./

Migration Report – Control Systems



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **To be approved By** | **Remarks/Revision Details** |
| 1 | 14-04-2021 | SOURAV DEY |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

Table of Contents

[Table of Figures 4](#_Toc68797560)

[1. First Order Equation: 5](#_Toc68797561)

[1.1 First Order equation of the Mass Damper: 5](#_Toc68797562)

[Plant Description 5](#_Toc68797563)

[Code: 5](#_Toc68797564)

[Math Analysis 11](#_Toc68797565)

[Comparison Analysis:(Speed, Accuracy and stability) 11](#_Toc68797566)

[1.2 First Order Mass Damper Open Loop: 12](#_Toc68797567)

[Plant Description 12](#_Toc68797568)

[Code: 12](#_Toc68797569)

[Math Analysis 16](#_Toc68797570)

[Comparison Analysis:(Speed, Accuracy and stability): 16](#_Toc68797571)

[1.3 First Order Mass Damper Closed Loop using Controllers: 17](#_Toc68797572)

[Plant Description 17](#_Toc68797573)

[Code: 17](#_Toc68797574)

[Math Analysis 24](#_Toc68797575)

[Comparison Analysis:(Speed, Accuracy and stability): 24](#_Toc68797576)

[2. Second Order Equation. 26](#_Toc68797577)

[2.1 Second Order Mass Spring Damper: 26](#_Toc68797578)

[Second Order MSD Equation 26](#_Toc68797579)

[Plant Description 26](#_Toc68797580)

[Code: 26](#_Toc68797581)

[Math Analysis: 32](#_Toc68797582)

[Comparison Analysis:(Speed, Accuracy and stability): 33](#_Toc68797583)

[2.2 Second order Open loop with controller: 34](#_Toc68797584)

[Second Order MSD Equation 34](#_Toc68797585)

[Plant Description 34](#_Toc68797586)

[Code: 34](#_Toc68797587)

[Math Analysis: 38](#_Toc68797588)

[Comparison Analysis:(Speed, Accuracy and stability): 39](#_Toc68797589)

[2.3 Second Order Spring Mass Damper Close loop using Controller. 40](#_Toc68797590)

[Second Order MSD Equation 40](#_Toc68797591)

[Plant Description 40](#_Toc68797592)

[Code: 40](#_Toc68797593)

[Math Analysis: 47](#_Toc68797594)

[Comparison Analysis:(Speed, Accuracy and stability): 48](#_Toc68797595)

[3. Analysis of First Order and Second Order using PID 49](#_Toc68797596)

[3.PID Analysis 49](#_Toc68797597)

[Second Order System PID Code 49](#_Toc68797598)

[Analysis of Second Order system with PID controllers 50](#_Toc68797599)

[First Order System PID Code 51](#_Toc68797600)

[Analysis of Second Order system with PID controllers 52](#_Toc68797601)

[4. Position of poles with different zeta values 53](#_Toc68797602)

[Roots of the Standard Equation 53](#_Toc68797603)

[Description: 53](#_Toc68797604)

[code 53](#_Toc68797605)

[Comparison Analysis: 58](#_Toc68797606)

[5. Movement of Poles 59](#_Toc68797607)

[Movement of Poles. 59](#_Toc68797608)

[Description: Here the movement of poles is shown along the real and 59](#_Toc68797609)

[Analysis : 60](#_Toc68797610)

[6. Simple Harmonic Motion 61](#_Toc68797611)

[Simple Harmonic Motion Equation 61](#_Toc68797612)

[Plant Description: 61](#_Toc68797613)

[Code: 61](#_Toc68797614)

[Analysis: 68](#_Toc68797615)

[7. Comparison of MATLAB and SciLab 69](#_Toc68797616)

[8. MATLAB 70](#_Toc68797617)

[9. SciLab 72](#_Toc68797618)

[10. Analysis from the scripts: 75](#_Toc68797619)

[11. Simulink Modelling 76](#_Toc68797620)

[12. SciLab Modelling: 77](#_Toc68797621)

[13. Comparison Analysis from the modelling: 78](#_Toc68797622)

## Table of Figures

[Figure 1: Bode plot of the 2nd Order Mass Spring Damper using MATLAB. 71](#_Toc68797623)

[Figure 2: SciLab code for 2nd Order Mass Spring Damper. 72](#_Toc68797624)

[Figure 3: Location of Poles. 73](#_Toc68797625)

[Figure 4: Bode plot of the 2nd Order Mass Spring Damper using SciLab. 74](#_Toc68797626)

[Figure 5: 2nd Order Mass Spring Damper Block diagram using Simulink. 76](#_Toc68797627)

[Figure 6: Time domain response of the 2nd order Mass Spring Damper using Simulink 76](#_Toc68797628)

[Figure 7: 2nd Order Mass Spring Damper Block diagram using SciLab. 77](#_Toc68797629)

[Figure 8: Time domain response of the 2nd order Mass Spring Damper using SciLab. 77](#_Toc68797630)

# First Order Equation:

## 1.1 First Order equation of the Mass Damper:

Plant Description 1

Code: 1

Math Analysis 6

Comparison Analysis:(Speed, Accuracy and stability) 6

%%1(a) First Order Equation  
% Author: Sourav Dey  
% PS Number: 99003785  
% Date: 7th April 2021.  
% Version: Matlab 2020b.

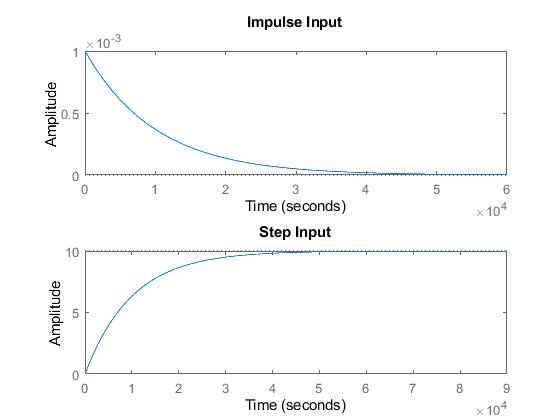
## Plant Description

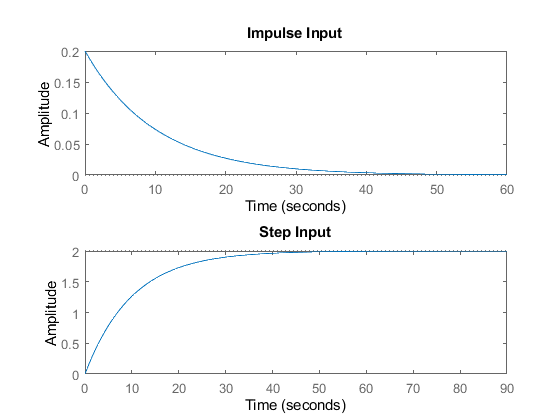
The Mass-damper first order system is taken as Plant. Equation: f= Bv + M v' f = force; B= coefficient of friction; M = mass ; v= velocity. Values: B1= 0.4 M1=1000; B2= 0.5 M2= 500; B3= 1.7 M3= 340;

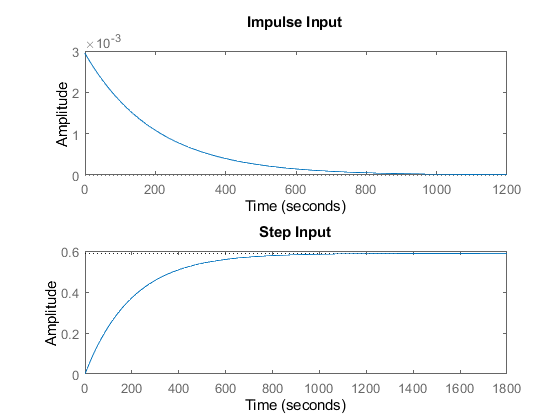
## Code:

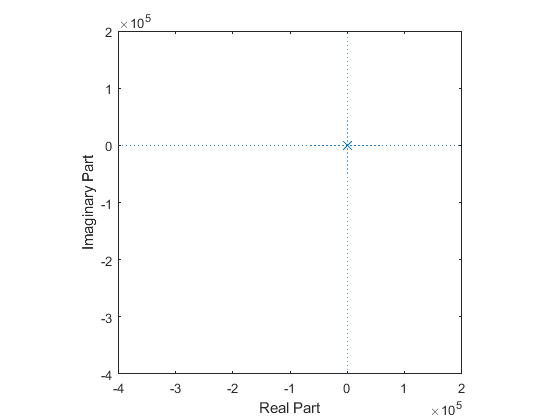
B1= ([0.1 0.5 1.7]);  
M1=([1000 5 340]);  
for i=1:3  
 sys = tf([1/M1(i)],[1,B1(i)/M1(i)])  
 figure(i);  
 subplot(2,1,1);  
 impulse(sys);  
 title('Impulse Input');  
 subplot(2,1,2);  
 step(sys);  
 title('Step Input');  
 [z,p,k]= tf2zp([1/M1(i)],[1,B1(i)/M1(i)])  
 figure(4);  
 zplane(z,p);  
 xlim([-4\*1e5 2\*1e5]);  
 ylim([-4\*1e5 2\*1e5]);  
 hold on;  
 S = stepinfo(sys)  
end

sys =  
   
 0.001  
 ----------  
 s + 0.0001  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -1.0000e-04  
  
  
k =  
  
 1.0000e-03  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 2.1970e+04  
 SettlingTime: 3.9121e+04  
 SettlingMin: 9.0450  
 SettlingMax: 9.9997  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 9.9997  
 PeakTime: 1.0546e+05  
  
  
sys =  
   
 0.2  
 -------  
 s + 0.1  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.1000  
  
  
k =  
  
 0.2000  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 21.9701  
 SettlingTime: 39.1207  
 SettlingMin: 1.8090  
 SettlingMax: 1.9999  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 1.9999  
 PeakTime: 105.4584  
  
  
sys =  
   
 0.002941  
 ---------  
 s + 0.005  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.0050  
  
  
k =  
  
 0.0029  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 439.4013  
 SettlingTime: 782.4149  
 SettlingMin: 0.5321  
 SettlingMax: 0.5882  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 0.5882  
 PeakTime: 2.1092e+03









## Math Analysis

Independent: Time(t) Dependent: Velocity(v) and Force(f) Constant: Mass(M) and Frictional Coefficient(B)

% Roots:(-B)/M  
  
% IVT:  
% 1. For step input: 0  
% 2. For impulse input: 1/M  
  
% FVT:  
% 1. For step input: 1/B  
% 2. For impulse input: 0  
  
% Time Response Results:  
% Rise Time :4tau = (4M)/B; where tau = M/B

## Comparison Analysis:(Speed, Accuracy and stability)

This is a first order system as we have only one pole which is present in the left hand side of the s-plane. Thus we can say the system is stable. The location of the pole determines the stability. Here, we have three different systems sys, sys1, sys2. The rise time system sys1 is less than other two. The settling time of the sys1 is also less which detemines the sys1 goes to stability is greater speed than other systems. From the above we can say that the second system or sys1 is most stable and has the highest speed. As we are dealing with first order systems it won't have any overshoots.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

## First Order Mass Damper Open Loop:

Plant Description 1

Code: 1

Math Analysis 5

Comparison Analysis:(Speed, Accuracy and stability): 5

%%1(b) First Order Equation  
% Author: Sourav Dey  
% PS Number: 99003785  
% Date: 7th April 2021.  
% Version: Matlab 2020b.

