

Control Theory

Laboratory 1

Mikołaj Suchoń and Witold Surdej

Task 1)

A)

```
clear
syms('DampRatio')
K=1;
D=0;
num=K;
T1=0.02;
T2=0.002;
den=[T1*T2,T1+T2,1,0];
sys=tf(num,den);
Mpw=1.84;
eq= Mpw==(2*DampRatio*sqrt(1-DampRatio^2))^-1
```

eq =

$$\frac{46}{25} = \frac{1}{2 \text{ DampRatio} \sqrt{1 - \text{DampRatio}^2}}$$

```
eq2= -Mpw==(2*DampRatio*sqrt(1-DampRatio^2))^-1
```

eq2 =

$$-\frac{46}{25} = \frac{1}{2 \text{ DampRatio} \sqrt{1 - \text{DampRatio}^2}}$$

```
s=solve(eq,DampRatio)
```

s =

$$\left(\frac{\sqrt{-\frac{625}{23 (\sqrt{1491} - 46)}}}{2}, \frac{25 \sqrt{23}}{46 \sqrt{\sqrt{1491} + 46}} \right)$$

```
s2=solve(eq2,DampRatio)
```

s2 =

$$\begin{pmatrix} -\frac{\sqrt{-\frac{625}{23(\sqrt{1491}-46)}}}{2} \\ -\frac{25\sqrt{23}}{46\sqrt{\sqrt{1491}+46}} \end{pmatrix}$$

```
ss=[s;s2]
```

```
ss =
```

$$\begin{pmatrix} \frac{\sqrt{-\frac{625}{23(\sqrt{1491}-46)}}}{2} \\ \frac{25\sqrt{23}}{46\sqrt{\sqrt{1491}+46}} \\ -\frac{\sqrt{-\frac{625}{23(\sqrt{1491}-46)}}}{2} \\ -\frac{25\sqrt{23}}{46\sqrt{\sqrt{1491}+46}} \end{pmatrix}$$

```
var = vpa(ss)
```

```
var =
```

$$\begin{pmatrix} 0.959015965051720659303678252829 \\ 0.28335204035954443610666490435945 \\ -0.959015965051720659303678252829 \\ -0.28335204035954443610666490435945 \end{pmatrix}$$

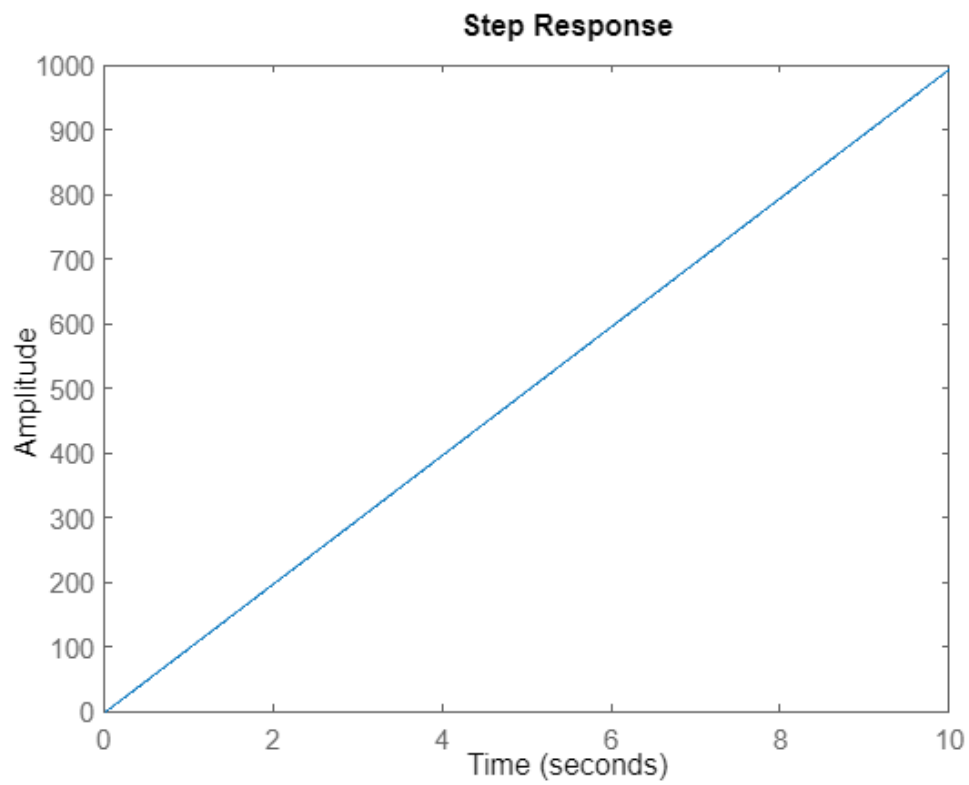
B)

```
damping_ratio = 0.2834; % check this damping ratio value.
```

```
% wn = 0:100:400;
% rlocus(sys);
% axis([-520 20 -400 400])
% sgrid(damping_ratio, wn);
% [K, closed_loop_poles] = rlocfind(sys)
% Already extracted K from rlocfind:
newData1 = load('-mat', 'K');
vars = fieldnames(newData1);
assignin('base', vars{3}, newData1.(vars{3}));
K
```

```
K = 99.4474
```

```
sys2=tf(K,den);
step(sys2)
```



```
sys_closed=feedback(sys2,1)
```

```
sys_closed =
```

