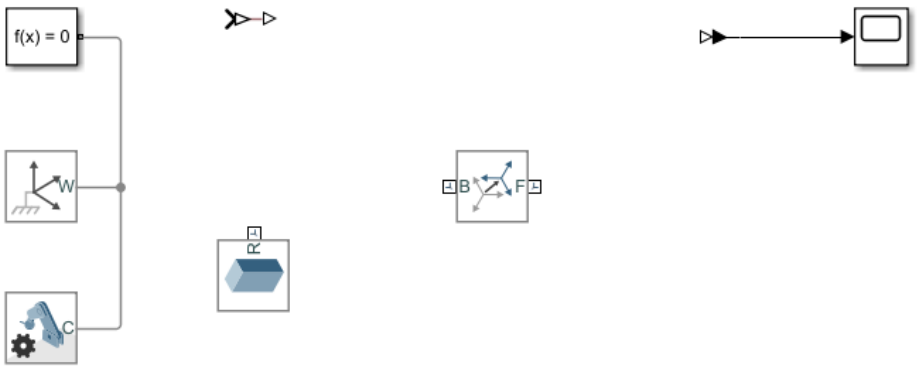
```
%%%Routh Hurwitz Automatically
% Let's make a script file for the generic RH Table
% to check the stability of any dynamic system
clear
clc
close all
%%%%%%INPUT COEFFICIENTS OF CHARACTERISTIC POLYNOMIAL HERE%%%%%
%Coefficient=[1,2,1];
syms K
%Coefficient=[1,2,24,48,-25,-50]; unstable
Coefficient=[1,5,K+3,K,3,0,0,1,2];
% Coefficient=[1,0,5,0,0,6,12,12,5];
%%%%Create the First Row of the Routh Table
rh = [];
first_row = [];
for i = 1:2:length(Coefficient)
    first_row = [first_row,Coefficient(i)]
end
while length(first_row) <= 1
    first_row = [first_row,0]
end
%disp('First Row')
rh = [rh;first_row]
%%%%Create the second row of the Routh Table
second_row = [];
for i = 2:2:length(Coefficient)
    second_row = [second_row,Coefficient(i)];
end
while length(second_row) < length(first_row)
    second_row = [second_row,0];
end
disp('Second Row')
rh = [rh;second_row]
routh_table_width = length(first_row);
%%%Now create the next rows
numberofrows = length(Coefficient)-2;
%%%Cross Check whether the system is at least second order
if numberofrows < 0
    disp('Not Possible')
    return
end


%% ********************
%%%Check for stability
%%%Grab the first column
rh
first_column = rh(:,1)
s = sign(first_column(1))
unstable = 0;
for i = 2:length(first_column)
    value = first_column(i);
    if s ~= sign(value)
        disp('Unstable ')
        unstable = 1;
        break
    end
end
if ~unstable
    disp('System is Stable')
end
%%%%%%Check%%%%%
%%% finding range of k
unstable=0;
for i=1:length(first_column)
    if symvar(first_column(i))==K
        answer=solve(first_column(i),K);
        fprintf(sprintf("System will be marginally stable at %.4f value of K",answer))
        if(subs(first_column(i),K,answer-1)<0)
            unstable=1;
            fprintf(sprintf("\nSystem will be unstable at value of K < %.4f",answer))
        elseif(subs(first_column(i),K,answer-1)>0)
            fprintf(sprintf("\nSystem will be stable at value of K < %.4f\n",answer))
        end
        if(unstable)
        fprintf(sprintf("\nSystem will be stable at value of K > %.4f\n",answer))
```

```
        end
        break;
    end
end
```

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

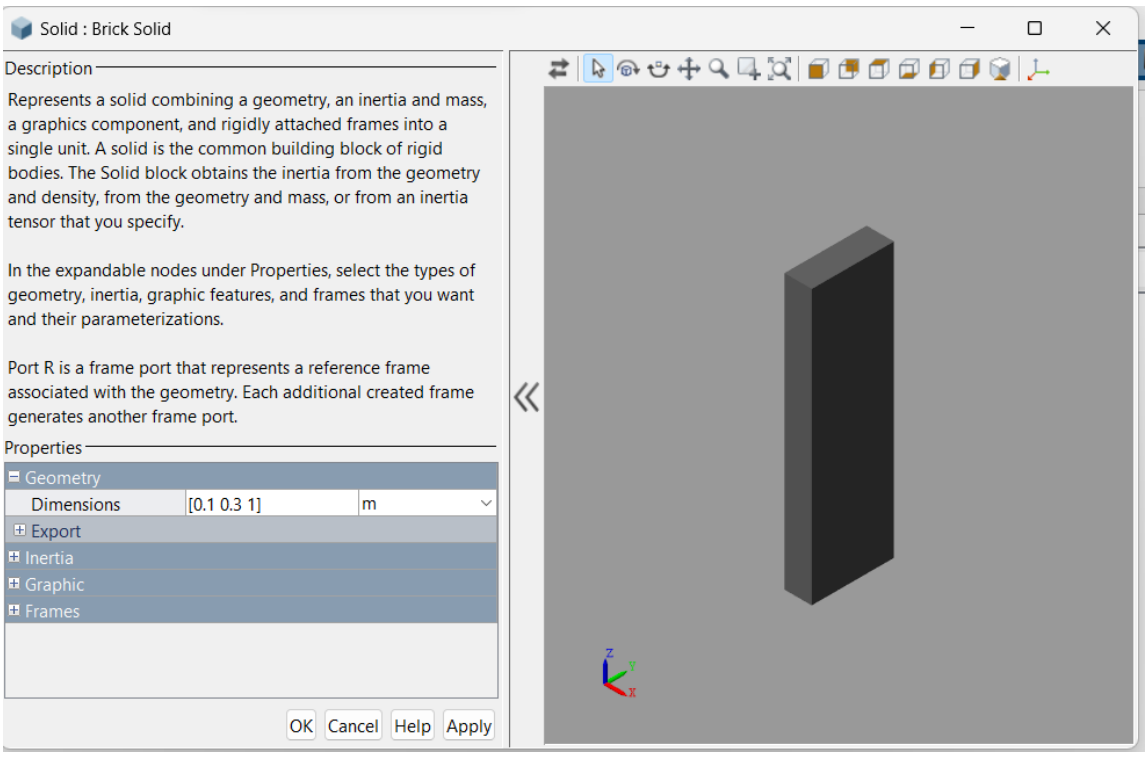
Lab#08: Introduction to MATLAB Toolbox MULTIBODY and its utilization for modeling, analysis, and cross-validation of dynamic systems: Part 1.

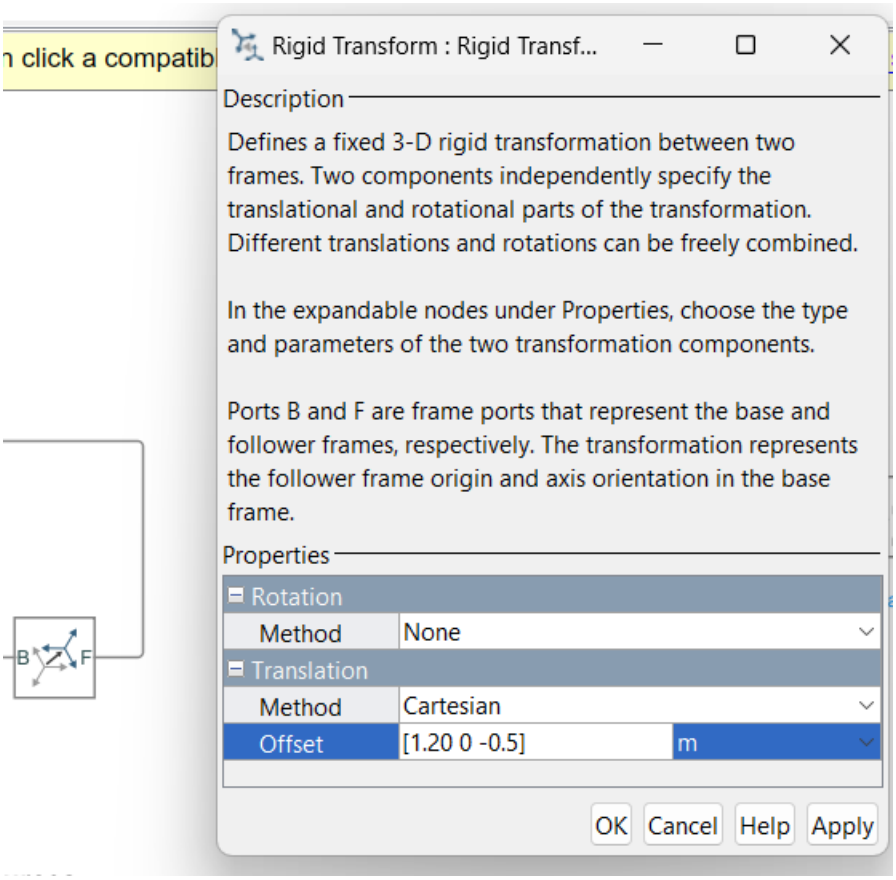
The fundamental aim of this lab is to explore MULTIBODY environment for modeling, analyzing, visualizing thd dynamics, and cross-validating the dynamic responses of translational mechanical systems under the various controllable parameters.



Simscape Multibody Resources

1. Find more multibody components in the Simscape Multibody library.
   For more information, see Simscape Multibody - Blocks.
2. Find components from other domains in the Simscape library.
3. Connect the components to form a physical network.
   For more information, see Essential Steps for Constructing a Physical Model and Creating a Multibody Model.
4. Visualize the simulation using Mechanics Explorer
5. Explore simulation results using sscexplore





In this example, simple mass-spring damper system will be visualized in the MULTIBODY environment for accessing and cross-validating the dynamic response of the system.

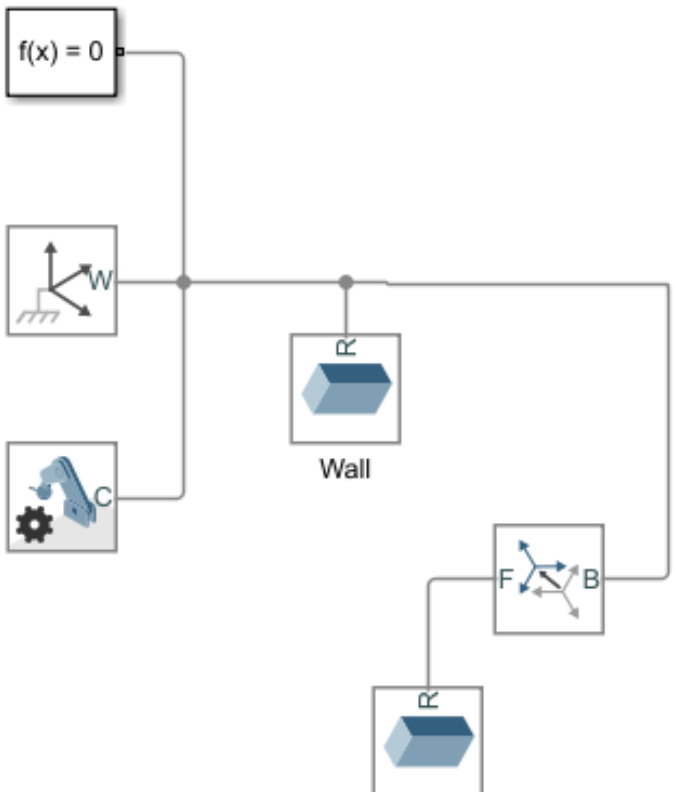
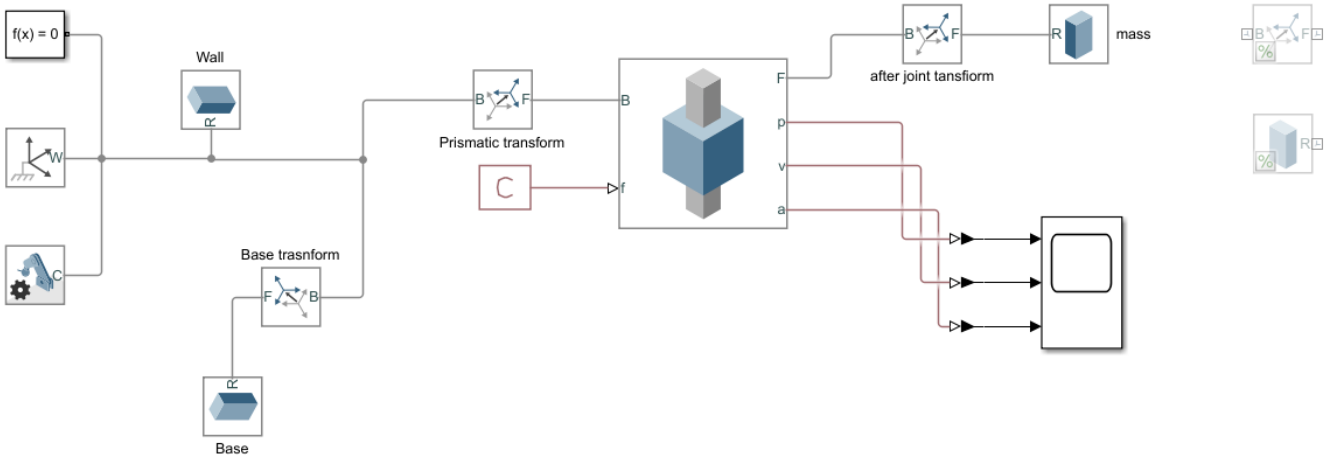
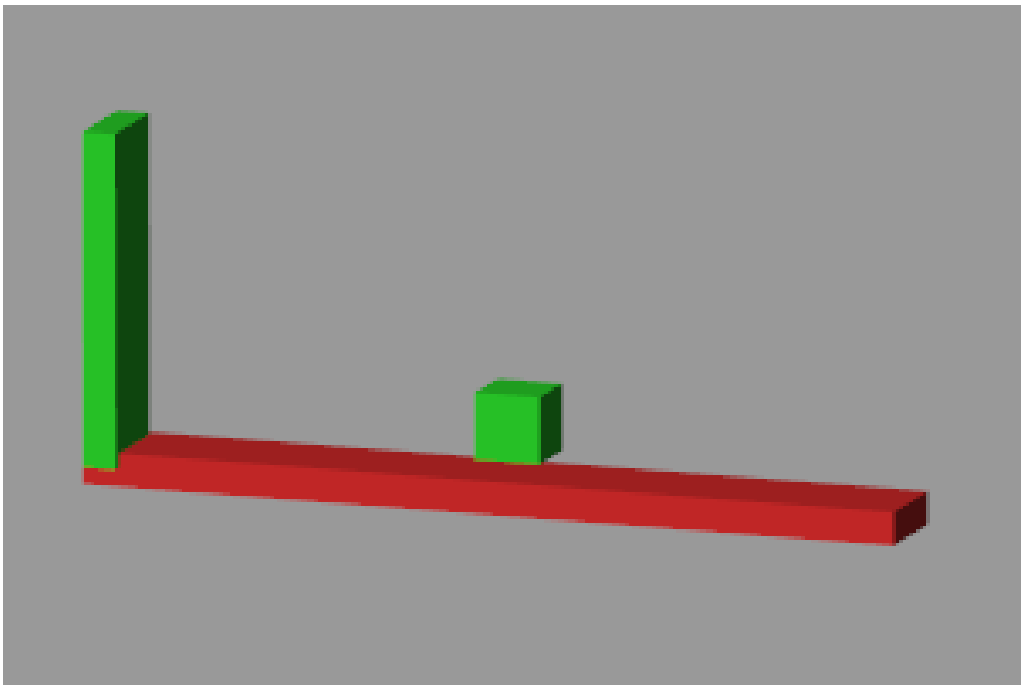


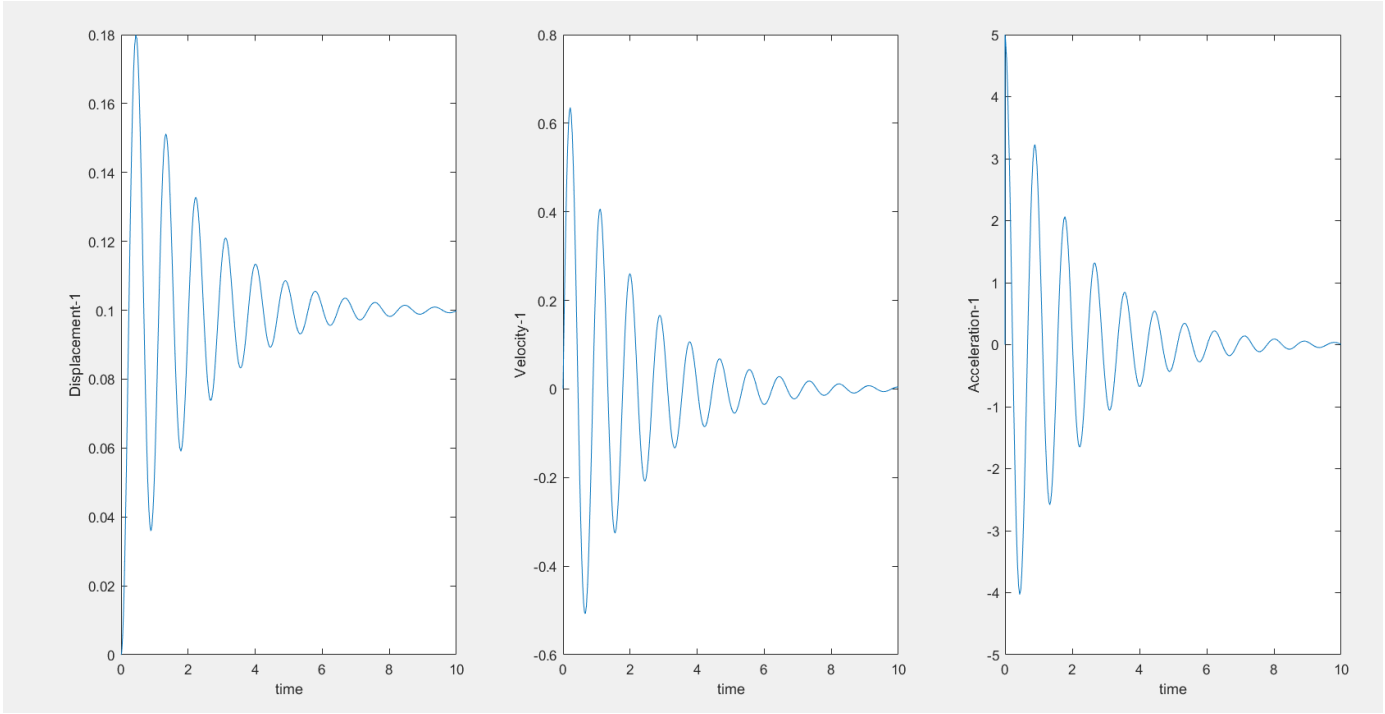
Figure shows the creation of wall and base for single mass sping damper system.





```
TR = [0 10];
X0 = [0;0];
[t,y]=ode45(@sim,TR,X0);
x=y(:,1);
v=y(:,2);
a1=diff(v)./diff(t);
subplot(1,3,1)
plot(t,x)
xlabel('time')
ylabel('Displacement-1')
subplot(1,3,2)
plot(t,v)
xlabel('time')
ylabel('Velocity-1')
subplot(1,3,3)
plot(t,[0;a1])
xlabel('time')
ylabel('Acceleration-1')

m=2;b=2;k=100;f=10;
num=[1];
den=[m b k];
tf(num,den)
```

```
function dy= sim(t,y)
   m=2;b=2;k=100;f=10;
   dy(1)=y(2);
   dy(2)=1/m*(f-b*y(2)-k*y(1))
   dy=dy';
end
```



All the examples from translational mechanical systems are explored by considering MULTIBODY environment. The cross-validation of

Lab#09: Implying the concept of PID Controller for tracking the response of dynamic systems: Via M-File and SIMSCAPE Approach.

In this lab, initially, the output response of a certain system is analyzed by applying the unit step input. It is observed that the system does not reach the target value and consequently generates the steadystate error. Therefore, a typical controller known as PID (Proportional Integral Derivate) is implemented with the different orders of the system to minimize the steady-state error. Furthermore, the response of the system is also analyzed by varying parametric values of proportional, integral, and derivate components of the controller.

Effect of Proportional (K_p) on the Response

Figure 1 shows the step response of a second-order system in which system M represents the response without implying the Proportional Controller. It can be concluded that by increasing the gain of the proportional term (K_p), the response tries to achieve the steady-state value. Therefore, by including the K_p, steady-state error and rise time will be decreased. However, at the same time, it increases the percentage overshoot which in turn increases the oscillation. It must be noted that this graph with steady-state output is achieved at K_p=24.

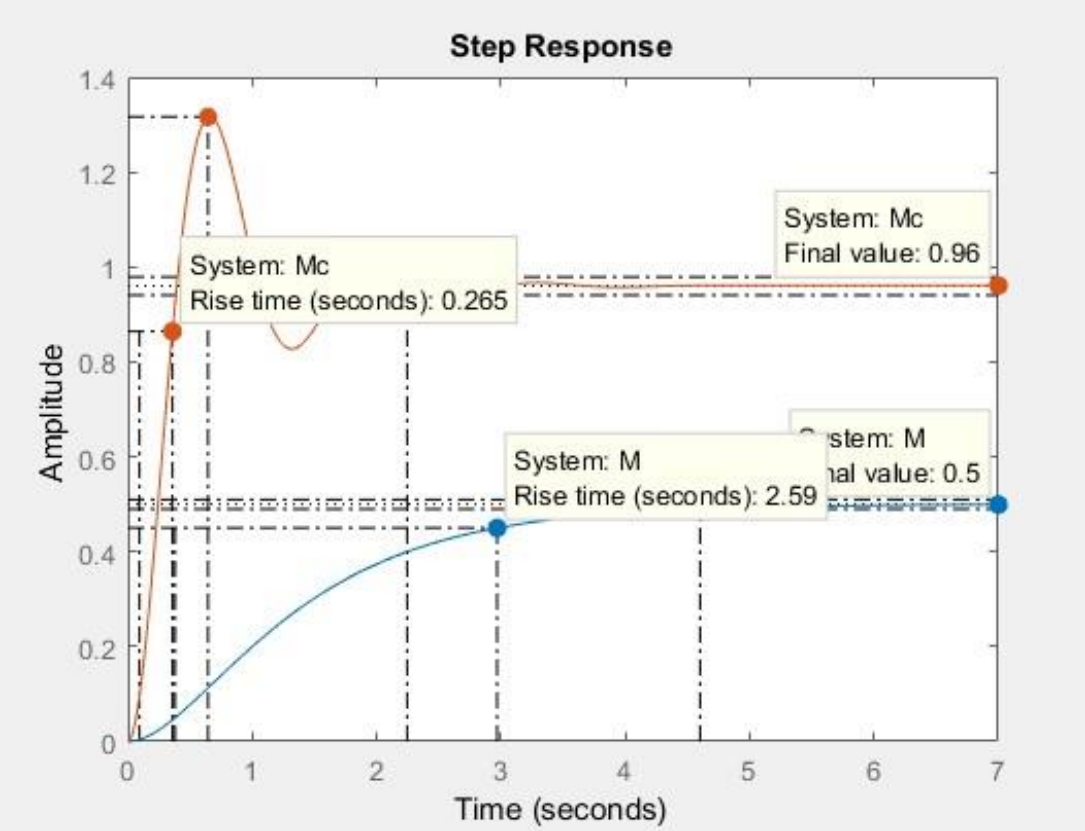


Figure 1: Response of System with K_p

Effect of Derivative (Kd) on the output response

In this section, the response of the system will be presented by changing the values of the derivative portion (K_d)of the PID controller. **Figure 2** reveals that by increasing the values of K_d the oscillations could be reduced to an acceptable level. This response of the system is achieved with K_d =8.