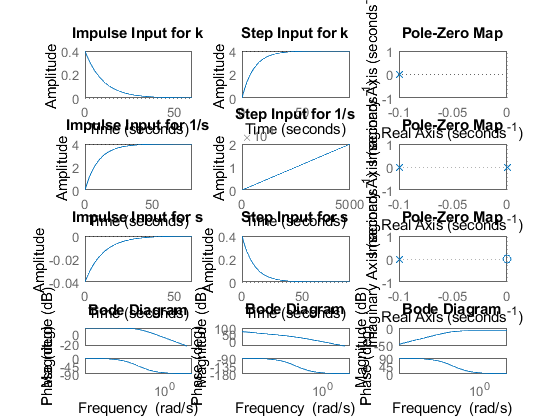
## Plant Description

The Mass-damper first order system is taken as Plant. Equation: f= Bv + M v' f = force; B= coefficient of friction; M = mass ; v= velocity. % Values: B1= 0.5 M1= 5; Here we have used three controllers i.e. Proportional(P), Integrator(1/s) Differentiator(s).

## Code:

clc;  
B1= 0.5;  
M1= 5;  
P = 2;  
  
 sys = tf([P/M1],[1,B1/M1])  
 subplot(4,3,1);  
 impulse(sys);  
 title('Impulse Input for k');  
 subplot(4,3,2);  
 step(sys);  
 title('Step Input for k');  
 subplot(4,3,3);  
 [z,p,k]= tf2zp([P/M1],[1,B1/M1])  
 pzmap(sys)  
 subplot(4,3,10);  
 bode(sys)  
  
 hold on;  
 S = stepinfo(sys)  
  
sys = tf([P/M1],[1,B1/M1,0])  
subplot(4,3,4);  
impulse(sys);  
title('Impulse Input for 1/s');  
subplot(4,3,5);  
step(sys);  
title('Step Input for 1/s');  
subplot(4,3,6);  
[z,p,k]= tf2zp([P/M1],[1,B1/M1,0])  
pzmap(sys)  
 subplot(4,3,11);  
 bode(sys)  
hold on;  
S = stepinfo(sys)  
sys = tf([P/M1,0],[1,B1/M1])  
subplot(4,3,7);  
impulse(sys);  
title('Impulse Input for s');  
subplot(4,3,8);  
step(sys);  
title('Step Input for s');  
subplot(4,3,9);  
[z,p,k]= tf2zp([P/M1,0],[1,B1/M1])  
pzmap(sys)  
 subplot(4,3,12);  
 bode(sys)  
hold on;  
S = stepinfo(sys)

sys =  
   
 0.4  
 -------  
 s + 0.1  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.1000  
  
  
k =  
  
 0.4000  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 21.9701  
 SettlingTime: 39.1207  
 SettlingMin: 3.6180  
 SettlingMax: 3.9999  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 3.9999  
 PeakTime: 105.4584  
  
  
sys =  
   
 0.4  
 -----------  
 s^2 + 0.1 s  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 0  
 -0.1000  
  
  
k =  
  
 0.4000  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf  
  
  
sys =  
   
 0.4 s  
 -------  
 s + 0.1  
   
Continuous-time transfer function.  
  
  
z =  
  
 0  
  
  
p =  
  
 -0.1000  
  
  
k =  
  
 0.4000  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 21.9701  
 SettlingTime: 39.1207  
 SettlingMin: 1.0521e-05  
 SettlingMax: 0.0382  
 Overshoot: 0  
 Undershoot: 7.2058e+17  
 Peak: 0.4000  
 PeakTime: 0



## Math Analysis

Independent: Time(t) Dependent: Velocity(v) and Force(f) Constant: Mass(M) and Frictional Coefficient(B)

% Roots:(-B)/M  
  
% IVT:  
% 1. For step input: 0  
% 2. For impulse input: 1/M  
  
% FVT:  
% 1. For step input: 1/B  
% 2. For impulse input: 0  
  
% Time Response Results:  
% Rise Time :4tau = (4M)/B; where tau = M/B

## Comparison Analysis:(Speed, Accuracy and stability):

First we are taking Proportional controller. From the graph we can say that the system is stable as the poles are on the left hand side of the s-plane. The proportional controller only increases the amplitude and does not affects any other aspect of a system.

% Second we are taking an integrator.An integrator adds a pole to  
% the system at origin. Thus the stability of the system changes from  
% stable to marginal stable. Use of integrator also reduces the  
% steady-state error.  
  
% Third we are taking an differentiator controller. A differentiator adds a  
% zero at origin. From the graph we can say that pole is present in left  
% hand side of s-plane.Zero makes a system from unstable to stable.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

## First Order Mass Damper Closed Loop using Controllers:

Plant Description 1

Code: 1

Math Analysis 8

Comparison Analysis:(Speed, Accuracy and stability): 8

%%1(c) First Order Equation with closed loop  
% Author: Sourav Dey  
% PS Number: 99003785  
% Date: 7th April 2021.  
% Version: Matlab 2020b.

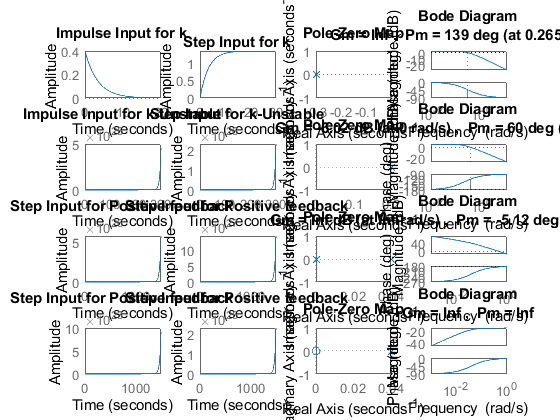
## Plant Description

The Mass-damper first order system is taken as Plant. Equation: f= Bv + M v' f = force; B= coefficient of friction; M = mass ; v= velocity.

## Code:

%Negative Feedback using gain input  
clc;  
B1= 0.5;  
M1= 5;  
P = 2;  
  
sys = tf([P],[M1,B1+1])  
subplot(4,4,1);  
impulse(sys);  
title('Impulse Input for k');  
subplot(4,4,2);  
step(sys);  
title('Step Input for k');  
subplot(4,4,3);  
[z,p,k]= tf2zp([P],[M1,B1+1])  
pzmap(sys)  
subplot(4,4,4)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
hold on;  
S = stepinfo(sys)  
  
B2= -2;  
M2= 5;  
P2 = 2;  
  
sys = tf([P2],[M2,B2+1])  
subplot(4,4,5);  
impulse(sys);  
title('Impulse Input for k-Unstable');  
subplot(4,4,6);  
step(sys);  
title('Step Input for k-Unstable');  
subplot(4,4,7);  
[z,p,k]= tf2zp([P2],[M2,B2+1])  
pzmap(sys)  
subplot(4,4,8)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
hold on;  
S = stepinfo(sys)  
  
%Positive Feedback using integral input  
B3= 0.8;  
M3= 5;  
  
  
sys = tf([1],[M3,B3-1,0])  
subplot(4,4,9);  
impulse(sys);  
title('Step Input for Positive feedback');  
subplot(4,4,10);  
step(sys);  
title('Step Input for Positive feedback');  
subplot(4,4,11);  
[z,p,k]= tf2zp([1],[M3,B3-1,0])  
pzmap(sys)  
subplot(4,4,12)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
hold on;  
S = stepinfo(sys)  
  
%Positive Feedback using differentiator input  
B4= 0.8;  
M4= 5;  
  
  
sys = tf([1,0],[M4,B4-1])  
subplot(4,4,13);  
impulse(sys);  
title('Step Input for Positive feedback');  
subplot(4,4,14);  
step(sys);  
title('Step Input for Positive feedback');  
subplot(4,4,15);  
[z,p,k]= tf2zp([1,0],[M4,B4-1])  
pzmap(sys)  
subplot(4,4,16)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
hold on;  
S = stepinfo(sys)

sys =  
   
 2  
 ---------  
 5 s + 1.5  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.3000  
  
  
k =  
  
 0.4000  
  
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 138.5925  
  
  
Wcg =  
  
 NaN  
  
  
Wcp =  
  
 0.2646  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 7.3234  
 SettlingTime: 13.0402  
 SettlingMin: 1.2060  
 SettlingMax: 1.3333  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 1.3333  
 PeakTime: 35.1528  
  
  
sys =  
   
 2  
 -------  
 5 s - 1  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 0.2000  
  
  
k =  
  
 0.4000  
  
  
Gm =  
  
 0.5000  
  
  
Pm =  
  
 59.9993  
  
  
Wcg =  
  
 0  
  
  
Wcp =  
  
 0.3464  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf  
  
  
sys =  
   
 1  
 -------------  
 5 s^2 - 0.2 s  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 0  
 0.0400  
  
  
k =  
  
 0.2000  
  
Warning: The closed-loop system is unstable.   
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 -5.1214  
  
  
Wcg =  
  
 Inf  
  
  
Wcp =  
  
 0.4463  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf  
  
  
sys =  
   
 s  
 ---------  
 5 s - 0.2  
   
Continuous-time transfer function.  
  
  
z =  
  
 0  
  
  
p =  
  
 0.0400  
  
  
k =  
  
 0.2000  
  
Warning: The closed-loop system is unstable.   
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 Inf  
  
  
Wcg =  
  
 NaN  
  
  
Wcp =  
  
 NaN  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf



## Math Analysis

Independent: Time(t) Dependent: Velocity(v) and Force(f) Constant: Mass(M) and Frictional Coefficient(B)

% Roots:(-B)/M  
% T.F. = G/1-G for -ve feedback where G is open loop T.F.  
% T.F. = G/1+G for +ve feedback where G is open loop T.F.  
% Here we take unity feedback.