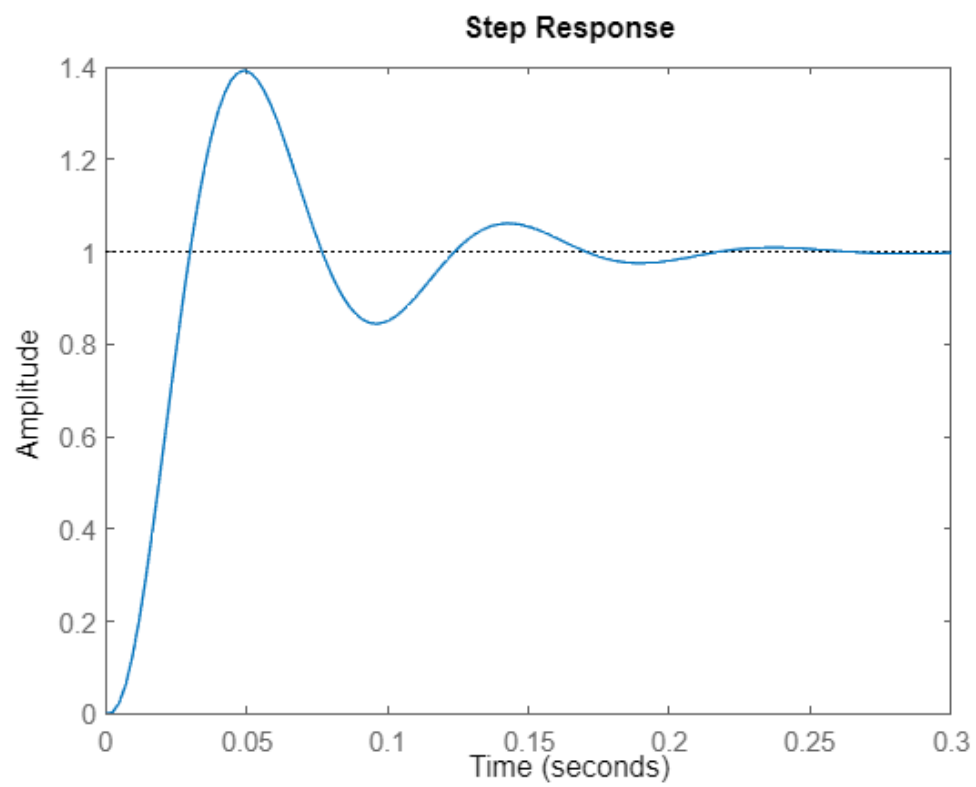
```

          99.45
-----
4e-05 s^3 + 0.022 s^2 + s + 99.45

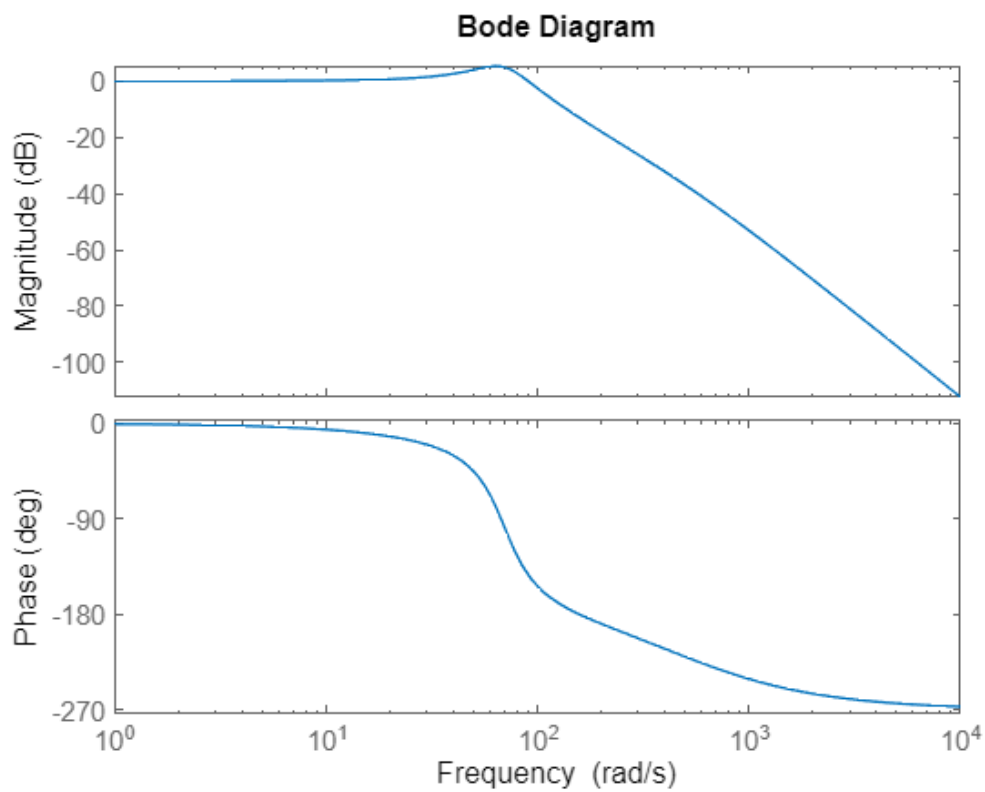
```

```
Continuous-time transfer function.
```

```
[y,tOut]=step(sys_closed);step(sys_closed)
```



```
bode(sys_closed);
```



Solving $20 \cdot \log(|T|) = 5.3$ we obtain $\sim T = 1.84$

```
T=1.84;
```

C)