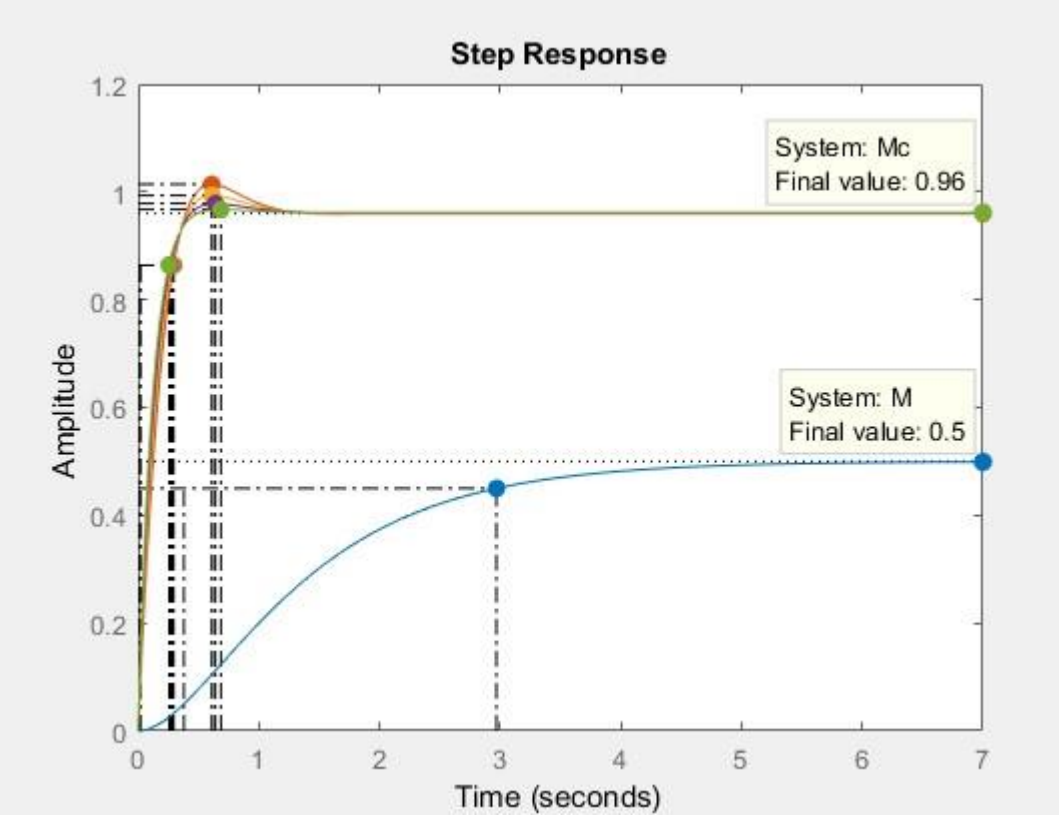


Figure 1: Response of System with K_d

Effect of Integral (KI) on the output response

From the above two graphs, it is observed that the system is still a little away from the target value. It means that steady-state error cannot be reduced by changing the values of K_d. Therefore, in this section, the response is analyzed with different values of K_I. **Figure 3** shows that the steady-state error can be minimized by adjusting the integral portion of the PID controller. In this system, it is required to obtain a steady state value of 1 and at the value of K_I =2, there will not be any steadystate error.

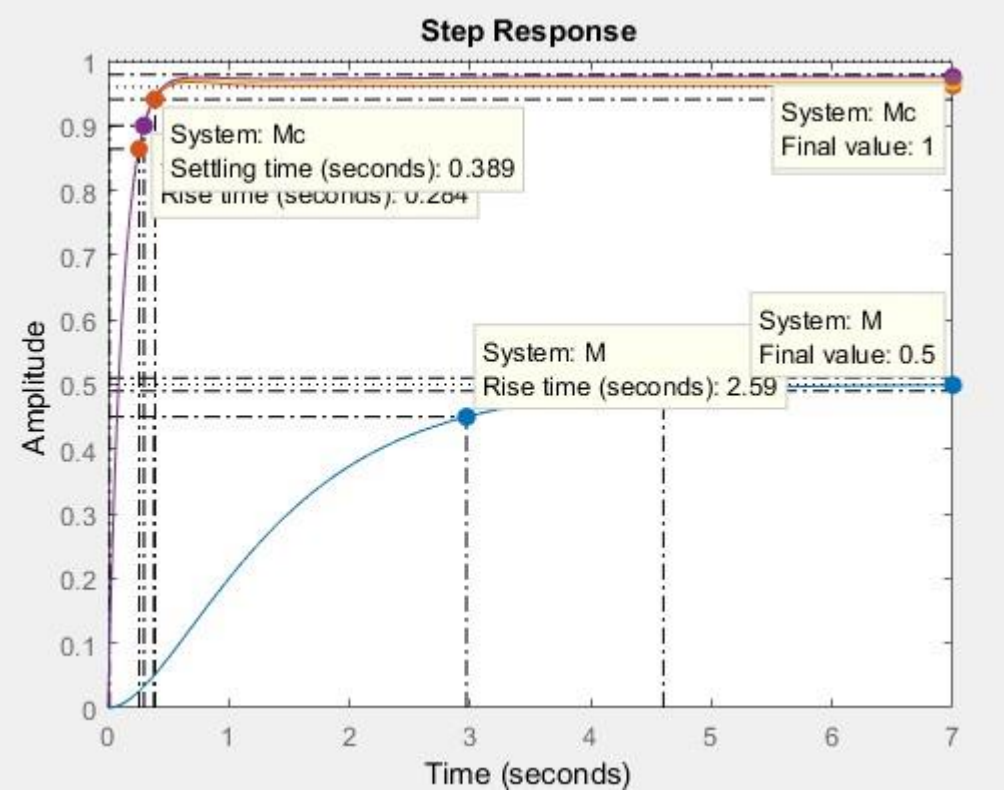


Figure 3: Response of System with K_I

Concluding Remarks

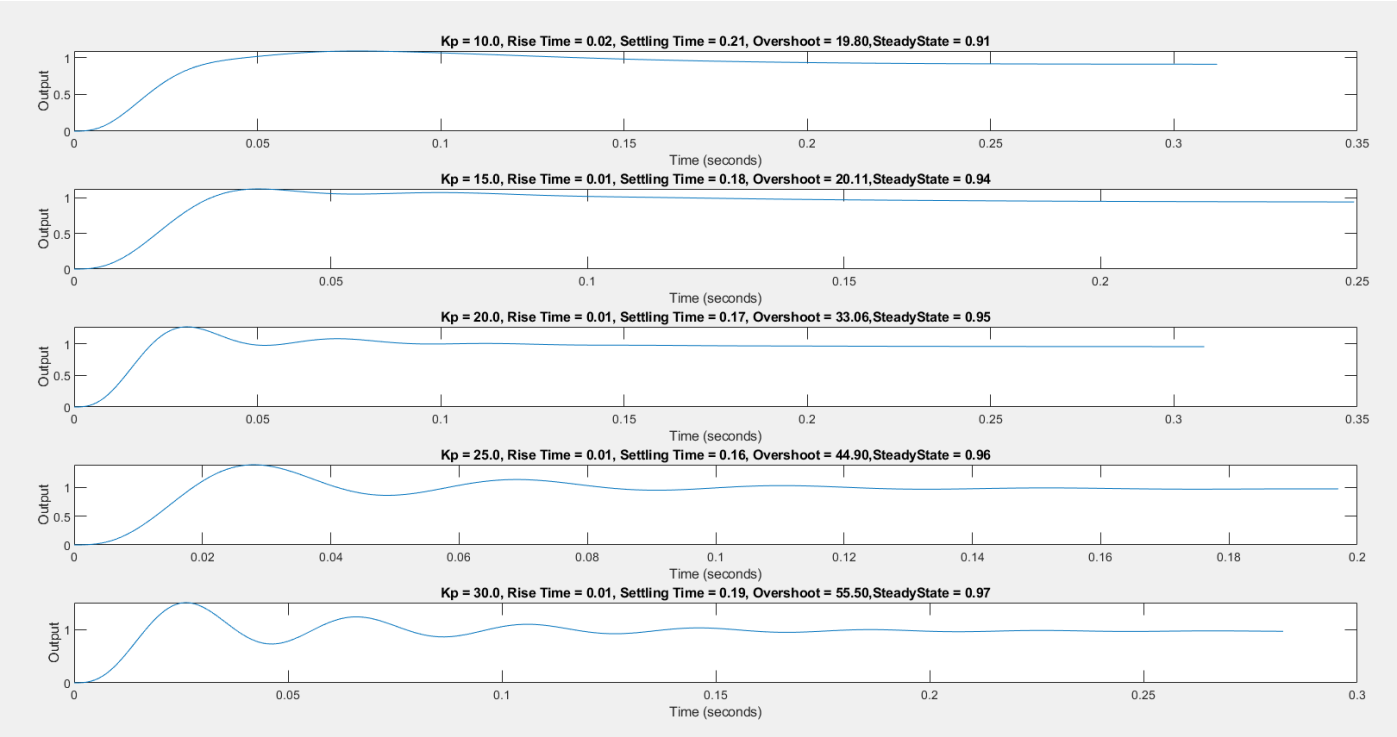
A PID controller calculates an error value as the difference between a measured output value and the desired setpoint. The controller attempts to minimize the error by adjusting the process through the use of individual components.

Table 1: Effect of PID Controller

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Minor Change	Decrease	Decrease	No Effect

For P controller that is KP

```
%% Implication of P-Controller to access the peformance parameters.
%increasing kp was giving more overshootand rise in Tr but it approacched
%Steady state value
kw=600000;  % Wheel Stiffness
m1=20;   % Unsprung mass
b=3600;   % Suspension damping and damping of tyre has been ignored
ks=48000;  % Suspension Stiffnes
m2=900;   % Sprung mass
A = [0 1 0 0;
    -(ks+kw)./m1 -b/m1 ks/m1 b/m1;
    0 0 0 1;
    ks/m2 b/m2 -ks/m2 -b/m2] % State space matrix A
B= [0; kw/m1; 0; 0]          % Input vector
C= [0 0 1 0] % Output matrix
D= 0;
[n,d]=ss2tf(A,B,C,D);
G=tf(n,d)
% Define the different values of Kp to test
Kp_values = [10, 15, 20, 25, 30];
% Loop over each value of Kp and simulate the response
for i = 1:length(Kp_values)
    Kp = Kp_values(i);
    K = pid(Kp, 0, 0);
    sys = feedback(K*G, 1);
    [y, t] = step(sys);
    % Calculate the rise time, settling time, and overshoot
    characteristics = stepinfo(sys);
    rise_time = characteristics.RiseTime;
    settling_time = characteristics.SettlingTime;
    overshoot = characteristics.Overshoot;
   [A, B, C, D] = ssdata(sys);
    ss_sys = ss(A, B, C, D);
    Steadystate = dcgain(ss_sys);
    Peaktime = characteristics.PeakTime;
    % Plot the step response and annotate the plot with the rise time,
    % settling time, and overshoot
    subplot(length(Kp_values), 1, i);
    plot(t, y);
    title(sprintf('Kp = %0.1f, Rise Time = %0.2f, Settling Time = %0.2f,
Overshoot = %0.2f,SteadyState = %0.2f%',...
    Kp, rise_time, settling_time, overshoot,Steadystate));
    xlabel('Time (seconds)');
    ylabel('Output');
    %xlim([0, 15]);
end
```

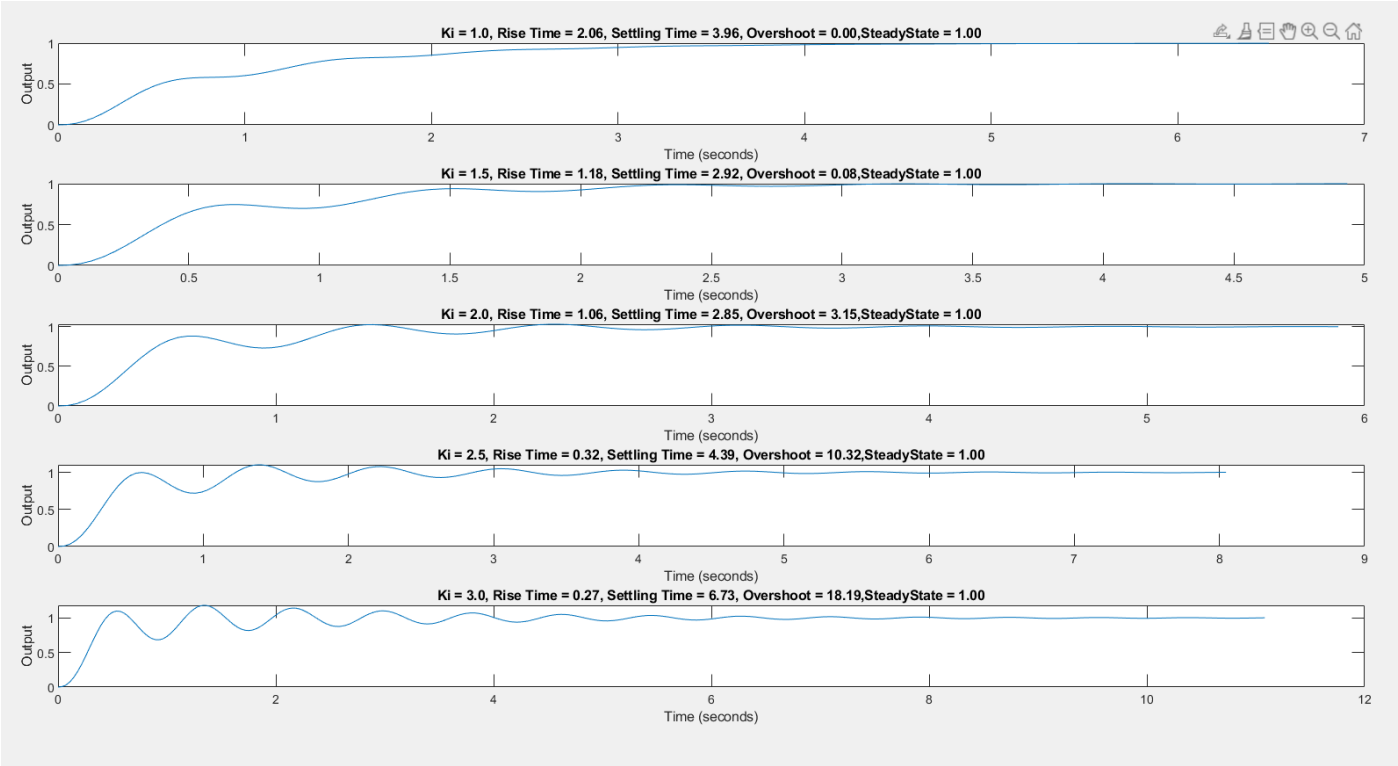


The rise time seems to be decreasing with KP. Overshoot increasing and the settling time decreases to a certain value, but starts increasing beyond that stable value.

For I controller THAT IS ki

Code:

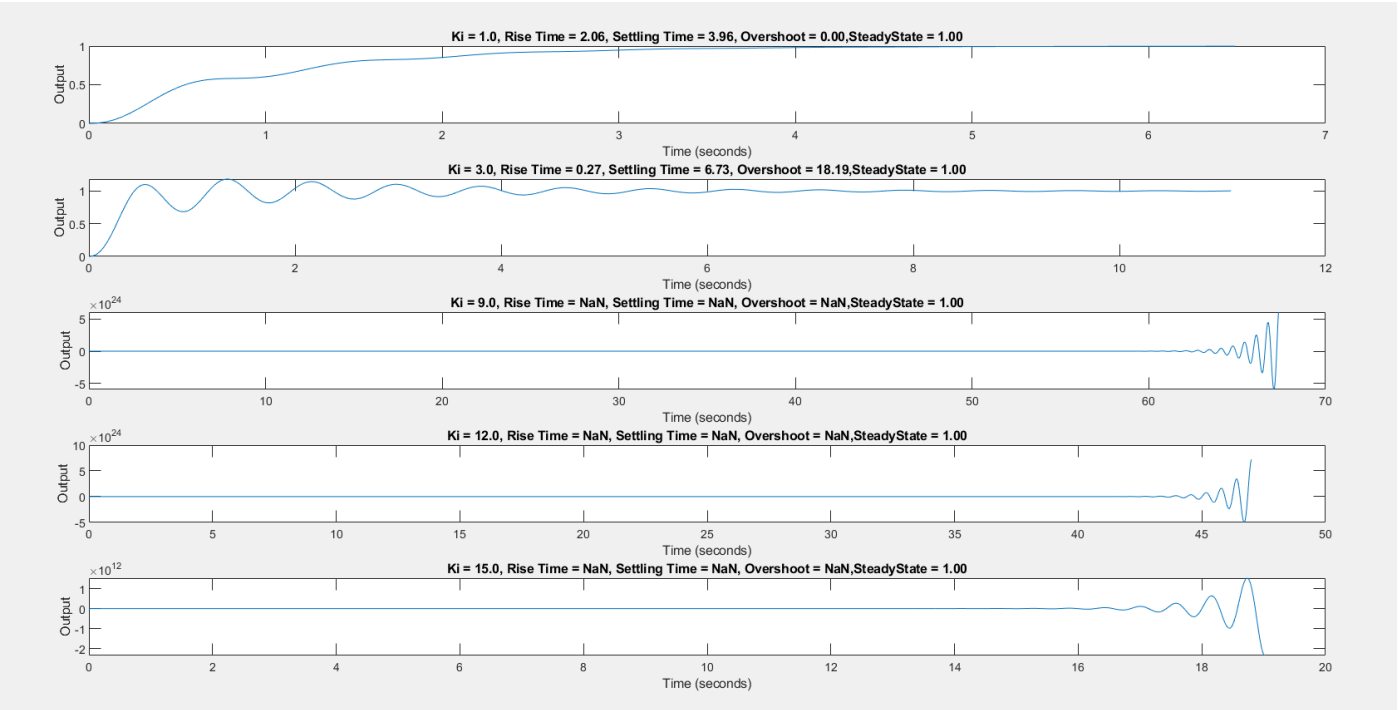
```
%% %% Implication of I-Controller to access the peformance parameters.
KI_values = [1, 1.5, 2, 2.5, 3];
% Loop over each value of Kp and simulate the response
for i = 1:length(KI_values)
    KI = KI_values(i);
    K = pid(0, KI, 0);
    sys = feedback(K*G, 1);
    [y, t] = step(sys);
    % Calculate the rise time, settling time, and overshoot
    characteristics = stepinfo(sys);
    rise_time = characteristics.RiseTime;
    settling_time = characteristics.SettlingTime;
    overshoot = characteristics.Overshoot;
   [A, B, C, D] = ssdata(sys);
    ss_sys = ss(A, B, C, D);
    Steadystate = dcgain(ss_sys);
    Peaktime = characteristics.PeakTime;
    % Plot the step response and annotate the plot with the rise time,
    % settling time, and overshoot
    subplot(length(KI_values), 1, i);
    plot(t, y);
    title(sprintf('Ki = %0.1f, Rise Time = %0.2f, Settling Time = %0.2f, Overshoot =
%0.2f,SteadyState = %0.2f%',...
    KI, rise_time, settling_time, overshoot,Steadystate));
    xlabel('Time (seconds)');
    ylabel('Output');
    %xlim([0, 15]);
end
```



Overshoot seems to be increasing with larger KIs, System is oscillating and settling time is increasing too. This suggests smaller values for ki in order to achieve stability.