## Comparison Analysis:(Speed, Accuracy and stability):

First we are analysing the system with negative feedback and p controller. Here, rise time and settling time of the system is reduced thus the system becomes more stable and more fast. We can also say that the system is reaching stability at a faster rate. Here the P controller increases the amplitude also. From the frequency response we can say that the T.F is stable.

% From the second graph we can say that the poles are on the left hand side  
% so the system is unstable. We can also validate by seeing the frequency  
% reponse as GM is negative, so the system is unstable.  
  
% Now we are analysing the system with positive feedback.  
% Here we are using two controllers a First Integrator and then  
% Differentiator.  
  
% For Integrator with positive feedback the stability of the system changes  
% from stable to unstable. The location of the poles are also on the right  
% hand side of the s-plane.  
% Here the phase margin is negative which says that the system is unstable.  
  
% For differentiator with positive feedback the system becomes unstable and  
% poles are also present in the right hand side of the s-plane.  
% From the frequency response we can see that the PM and GM becomes  
% infinity which implies that the system is unable to accept signals  
% without errors. Thus more errors get accumulated and the system becomes  
% unstable. This is an exception case.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

# Second Order Equation.

## 2.1 Second Order Mass Spring Damper:

# Second Order MSD Equation

Author: Sourav Dey PS Number: 99003785 Date: 7th April 2021. Version: Matlab 2020b.

Plant Description 1

Code: 1

Math Analysis: 6

Comparison Analysis:(Speed, Accuracy and stability): 7

## Plant Description

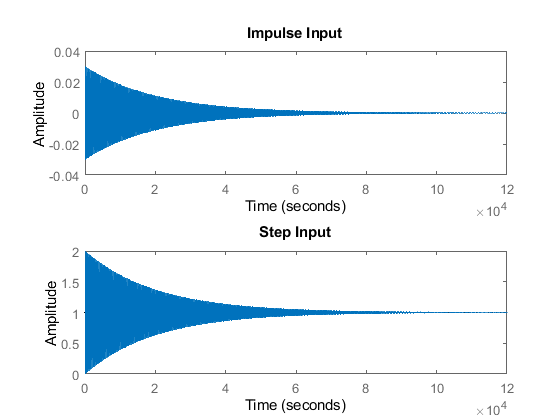
The Mass-damper Spring Second order system is taken as Plant. It is used in as suspension.

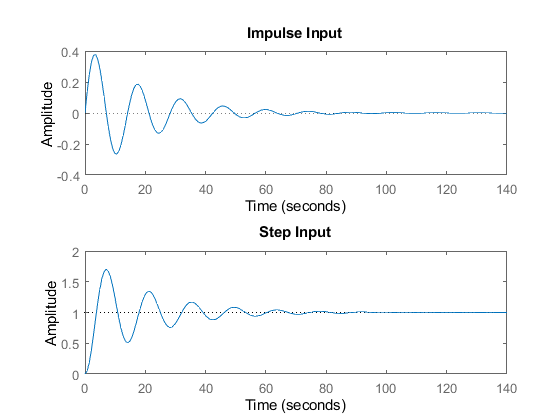
% Equation: Mx''(t)+ Bx'(t) + Kx(t)= Kf(t).  
% f = force; B= coefficient of friction; M = mass ; v= velocity; k=spring  
% constant.  
% Values: K1= 0.1 B1= 0.4 M1=1000 Wn=0.03 ; K2= 1 B2= 0.5 M2= 500 Wn=0.44;  
% K3= 3 B3= 1.7 M3= 340 Wn=0.09;

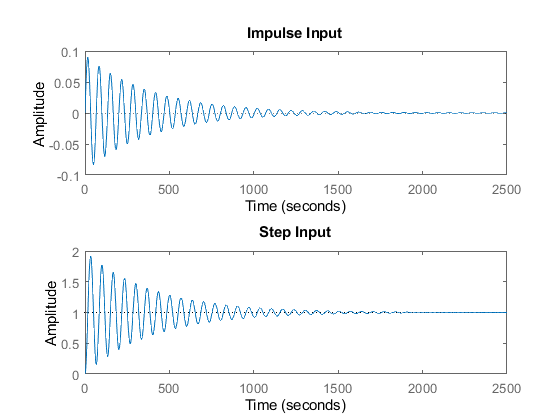
## Code:

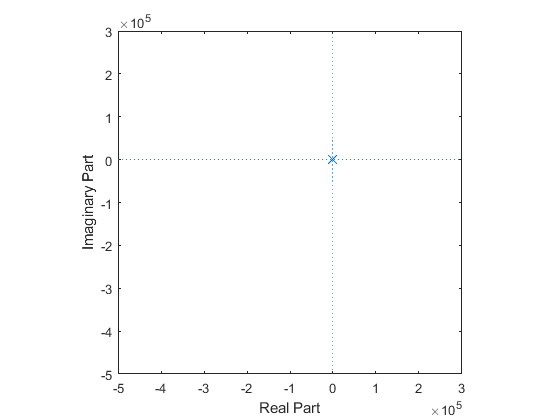
B1= ([0.1 0.5 1.7]);  
M1=([1000 5 340]);M1=([1000 5 340]);  
K1 = ([0.9 1 3]);  
for i=1:3  
 sys = tf([K1(i)/M1(i)],[1,B1(i)/M1(i),K1(i)/M1(i)])  
 figure(i);  
 subplot(2,1,1);  
 impulse(sys);  
 title('Impulse Input');  
 subplot(2,1,2);  
 step(sys);  
 title('Step Input');  
 [z,p,k]= tf2zp([K1(i)/M1(i)],[1,B1(i)/M1(i),K1(i)/M1(i)])  
 figure(4);  
 zplane(z,p);  
 xlim([-5\*1e5 3\*1e5]);  
 ylim([-5\*1e5 3\*1e5]);  
 hold on;  
 S = stepinfo(sys)  
end

sys =  
   
 0.0009  
 -----------------------  
 s^2 + 0.0001 s + 0.0009  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.0001 + 0.0300i  
 -0.0001 - 0.0300i  
  
  
k =  
  
 9.0000e-04  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 34.7791  
 SettlingTime: 7.8226e+04  
 SettlingMin: 0.0104  
 SettlingMax: 1.9948  
 Overshoot: 99.4778  
 Undershoot: 0  
 Peak: 1.9948  
 PeakTime: 104.7198  
  
  
sys =  
   
 0.2  
 -----------------  
 s^2 + 0.1 s + 0.2  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.0500 + 0.4444i  
 -0.0500 - 0.4444i  
  
  
k =  
  
 0.2000  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 2.5448  
 SettlingTime: 78.1524  
 SettlingMin: 0.5072  
 SettlingMax: 1.7021  
 Overshoot: 70.2118  
 Undershoot: 0  
 Peak: 1.7021  
 PeakTime: 7.0248  
  
  
sys =  
   
 0.008824  
 ------------------------  
 s^2 + 0.005 s + 0.008824  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.0025 + 0.0939i  
 -0.0025 - 0.0939i  
  
  
k =  
  
 0.0088  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 11.3230  
 SettlingTime: 1.5426e+03  
 SettlingMin: 0.1540  
 SettlingMax: 1.9198  
 Overshoot: 91.9760  
 Undershoot: 0  
 Peak: 1.9198  
 PeakTime: 33.4448









## Math Analysis:

Independent: Time(t) Dependent: Velocity(v) and Force(f) Constant: Mass(M), Frictional Coefficient(B), Spring constant(K)

% Roots:((-B/M)+-sqrt(sq(B/M)-4K/M))/2  
  
% Time Response Results:  
% K1= 0.9 B1= 0.4 M1=1000  
% Rise Time :34.77  
% settling time:7.82e+04  
% Overshoot:99.47  
% Undershoot:0  
% PeakTime:104.71  
  
%K2= 1 B2= 0.5 M2= 500  
% Rise Time :2.54  
% settling time:78.15  
% Overshoot:70.21  
% Undershoot:0  
% PeakTime:7.024  
  
%K3= 3 B3= 1.7 M3= 340  
% Rise Time :11.32  
% settling time:1.54e+03  
% Overshoot:91.97  
% Undershoot:0  
% PeakTime:33.44

## Comparison Analysis:(Speed, Accuracy and stability):

For the first system, the poles are present on the left-hand side of the

% s-plane making it stable.  
  
% For the second system the poles are present on the L.H.S of the s-plane  
% making the system stable.  
  
% For the third system the poles are present on thr L.H.S of the s-plane  
% making the system stable .  
  
% Among the three systems second system has the least rising time and  
% settling time making it most stable and highest speed.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

## 2.2 Second order Open loop with controller:

# Second Order MSD Equation

Author: Sourav Dey PS Number: 99003785 Date: 7th April 2021. Version: Matlab 2020b.

Plant Description 1

Code: 1

Math Analysis: 5

Comparison Analysis:(Speed, Accuracy and stability): 6

## Plant Description

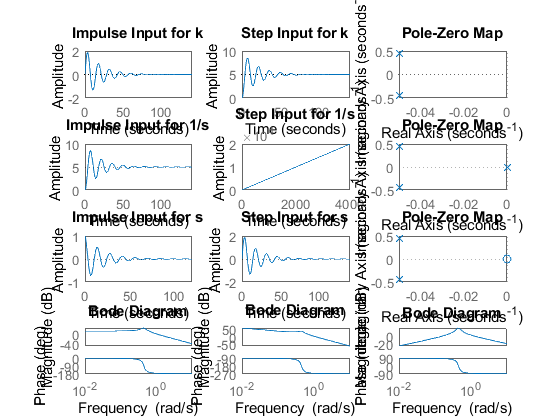
The Mass-damper Spring Second order system is taken as Plant. It is used in as suspension.