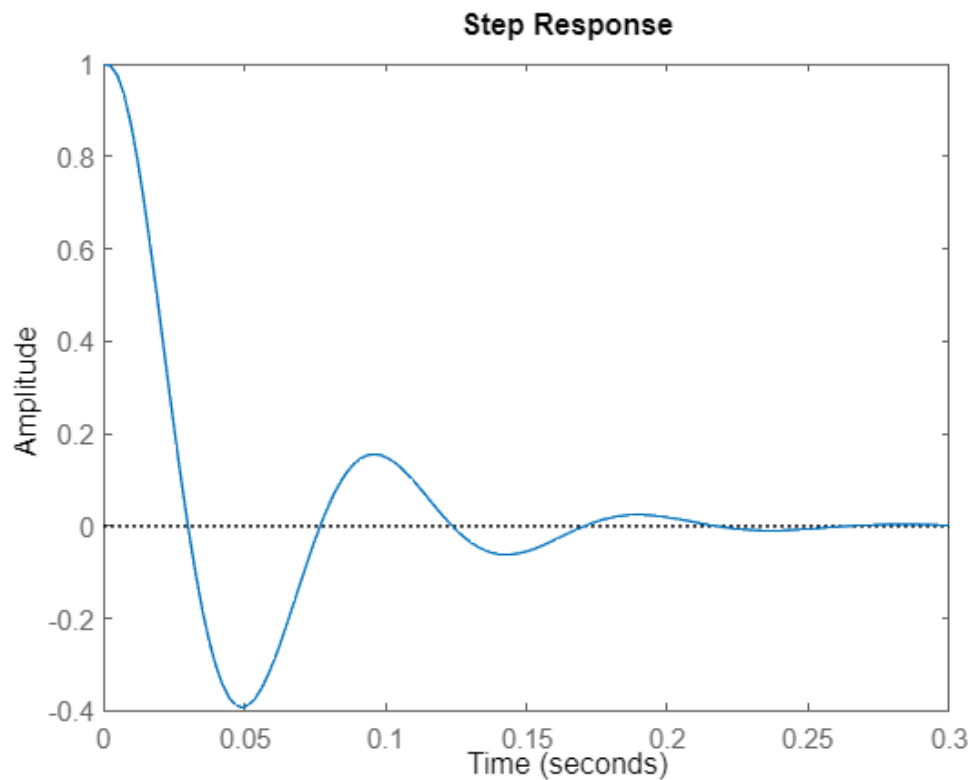
```
S=1-sys_closed
```

```
S =
```

$$\frac{4e-05 s^3 + 0.022 s^2 + s}{4e-05 s^3 + 0.022 s^2 + s + 99.45}$$

Continuous-time transfer function.

```
step(S)
```



```
wb=120;  
[mag, phase, wout] = bode(S);  
[~,phase1]=bode(S,wb);  
[~,phase2]=bode(S,wb/2);  
[~,phase3]=bode(S,wb/4);  
  
mag_db = 20*log10(mag);  
figure;  
subplot(2,1,1);
```

```
magdb=mag_db(1,:);
% semilogx(wout, magdb, '-',wb,mag1,'*',wb/2,mag2,'*',wb/3,mag3,'*');
```

From bode plot:

```
syms S
eq = 20*log10(abs(S))==3.2;
S1=vpa(solve(eq,S))
```

```
S1 = 1.4454397707459275119314815458354
```

```
eq = 20*log10(abs(S))==4.46;
S2=vpa(solve(eq,S))
```

```
S2 = 1.6710906143107075834731238646362
```

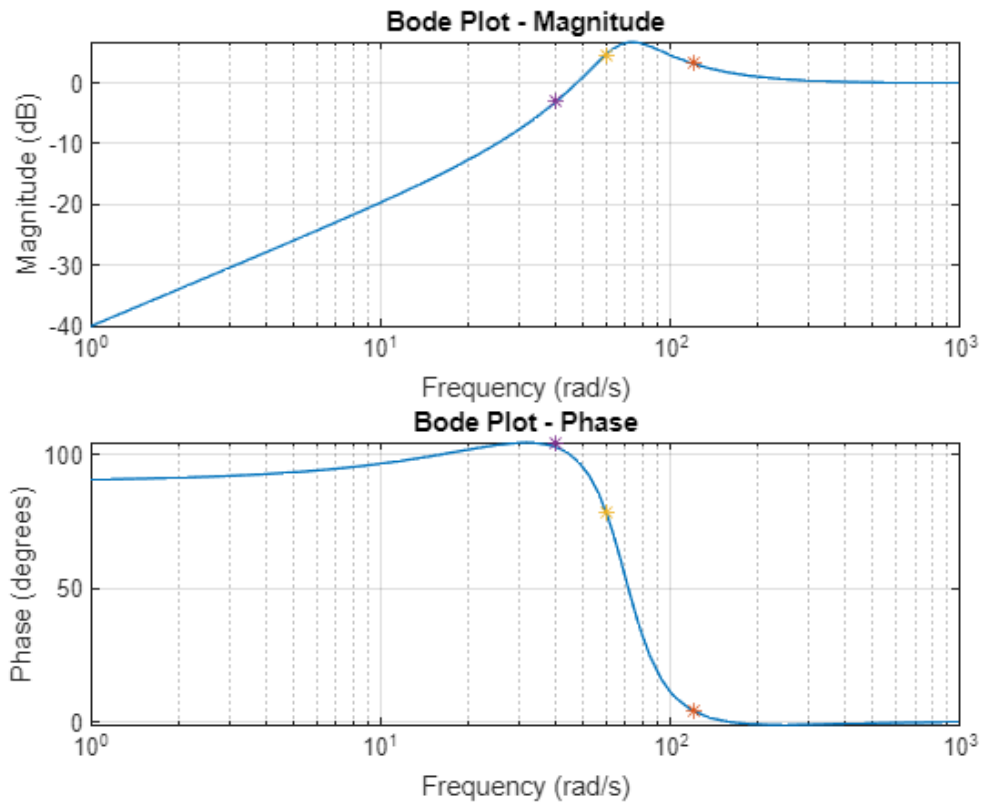
```
eq = 20*log10(abs(S))== -3.1;
S3=vpa(solve(eq,S))
```

```
S3 = 0.6998419960022734988243713460702
```

```
semilogx(wout, magdb, '-',wb,3.2,'*',wb/2,4.46,'*',wb/3,-3.1,'*');
grid on;
xlabel('Frequency (rad/s)');
ylabel('Magnitude (dB)');
title('Bode Plot - Magnitude');

subplot(2,1,2);
phase=phase(1,:);

semilogx(wout, phase, '-',wb,phase1,'*',wb/2,phase2,'*',wb/3,phase3,'*');
grid on;
xlabel('Frequency (rad/s)');
ylabel('Phase (degrees)');
title('Bode Plot - Phase');
```