CODE: (change values of Ki)

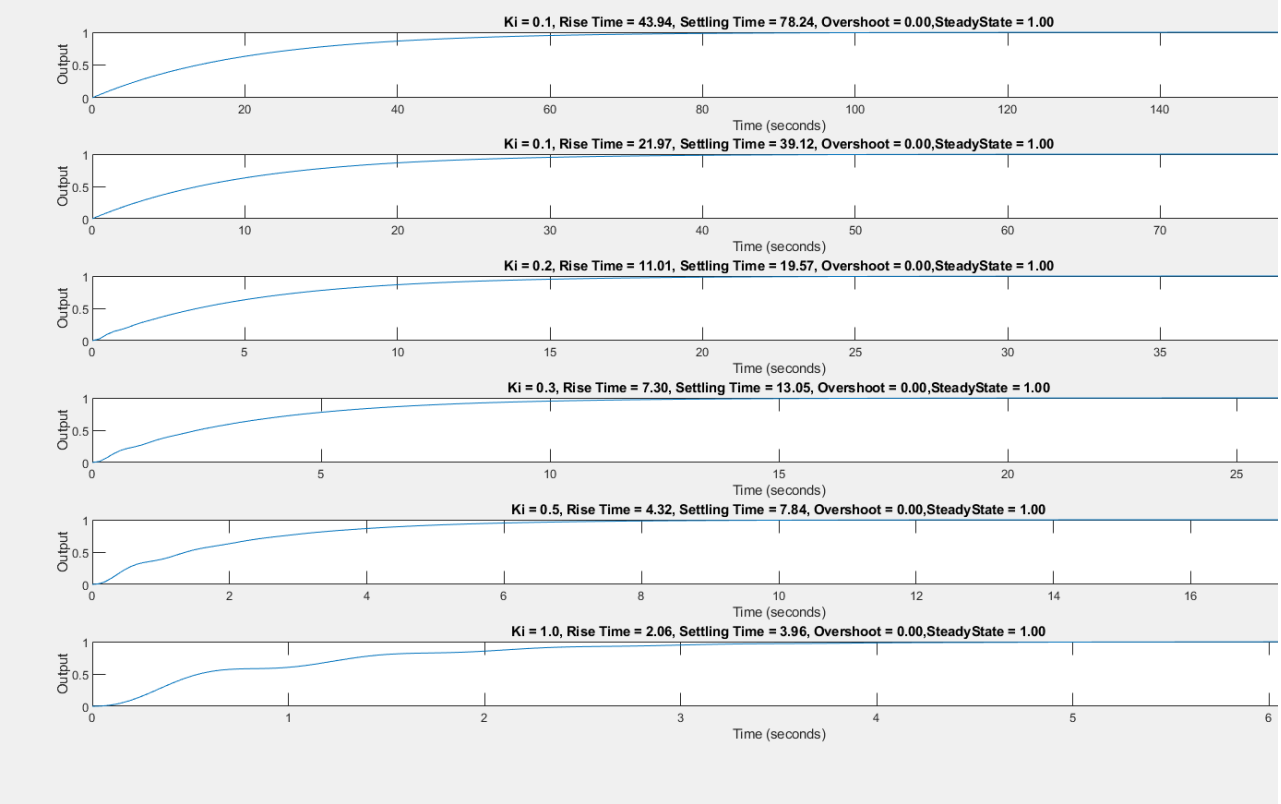
```
KI_values = [1, 3, 9, 12, 15];
```



Too large of the ki tends to render the system unstable with exponentially increasing graph after some time.

For even smaller values:

```
KI_values = [0.05,0.1,0.2,0.3,0.5,1]
```



System tends to remain stable and slowly achieve steady state(note the settling time). This kind of behavior is much preferable as syste does not steer towards uncontrolled oscilaations rather gradually attains steady state with minimum overshoot.

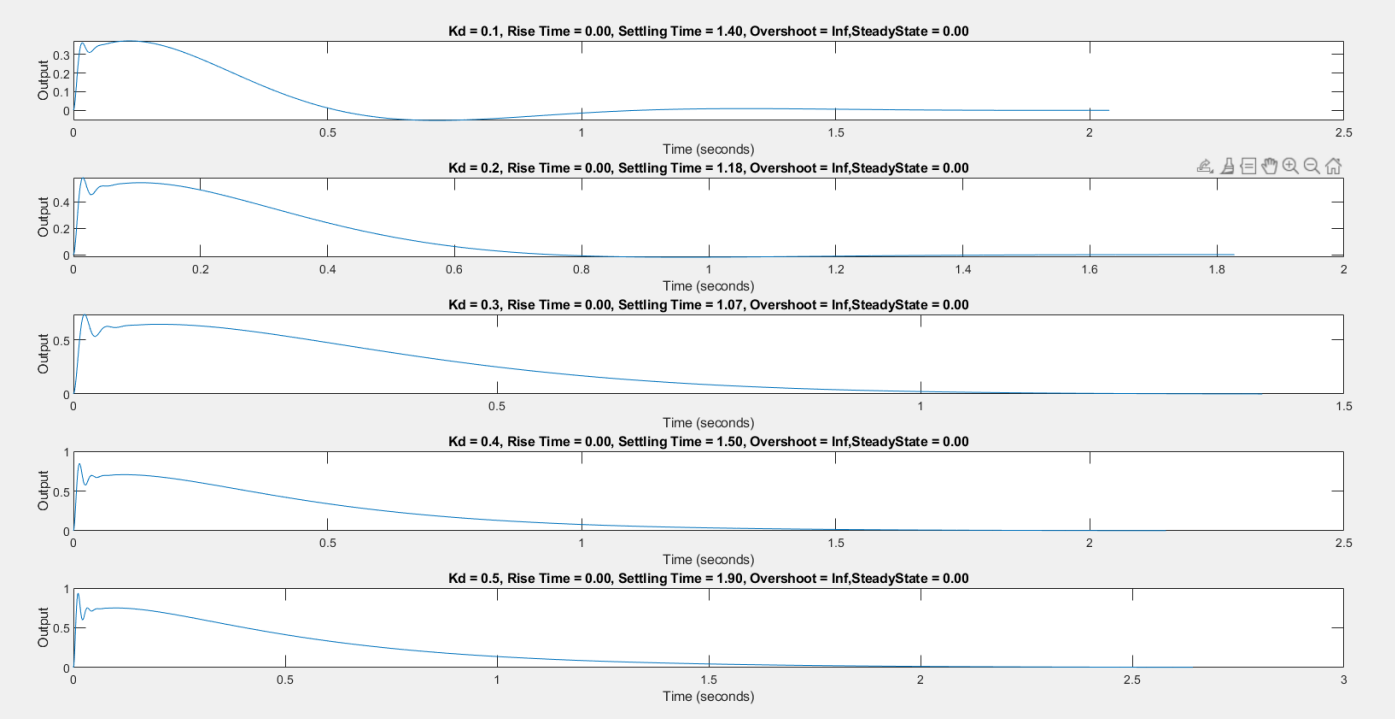
Consequently the KI values ought to be small for ideal controller gain.

For KD controller:

Code:

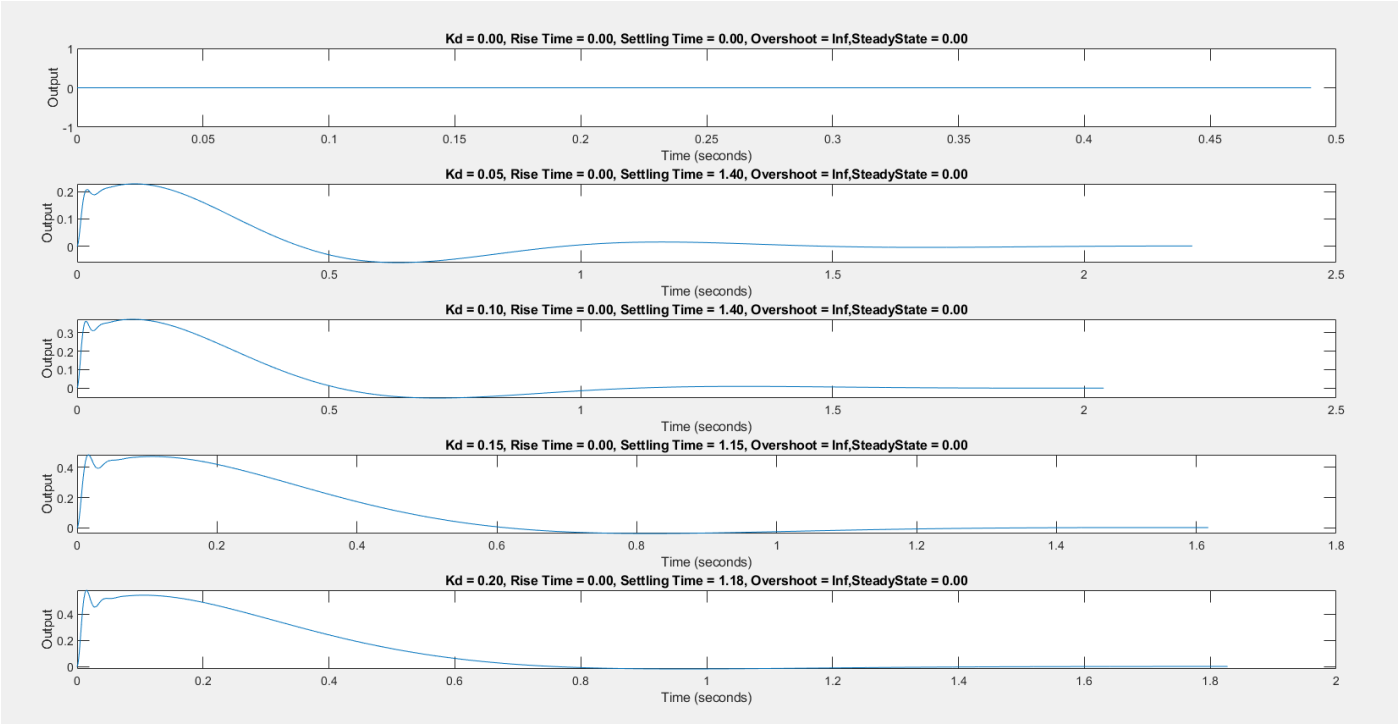
```
%% Implication of D-Controller on the transfer function
Kd_values = [0.1,0.2,0.3,0.4,0.5];
% Loop over each value of Kp and simulate the response
for i = 1:length(Kd_values)
    Kd = Kd_values(i);
    K = pid(0,0,Kd);
    sys = feedback(K*G, 1);
    [y, t] = step(sys);
    % Calculate the rise time, settling time, and overshoot
    characteristics = stepinfo(sys);
    rise_time = characteristics.RiseTime;
```

```
    settling_time = characteristics.SettlingTime;
    overshoot = characteristics.Overshoot;
   [A, B, C, D] = ssdata(sys);
    ss_sys = ss(A, B, C, D);
    Steadystate = dcgain(ss_sys);
    Peaktime = characteristics.PeakTime;
    subplot(length(Kd_values), 1, i);
    plot(t, y);
    title(sprintf('Kd = %0.1f, Rise Time = %0.2f, Settling Time =
%0.2f, Overshoot = %0.2f,SteadyState = %0.2f%',...
    Kd, rise_time, settling_time, overshoot,Steadystate));
    xlabel('Time (seconds)');
    ylabel('Output');
    %xlim([0, 15]);
end
```



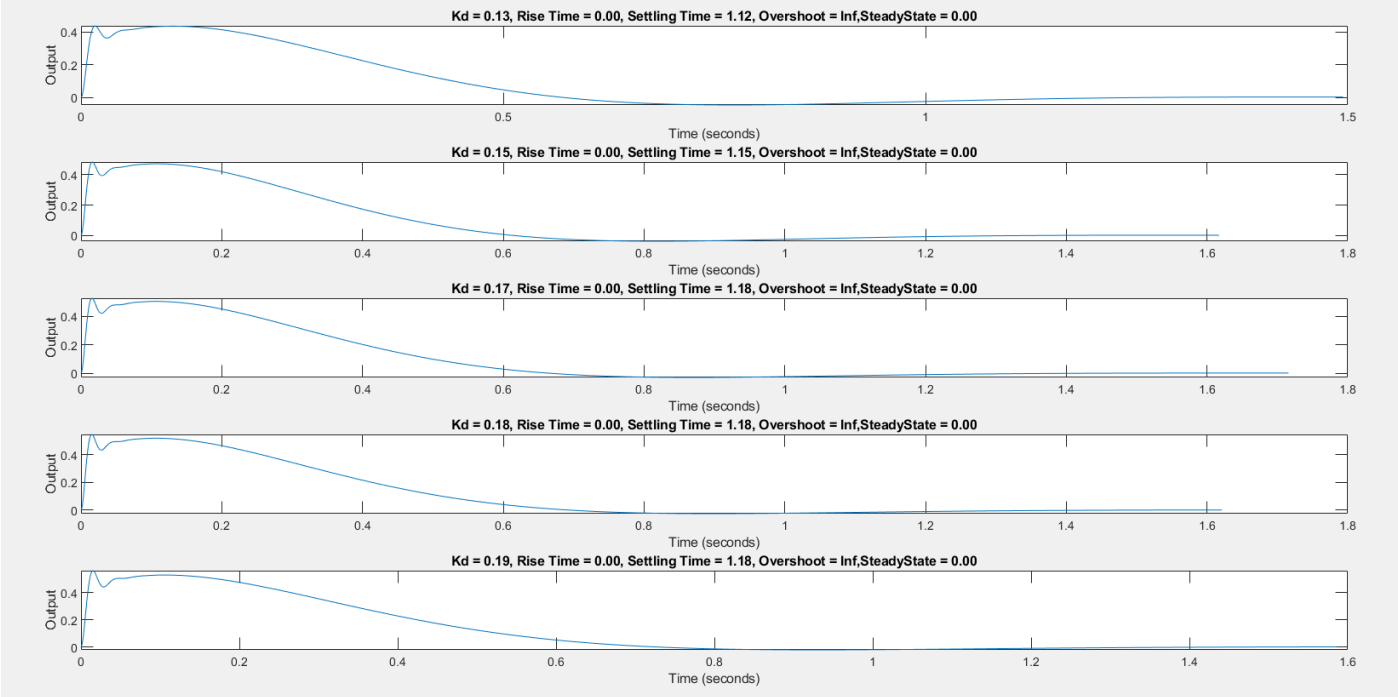
Overshoot is increasing with larger Kd values and settling time too.

```
Kd_values = [0,0.05,0.1,0.15,0.2];
```



Notice how settling time was decreasing upto kd=0.15 but started increasing thereof. This indicates that stable kd value is around this value

```
Kd_values = [0.13,0.15,0.17,0.18,0.19];
```



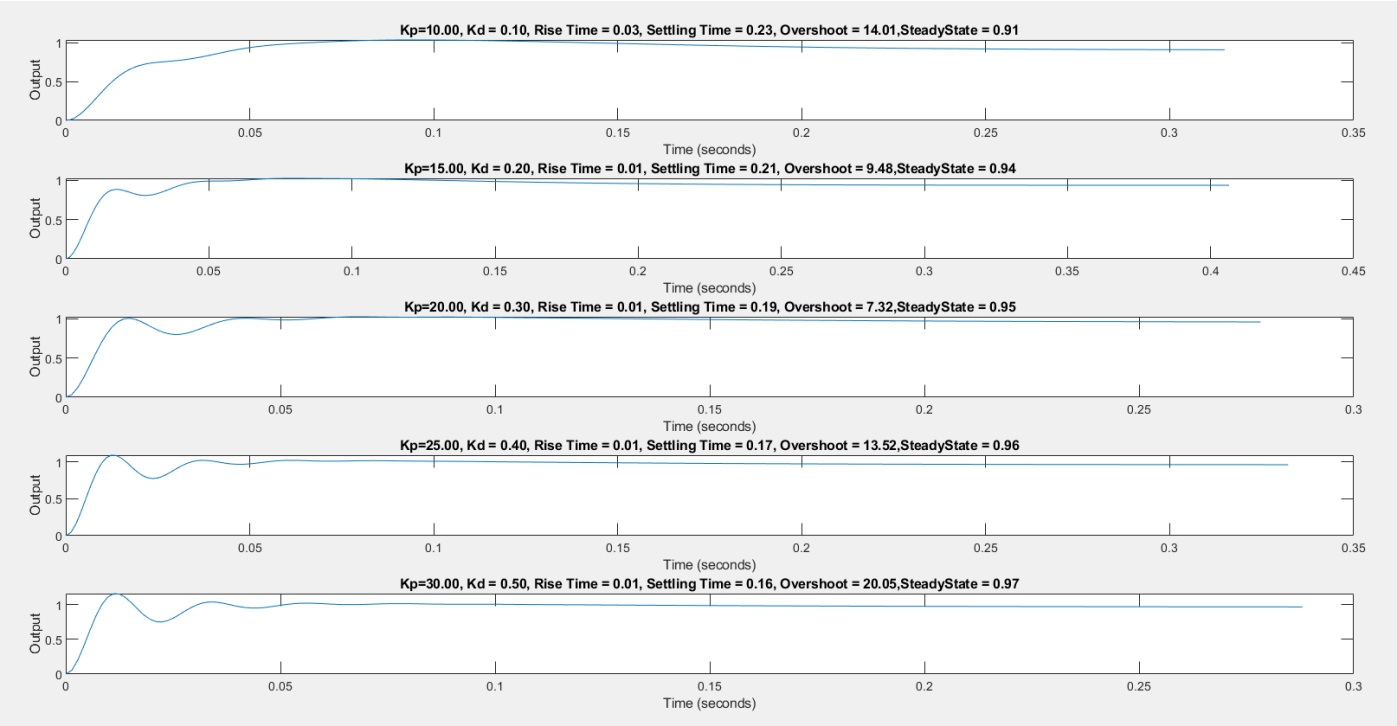
This suggests a stable value range for Kd in case of D-controller.

FOR PD controller:

Code:

Add kp kd to the pid() function and edit the graph displays accordingly

```
%% Implication of D-Controller on the transfer function
Kd_values = [0.1,0.2,0.3,0.4,0.5];
%Kd_values = [0.13,0.15,0.17,0.18,0.19];
% Loop over each value of Kp and simulate the response
Kp_values = [10, 15, 20, 25, 30];

for i = 1:length(Kd_values)
    Kd = Kd_values(i);
    Kp = Kp_values(i);
    K = pid(Kp,0,Kd);
    sys = feedback(K*G, 1);
    [y, t] = step(sys);
    % Calculate the rise time, settling time, and overshoot
    characteristics = stepinfo(sys);
    rise_time = characteristics.RiseTime;
    settling_time = characteristics.SettlingTime;
    overshoot = characteristics.Overshoot;
   [A, B, C, D] = ssdata(sys);
    ss_sys = ss(A, B, C, D);
    Steadystate = dcgain(ss_sys);
    Peaktime = characteristics.PeakTime;
    subplot(length(Kd_values), 1, i);
    plot(t, y);
    title(sprintf('Kp=%0.2f, Kd = %0.2f, Rise Time = %0.2f,
Settling Time = %0.2f, Overshoot = %0.2f,SteadyState =
%0.2f%',...
    Kp,Kd, rise_time, settling_time, overshoot,Steadystate));
    xlabel('Time (seconds)');
    ylabel('Output');
    %xlim([0, 15]);
End
```



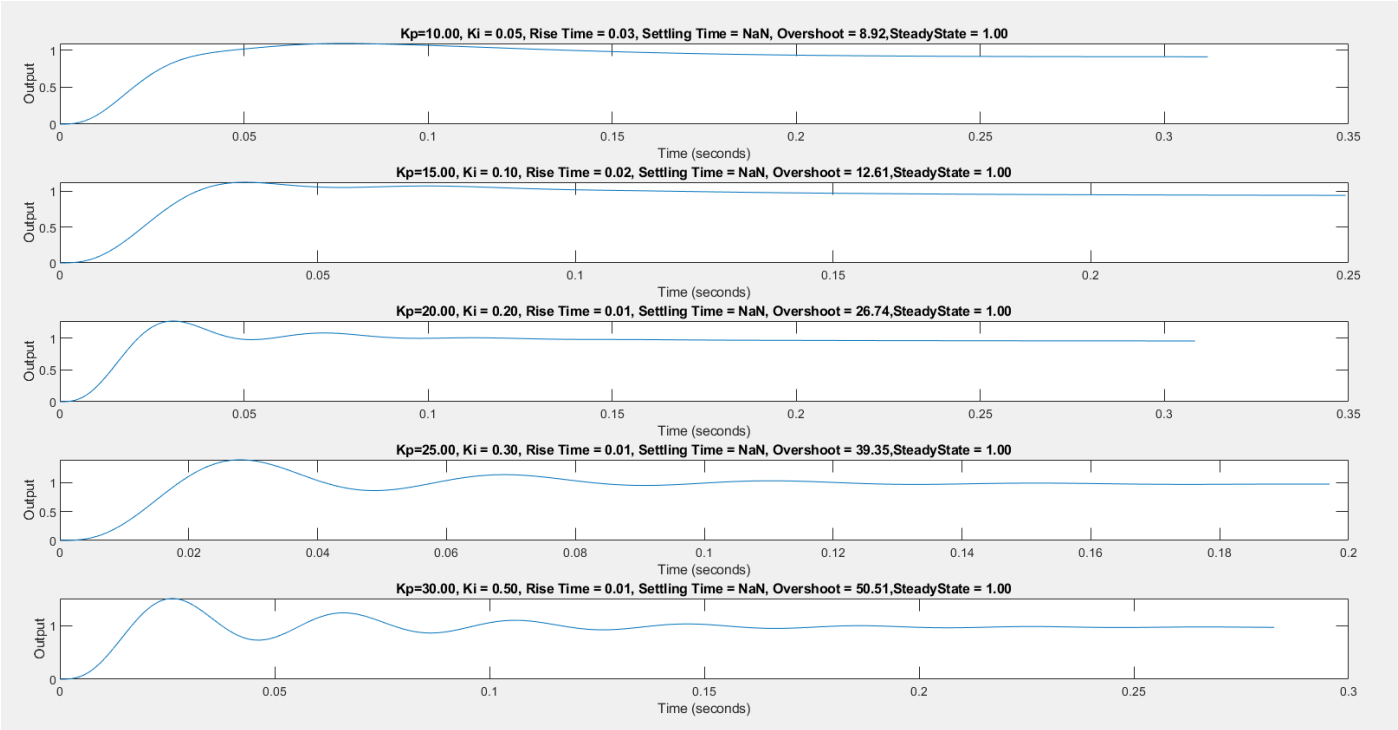
Notice the time characteristics. Kp=20,Kd=0.3 gives least overshoot. Rise time seems to have been saturated. Where as the overshoot goes on increasing beyond Kp=20. This suggests a stable value range for Kp and ki

PI Controller:

Code:

```
%% Implication of PI-Controller on the transfer function

Kp_values = [10, 15, 20, 25, 30];
Ki_values = [0.05,0.1,0.2,0.3,0.5,1]

for i = 1:length(Kd_values)
    Kp = Kp_values(i);
    Ki = Ki_values(i);
    K = pid(Kp,Ki,0);
    sys = feedback(K*G, 1);
    [y, t] = step(sys);
    % Calculate the rise time, settling time, and overshoot
    characteristics = stepinfo(sys);
    rise_time = characteristics.RiseTime;
    settling_time = characteristics.SettlingTime;
    overshoot = characteristics.Overshoot;
   [A, B, C, D] = ssdata(sys);
    ss_sys = ss(A, B, C, D);
    Steadystate = dcgain(ss_sys);
    Peaktime = characteristics.PeakTime;
    subplot(length(Kd_values), 1, i);
    plot(t, y);
```

```
    title(sprintf('Kp=%0.2f, Ki = %0.2f, Rise Time = %0.2f,
Settling Time = %0.2f, Overshoot = %0.2f,SteadyState =
%0.2f%',...
    Kp,Ki, rise_time, settling_time, overshoot,Steadystate));
    xlabel('Time (seconds)');
    ylabel('Output');
    %xlim([0, 15]);
End
```