% Equation: Mx''(t)+ Bx'(t) + Kx(t)= Kf(t).  
% f = force; B= coefficient of friction; M = mass ; v= velocity;  
% k=spring constant.  
% Here we have used three controllers i.e. Proportional(P), Integrator(1/s),  
% Differentiator(s).  
% Values: K1= 1 B1= 0.5 M1= 500 Wn=0.44;

## Code:

B1= 0.5  
M1= 5;  
K1 =1;  
P=5;  
sys = tf([P\*K1/M1],[1,B1/M1,K1/M1])  
subplot(4,3,1);  
impulse(sys);  
title('Impulse Input for k');  
subplot(4,3,2);  
step(sys);  
title('Step Input for k');  
subplot(4,3,3);  
[z,p,k]= tf2zp([P\*K1/M1],[1,B1/M1,K1/M1])  
pzmap(sys)  
subplot(4,3,10);  
bode(sys)  
hold on;  
S = stepinfo(sys)  
  
sys = tf([P\*K1/M1],[1,B1/M1,K1/M1,0])  
subplot(4,3,4);  
impulse(sys);  
title('Impulse Input for 1/s');  
subplot(4,3,5);  
step(sys);  
title('Step Input for 1/s');  
subplot(4,3,6);  
[z,p,k]= tf2zp([P\*K1/M1],[1,B1/M1,K1/M1,0])  
pzmap(sys)  
subplot(4,3,11);  
bode(sys)  
hold on;  
S = stepinfo(sys)  
  
sys = tf([P\*K1/M1,0],[1,B1/M1,K1/M1])  
subplot(4,3,7);  
impulse(sys);  
title('Impulse Input for s');  
subplot(4,3,8);  
step(sys);  
title('Step Input for s');  
subplot(4,3,9);  
[z,p,k]= tf2zp([P\*K1/M1,0],[1,B1/M1,K1/M1])  
pzmap(sys)  
subplot(4,3,12);  
bode(sys)  
hold on;  
S = stepinfo(sys)

B1 =  
  
 0.5000  
  
  
sys =  
   
 1  
 -----------------  
 s^2 + 0.1 s + 0.2  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.0500 + 0.4444i  
 -0.0500 - 0.4444i  
  
  
k =  
  
 1  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 2.5448  
 SettlingTime: 78.1524  
 SettlingMin: 2.5361  
 SettlingMax: 8.5106  
 Overshoot: 70.2118  
 Undershoot: 0  
 Peak: 8.5106  
 PeakTime: 7.0248  
  
  
sys =  
   
 1  
 ---------------------  
 s^3 + 0.1 s^2 + 0.2 s  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 0.0000 + 0.0000i  
 -0.0500 + 0.4444i  
 -0.0500 - 0.4444i  
  
  
k =  
  
 1  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf  
  
  
sys =  
   
 s  
 -----------------  
 s^2 + 0.1 s + 0.2  
   
Continuous-time transfer function.  
  
  
z =  
  
 0  
  
  
p =  
  
 -0.0500 + 0.4444i  
 -0.0500 - 0.4444i  
  
  
k =  
  
 1  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 0  
 SettlingTime: 81.5509  
 SettlingMin: -1.3280  
 SettlingMax: 1.8877  
 Overshoot: Inf  
 Undershoot: Inf  
 Peak: 1.8877  
 PeakTime: 3.5124



## Math Analysis:

Independent: Time(t) Dependent: Velocity(v) and Force(f) Constant: Mass(M), Frictional Coefficient(B), Spring constant(K)

% Roots:((-B/M)+-sqrt(sq(B/M)-4K/M))/2  
  
% Poles and Zero Calculation:  
  
  
% IVT:  
% 1. For step input: 0  
% 2. For impulse input: 0  
  
% FVT:  
% 1. For step input: 1  
% 2. For impulse input: K/M  
  
% Time Response Results:  
% K1= 0.9 B1= 0.4 M1=1000  
% Rise Time :  
% settling time:  
% Overshoot:  
% Undershoot:  
% PeakTime:  
  
%K2= 1 B2= 0.5 M2= 500  
% Rise Time :  
% settling time:  
% Overshoot:  
% Undershoot:  
% PeakTime:  
  
%K3= 3 B3= 1.7 M3= 340  
% Rise Time :  
% settling time:  
% Overshoot:  
% Undershoot:  
% PeakTime:

## Comparison Analysis:(Speed, Accuracy and stability):

When add a P controller only the amplitude increases rest all remains same as open loop without controller.

% When we add a integrator then a pole is added in origin which makes the  
% system from stable to marginally stable.  
  
% When we add a differentiator to the system a zero is added in origin  
% which makes a stable system unstable.The overshoot also increrases.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

## 2.3 Second Order Spring Mass Damper Close loop using Controller.

# Second Order MSD Equation

Author: Sourav Dey PS Number: 99003785 Date: 7th April 2021. Version: Matlab 2020b.

Plant Description 1

Code: 1

Math Analysis: 8

Comparison Analysis:(Speed, Accuracy and stability): 8

## Plant Description

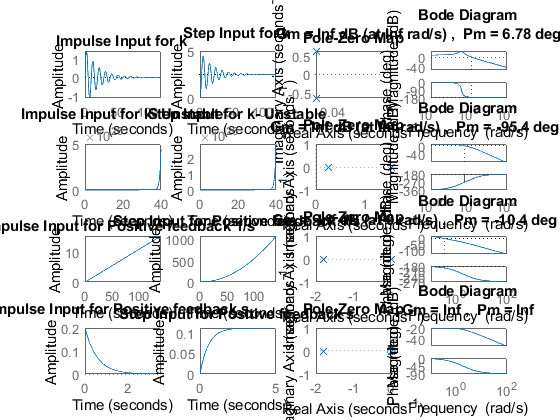
The Mass-damper Spring Second order system is taken as Plant. It is used in as suspension.

% Equation: Mx''(t)+ Bx'(t) + Kx(t)= Kf(t).  
% f = force; B= coefficient of friction; M = mass ; v= velocity;  
% k=spring constant.  
% Values: K=1 B= 0.5 M= 500 P=5 Wn=0.44;

## Code:

%For negative feedback  
B1= 0.5  
M1= 5;  
K1 =1;  
P=5;  
  
sys = tf([P\*K1],[M1,B1,2\*K1])  
subplot(4,4,1);  
impulse(sys);  
title('Impulse Input for k');  
subplot(4,4,2);  
step(sys);  
title('Step Input for k');  
subplot(4,4,3);  
[z,p,k]= tf2zp([P\*K1],[M1,B1,2\*K1])  
pzmap(sys)  
subplot(4,4,4)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
  
hold on;  
S = stepinfo(sys)  
  
B2= -9  
M2= 5;  
K2=1;  
P2=5;  
  
sys = tf([P2\*K2],[M2,B2,2\*K2])  
subplot(4,4,5);  
impulse(sys);  
title('Impulse Input for k- Unstable');  
subplot(4,4,6);  
step(sys);  
title('Step Input for k- Unstable');  
subplot(4,4,7);  
[z,p,k]= tf2zp([P2\*K2],[M2,B2,2\*K2])  
pzmap(sys)  
subplot(4,4,8)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
hold on;  
S = stepinfo(sys)  
  
% For Positive feedback using I & D  
  
B3= 9  
M3= 5;  
K3=1;  
  
  
sys = tf([K3],[M3,B3,0,0])  
subplot(4,4,9);  
impulse(sys);  
title('Impulse Input for Positive feedback 1/s ');  
subplot(4,4,10);  
step(sys);  
title('Step Input for Positive feedback 1/s');  
subplot(4,4,11);  
[z,p,k]= tf2zp([K3],[M3,B3,0,0])  
pzmap(sys)  
subplot(4,4,12)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
hold on;  
S = stepinfo(sys)  
  
  
B4= 9  
M4= 5;  
K4=1;  
  
  
sys = tf([K4,0],[M4,B4,0])  
subplot(4,4,13);  
impulse(sys);  
title('Impulse Input for Positive feedback s ');  
subplot(4,4,14);  
step(sys);  
title('Step Input for Positive feedback s');  
subplot(4,4,15);  
[z,p,k]= tf2zp([K4,0],[M4,B4,0])  
pzmap(sys)  
subplot(4,4,16)  
bode(sys)  
margin(sys)  
[Gm,Pm,Wcg,Wcp] = margin(sys)  
hold on;  
S = stepinfo(sys)

B1 =  
  
 0.5000  
  
  
sys =  
   
 5  
 -----------------  
 5 s^2 + 0.5 s + 2  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.0500 + 0.6305i  
 -0.0500 - 0.6305i  
  
  
k =  
  
 1  
  
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 6.7782  
  
  
Wcg =  
  
 Inf  
  
  
Wcp =  
  
 1.1803  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 1.7526  
 SettlingTime: 75.6433  
 SettlingMin: 0.9814  
 SettlingMax: 4.4486  
 Overshoot: 77.9429  
 Undershoot: 0  
 Peak: 4.4486  
 PeakTime: 4.9673  
  
  
B2 =  
  
 -9  
  
  
sys =  
   
 5  
 ---------------  
 5 s^2 - 9 s + 2  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 1.5403  
 0.2597  
  
  
k =  
  
 1  
  
Warning: The closed-loop system is unstable.   
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 -95.4008  
  
  
Wcg =  
  
 Inf  
  
  
Wcp =  
  
 0.5531  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf  
  
  
B3 =  
  
 9  
  
  
sys =  
   
 1  
 -------------  
 5 s^3 + 9 s^2  
   
Continuous-time transfer function.  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 0  
 0  
 -1.8000  
  
  
k =  
  
 0.2000  
  
Warning: The closed-loop system is unstable.   
  
Gm =  
  
 0  
  
  
Pm =  
  
 -10.4065  
  
  
Wcg =  
  
 0  
  
  
Wcp =  
  
 0.3306  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf  
  
  
B4 =  
  
 9  
  
  
sys =  
   
 s  
 -----------  
 5 s^2 + 9 s  
   
Continuous-time transfer function.  
  
  
z =  
  
 0  
  
  
p =  
  
 0  
 -1.8000  
  
  
k =  
  
 0.2000  
  
Warning: The closed-loop system is unstable.   
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 Inf  
  
  
Wcg =  
  
 NaN  
  
  
Wcp =  
  
 NaN  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 1.2206  
 SettlingTime: 2.1734  
 SettlingMin: 0.1005  
 SettlingMax: 0.1111  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 0.1111  
 PeakTime: 5.8588



## Math Analysis:

Independent: Time(t) Dependent: Velocity(v) and Force(f) Constant: Mass(M), Frictional Coefficient(B), Spring constant(K)

% Roots:((-B/M)+-sqrt(sq(B/M)-4K/M))/2  
  
% T.F. = G/1-G for -ve feedback where G is open loop T.F.  
% T.F. = G/1+G for +ve feedback where G is open loop T.F.  
% Here we take unity feedback.