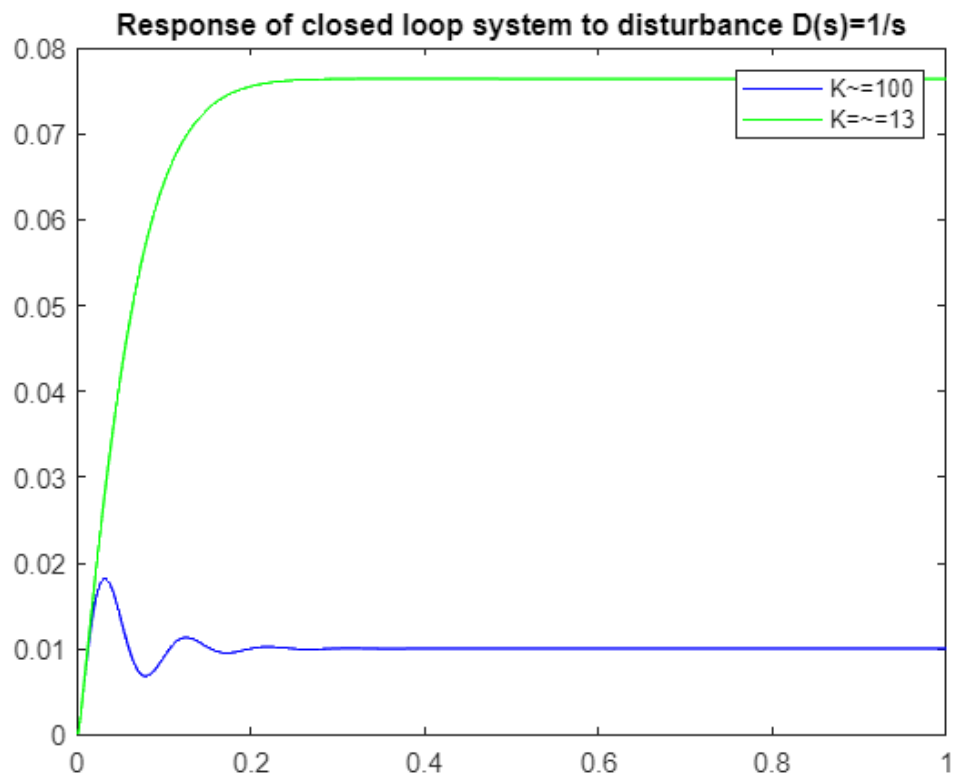


D)

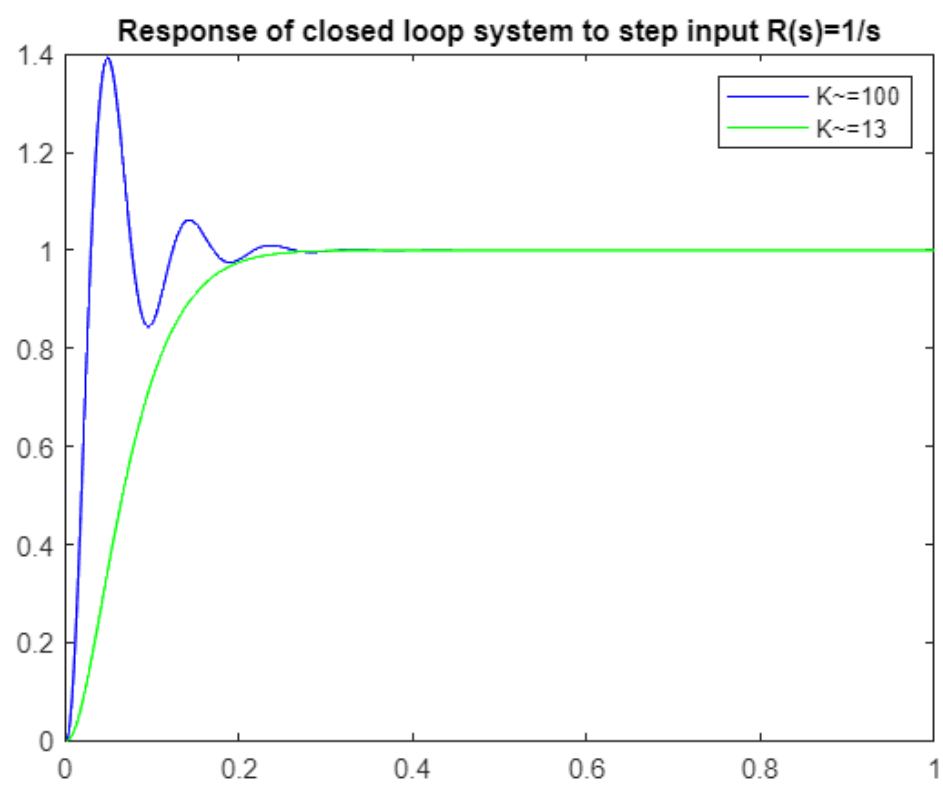
```
newData1 = load('-mat', 'K1');
vars = fieldnames(newData1);
assignin('base', vars{4}, newData1.(vars{3}));
K1
```

K1 = 13.0863

```
simou=sim("task1.slx");
figure
plot(simou.tout,simou.simout,'b',simou.tout,simou.simout1,'g')
title('Response of closed loop system to disturbance D(s)=1/s')
legend('K~100', 'K=~13')
```



```
plot(simou.tout,simou.simout2,'b',simou.tout,simou.simout3,'g')  
title('Response of closed loop system to step input  $R(s)=1/s$ ')  
legend('K=100','K=13')
```

```
clear
clc
```

a) Displaying the initial system

```
%given data
K=1;
tau=1/25;
D=0;
Kv=100;
ess=1/Kv;
%transfer funtion
s=tf('s');
%open loop system
%Gp = K/(s*(tau*s+1));
num=K;
den=[tau,1,0];
Gp=tf(num,den)
```