For PI controller the increasing kp and ki render system slightly instable. Higher values are of no use as rise time seems to be saturated at kp=20, ki=0.2. For PID controller we should play around these values

PID controller:

Code:

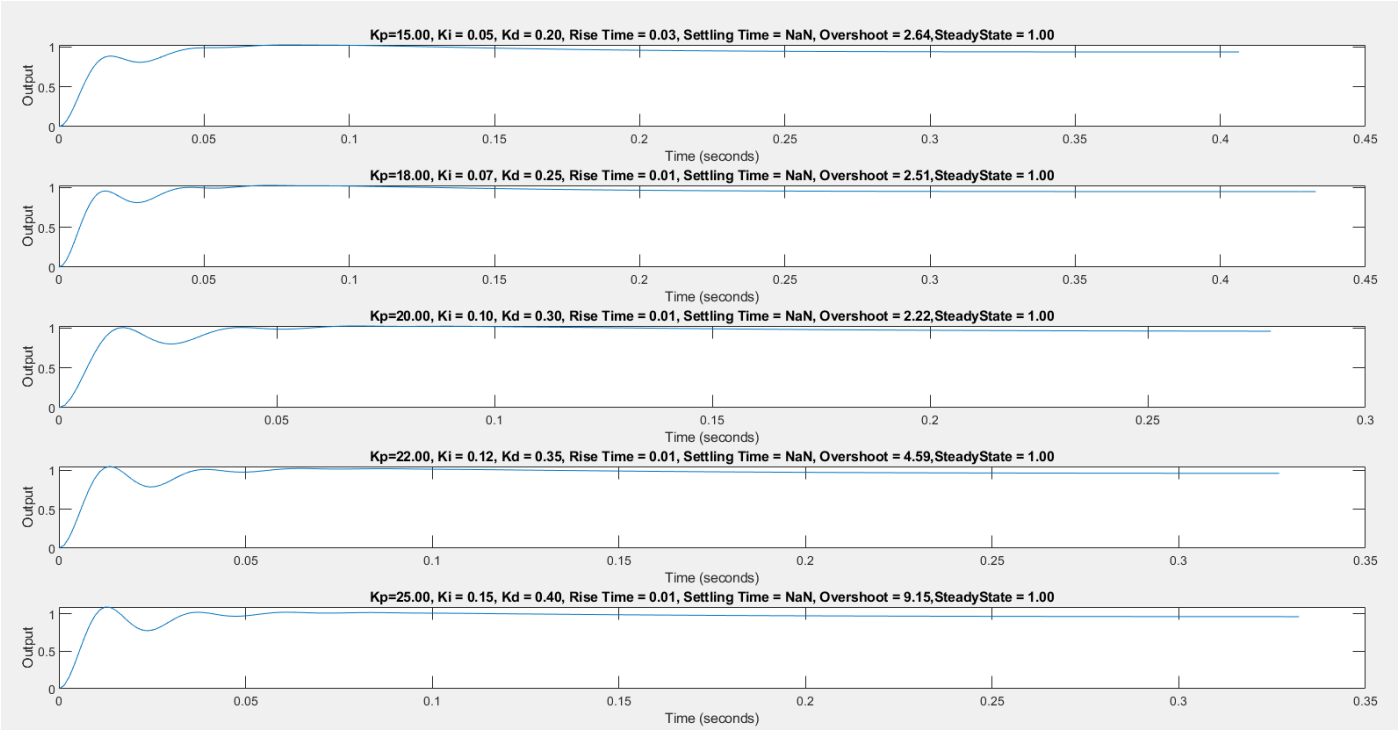
```
%% Implication of PID-Controller on the transfer function

Kp_values = [15,18,20,22,25];
Ki_values = [0.05,0.07,0.1,0.12,0.15];
Kd_values = [0.1,0.15,0.2,0.25,0.3];

for i = 1:length(Kd_values)
    Kp = Kp_values(i);
    Ki = Ki_values(i);
    Kd = Kd_values(i);
    K = pid(Kp,Ki,Kd);
    sys = feedback(K*G, 1);
```

```
    [y, t] = step(sys);
    % Calculate the rise time, settling time, and overshoot
    characteristics = stepinfo(sys);
    rise_time = characteristics.RiseTime;
    settling_time = characteristics.SettlingTime;
    overshoot = characteristics.Overshoot;
   [A, B, C, D] = ssdata(sys);
    ss_sys = ss(A, B, C, D);
    Steadystate = dcgain(ss_sys);
    Peaktime = characteristics.PeakTime;
    subplot(length(Kd_values), 1, i);
    plot(t, y);
    title(sprintf('Kp=%0.2f, Ki = %0.2f, Kd = %0.2f, Rise Time =
%0.2f, Settling Time = %0.2f, Overshoot = %0.2f,SteadyState =
%0.2f%',...
    Kp,Ki,Kd, rise_time, settling_time, overshoot,Steadystate));
    xlabel('Time (seconds)');
    ylabel('Output');
    %xlim([0, 15]);
End
```

Note the Ki values. Changing a few decimals affects the overshoot drastically.

It is important to start with 1 type of controller first and then build up with other gains in order to converge onto ideal values. P controller, D controller, PI, PD and finally PID. Or else it will be difficult to converge onto stability.

Notice how settling time is not definite. This is due to the fact that system only attains near steady state value. Also you need to compromise on other factors in order to leverage other characteristics.


Kp=15.00, Ki = 0.05, Kd = 0.20, Rise Time = 0.03, Settling Time = NaN, Overshoot = 2.64,SteadyState = 1.00
Kp=18.00, Ki = 0.07, Kd = 0.25, Rise Time = 0.01, Settling Time = NaN, Overshoot = 2.51,SteadyState = 1.00
Kp=20.00, Ki = 0.10, Kd = 0.30, Rise Time = 0.01, Settling Time = NaN, Overshoot = 2.22,SteadyState = 1.00
Kp=22.00, Ki = 0.12, Kd = 0.35, Rise Time = 0.01, Settling Time = NaN, Overshoot = 4.59,SteadyState = 1.00
Kp=25.00, Ki = 0.15, Kd = 0.40, Rise Time = 0.01, Settling Time = NaN, Overshoot = 9.15,SteadyState = 1.00
Output
Time (seconds)

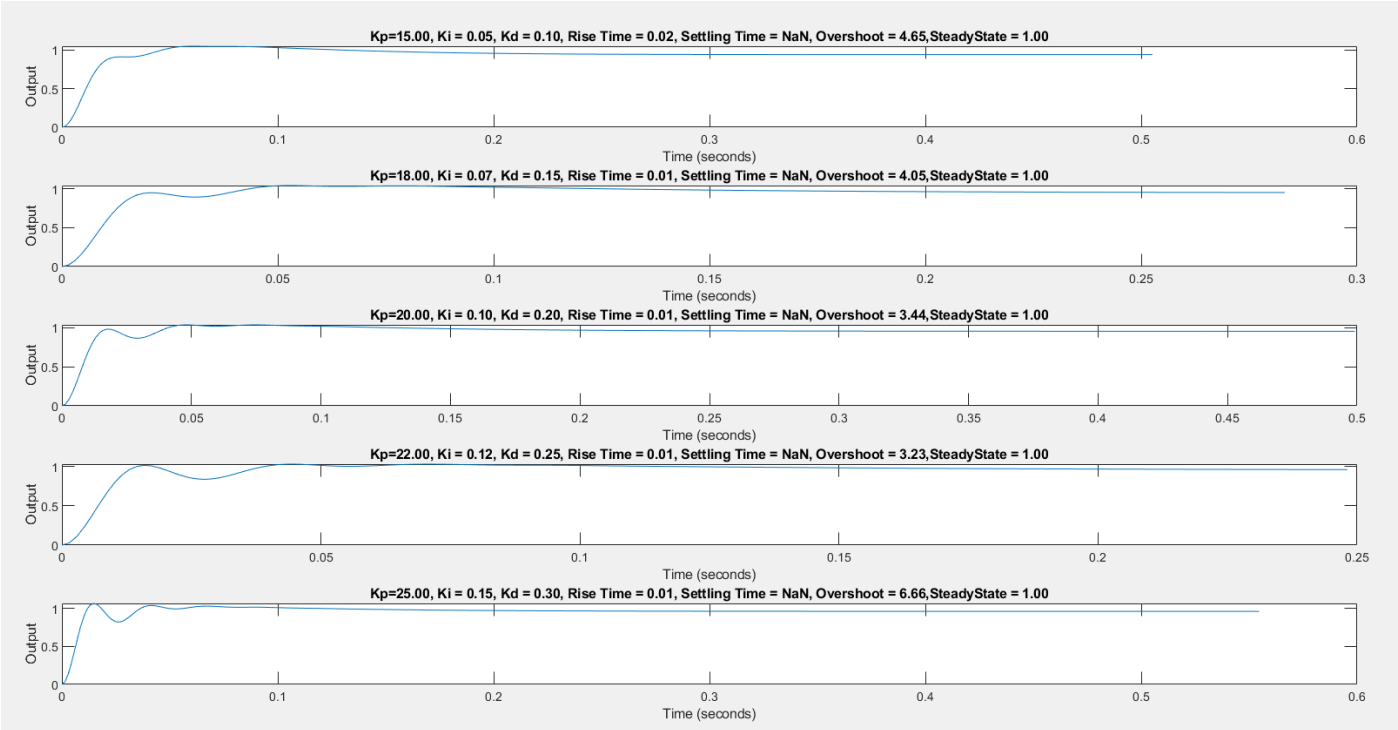

Kp=15.00, Ki = 0.05, Kd = 0.10, Rise Time = 0.02, Settling Time = NaN, Overshoot = 4.65,SteadyState = 1.00
Kp=18.00, Ki = 0.07, Kd = 0.15, Rise Time = 0.01, Settling Time = NaN, Overshoot = 4.05,SteadyState = 1.00
Kp=20.00, Ki = 0.10, Kd = 0.20, Rise Time = 0.01, Settling Time = NaN, Overshoot = 3.44,SteadyState = 1.00
Kp=22.00, Ki = 0.12, Kd = 0.25, Rise Time = 0.01, Settling Time = NaN, Overshoot = 3.23,SteadyState = 1.00
Kp=25.00, Ki = 0.15, Kd = 0.30, Rise Time = 0.01, Settling Time = NaN, Overshoot = 6.66,SteadyState = 1.00
Output
Time (seconds)

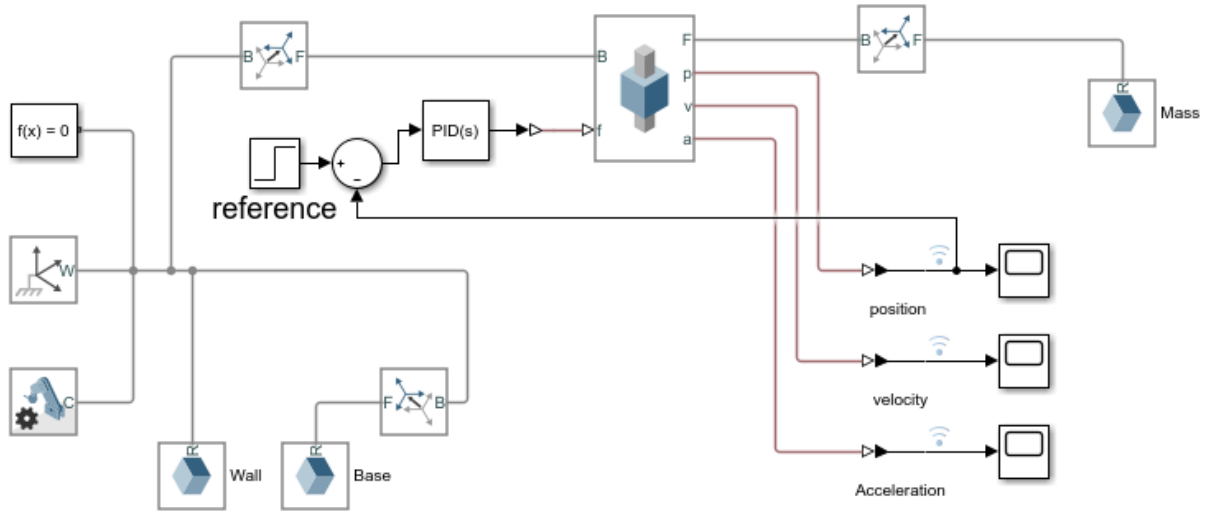
TABLE FOR CONTROLLERS INFLUENE ON 2ND ORDER CHARACERISTICS

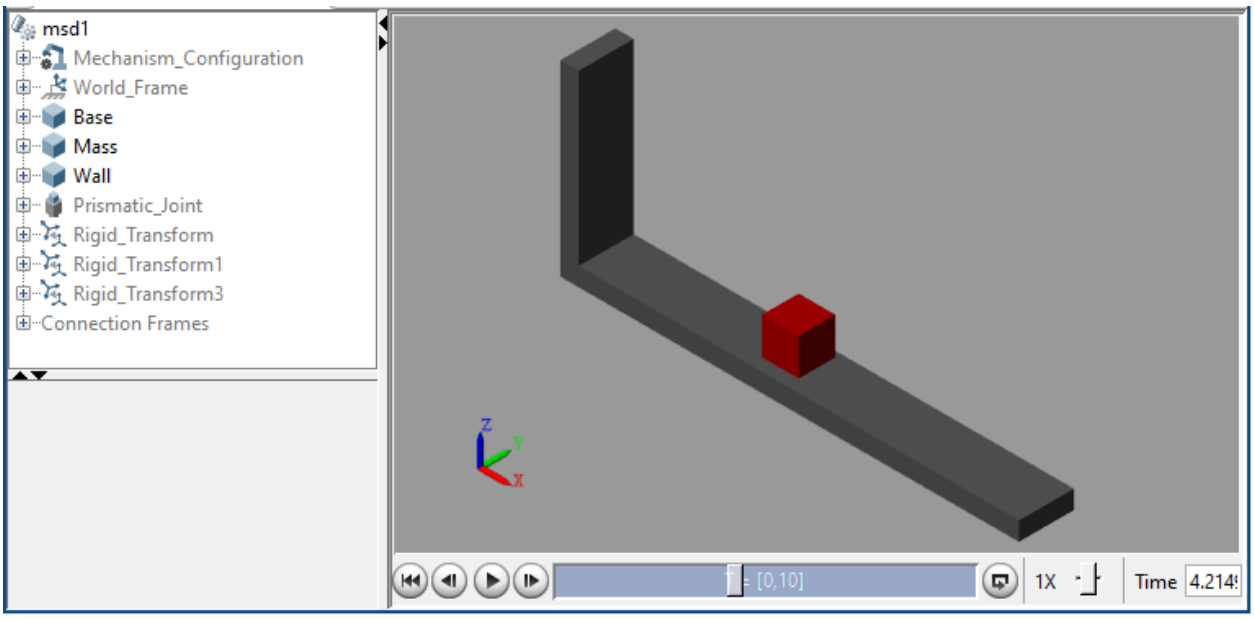
controller	TS	TR	% overshoot	SS value
P	Increasing	Decreasing upto saturation	Increasing until too large(oscillations begin)	Increases to reach SS. Error decreased
I	Decreasing to a certain value & increasing after that	Decrease	Increases to Constant	Constant No error
D	Decreasing to a certain value & increasing after that	Constant	Increasing	Constant
PI	Increase	Decrease	Increase	Constant
PD	Decrease to saturation	Decrease	Decreases to a certain value then starts increasing	Increase
PID	decrease	Decreasing upto saturation	Decreases to a certain value then starts increasing	Constant

Lab#10: Exploring MULTIBODY environment for passive and active dynamic systems: Modeling, Analyzing, Cross-validating, and Evaluating three (M-File, SIMSCAPE, MULTIBODY) different Approaches.

The fundamental aim of this lab is to imply the concepts of MULTIBODY and PID Controllers for modeling, analyzing, and cross-validating the dynamic responses of passive as well as active dynamic systems.Therefore, in this lab, we shall be trying to incorporate the previously developed dynamic systems for comprehending the significance of PID controller.

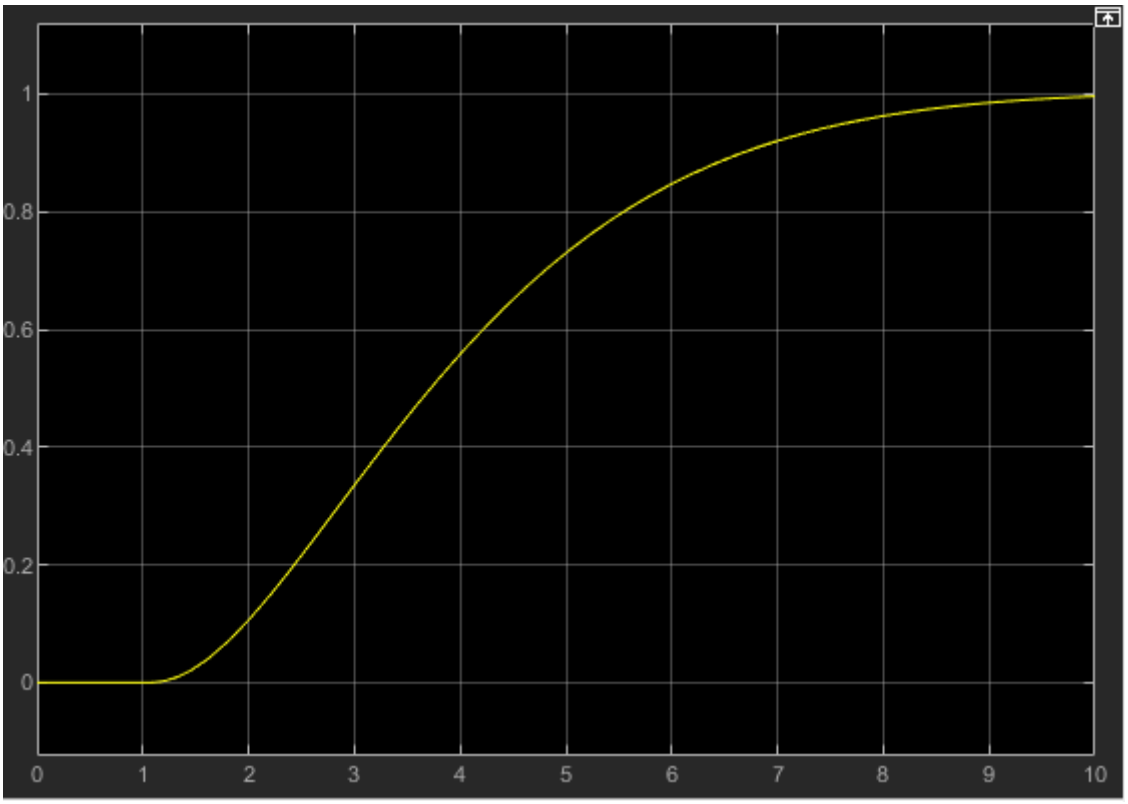
Active Single Mass-Spring Damper System



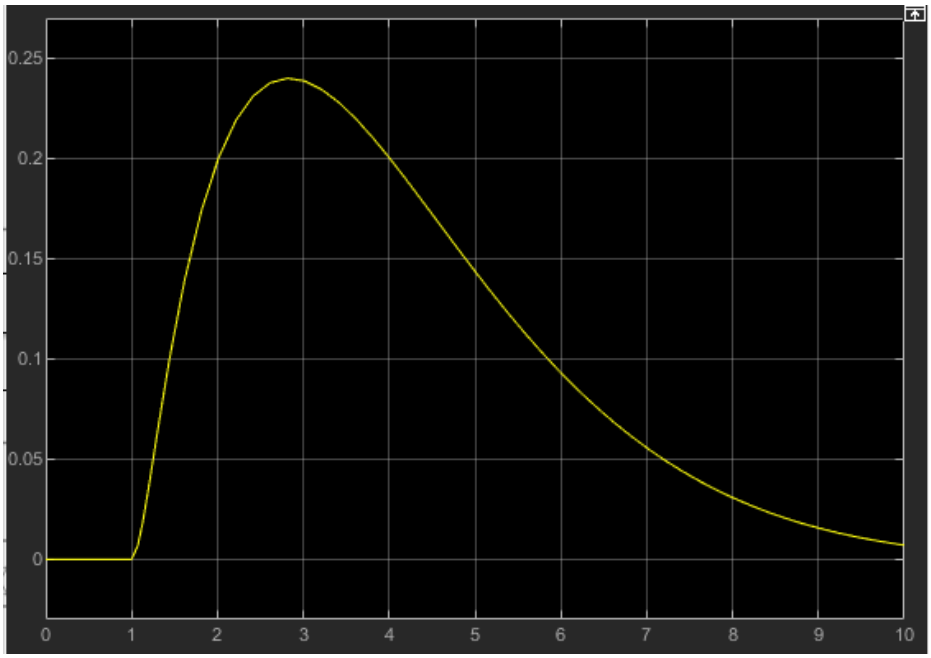


Responses:

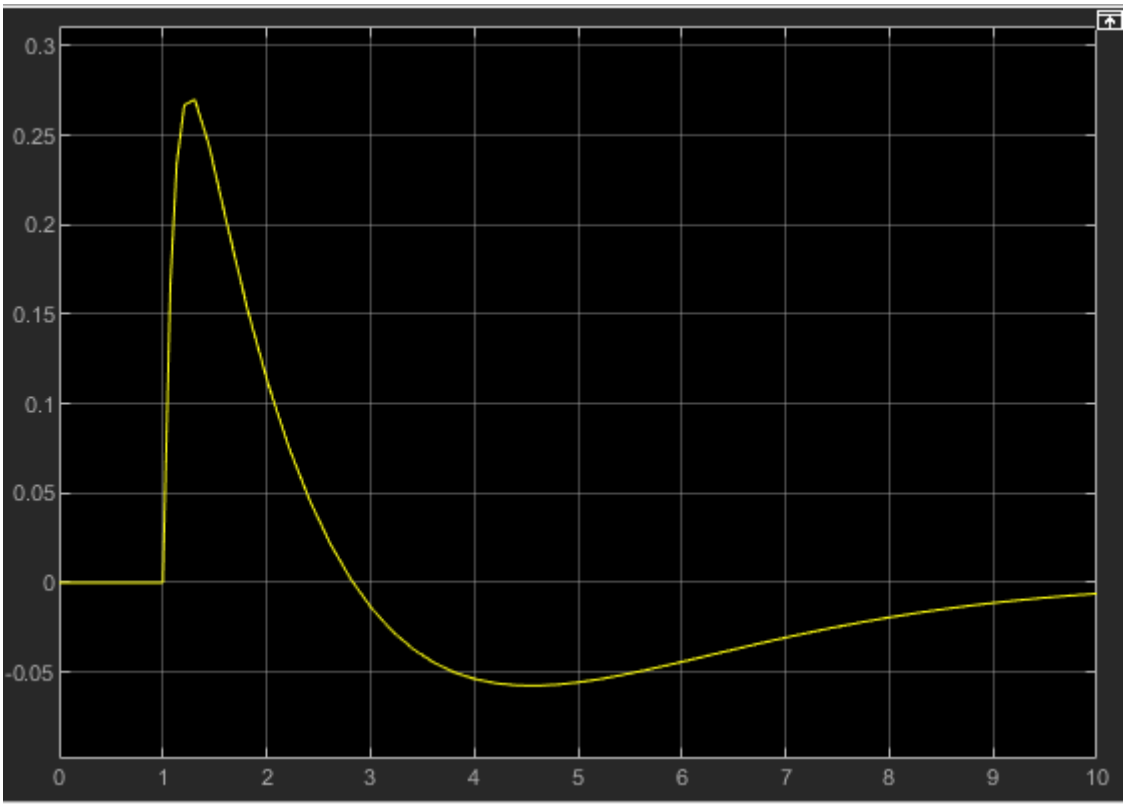
Position:



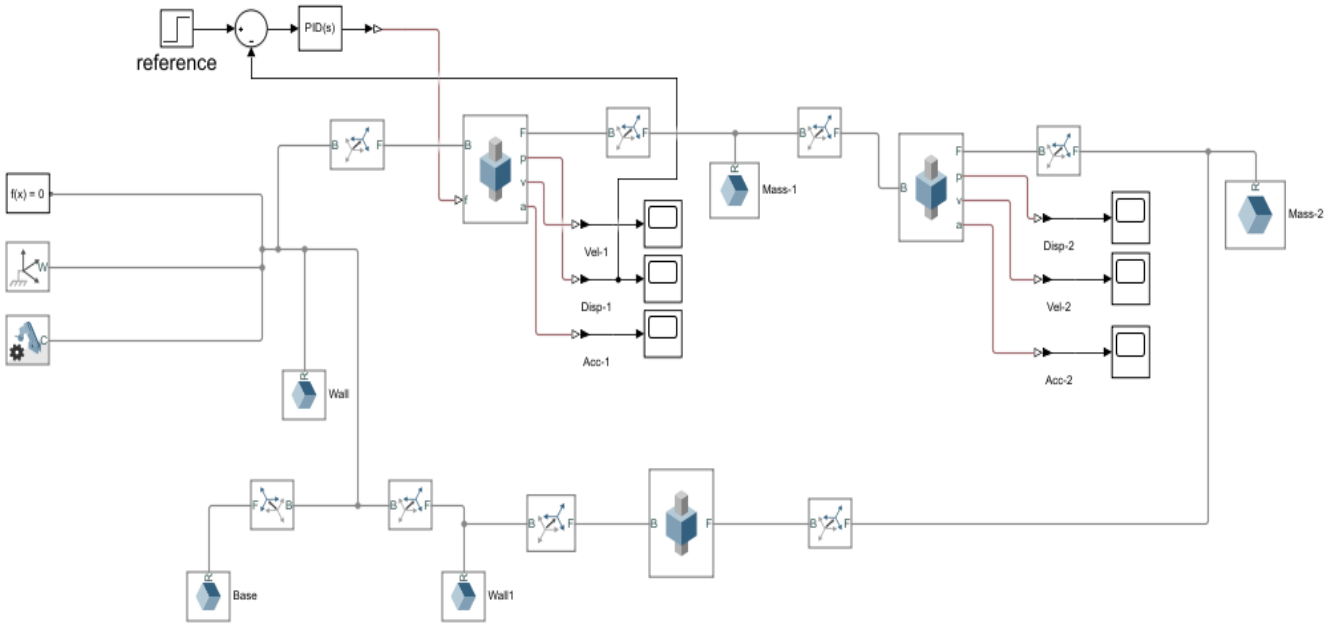
Velocity:

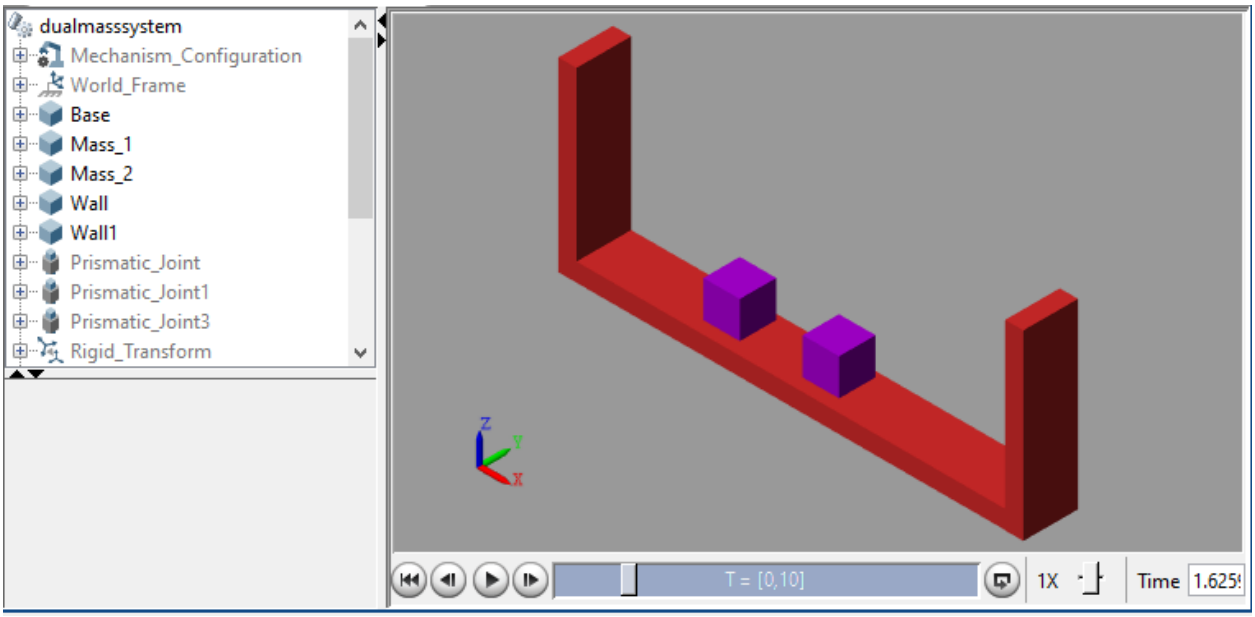


Acceleration:



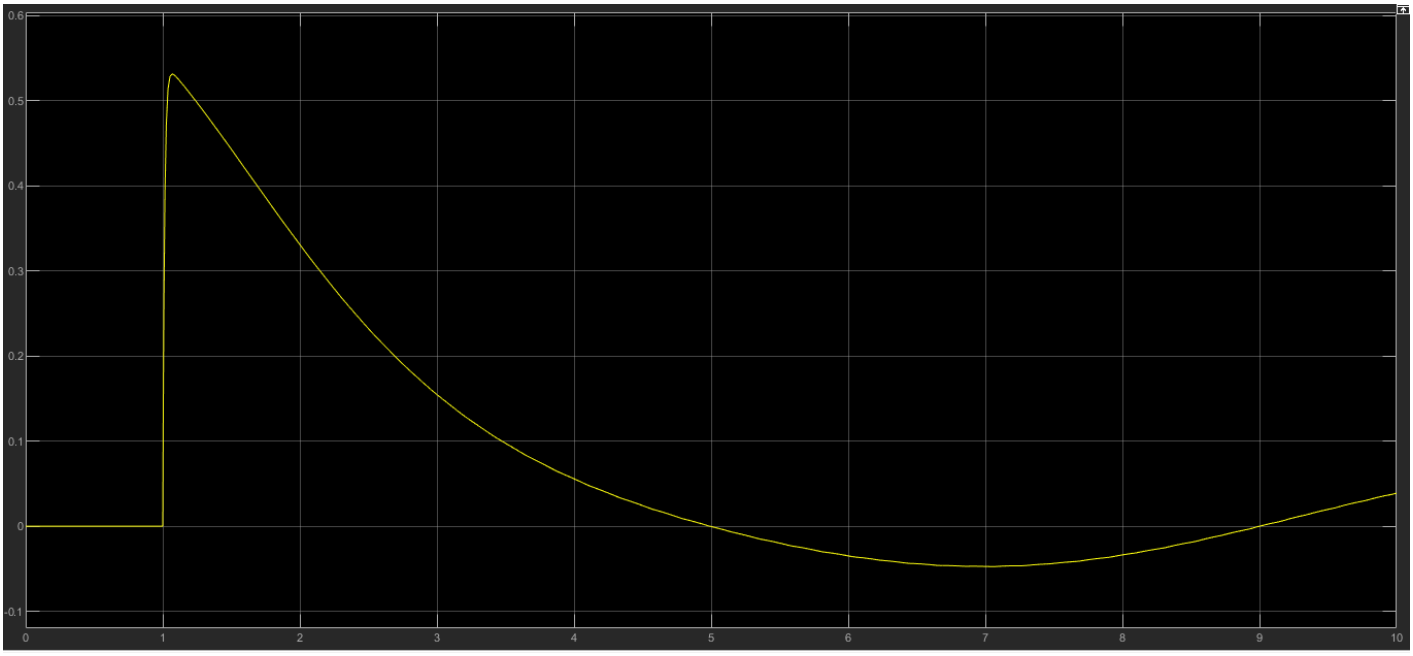
Dual Mass-Spring Damper System



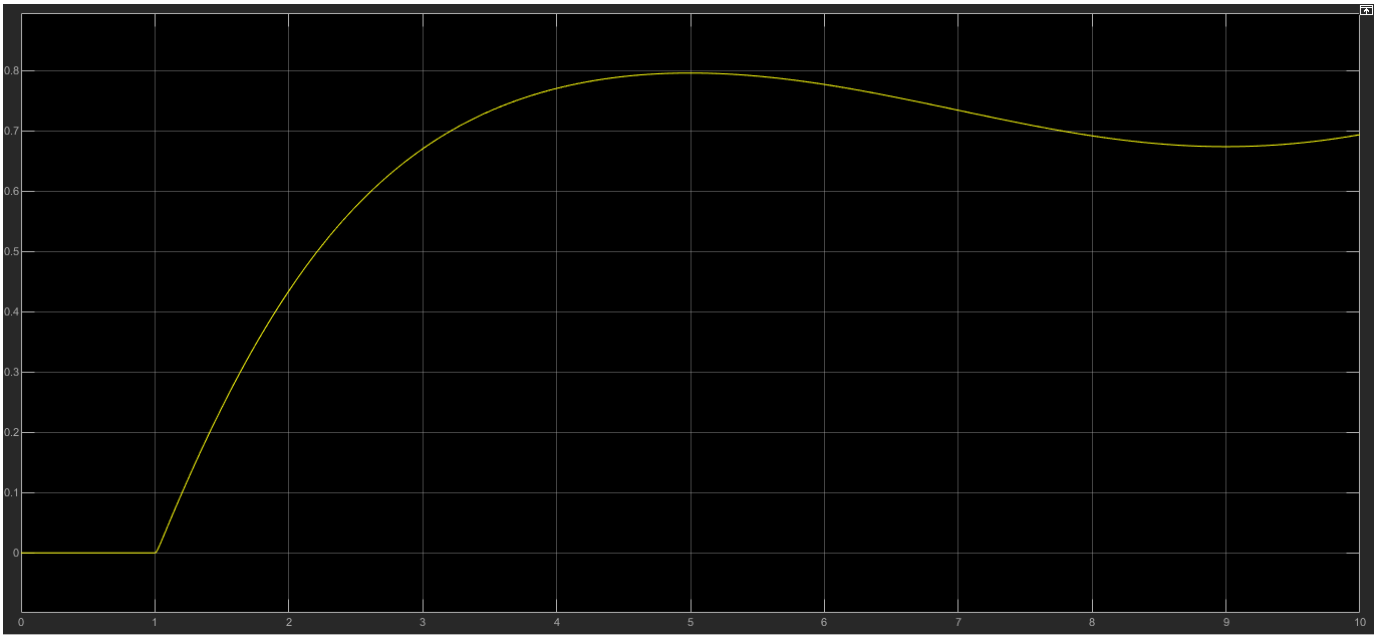


Responses:

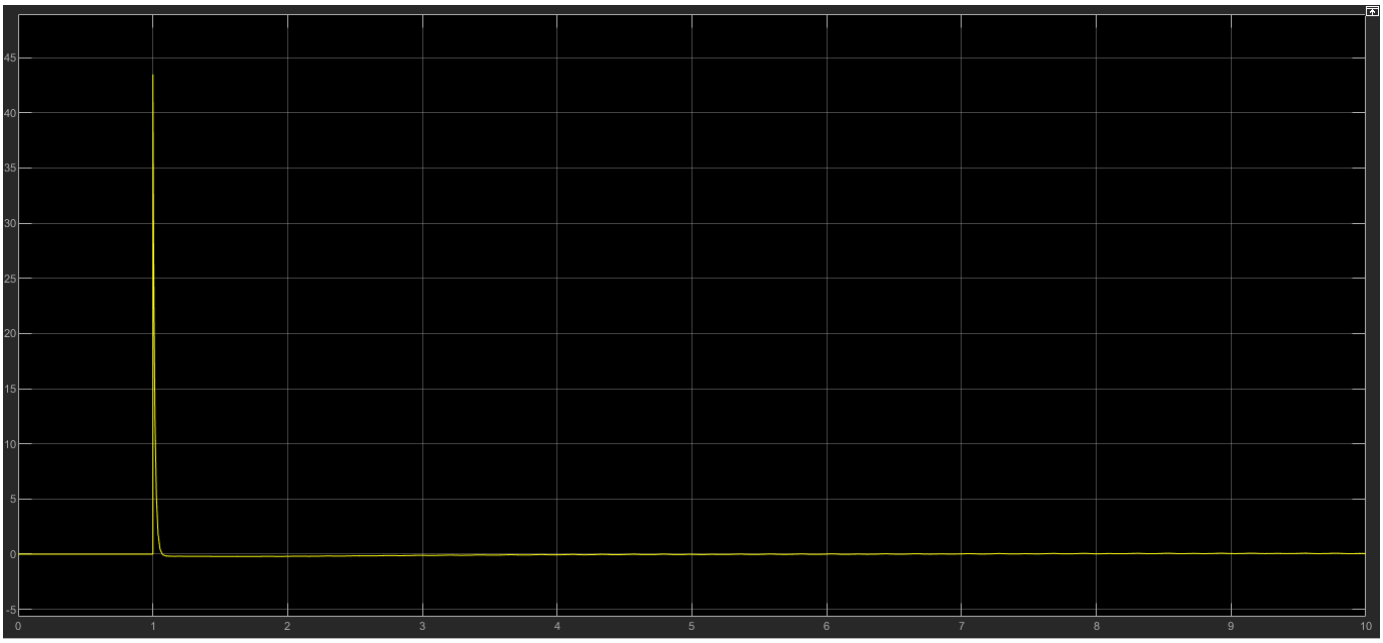
Position of m1:



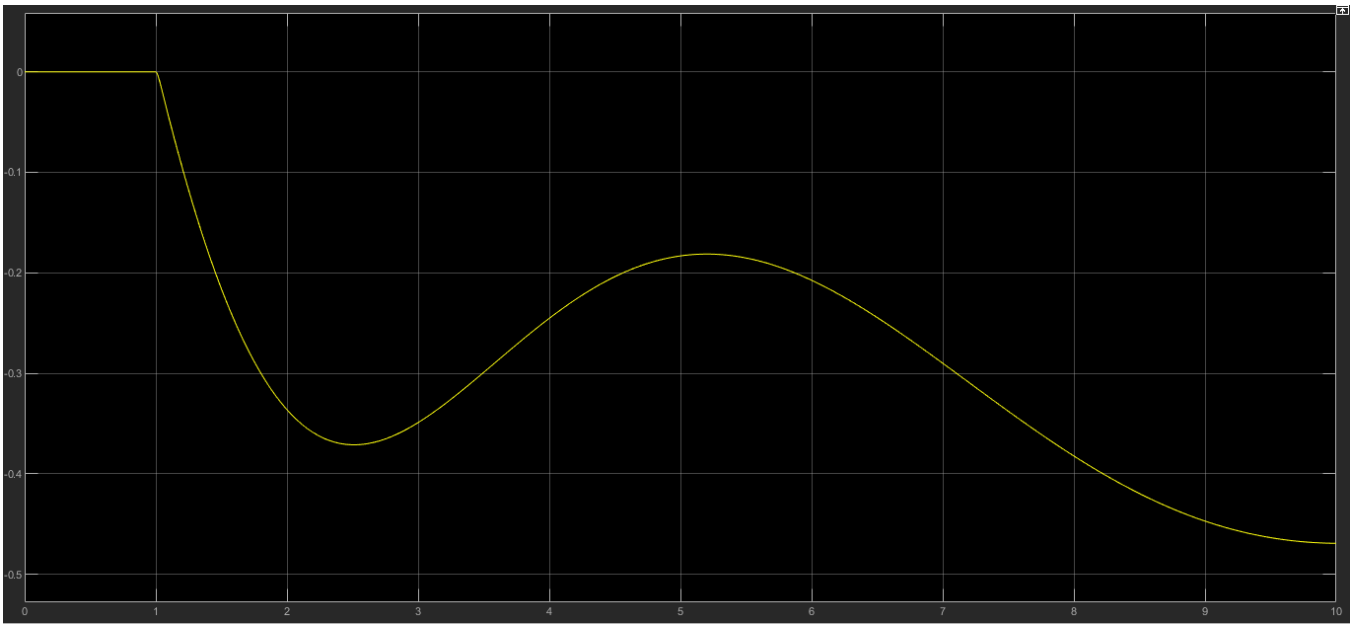
Velocity of m1:



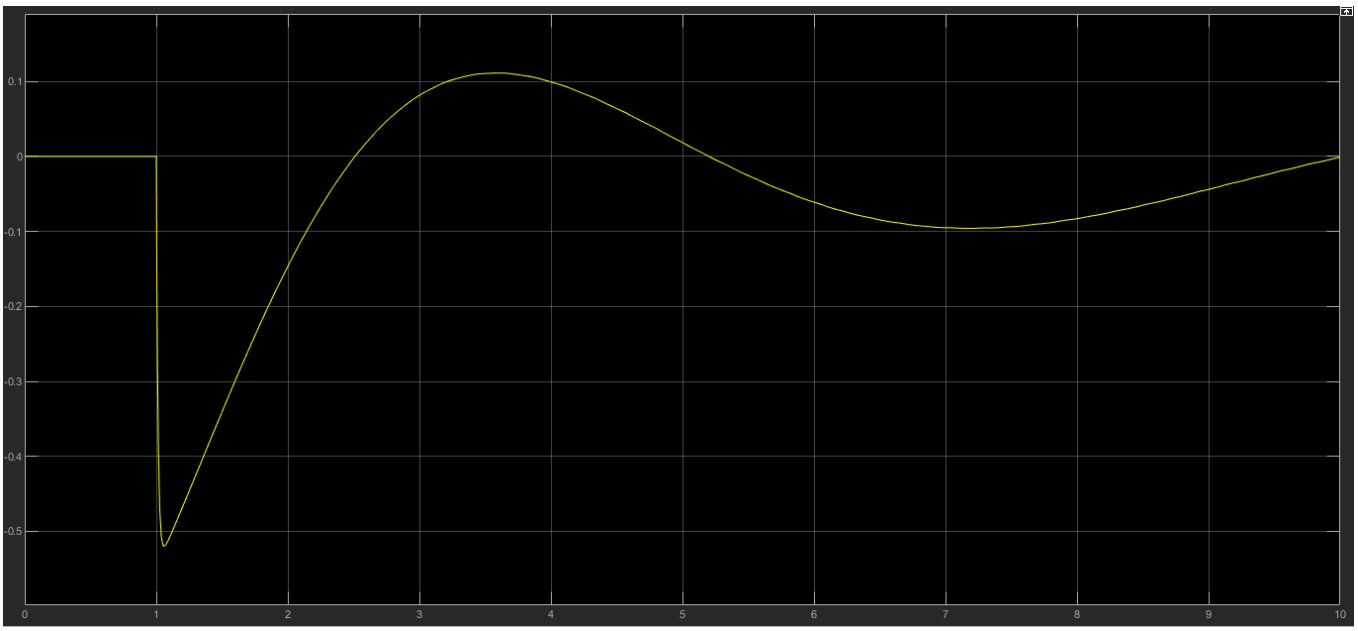
Acceleration of m1:



Position of m2:

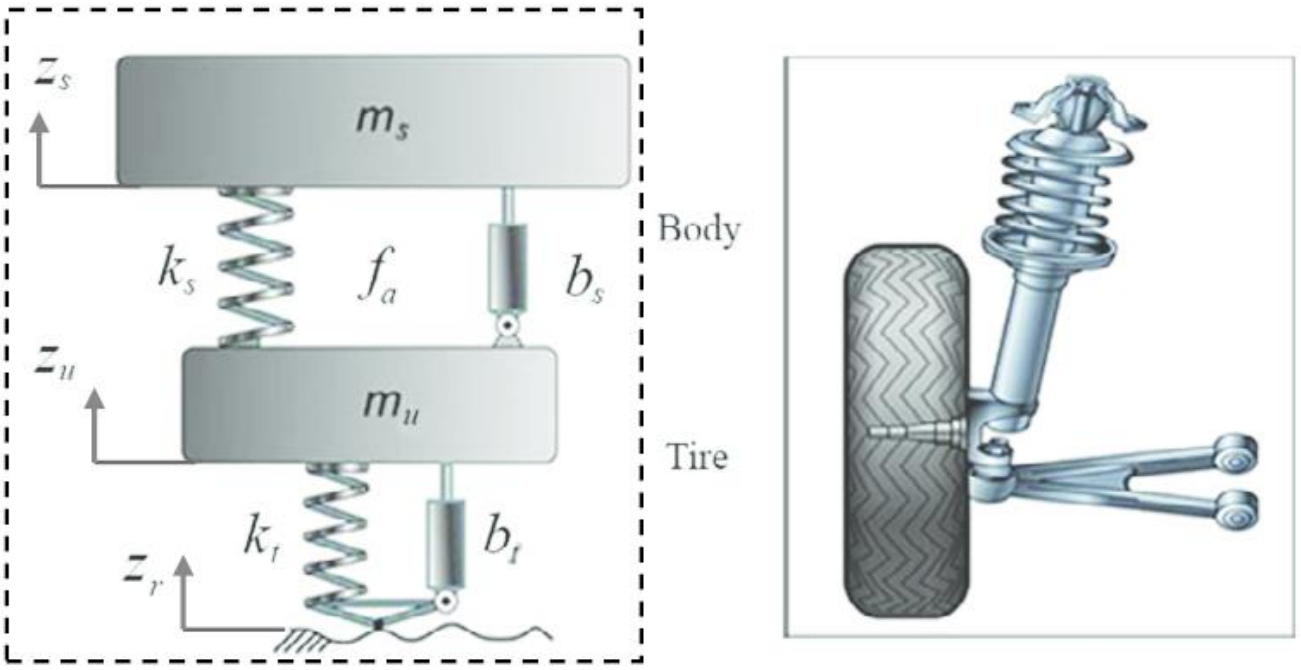


Velocity of m2:

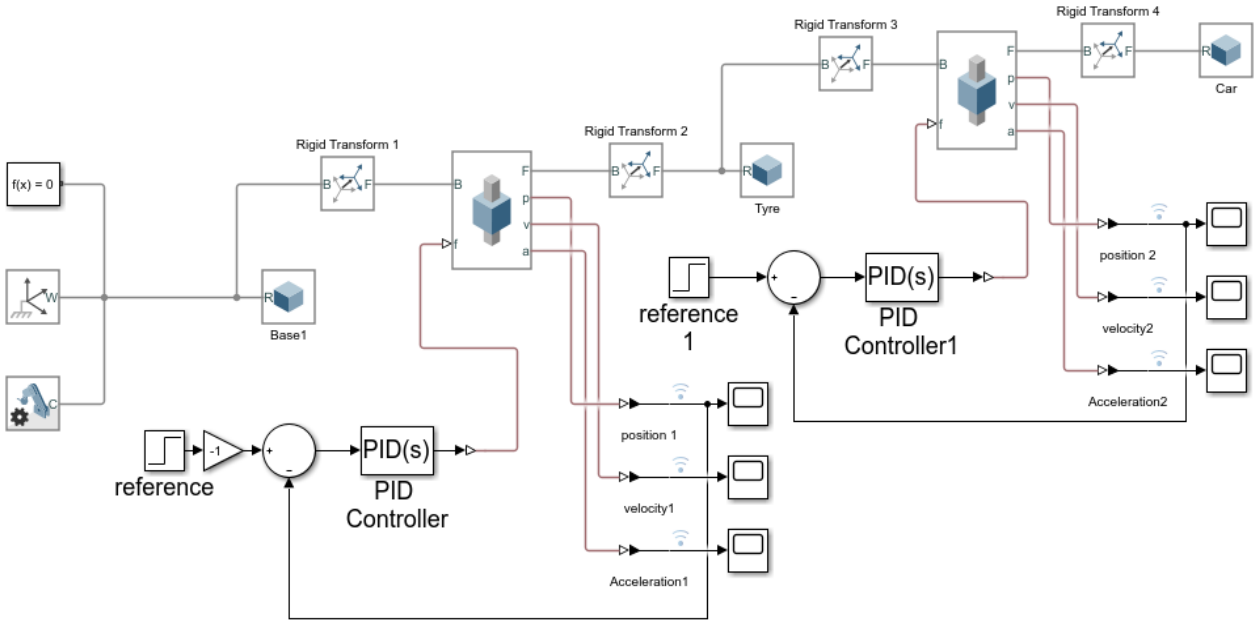


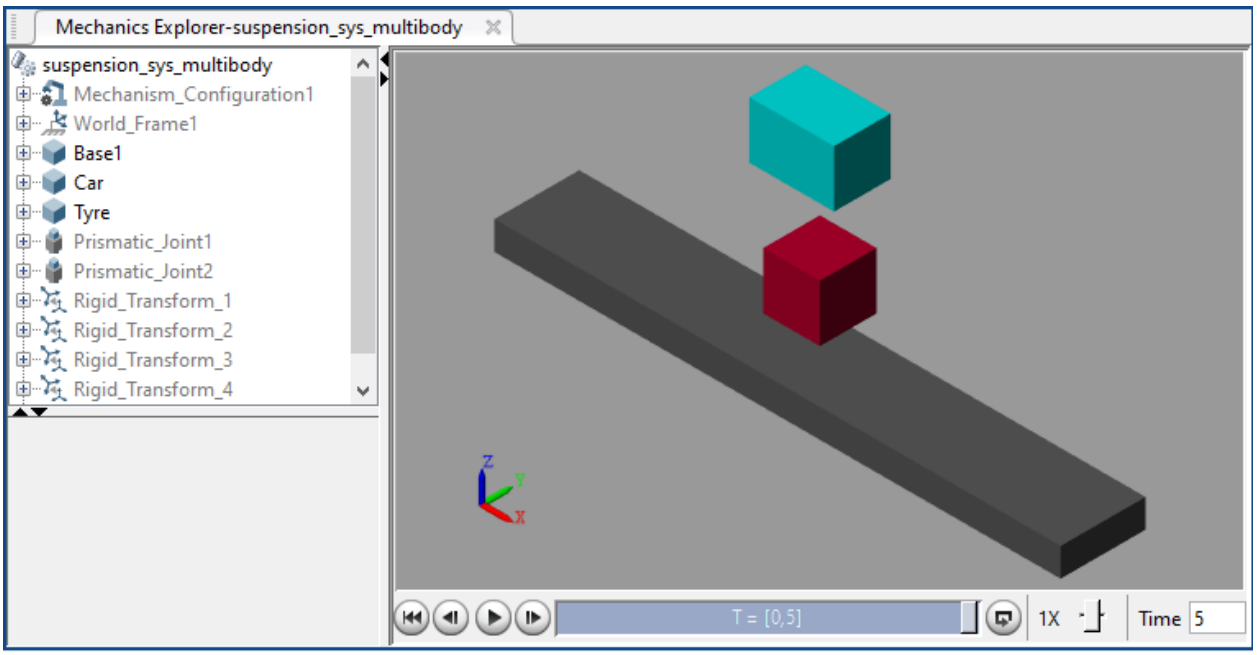
Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

Quarter Car Model



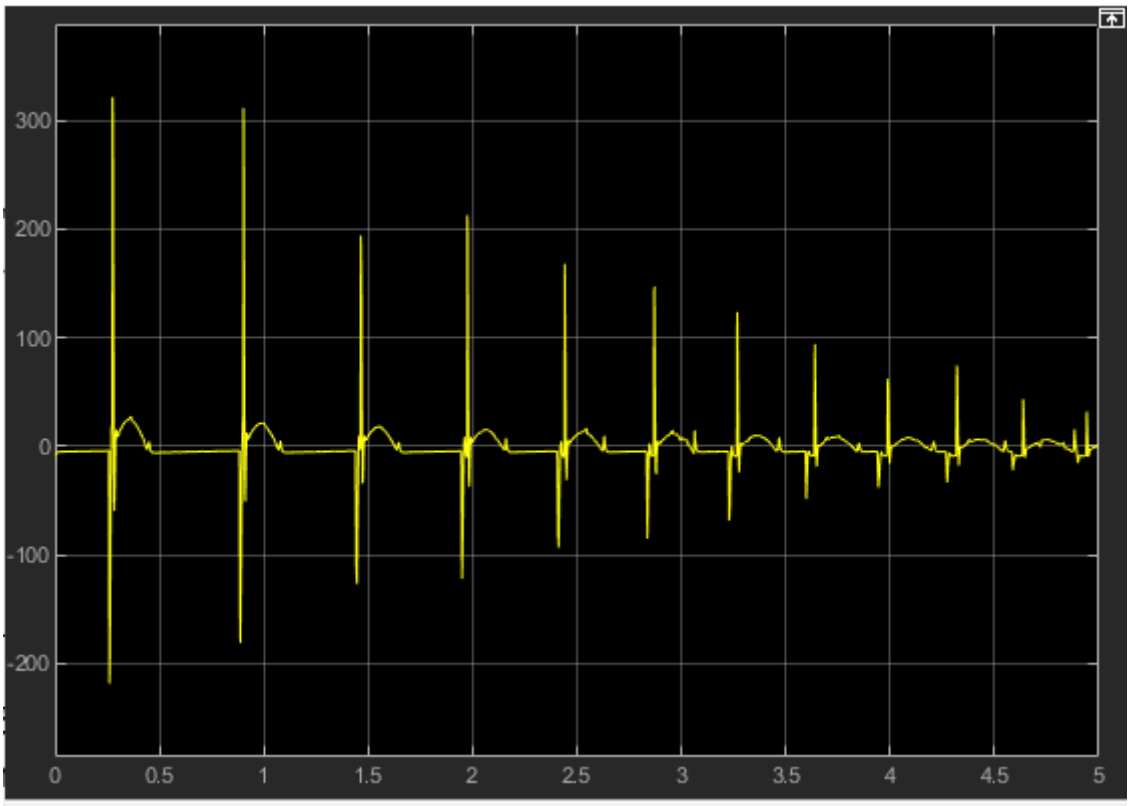
The multibody model of suspension system is desgned by implying all those established theporetical interpretations of previous labs. The goal of this task will be tracking the displacement of the sprung mass and the overall model is shown in Figure.



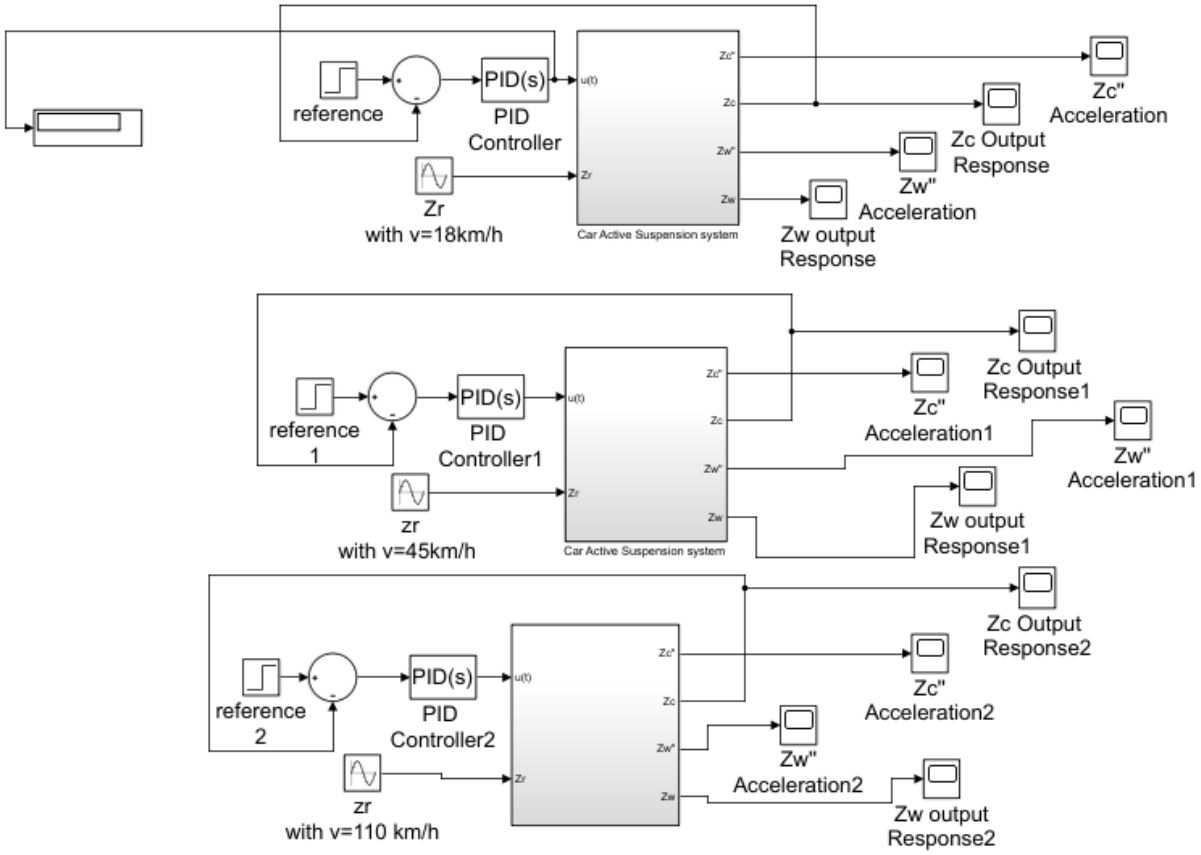


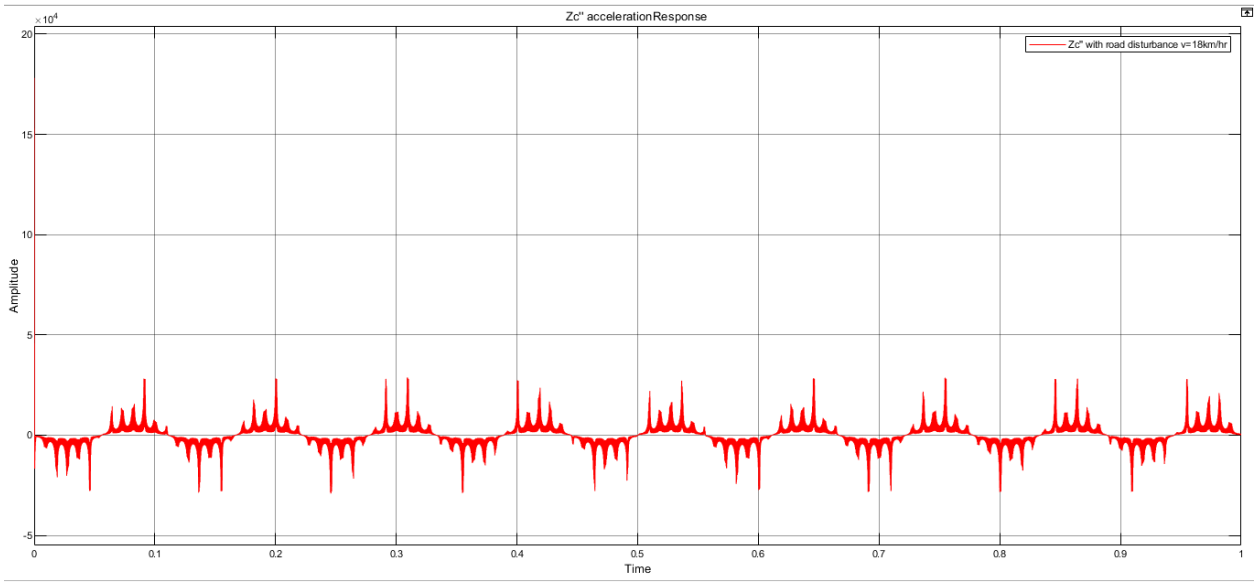
Responses in terms of road disturbances:

Acceleration of mc:



SIMSCAPE MODEL



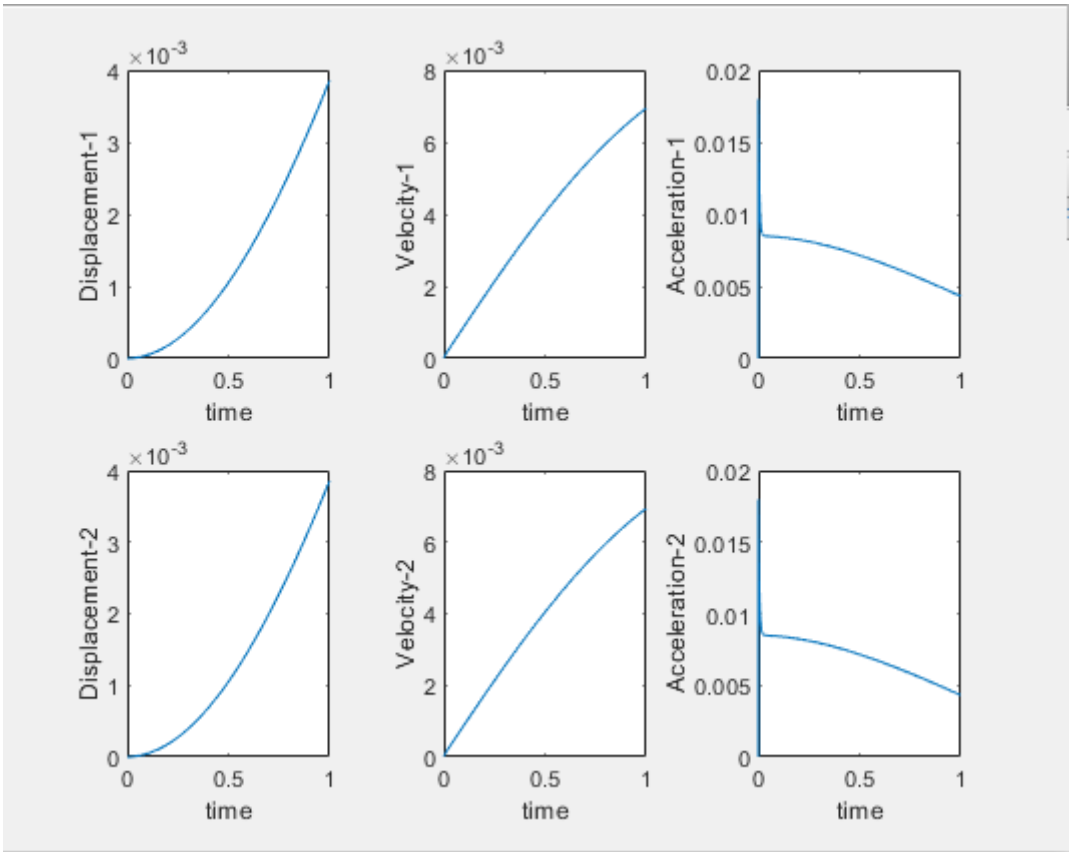


Function:

```
function dy= ActiveSuspensionSystem(t,y)
   u=21.62; kc=1400; kw=2200; bc=1200; bw=1100; mc=1200; mw=10;
   zr=0.2*sin(18*pi);
   dy(1)=y(2);
   dy(3)=y(4);
   dy(2)= (1/mc)*(-kc*(y(1)-y(3))-bc*(y(2)-y(4))+u);
   dy(4)= (1/mw)*(-u+kc*(y(1)-y(3))+bc*(y(2)-y(4))-kw*(y(3)-zr)-bw*(y(4)));
   dy=dy';
end
```

Code:

```
clc;
TR = [0 1];
X0 = [0;0;0;0];
[t,y]=ode45(@ActiveSuspensionSystem,TR,X0);

x1=y(:,1);
v1=y(:,2);
a1=diff(v1)./diff(t);
x2=y(:,1);
v2=y(:,2);
a2=diff(v2)./diff(t);

subplot(2,3,1)
plot(t,x1)
xlabel('time')
ylabel('Displacement-1')
subplot(2,3,2)
plot(t,v1)
xlabel('time')
ylabel('Velocity-1')
subplot(2,3,3)
plot(t,[0;a1])
xlabel('time')
ylabel('Acceleration-1')

subplot(2,3,4)
plot(t,x2)
xlabel('time')
ylabel('Displacement-2')
subplot(2,3,5)
plot(t,v2)
xlabel('time')
ylabel('Velocity-2')
subplot(2,3,6)
plot(t,[0;a2])
xlabel('time')
ylabel('Acceleration-2')
```


Displacement-1
Velocity-1
Acceleration-1
Displacement-2
Velocity-2
Acceleration-2
time
×10^-3

Department of Mechatronics and Control Engineering
University of Engineering and Technology, Lahore Pakistan

LAB HOME TASK-1
(Section A)

Aim and Objectives:

The fundamental aim of this home task-1 is to imply the MATLAB programming skills and classical control approach for the synthesizing (In terms of the characteristics) the dynamic systems (1st and 2nd order systems). Firstly, the first-order system shall be comprehensively analyzed to find the time constant as well as the DC gain of the respective system. Afterward, a detailed discussion will be considered in analyzing the transient characteristics of the 2nd order system. Lastly, MATLAB codes shall be modified in such a manner so that the transfer function from the SIMSCAPE model could lead to the comprehensive analysis of the time-domain characteristics.

5. To analyze the time-domain characteristics of 1st-order systems (DC Gain and Time Constant).
6. To improve the code in such a way that the transfer function from SIMSCAPE model (RC Series Circuit) could result in a comprehensive analysis (In terms of Time Constant and DC Gain).
7. To analyze the time-domain characteristics of 2nd order systems (DC Gain and Time Constant).

Assigned Tasks:

Task 1:

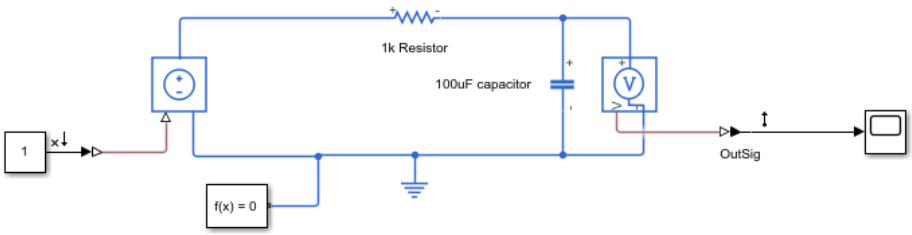
Problem Statement:

Consider the RC series circuit in the SIMSCAPE environment (Already done in the previous labs) and find its transfer function from the developed model. Afterward, your transfer function will be in the workspace; therefore, you may start reading the numerator as well as denominator sequentially (in m-files) for the forthcoming analysis (To find DC gain, Time constant, etc., just the way we have done in the lab) of the system. You may incorporate the loop for implying the step signal of different magnitudes (1 to 5) in which you are also required to incorporate the "Generic form of 1st order system in time domain" so that you could cross-validate the step responses of built-in command and your developed mathematical model (You may refer to your lecture slides). Finally, a graph should be plotted between the test signal magnitude (On X-axis) and the Steady State Value (On the y-axis). Most importantly, you should also see the step responses (both built-in command as well as through the developed model) of different magnitudes in the separate graph.