## Comparison Analysis:(Speed, Accuracy and stability):

Negative Feedback When we use a negative feedback the system becomes stable at a faster rate as the rise time and settling time decrease. From the first graph it can be validated by seeing the location of the poles.

% From the second graph it can be determined that the system is unstable  
% and it can be validated by the locaation of the poles and phase margin is  
% negative.  
  
% When we use a positive feedback with I controller it makes the stable  
% system marginally stable as the error never crosses the threshold value.  
% Poles of the system is present on L.H.S of the S-plane but there is a  
% pole present in origin.  
% The GM and PM is also negative which says the system is reaching  
% unstability.  
  
% When we use a positive feedback with D controller Zero is added in  
% origin which pulls the poles near origin.  
% The stabilty of the system reduces i.e it becomes unstable.  
  
% From the frequency response we can say that the system is unstable as PM  
% and GM both is infinity. When GM becomes infinity the system cannot  
% accept signals without errors thus more errors get accumulated and system  
% becomes unstable.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

## 

# Analysis of First Order and Second Order using PID

# 3.PID Analysis

Author: Sourav Dey PS Number: 99003785 Date: 7th April 2021. Version: Matlab 2020b.

Second Order System PID Code 1

Analysis of Second Order system with PID controllers 2

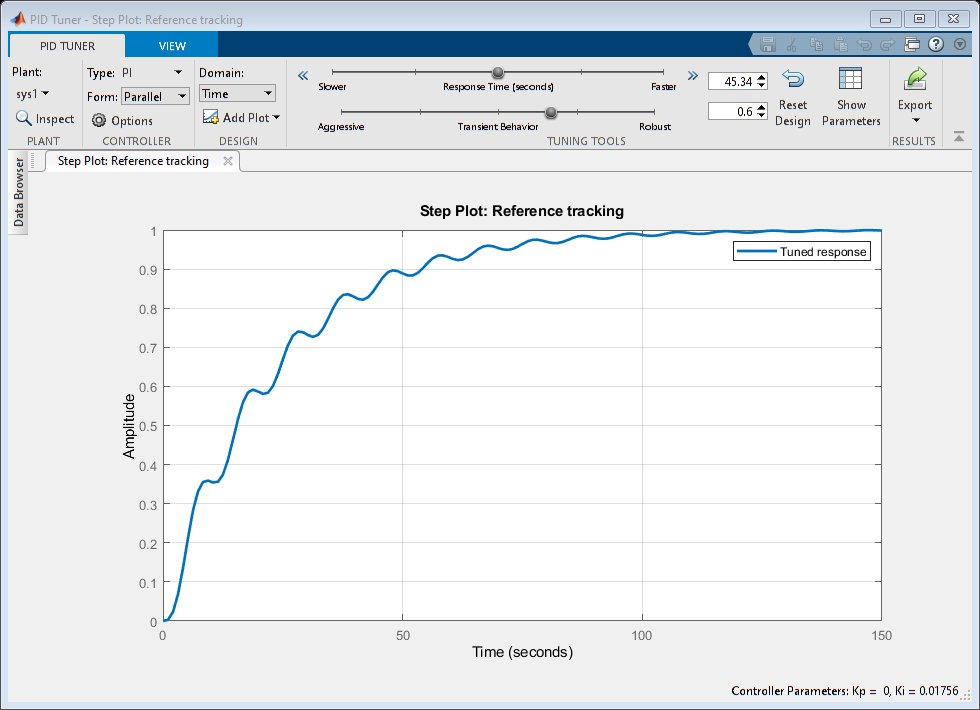
First Order System PID Code 3

Analysis of Second Order system with PID controllers 4

## Second Order System PID Code

B1= 0.5  
M1= 5;  
K1 =1;  
P1=5;  
  
sys1 = tf([P1\*K1],[M1,B1,2\*K1])  
pidTuner(sys1)

B1 =  
  
 0.5000  
  
  
sys1 =  
   
 5  
 -----------------  
 5 s^2 + 0.5 s + 2  
   
Continuous-time transfer function.



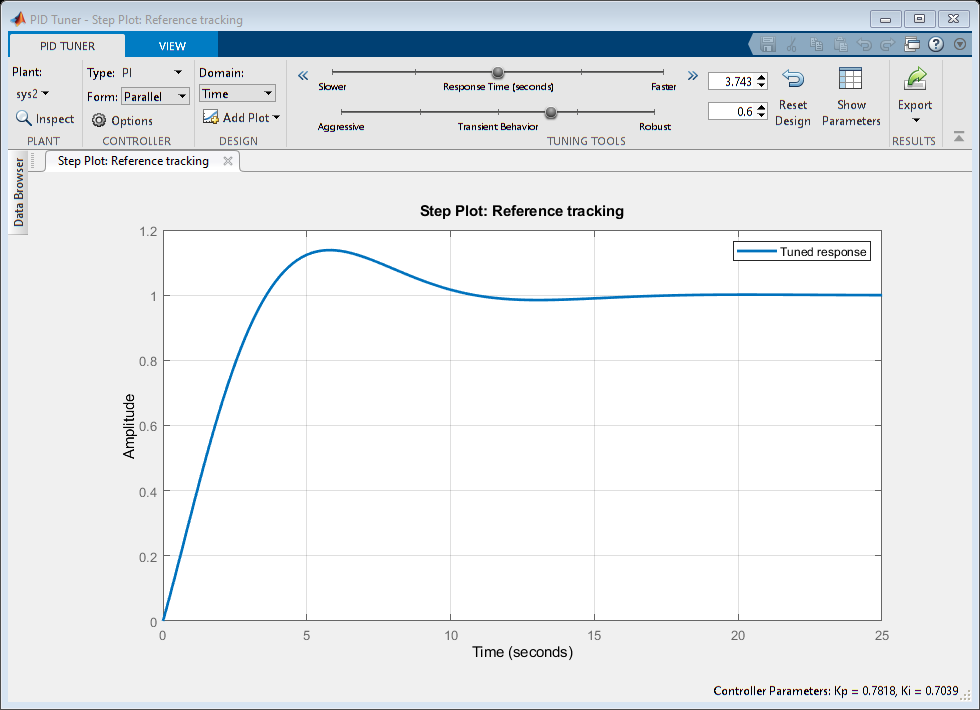
## Analysis of Second Order system with PID controllers

% PI Controller  
% Ideal values given by the pid tuner:  
% Kp =0 Ki =0.0174  
% Rise time=50.4s  
% Settling time= 93.4s  
% Overshoot= 0.00872%  
  
% To make 0% overshoot we are decreasing the speed of the system  
  
% The new values are given below:  
% Kp =0 Ki =0.015  
% Rise time=51.1s  
% Settling time= 94.3s  
% Overshoot= 0  
  
% PD Controller  
% Ideal values given by the pid tuner:  
% Kp =27.35 Kd =6.251  
% Rise time=0.017s  
% Settling time= 1.35s  
% Overshoot= 24.7%  
  
% To make overshoot 0% we are increasing the speed.  
% The new values are given below:  
  
% Kp =26.97 Kd =6.34  
% Rise time=0.0179s  
% Settling time= 0.13s  
% Overshoot= 0%  
  
% PID Controller  
% Ideal values given by the pid tuner:  
  
% This is the best possible reponse given by the system i.e. we cannot  
% decrease the overshoot further more.  
  
% Kp =26.97 Kd =6.34 Ki =0.68  
% Rise time=0.495s  
% Settling time= 9.3s  
% Overshoot= 12.4%

## First Order System PID Code

B2= 0.5;  
M2= 5;  
P2 = 2;  
  
sys2 = tf([P2],[M2,B2+1])  
pidTuner(sys2)

sys2 =  
   
 2  
 ---------  
 5 s + 1.5  
   
Continuous-time transfer function.



## Analysis of Second Order system with PID controllers

PI Controller Ideal values given by the pid tuner: Kp =0.781 Ki =0.70 Rise time=2.7s Settling time= 9.87s Overshoot= 13.8%

% Changing the values of Kp and ki to get the best possible stability of  
% the system i.e decreasing the overshoot.  
  
% The new values are given below:  
% Kp =1.25 Ki =0.46  
% Rise time=3.59s  
% Settling time= 5.39s  
% Overshoot= 1.33%  
  
% PD Controller  
% Ideal values given by the pid tuner:  
% Kp =53.18 Kd =0  
% Rise time=0.102s  
% Settling time= 0.181s  
% Overshoot= 0%  
  
% As we are getting the overshoot 0% so we don't need to change any  
% parameters.  
  
% PID Controller  
% Ideal values given by the pid tuner:  
  
% Kp =1.07 Kd =0 Ki =0.53  
% Rise time=3.04s  
% Settling time= 10.6s  
% Overshoot= 6.08%  
  
% The above parameters gives the best response for the system so we don't  
% need to change any parameters.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

# Position of poles with different zeta values

# Roots of the Standard Equation

Author: Sourav Dey PS Number: 99003785 Date: 7th April 2021. Version: Matlab 2020b.

Description: 1

code 1

Comparison Analysis: 6

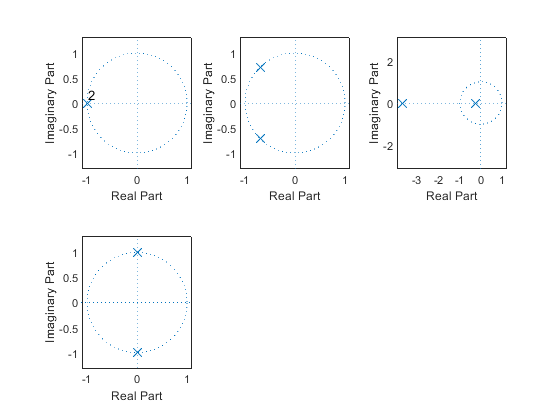
## Description:

Here standard Second Order Equation is taken as the plant.