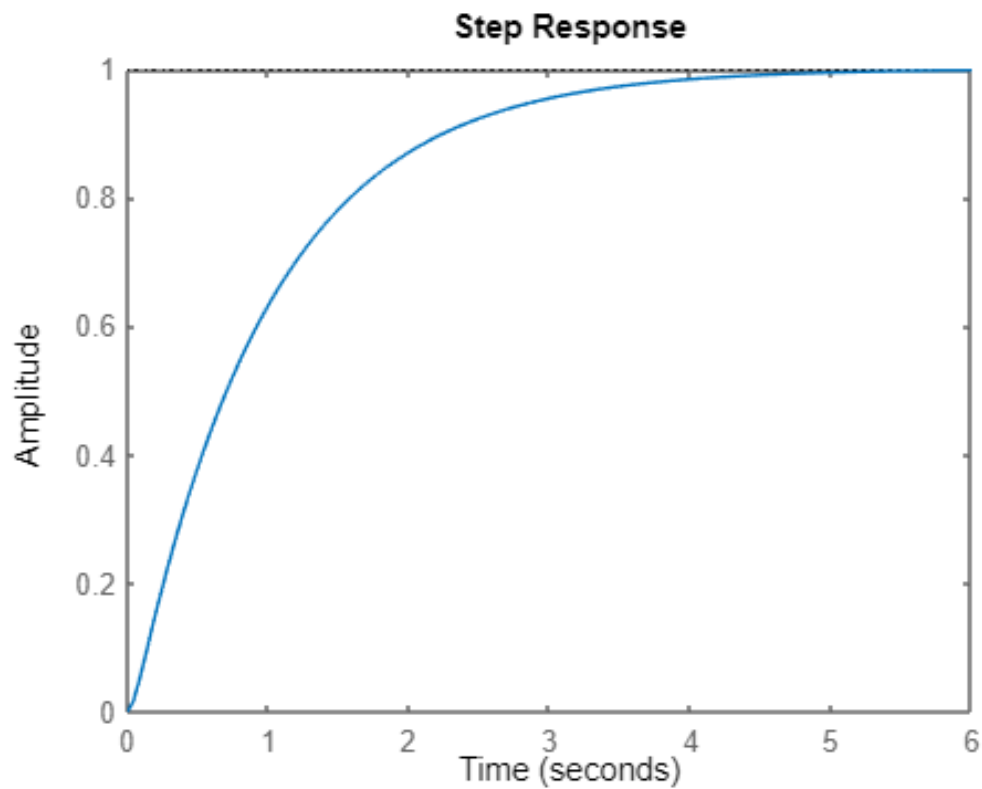
Gp =

$$\frac{1}{0.04 s^2 + s}$$

Continuous-time transfer function.

The initial system has 2 poles at p1=0 and p2=-25

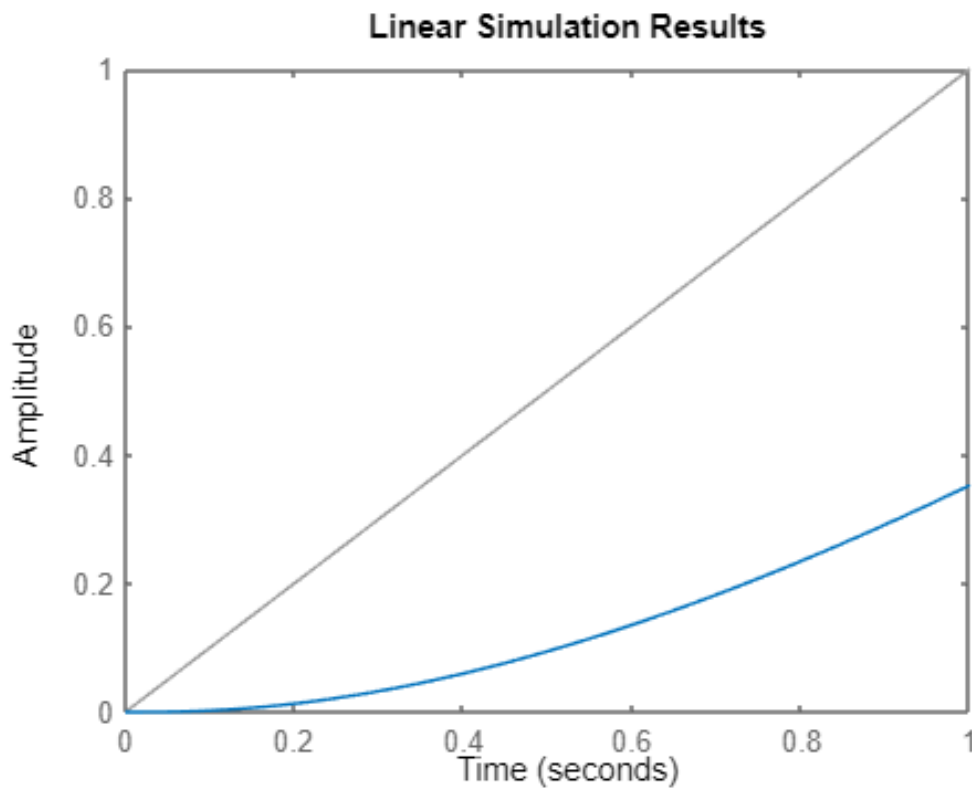
```
%closed loop system
Gp_closed=feedback(Gp,1);
%plotting closed_loop step
figure()
step(Gp_closed)
```



```
%displaying step info
stepinfo(Gp_closed)
```

```
ans = struct with fields:
    RiseTime: 2.1071
    TransientTime: 3.7915
    SettlingTime: 3.7915
    SettlingMin: 0.9002
    SettlingMax: 0.9993
    Overshoot: 0
    Undershoot: 0
    Peak: 0.9993
    PeakTime: 7.0166
```

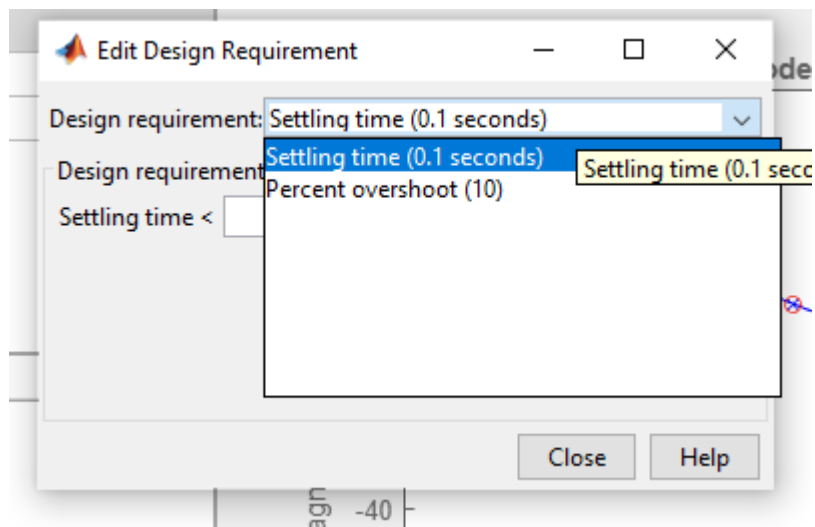
```
%linear simulation of the original system
t = 0:0.01:1;
figure()
lsim(Gp_closed,t,t)
```