M File:

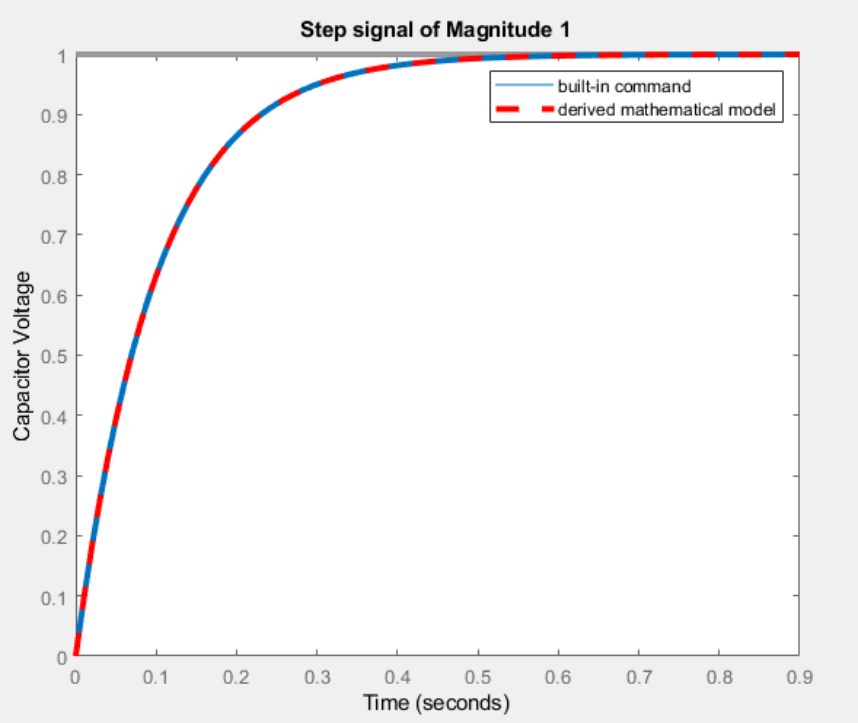
```
clc;
syms s
fprintf("----- Analysis of first order system -----");
fprintf("\n>> The example is of a series RC circuit.");
fprintf("\n-------------------------------------------------\n");
fprintf("The transfer function : Vout/Vin = \n");tf(linsys1)
fprintf("Some important parameters are : \n");
[num,den]=tfdata(tf(linsys1));
num=cell2mat(num);den=cell2mat(den);
k=num(end)/den(end); %putting s=0 results in only the constant values/last entries
of matrix
tau=den(1)/den(2);
Ts=4*tau;
text=sprintf("DC Gain = %d\n",k);
fprintf(text);
text=sprintf("Time Constant = %.4f s\n",tau);
fprintf(text);
text=sprintf("Settling Time ~ %.4f s\n",Ts);
fprintf(text);

signalMag=[];
SteadyVal=[];
for i=1:5
    figure;
    t=0:0.01:(tau+2*Ts);
    u=i*ones(1,length(t));
    lsim(num,den,u,t);
    set(findall(gcf,'type','line'),'linewidth',3);
    hold on;
    y=i*k*(1-exp((-1*t)./tau));
    plot(t,y,'r--','lineWidth',3);
    title(sprintf("Step signal of Magnitude %d",i));
    xlabel("Time");
    ylabel("Capacitor Voltage");
    legend("built-in command","derived mathematical model");
    signalMag(end+1)=i;
    SteadyVal(end+1)=y(end);
end

figure;
scatter(signalMag,SteadyVal);
title("Test Magnitude vs Steady State Value");
xlabel("Test Magnitude");
ylabel("Steady State Value");
```

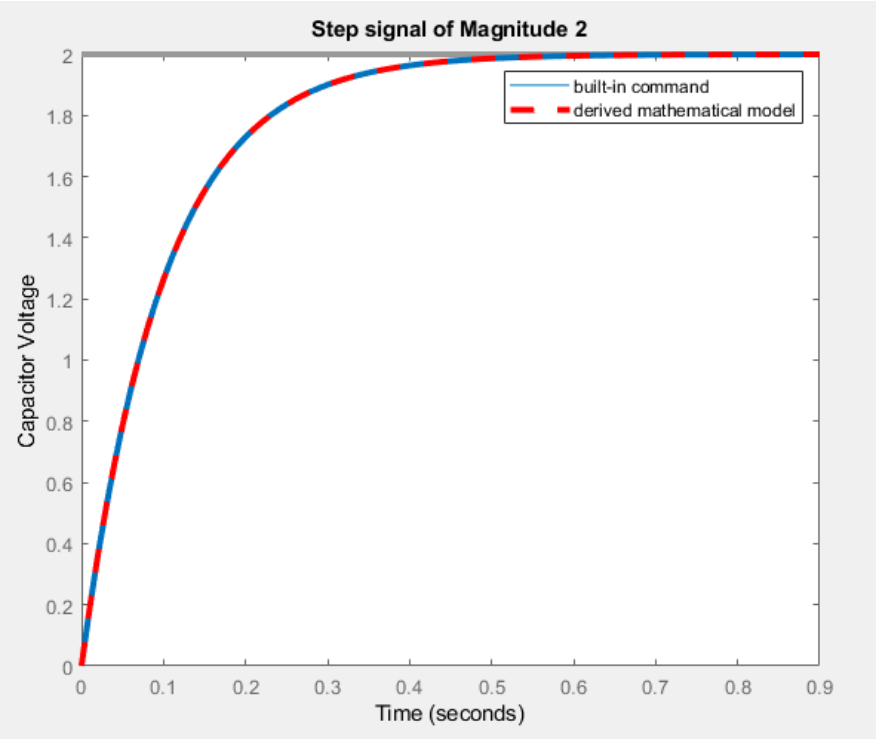
SIMSCAPE model:

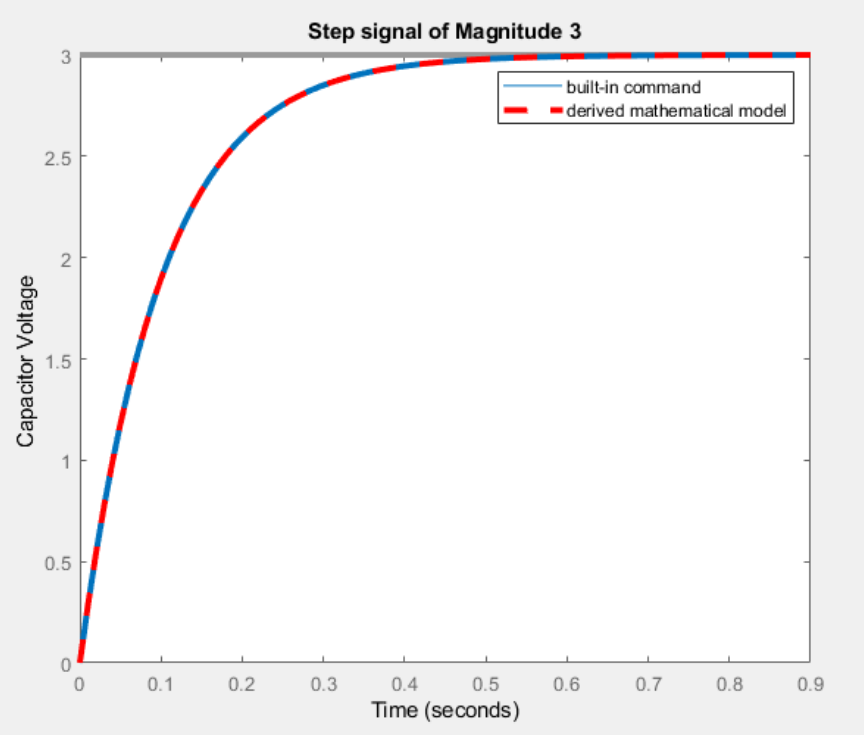


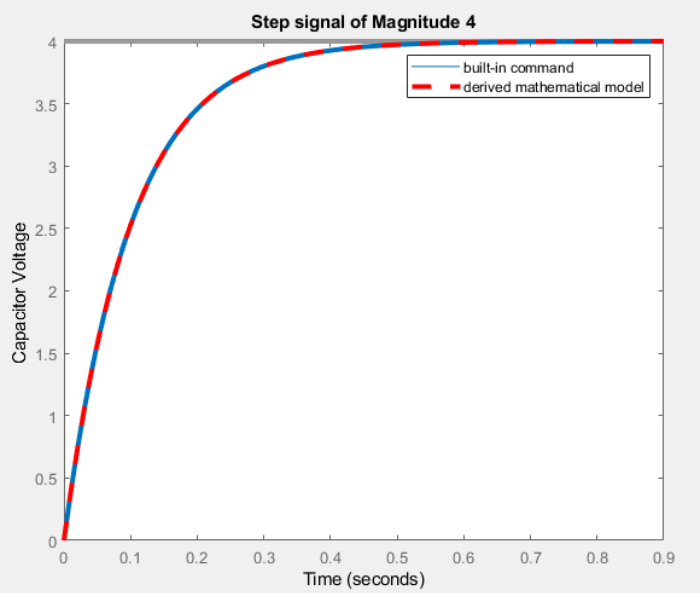
Results:

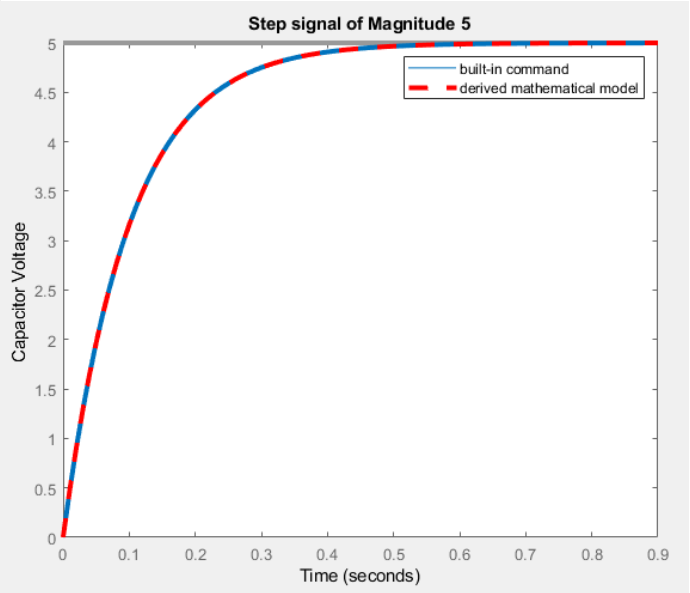
```
----- Analysis of first order system -----
>> The example is of a series RC circuit.
-----------------------------------------------
The transfer function : Vout/Vin =

ans =
  |
   From input "Constant" to output "Vc":
     10
   ------
   s + 10

Name: Linearization at model initial condition
Continuous-time transfer function.

Some important parameters are :
DC Gain = 1
Time Constant = 0.1000 s
Settling Time ~ 0.4000 s
```

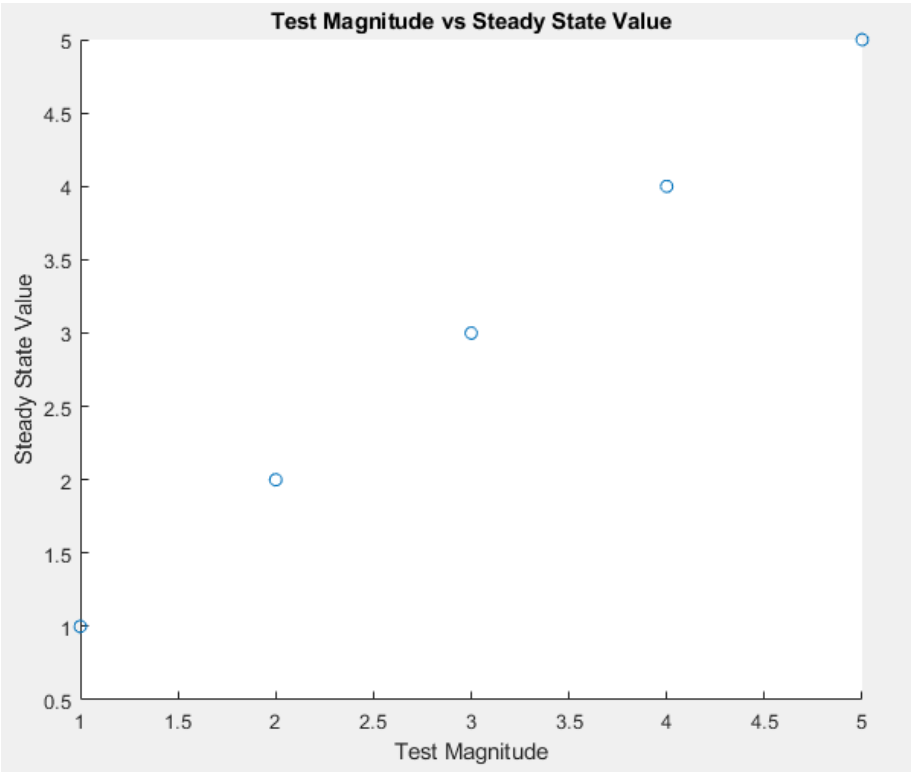



Step signal of Magnitude 2
built-in command
derived mathematical model
Capacitor Voltage
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
0.1
0.3
0.5
0.7
0.9


Step signal of Magnitude 3
built-in command
derived mathematical model
Capacitor Voltage
Time (seconds)
0
0.5
1
1.5
2
2.5
3
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


Step signal of Magnitude 4
built-in command
derived mathematical model
Capacitor Voltage
Time (seconds)


Step signal of Magnitude 5
built-in command
derived mathematical model
Capacitor Voltage
Time (seconds)


Test Magnitude vs Steady State Value
Steady State Value
Test Magnitude

Comments/Remarks:

The outputs obtained strengthen the concepts studied in the theory lectures. The graphs obtained are similar to the generic responses and both the graphs obtained by mathematical interpretations as well as by the built in functions are exactly the same. A thing to note is the validation of the formula that steady state value is equal to the product of DC gain and the magnitude of step signal. Another interesting thing is that the x-axis (in other words the time characteristics) is independent of the magnitude of step signal!

Task 2:

Problem Statement:

Consider any rotational 2nd order system (in generic form) in the m-files and of course, you may reutilize your previously developed code for finding the transfer function. Afterward, do the following action items in a single m-file:

1. Modify the code in such a manner that the different parameters (Numerator and Denominator) of the above-obtained transfer function (In your case it will be a rotational 2nd order system) should have the comparison with the generic form of the second order system. Subsequently, you will be achieving the generic forms of the Natural frequency, Damped Natural Frequency, Damping Ratio, DC Gain of the model, and time domain characteristics (Recall the most recent lectures and of course, you may take help from the shared m-files).
2. Ask the user for the random values of J, D, and K (the Major three components if and only if you have not selected the geared system) from the user, and based on those values, respond to the user with the "Category of the system" (Due to given values). Following the progression, the response of that category (For example Overdamped) should also be generated; however, the response should be in the form of SUBPLOT in which 1st SUBPLOT would be the response using the built-in step command and 2nd SUBPLOT should be due to the developed function of the corresponding category (You may refer to the lecture slides and shared m-files).
3. You are well aware of all the categories of the system (Undamped, Underdamped, Overdamped, and Critical Damped). For example, based on the user-entered values, you have got the underdamped system (It could be any category) so your next goal is to ask the user the preference for the remaining three categories: "Which category do you want to see first, second, and in third pref.?" and code should be smartly written in such a manner that based on the user choice it should automatically change the parametric values of B (By keeping J and K constant: Recall the example from your lecture slides) to see the responses as per user-preferences in subplots (again SUBPLOT-1 will be using built-in step command and SUBPLOT-2 will be from the mathematical developed function). Furthermore, the code should also return the parametric values of the dynamic system (on which you have got that preferred category) along with all the time-domain characteristics (i.e., Rise Time, Settling Time, Percentage Overshoot, etc.) in each category of the system. Most importantly, after each category of the system, the code should ask the user to press "enter" for analyzing the sequential category as this would be more user-friendly to investigate the parameters in that category of the system.
4. After achieving Objective-3, finally, a list of graphs should be generated in the following sequence:
    - Bar-Chart in which the x-axis should be consisting of four different categories and on the y-axis, the respective natural and damped natural frequencies (Coincident charts (One for natural and the other for damped frequency) for each category) should be enlightened.
    - Graphs between categories of the system and certain dominant characteristics such as "Rise Time" and "Settling Time".

M File:

```
clc;
syms s
fprintf("----- Analysis of second order system -----");
fprintf("\n>> The example is of a rotational mass spring damper system.");
J=input("\nPlease enter the value of Inertia (J, in kg) = ");
D=input("Please enter the value of Rotational damper (D, in N-m-s/rad) = ");
K=input("Please enter the value of Rotational spring (K, in N-m/rad) = ");
G=1/(J*s^2 + D*s + K);
fprintf("\n-----------------------------------------------\n");
fprintf("The transfer function : Theta(s)/T(s) = \n");pretty(G);
fprintf("Some important parameters are : \n");
% important parameters
[num,den]=numden(G);
num=sym2poly(num);den=sym2poly(den);
k=1/K;
w=sqrt(K/J);
zetta=D/(2*sqrt(J*K));
wd=w*sqrt(1-zetta^2);
tau=1/(zetta*w);
if zetta==0
    tau=0;
elseif zetta>1
    tau=1/(abs(max((-zetta*w-w*sqrt(zetta^2-1)),(-zetta*w+w*sqrt(zetta^2-1)))));
end
Ts=4*tau;
PO=exp(-zetta*pi/sqrt(1-zetta^2));
Tr=1.8/w;
Tp=pi/wd;
% printing parameters
fprintf(sprintf("DC Gain = %d\n",k));
fprintf(sprintf("Damping Ratio = %.2f\n",zetta));
fprintf(sprintf("Undamped natural frequency = %.2f\n",w));
fprintf(sprintf("Damped frequency = %.2f\n",wd));
fprintf(sprintf("Time Constant = %.4f s\n",tau));
fprintf(sprintf("Settling Time ~ %.4f s\n",Ts));
fprintf(sprintf("Rise Time ~ %.4f s\n",Tr));

% category of system
categ=["Under damped", "Critically damped", "Over damped", "Undamped"];
CategInOrder=[""];
undampFreq=[];
dampFreq=[];
timePer=[];
```

```
t=0:0.01:(tau+2*Ts);
if zetta<1 && zetta>0
    fprintf(sprintf("Peak Time ~ %.4f s\n",Tp));
    fprintf(sprintf("Percentage Overshoot ~ %.2f %s \n",PO*100,"%%"));
    fprintf("--> The category of the system is : 'Under Damped'\n");
    phi=atan(zetta/(1-zetta^2));
    y=k*(1-(1/sqrt(1-zetta^2))*exp(-zetta*w*t).*cos(w*t*sqrt(1-zetta^2) - phi));
    cat="Under damped";
elseif zetta==1
     fprintf("--> The category of the system is : 'Critically Damped'\n");
     y=k*(1-(1+w*t).*exp(-w*t));
     cat="Critically damped";
elseif zetta>1
     fprintf("--> The category of the system is : 'Over Damped'\n");
     p1=-zetta*w-w*sqrt(zetta^2-1);
     p2=-zetta*w+w*sqrt(zetta^2-1);
     y=k*(1 + (-0.5+zetta/(2*sqrt(zetta^2-1))).*exp(p1*t)+(-0.5-
zetta/(2*sqrt(zetta^2-1))).*exp(p2*t));
     cat="Over damped";
else
     fprintf("--> The category of the system is : 'Undamped'\n");
     t=0:0.01:5;
     y=k*(1-cos(w*t));
     cat="Undamped";
end

subplot(1,2,1);plot(t,y,'r');
xlabel('Time');ylabel('System Response');
title('Response by mathematical formula');
subplot(1,2,2);step(num,den,t);
xlabel('Time');ylabel('System Response');
title('Response by built in step func');
CategInOrder(end+1)=cat;
dampFreq(end+1)=wd;
undampFreq(end+1)=w;
timePer(end+1)=tau;
% asking for categories that are left off
fprintf("\n\n");
fprintf(sprintf("Since you have selected %s, following categories have been left
off:\n",cat));
j=1;
leftCateg=[""];
for i = 1:4
    if categ(i)~=cat
        fprintf(sprintf("%d. %s ",j,categ(i)));
        leftCateg(end+1)=categ(i);
        j=j+1;
    end
end
fprintf("\n\nNow type an order in which you'd like to display your system.\n");
fprintf("For Example, typing [1,2,3] would result in displaying the parameters\n");
fprintf(sprintf("of %s system, then for %s system\n",leftCateg(2),leftCateg(3)));
fprintf(sprintf("and finally for %s system.\n",leftCateg(4)));
order=input("Type your desired order >> ");
fprintf('\n\n');
```

```
for i=1:3
    cat=leftCateg(order(i)+1);
    if cat=="Under damped"
        zetta1=0.5;
        D=zetta1*(2*sqrt(J*K));
        tau=1/(zetta1*w);
        if zetta1==0
            tau=0;
        elseif zetta1>1
            tau=1/(abs(max((-zetta1*w-w*sqrt(zetta1^2-1)),(-
zetta1*w+w*sqrt(zetta1^2-1)))));
        end
        Ts=4*tau;
        t=0:0.01:(tau+2*Ts);
        phi=atan(zetta1/(1-zetta1^2));
        y=k*(1-(1/sqrt(1-zetta1^2))*exp(-zetta1*w*t).*cos(w*t*sqrt(1-zetta1^2) -
phi));
    elseif cat=="Critically damped"
         zetta1=1;
         D=zetta1*(2*sqrt(J*K));
         tau=1/(zetta1*w);
         if zetta1==0
            tau=0;
         elseif zetta1>1
            tau=1/(abs(max((-zetta1*w-w*sqrt(zetta1^2-1)),(-
zetta1*w+w*sqrt(zetta1^2-1)))));
         end
         Ts=4*tau;
         t=0:0.01:(tau+2*Ts);
         y=k*(1-(1+w*t).*exp(-w*t));
    elseif cat=="Over damped"
         zetta1=1.5;
         D=zetta1*(2*sqrt(J*K));
         tau=1/(zetta1*w);
         if zetta1==0
            tau=0;
         elseif zetta1>1
            tau=1/(abs(max((-zetta1*w-w*sqrt(zetta1^2-1)),(-
zetta1*w+w*sqrt(zetta1^2-1)))));
         end
         Ts=4*tau;
         t=0:0.01:(tau+2*Ts);
         p1=-zetta1*w-w*sqrt(zetta1^2-1);
         p2=-zetta1*w+w*sqrt(zetta1^2-1);
         y=k*(1 + (-0.5+zetta1/(2*sqrt(zetta1^2-1))).*exp(p1*t)+(-0.5-
zetta1/(2*sqrt(zetta1^2-1))).*exp(p2*t));
```

```
else
          zetta1=0;
          D=zetta1*(2*sqrt(J*K));
          tau=1/(zetta1*w);
          if zetta1==0
             tau=0;
          elseif zetta1>1
             tau=1/(abs(max((-zetta1*w-w*sqrt(zetta1^2-1)),(-
zetta1*w+w*sqrt(zetta1^2-1)))));
          end
          Ts=4*tau;
          t=0:0.01:5;
          y=k*(1-cos(w*t));
    end
    % remaining parameters
    wd=w*sqrt(1-zetta1^2);
    PO=exp(-zetta1*pi/sqrt(1-zetta1^2));
    Tr=1.8/w;
    Tp=pi/wd;

    fprintf(sprintf("%d) %s system\n",i,cat));
    fprintf("Some important parameters are : \n");
    fprintf(sprintf("J=%d, D=%d, K=%d\n",J,D,K));
    fprintf(sprintf("DC Gain = %d\n",k));
    fprintf(sprintf("Damping Ratio = %.2f\n",zetta1));
    fprintf(sprintf("Undamped natural frequency = %.2f\n",w));
    fprintf(sprintf("Damped frequency = %.2f\n",wd));
    fprintf(sprintf("Time Constant = %.4f s\n",tau));
    fprintf(sprintf("Settling Time ~ %.4f s\n",Ts));
    fprintf(sprintf("Rise Time ~ %.4f s\n",Tr));
    if cat=="Under damped"
        fprintf(sprintf("Peak Time ~ %.4f s\n",Tp));
        fprintf(sprintf("Percentage Overshoot ~ %.2f %s \n",PO*100,"%%"));
    end
    fprintf("--------------------\n\n");
    % plotting results
    figure;
    num=1;den=[J,D,K];
    subplot(1,2,1);plot(t,y,'r');
    xlabel('Time');ylabel('System Response');
    title('Response by mathematical formula');
    subplot(1,2,2);step(num,den,t);
    xlabel('Time');ylabel('System Response');
    title('Response by built in step func');
    sgtitle(sprintf("%s",cat));

    if i~=3
    x=input("\n--- Press enter for the next system ---\n\n");
    end
```