## code

zeta=1;  
TF=tf([1],[1,(2\*zeta),1])  
sys = tf([1],[1,(2\*zeta),1])  
figure  
subplot(2,3,1)  
S = stepinfo(sys)  
[z,p,k]= tf2zp([1],[1,(2\*zeta),1])  
zplane(z,p)  
  
zeta=0.7 ;  
TF=tf([1],[1,(2\*zeta),1])  
sys = tf([1],[1,(2\*zeta),1])  
subplot(2,3,2)  
S = stepinfo(sys)  
[z,p,k]= tf2zp([1],[1,(2\*zeta),1])  
zplane(z,p)  
  
zeta=2;  
TF=tf([1],[1,(2\*zeta),1])  
sys = tf([1],[1,(2\*zeta),1])  
subplot(2,3,3)  
S = stepinfo(sys)  
[z,p,k]= tf2zp([1],[1,(2\*zeta),1])  
zplane(z,p)  
  
zeta=0;  
TF=tf([1],[1,(2\*zeta),1])  
sys = tf([1],[1,(2\*zeta),1])  
subplot(2,3,4)  
S = stepinfo(sys)  
[z,p,k]= tf2zp([1],[1,(2\*zeta),1])  
zplane(z,p)

TF =  
   
 1  
 -------------  
 s^2 + 2 s + 1  
   
Continuous-time transfer function.  
  
  
sys =  
   
 1  
 -------------  
 s^2 + 2 s + 1  
   
Continuous-time transfer function.  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 3.3579  
 SettlingTime: 5.8339  
 SettlingMin: 0.9000  
 SettlingMax: 0.9994  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 0.9994  
 PeakTime: 9.7900  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -1  
 -1  
  
  
k =  
  
 1  
  
  
TF =  
   
 1  
 ---------------  
 s^2 + 1.4 s + 1  
   
Continuous-time transfer function.  
  
  
sys =  
   
 1  
 ---------------  
 s^2 + 1.4 s + 1  
   
Continuous-time transfer function.  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 2.1268  
 SettlingTime: 5.9789  
 SettlingMin: 0.9001  
 SettlingMax: 1.0460  
 Overshoot: 4.5986  
 Undershoot: 0  
 Peak: 1.0460  
 PeakTime: 4.4078  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -0.7000 + 0.7141i  
 -0.7000 - 0.7141i  
  
  
k =  
  
 1  
  
  
TF =  
   
 1  
 -------------  
 s^2 + 4 s + 1  
   
Continuous-time transfer function.  
  
  
sys =  
   
 1  
 -------------  
 s^2 + 4 s + 1  
   
Continuous-time transfer function.  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: 8.2308  
 SettlingTime: 14.8789  
 SettlingMin: 0.9017  
 SettlingMax: 0.9993  
 Overshoot: 0  
 Undershoot: 0  
 Peak: 0.9993  
 PeakTime: 27.3269  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 -3.7321  
 -0.2679  
  
  
k =  
  
 1  
  
  
TF =  
   
 1  
 -------  
 s^2 + 1  
   
Continuous-time transfer function.  
  
  
sys =  
   
 1  
 -------  
 s^2 + 1  
   
Continuous-time transfer function.  
  
  
S =   
  
 struct with fields:  
  
 RiseTime: NaN  
 SettlingTime: NaN  
 SettlingMin: NaN  
 SettlingMax: NaN  
 Overshoot: NaN  
 Undershoot: NaN  
 Peak: Inf  
 PeakTime: Inf  
  
  
z =  
  
 0×1 empty double column vector  
  
  
p =  
  
 0.0000 + 1.0000i  
 0.0000 - 1.0000i  
  
  
k =  
  
 1



## Comparison Analysis:

% 1st value lise on negative x axis means: Critically-damped case & stable  
% 2nd value lise in 2nd & 3rd quadrant means: Under-damp case & stable  
% 3rd value lise on negative x axis means: Overdamped case & stable  
% 4th value lise on positive and negative y axis means:Undamped, roots are  
% imaginery and complex conjugate. We can also say unstable.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

# Movement of Poles

# Movement of Poles.

Author: Sourav Dey PS Number: 99003785 Date: 7th April 2021. Version: Matlab 2020b.

Description: Here the movement of poles is shown along the real and 1

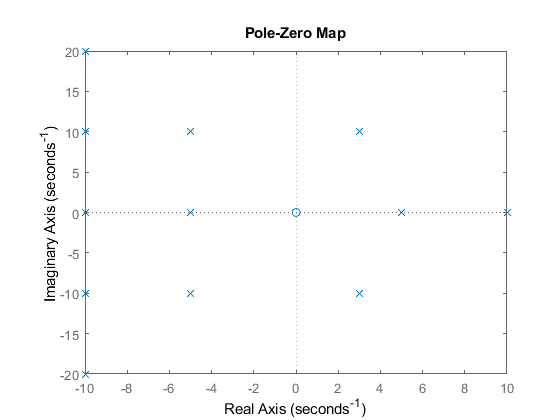
Analysis : 2

## Description: Here the movement of poles is shown along the real and

imaginary axis.

poles = [-10+20i -10-20i -5+10i -5-10i -10+10i -10-10i 3+10i 3-10i -5+0i +5+0i -10+0i +10-0i ];  
  
zeros = [0 0];  
  
gain = 0.9;  
  
s=zpk(zeros,poles,gain);  
  
pzplot(s)  
  
[wn,zeta] = damp(s)

wn =  
  
 5.0000  
 5.0000  
 10.0000  
 10.0000  
 10.4403  
 10.4403  
 11.1803  
 11.1803  
 14.1421  
 14.1421  
 22.3607  
 22.3607  
  
  
zeta =  
  
 1.0000  
 -1.0000  
 1.0000  
 -1.0000  
 -0.2873  
 -0.2873  
 0.4472  
 0.4472  
 0.7071  
 0.7071  
 0.4472  
 0.4472



## Analysis :

1. If we move the roots along Wn, then frequency will increase and overshoot remains same. 2. If we move the poles along jw axis, then overshoot increases and frequency also increases. 3. If we move along zetaWn axis or x-axis. 3a. If we move to right hand side then overshoot increases and frequency decreases. 3b. If we move to left hand side of s-plane overshoot decreases and frequency increases.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

# Simple Harmonic Motion

Simple Harmonic Motion Equation 1

Plant Description: 1

Code: 1

Analysis: 7

## Simple Harmonic Motion Equation

Author: Sourav Dey PS Number: 99003785 Date: 7th April 2021. Version: Matlab 2020b.

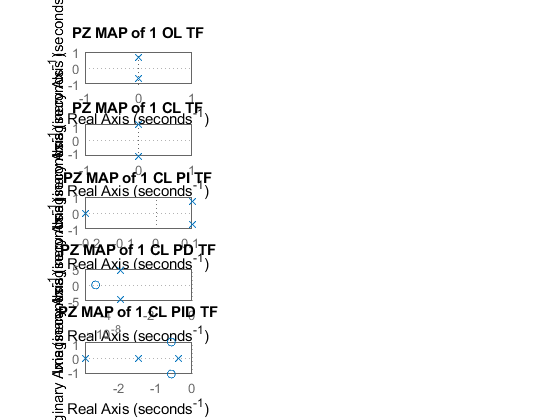
## Plant Description:

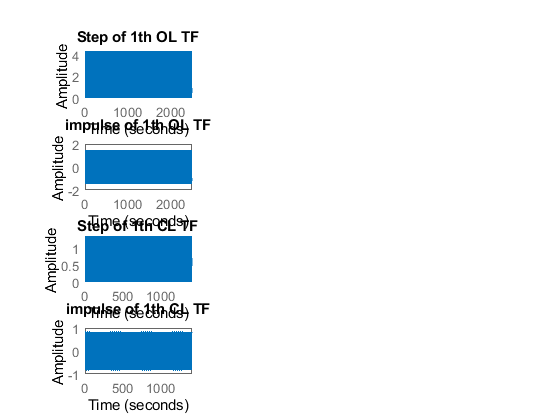
Simple Harmonic Motion is the plant taken here. Equation : fnet = Kx + Mx" K = Spring constant M = Mass; Applocation:Application of S.H.M are as follows: Car shock Absorber. Musical instruments. Bunjee Jumping.

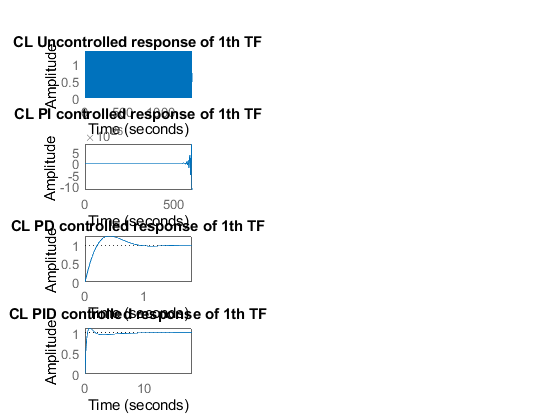
## Code:

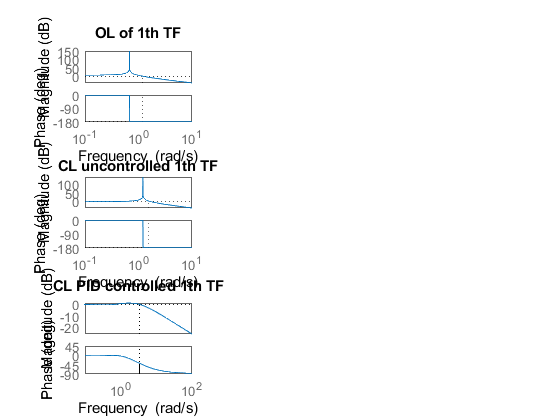
clc  
clear all;  
K = ([2.3 0]);  
M = ([5 0]);  
  
  
for i = 1  
 sys\_ol = tf([1],[1,0,K/M])  
 sys\_cl = feedback(sys\_ol,1);  
  
 [GC\_PI,info\_PI] = pidtune(sys\_ol,'PI');  
 sys\_cl\_PI = feedback(sys\_ol \* GC\_PI,1);  
  
 [GC\_PD,info\_PD] = pidtune(sys\_ol,'PD');  
 sys\_cl\_PD = feedback(sys\_ol \* GC\_PD,1);  
  
 [GC\_PID,info\_PID] = pidtune(sys\_ol,'PID');  
 sys\_cl\_PID = feedback(sys\_ol \* GC\_PID,1);  
  
  
  
 % PZ map  
 figure(1);  
 subplot(5,3,i);  
 pzmap(sys\_ol)  
 title(['PZ MAP of ', num2str(i) ,' OL TF']);  
  
 subplot(5,3,i+3);  
 pzmap(sys\_cl)  
 title(['PZ MAP of ', num2str(i) ,' CL TF']);  
  
 subplot(5,3,i+6);  
 pzmap(sys\_cl\_PI)  
 title(['PZ MAP of ', num2str(i) ,' CL PI TF']);  
  
 subplot(5,3,i+9);  
 pzmap(sys\_cl\_PD);  
 title(['PZ MAP of ', num2str(i) ,' CL PD TF']);  
  
 subplot(5,3,i+12);  
 pzmap(sys\_cl\_PID)  
 title(['PZ MAP of ', num2str(i) ,' CL PID TF']);  
  