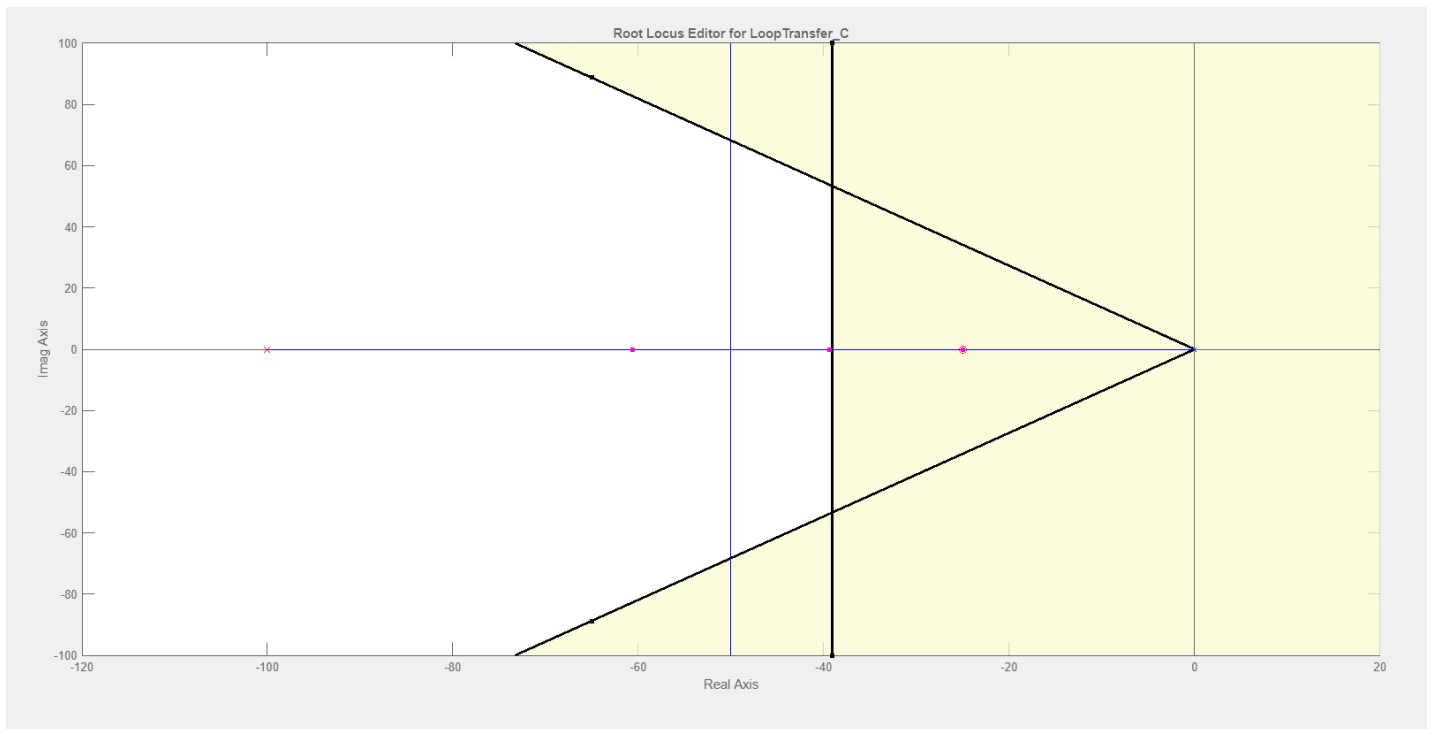
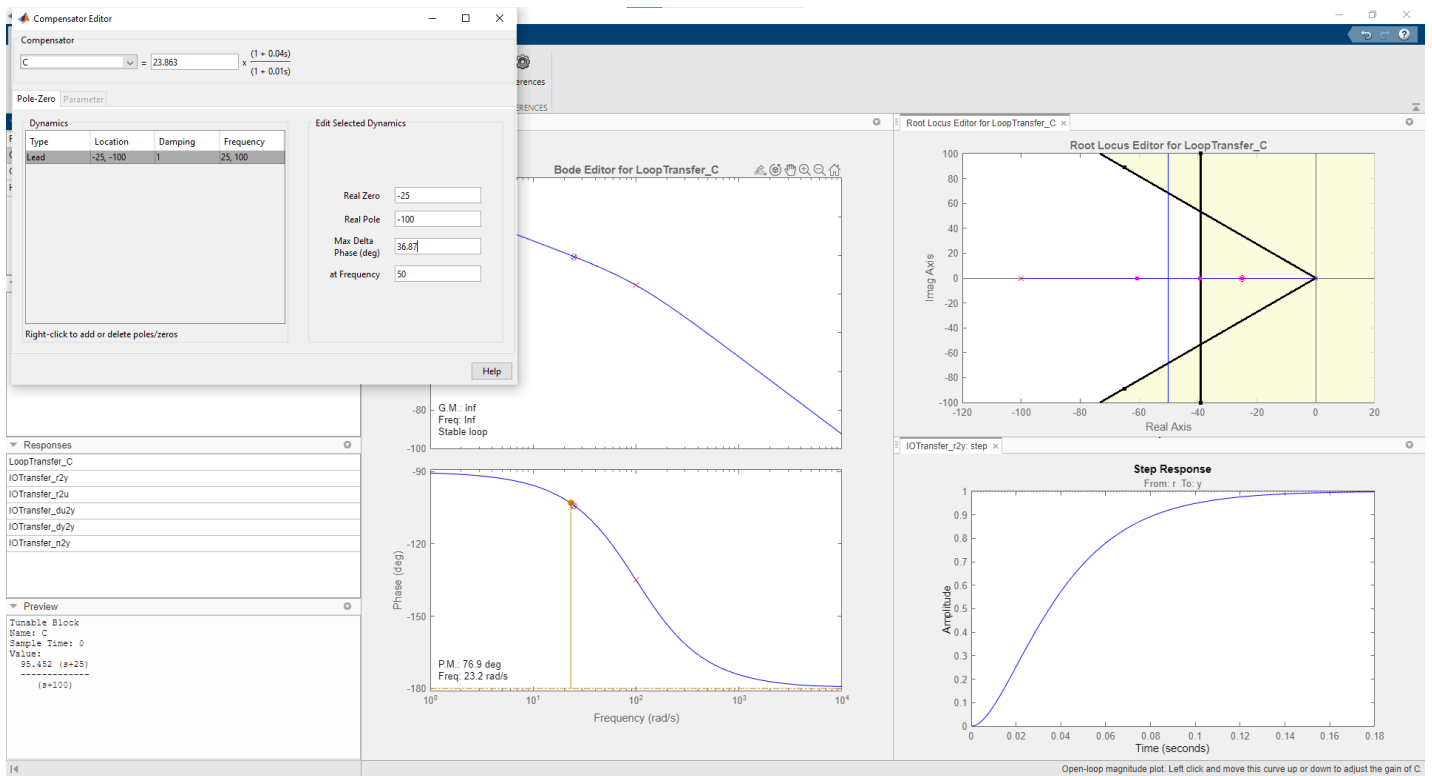