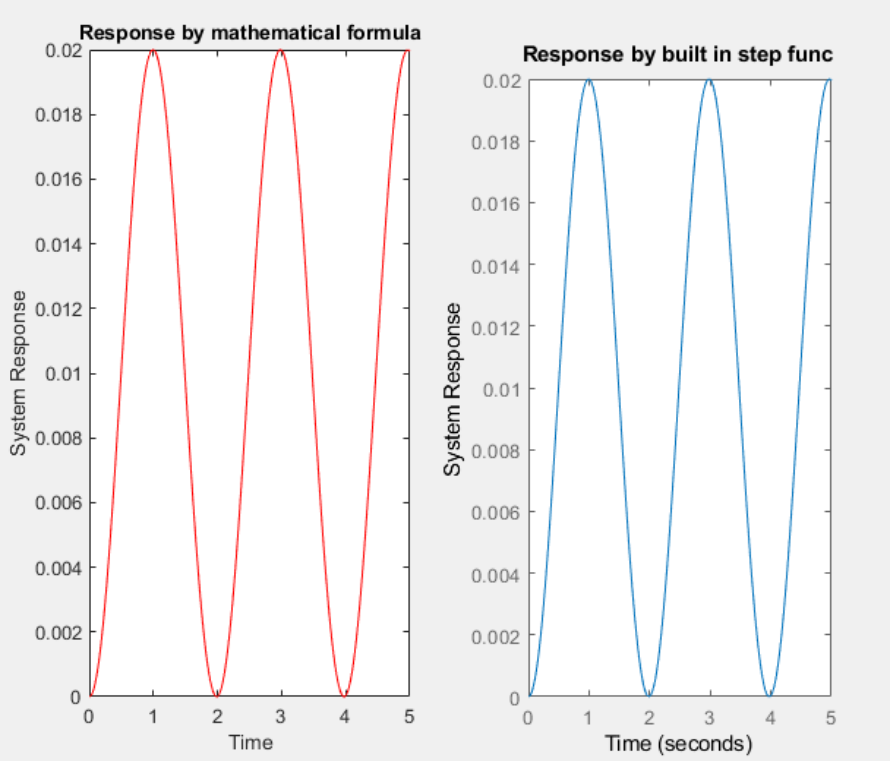
```
    CategInOrder(end+1)=cat;
    dampFreq(end+1)=wd;
    undampFreq(end+1)=w;
    timePer(end+1)=tau;
end
figure;
bar([1,2,3,4],dampFreq,0.5,'FaceColor',[0.2 0.2 0.5]);
hold on
bar([1,2,3,4],undampFreq,0.25,'FaceColor',[0 0.7 0.7]);
xlabel(sprintf("Categories (in order: %s, %s, %s,
%s)",CategInOrder(2),CategInOrder(3),CategInOrder(4),CategInOrder(5)));ylabel("Natur
al Frequencies");
legend("Damped Frequency","Undamped Frequency");
figure;
scatter([1,2,3,4],timePer);
title("Categories vs Time constant");
xlabel(sprintf("Categories (in order: %s, %s, %s,
%s)",CategInOrder(2),CategInOrder(3),CategInOrder(4),CategInOrder(5)));ylabel("Time
Constant");
```

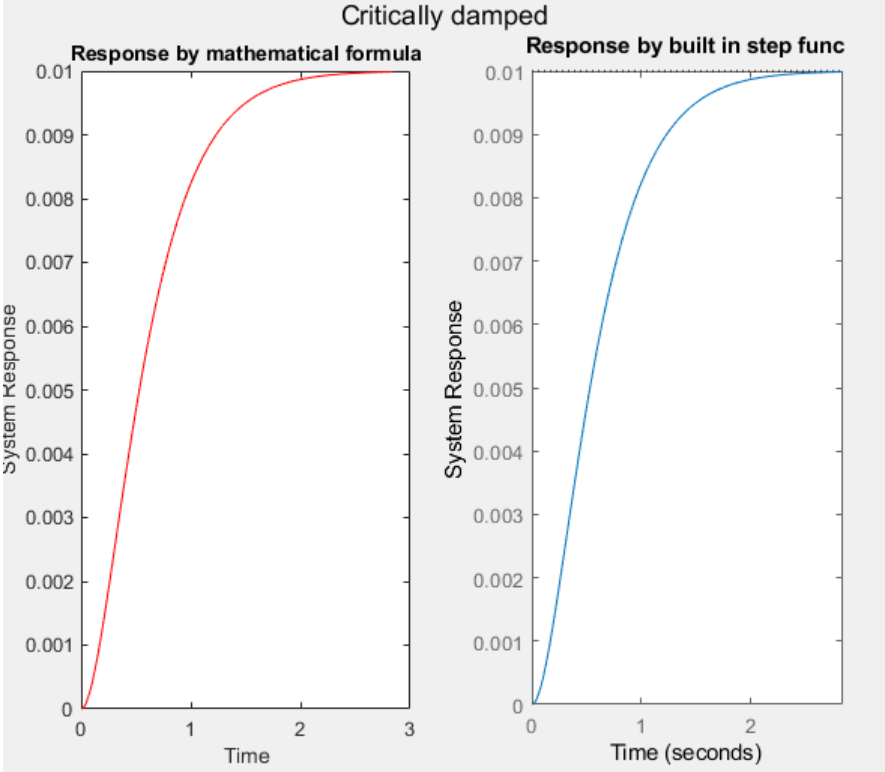
Results:

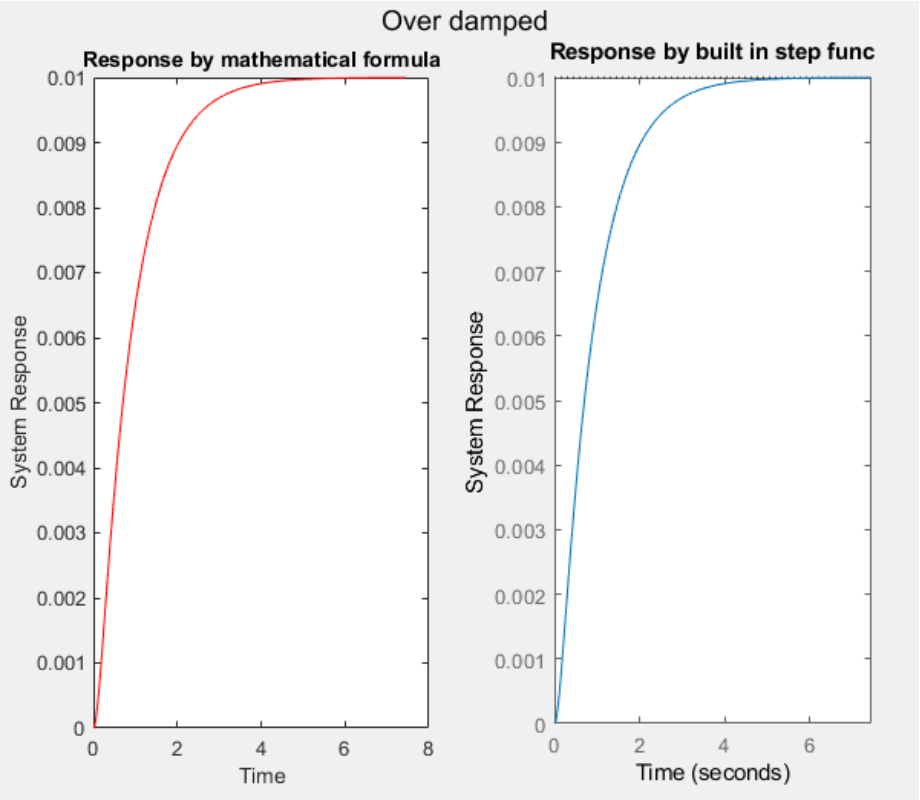
```
----- Analysis of second order system -----
>> The example is of a rotational mass spring damper system.
Please enter the value of Inertia (J, in kg) = 10
Please enter the value of Rotational damper (D, in N-m-s/rad) = 0
Please enter the value of Rotational spring (K, in N-m/rad) = 100

-----------------------------------------------
The transfer function : Theta(s)/T(s) =
     1
-----------
    2
10 s  + 100

Some important parameters are :
DC Gain = 1.000000e-02
Damping Ratio = 0.00
Undamped natural frequency = 3.16
Damped frequency = 3.16
Time Constant = 0.0000 s
Settling Time ~ 0.0000 s
Rise Time ~ 0.5692 s
--> The category of the system is : 'Undamped'


Since you have selected Undamped, following categories have been left off:
1. Under damped 2. Critically damped 3. Over damped

Now type an order in which you'd like to display your system.
For Example, typing [1,2,3] would result in displaying the parameters
of Under damped system, then for Critically damped system
and finally for Over damped system.
Type your desired order >> [2,3,1]
```

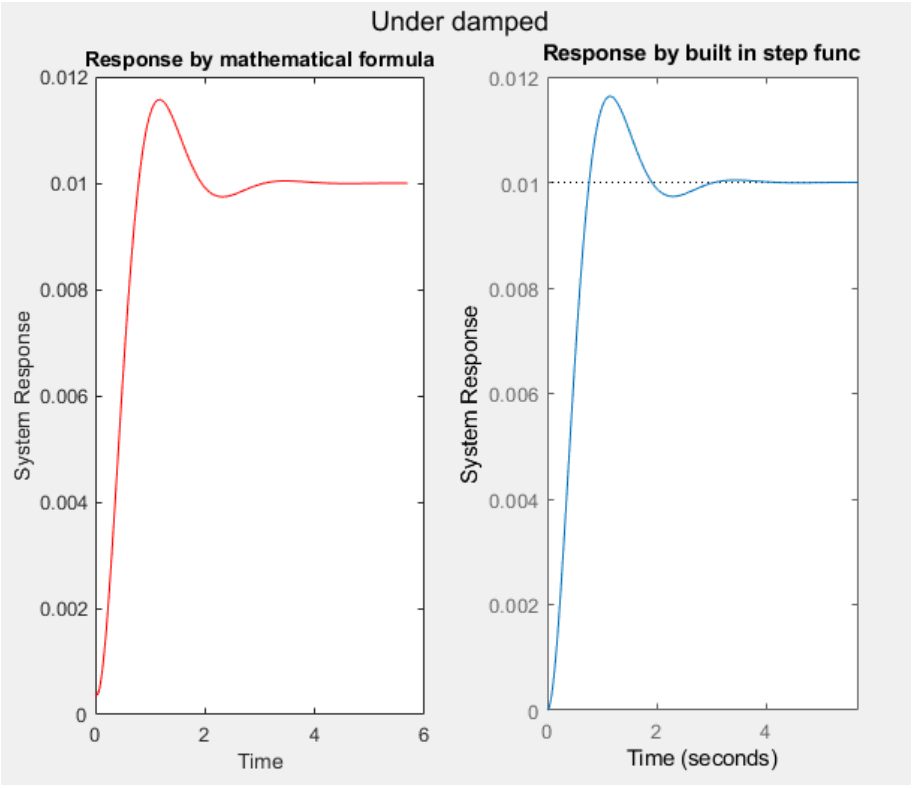
```
1) Critically damped system
Some important parameters are :
J=10, D=6.324555e+01, K=100
DC Gain = 1.000000e-02
Damping Ratio = 1.00
Undamped natural frequency = 3.16
Damped frequency = 0.00
Time Constant = 0.3162 s
Settling Time ~ 1.2649 s
Rise Time ~ 0.5692 s
--------------------


--- Press enter for the next system ---

2) Over damped system
Some important parameters are :
J=10, D=9.486833e+01, K=100
DC Gain = 1.000000e-02
Damping Ratio = 1.50
Undamped natural frequency = 3.16
Damped frequency = 0.00
Time Constant = 0.8279 s
Settling Time ~ 3.3116 s
Rise Time ~ 0.5692 s
--------------------


--- Press enter for the next system ---


3) Under damped system
Some important parameters are :
J=10, D=3.162278e+01, K=100
DC Gain = 1.000000e-02
Damping Ratio = 0.50
Undamped natural frequency = 3.16
Damped frequency = 2.74
Time Constant = 0.6325 s
Settling Time ~ 2.5298 s
Rise Time ~ 0.5692 s
Peak Time ~ 1.1471 s
Percentage Overshoot ~ 16.30 %
--------------------
```

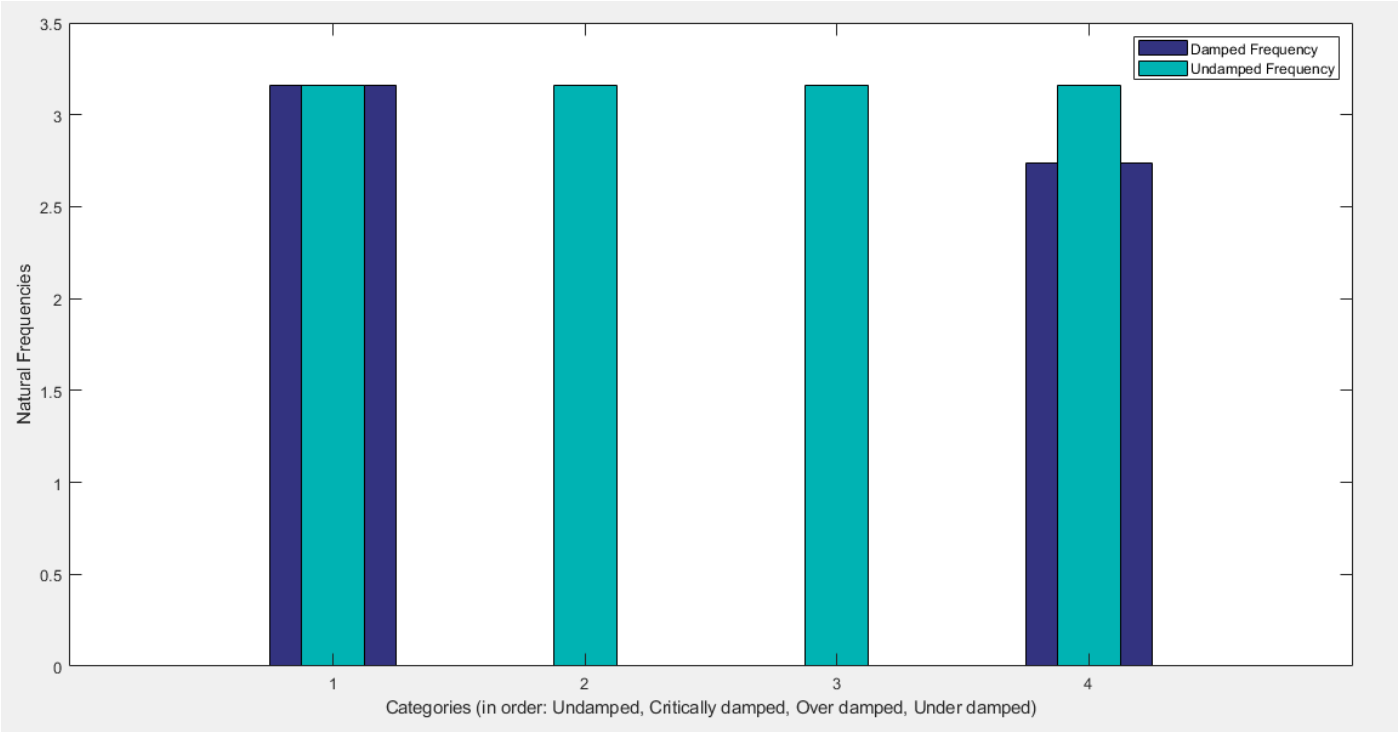
Graphical Results:

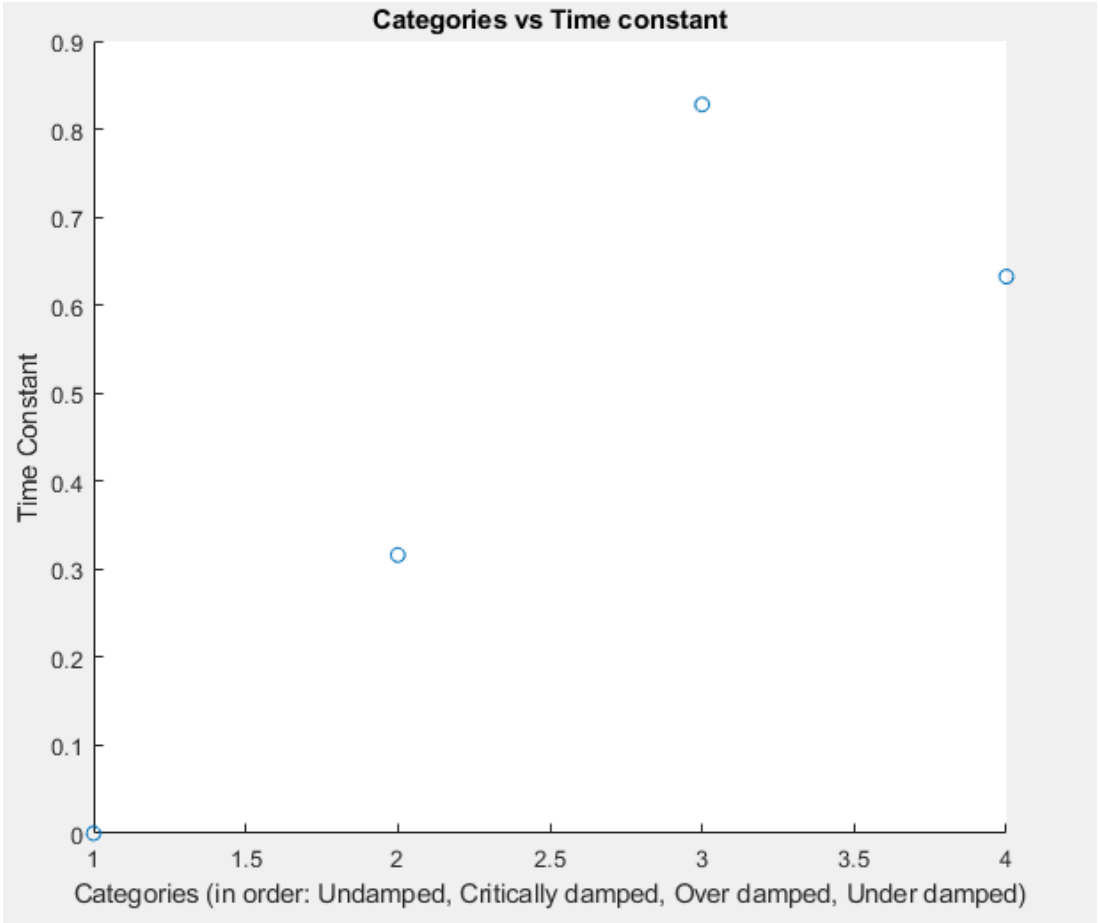





Over damped
Response by mathematical formula
Response by built in step func
System Response
Time
Time (seconds)
0.01
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
0
2
4
6
8


Under damped
Response by mathematical formula
Response by built in step func
System Response
Time
Time (seconds)
0.012
0.01
0.008
0.006
0.004
0.002
0
0
2
4
6


Damped Frequency
Undamped Frequency
Natural Frequencies
0
0.5
1
1.5
2
2.5
3
3.5
1
2
3
4
Categories (in order: Undamped, Critically damped, Over damped, Under damped)


Categories vs Time constant
Time Constant
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.5
2
2.5
3
3.5
4
Categories (in order: Undamped, Critically damped, Over damped, Under damped)

Comments/Remarks:

The first four graphs are quite generic and are correctly cross-validated. The last two tasks are quite interesting. Firstly as undamped natural frequency is independent of zetta or damping so the magnitude of omega n is constant in all the categories. The damped natural frequency on the other hand depends on damping so it is varied throughout i.e., in case of undamped there is no damping so damped frequency is maximum (equal to undamped frequency), in case of under damped system there is some level of damping present which causes the damped frequency to reduce, in case of critically damped system the system damping becomes equal to critical damping (at which there are no oscillations!) and hence damped frequency is zero (similar is the case for over damped system).

In the last graph it can be seen that critically damped is ideal as it reaches 63% of its final value fastest as compared to the other two. Underdamped is also quite better but it is not desirable due to the overshoot.