  
% input response plots  
 figure(2);  
 subplot(4,3,i);  
 step(sys\_ol)  
 title(['Step of ', num2str(i) ,'th OL TF']);  
  
 subplot(4,3,i+3);  
 impulse(sys\_ol)  
 title(['impulse of ', num2str(i) ,'th OL TF']);  
  
 subplot(4,3,i+6);  
 step(sys\_cl)  
 title(['Step of ', num2str(i) ,'th CL TF']);  
  
 subplot(4,3,i+9);  
 impulse(sys\_cl)  
 title(['impulse of ', num2str(i) ,'th CL TF']);  
  
  
  
% controller plots  
 figure(3);  
 subplot(4,3,i);  
 step(sys\_cl)  
 title(['CL Uncontrolled response of ', num2str(i) ,'th TF']);  
  
 subplot(4,3,i+3);  
 step(sys\_cl\_PI)  
 title(['CL PI controlled response of ', num2str(i) ,'th TF']);  
  
 subplot(4,3,i+6);  
 step(sys\_cl\_PD)  
 title(['CL PD controlled response of ', num2str(i) ,'th TF']);  
  
 subplot(4,3,i+9);  
 step(sys\_cl\_PID)  
 title(['CL PID controlled response of ', num2str(i) ,'th TF']);  
  
% Bode plots  
 figure(4)  
 subplot(3,3,i)  
 bode(sys\_ol)  
 margin(sys\_ol)  
 [Gm,Pm,Wcg,Wcp] = margin(sys\_ol)  
 title(['OL of ',num2str(i),'th TF']);  
  
 subplot(3,3,i+3)  
 bode(sys\_cl)  
 margin(sys\_cl)  
 [Gm,Pm,Wcg,Wcp] = margin(sys\_cl)  
 title(['CL uncontrolled ',num2str(i),'th TF']);  
  
 subplot(3,3,i+6)  
 bode(sys\_cl\_PID)  
 margin(sys\_cl\_PID)  
 [Gm,Pm,Wcg,Wcp] = margin(sys\_cl\_PID)  
 title(['CL PID controlled ',num2str(i),'th TF']);  
  
end

sys\_ol =  
   
 1  
 ----------  
 s^2 + 0.46  
   
Continuous-time transfer function.  
  
Warning: The closed-loop system is unstable.   
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 0  
  
  
Wcg =  
  
 Inf  
  
  
Wcp =  
  
 1.2084  
  
Warning: The closed-loop system is unstable.   
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 0  
  
  
Wcg =  
  
 Inf  
  
  
Wcp =  
  
 1.5684  
  
  
Gm =  
  
 Inf  
  
  
Pm =  
  
 141.5662  
  
  
Wcg =  
  
 NaN  
  
  
Wcp =  
  
 3.3630









## Analysis:

% From the first graph we get conjugate poles on the real axis which means  
% the system is unstable. The time response graphs shows the real nature of  
% simple harmonic motion. The GM is infinity and PM is 0 also becomes  
% negative in frequency response plot.  
  
% From the second graph we can see the change in the time response graph as  
% we connect a feedback. Position of the poles remains same.  
  
% Here we are using a PI controller. Pole gets added in the origin. The  
% system is unstable as we get a conjugate pole on the R.H.S of the  
% s-plane. Here the amplitude increases due to P controller.  
  
% Here we are using PD controller. Zero gets added to the system. The  
% stability of the system increses. Poles and zero is present of the L.H.S  
% of the s-plane.  
  
% Here we are using PID controller. The stability of the system increses as  
% a pole gets added on the L.H.S of the s-plane. The rise time and settling  
% time decreses. So speed of the system increases. Overshoot also  
% decreases. From the frequency response we can see that the GM becomes  
% infinity due to the effect of P controller and we have a positive PM  
% value. Thus the system with PID controller becomes stable.

[Published with MATLAB® R2020b](https://www.mathworks.com/products/matlab)

# Comparison of MATLAB and SciLab

This document contains the migration of system transfer function from MATLAB to SciLab.

It contains the different scripts and block diagrams present in both the tools.

The spring mass damper is taken as the system. It is a second order system as the order of the

System is 2.

Equation of the system is given below:

**Mx''(t)+ Bx'(t) + Kx(t)= Kf(t)**

f = force; B= coefficient of friction; M = mass; v= velocity; k=spring constant.

Applications of the system are as follows:

* It is used in the suspension system of the car.

# MATLAB

Plant Description:

The Mass-damper Spring Second order system is taken as Plant. It is used in as suspension.

% Equation: Mx''(t)+ Bx'(t) + Kx(t)= Kf(t).  
% f = force; B= coefficient of friction; M = mass; v= velocity; k=spring  
% constant.  
% Values: K1= 0.1 B1= 0.4 M1=1000 Wn=0.03;

Code:

B1= ([0.1 0.5 1.7]);  
M1=([1000 5 340]);M1=([1000 5 340]);  
K1 = ([0.9 1 3]);  
for i=1:3  
 sys = tf([K1(i)/M1(i)],[1,B1(i)/M1(i),K1(i)/M1(i)])  
 figure(i);  
 subplot(2,1,1);  
 impulse(sys);  
 title('Impulse Input');  
 subplot(2,1,2);  
 step(sys);  
 title('Step Input');  
 [z,p,k]= tf2zp([K1(i)/M1(i)],[1,B1(i)/M1(i),K1(i)/M1(i)])  
 figure(4);  
 zplane(z,p);  
 xlim([-5\*1e5 3\*1e5]);  
 ylim([-5\*1e5 3\*1e5]);  
 hold on;  
 S = stepinfo(sys)  
end

Transfer Function:

sys =  
   
 0.0009  
 -----------------------  
 s^2 + 0.0001 s + 0.0009

Position of Poles:

p =  
  
 -0.0001 + 0.0300i  
 -0.0001 - 0.0300i

Risetime: 34.7791  
 Settling Time: 7.8226e+04  
 SettlingMin: 0.0104  
 SettlingMax: 1.9948  
 Overshoot: 99.4778  
 Undershoot: 0  
 Peak: 1.9948  
 PeakTime: 104.7198

Bode plot:

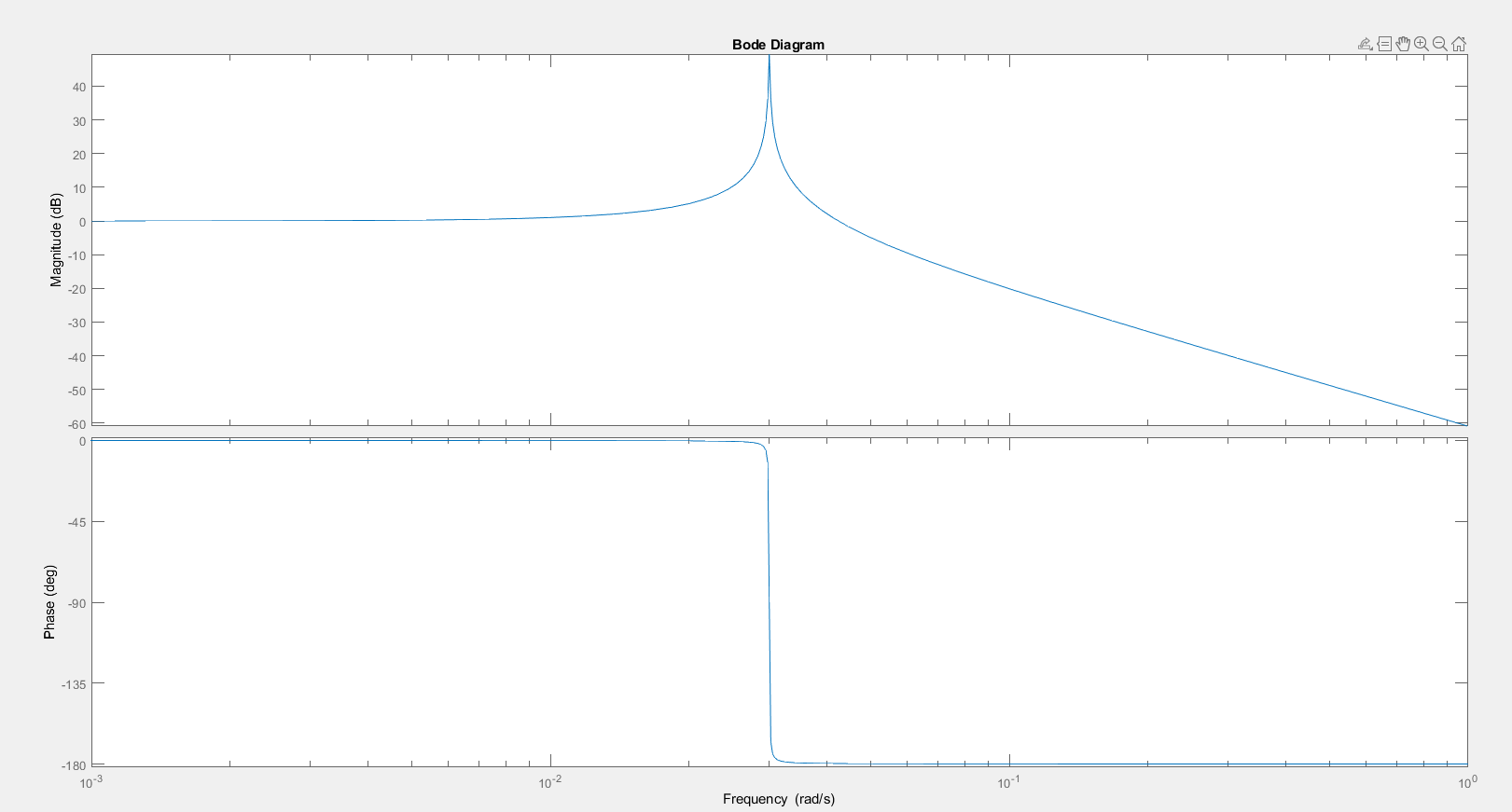


Figure 1: Bode plot of the 2nd Order Mass Spring Damper using MATLAB.