As seen on the plot above the response of the system significantly lags behind the expected ramp of the step response. We will now design a lead-lag compensator using Matlab's sisotool to fix it.

b) Designing an initial lead-lag compensator using sisotool

```
%controlSystemDesigner(compensator1.mat);
```

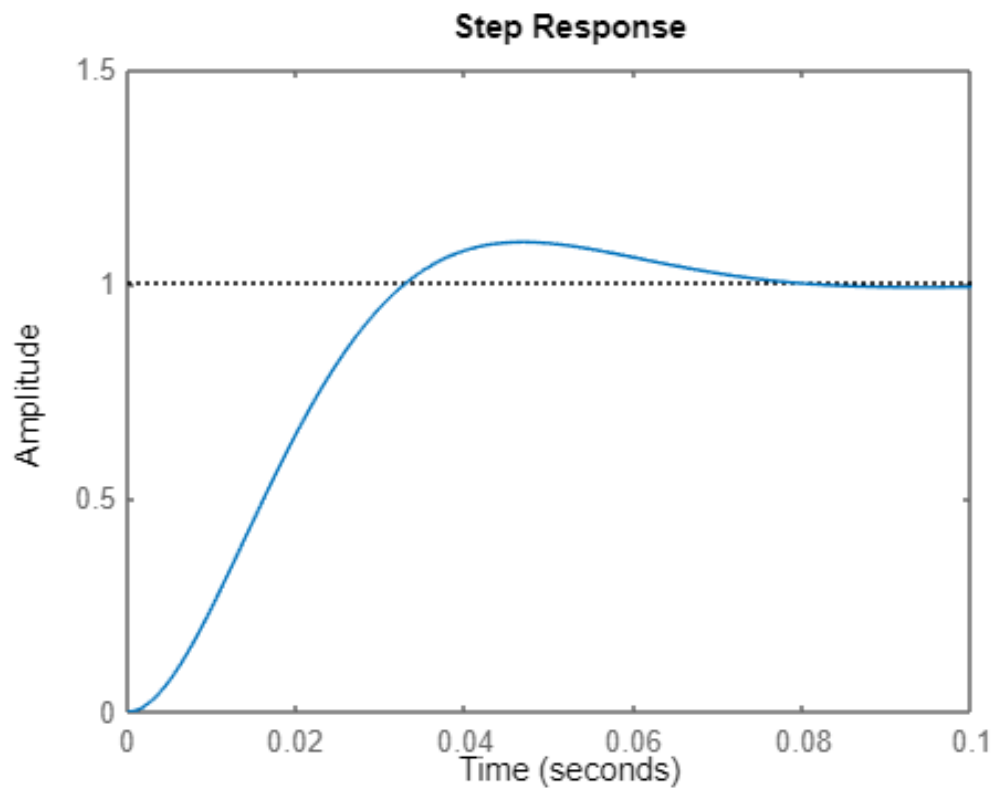




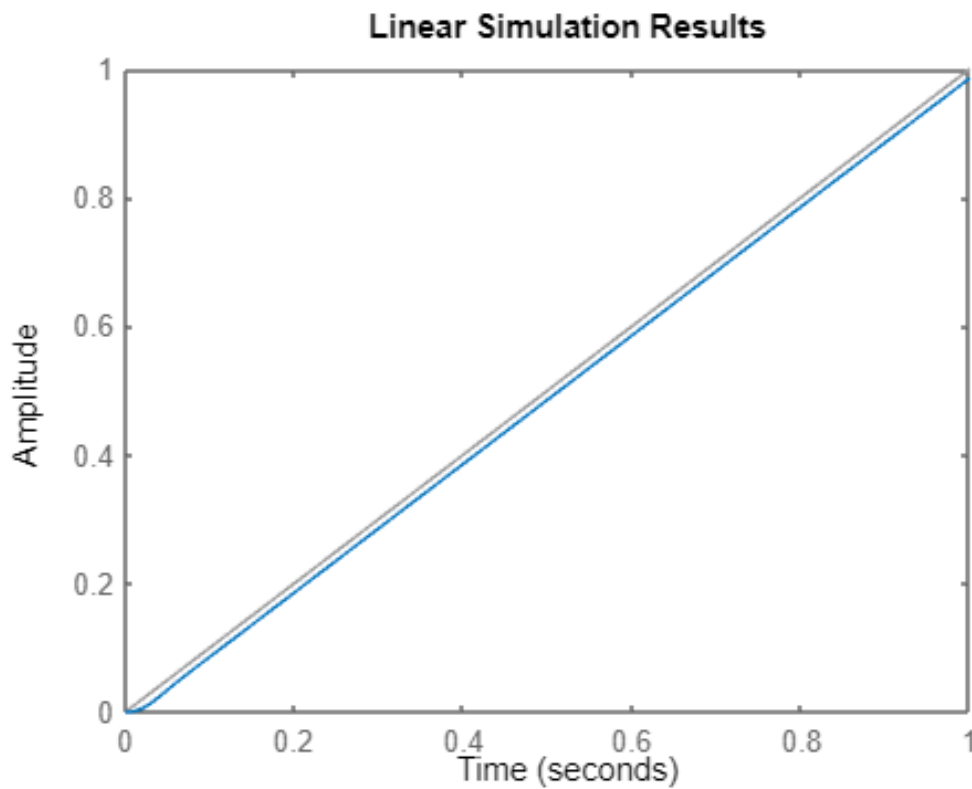
Based on the results obtained we can now test our compensator, and see if it fits all the requirements.

```
%compensator equation
Gc = (2.8*s + 70)/(0.01*s + 1);
%creating a closed loop compensated system
Gp_closed_comp = feedback(Gc*Gp,1);
%plotting results
figure();
```

```
step(Gp_closed_comp); ylim([0 1.5]);
```



```
figure();  
lsim(Gp_closed_comp,t,t);
```




```
%checking if the results fit the requirements:
stepinfo(Gp_closed_comp)
```

```
ans = struct with fields:
    RiseTime: 0.0221
    TransientTime: 0.0710
    SettlingTime: 0.0710
    SettlingMin: 0.9127
    SettlingMax: 1.0962
    Overshoot: 9.6167
    Undershoot: 0
    Peak: 1.0962
    PeakTime: 0.0470
```