# SciLab

Code:

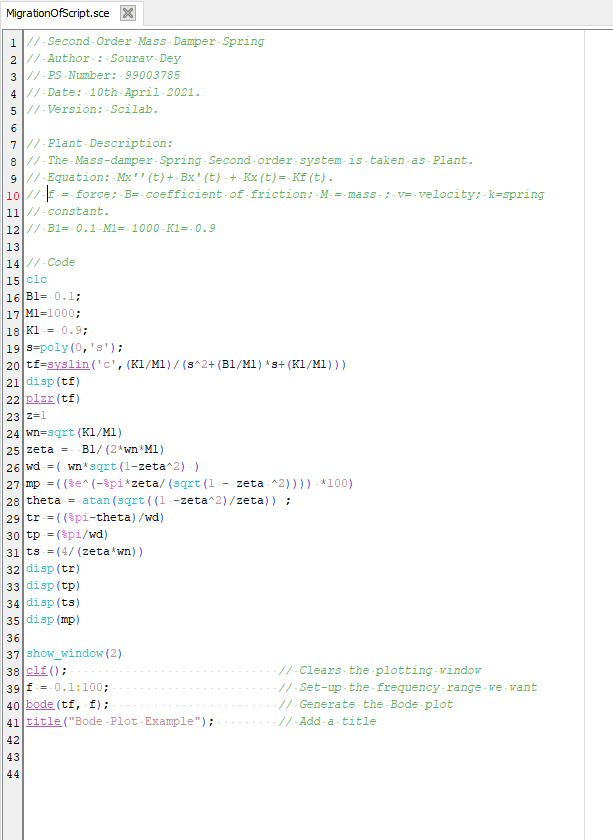


Figure 2: SciLab code for 2nd Order Mass Spring Damper.

Location of Poles:

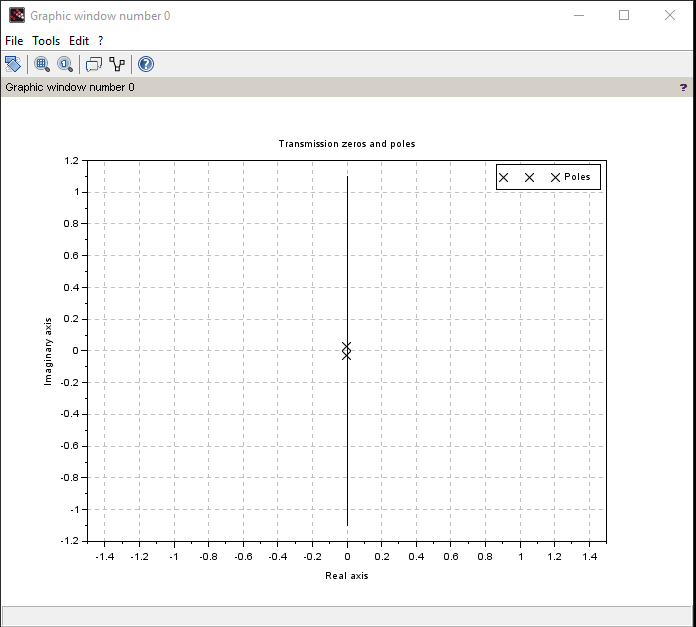


Figure 3: Location of Poles.

Bode Plots:

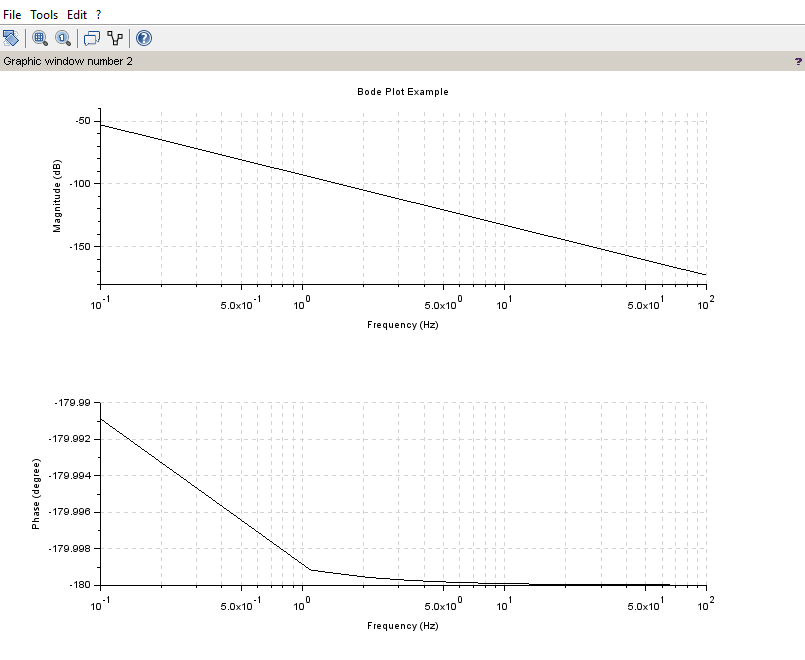


Figure 4: Bode plot of the 2nd Order Mass Spring Damper using SciLab.

# Analysis from the scripts:

* The position of the poles for the given transfer function remains same in both the tools.
* The frequency response and the time domain responses in the respective tool gives the same stability of the system.
* The rise time, settling time and the peak value remains the same.

# Simulink Modelling

Model:

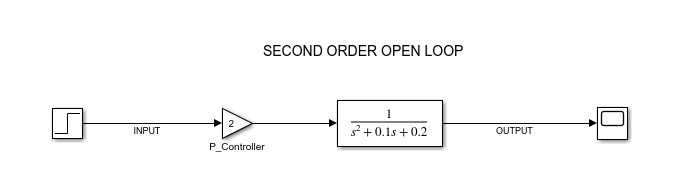


Figure 5: 2nd Order Mass Spring Damper Block diagram using Simulink.

Output Response:

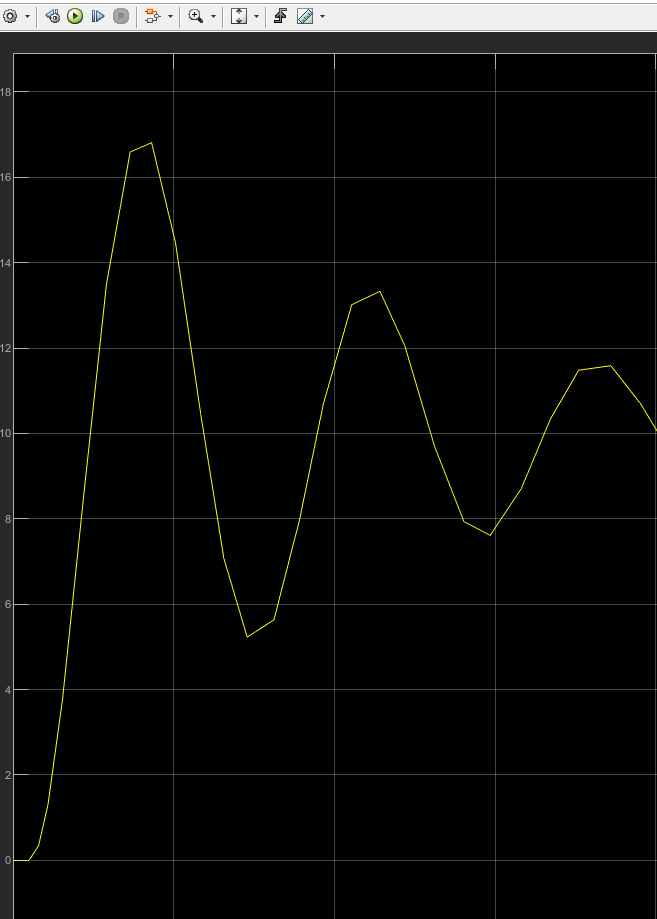


Figure 6: Time domain response of the 2nd order Mass Spring Damper using Simulink

# SciLab Modelling:

Model:

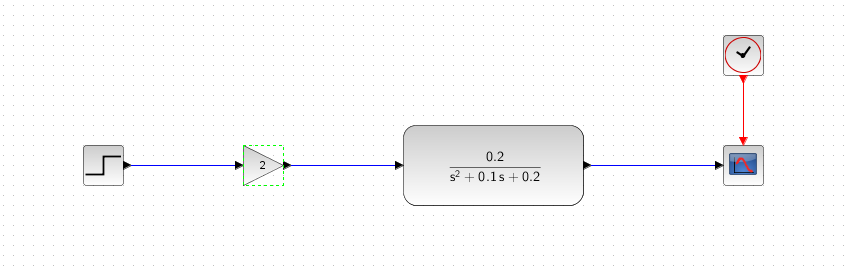


Figure 7: 2nd Order Mass Spring Damper Block diagram using SciLab.

Output Response:

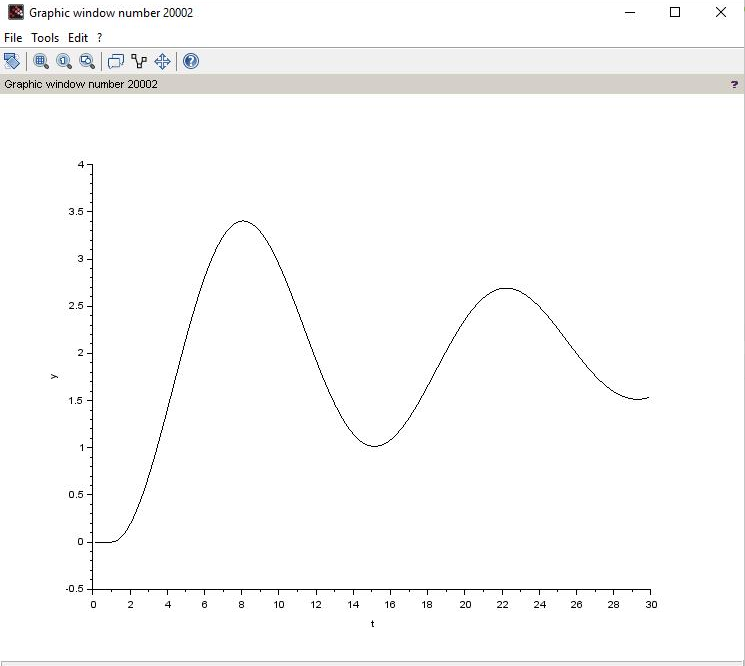


Figure 8: Time domain response of the 2nd order Mass Spring Damper using SciLab.

# Comparison Analysis from the modelling:

Above we have built the transfer function of the second order mass spring damper in both Simulink

and SciLab.

As we go through both the graph we find output time domain response to be similar in both the tools.

The settling time and peak also remains the same.