Both percentage overshoot ($9.617\% < 10\%$) & settling time ($0.0710 < 0.01$) are within the design requirements , but Steady state error ($0.014 > 0.01$) is still too big. We can fix this by adjusting values of K_c and position of the pole p .

```
%controlSystemDesigner(compensator2.mat);
```


Compensator Editor

Compensator

C

=

100

x

$$\frac{(1 + 0.04s)}{(1 + 0.0033s)}$$

Pole-Zero

Parameter

Dynamics

Type	Location	Damping	Frequency
Lead	-25, -300	1	25, 300

Right-click to add or delete poles/zeros

Edit Selected Dynamics

Real Zero

-25

Real Pole

-300

Max Delta Phase (deg)

57.796

at Frequency

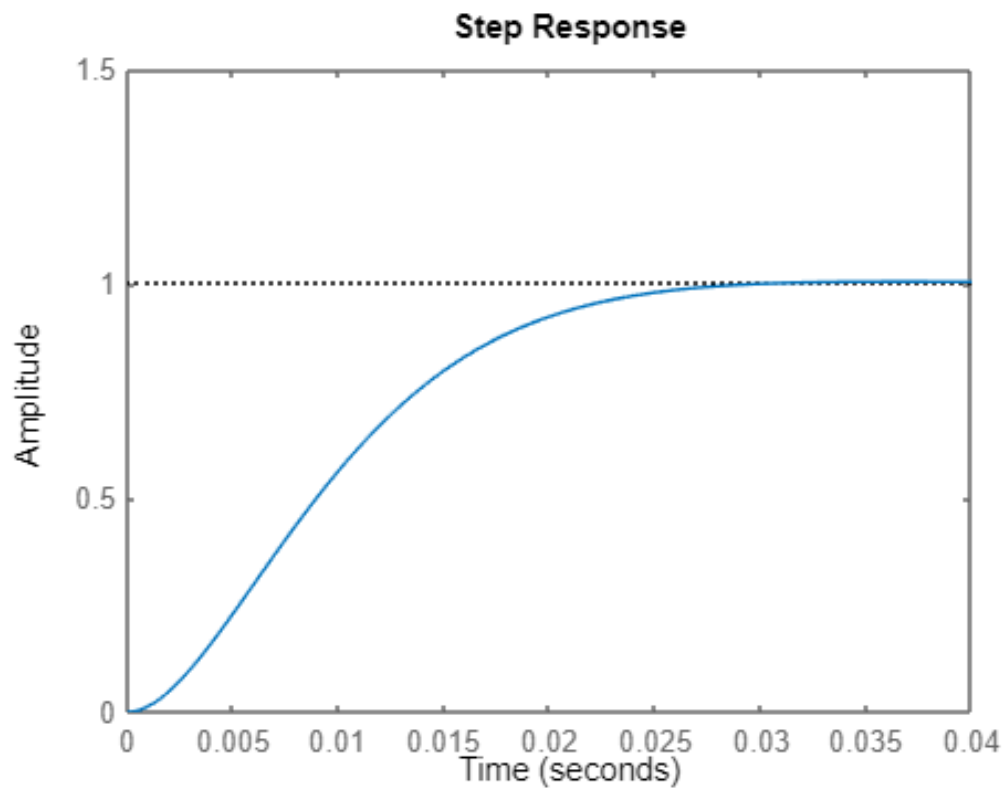
86.603

Help

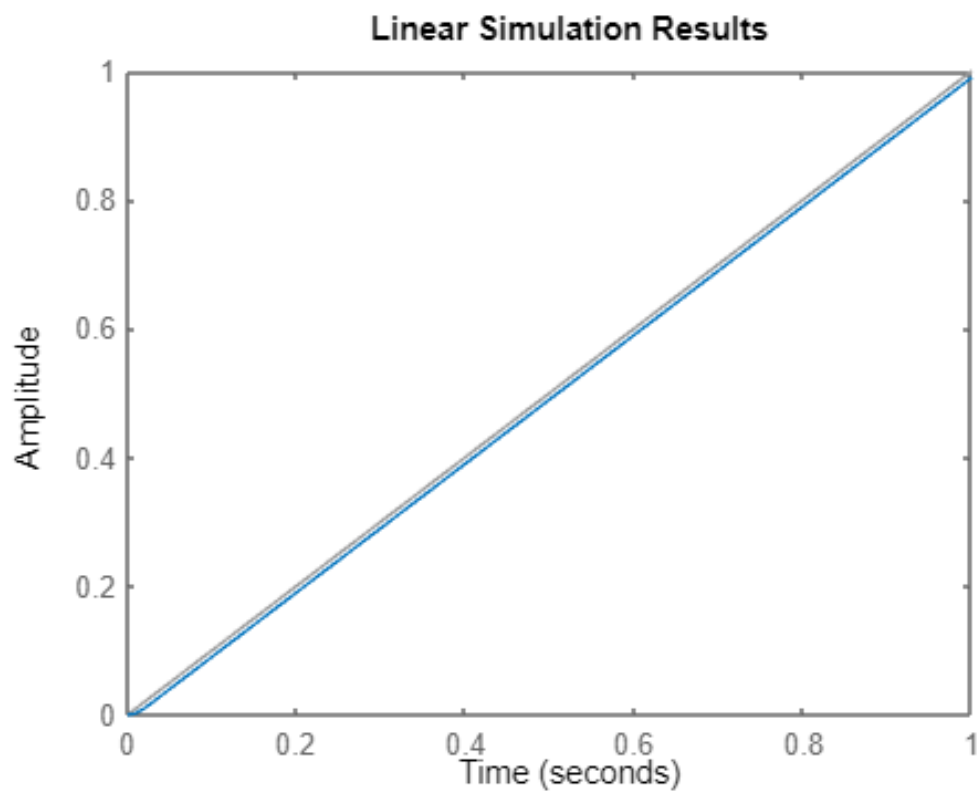
```

%checking results
%compensator equation
Gc_new = (4*s + 100)/(0.0033*s + 1);
%creating a closed loop compensated system
Gp_closed_comp_new = feedback(Gc_new*Gp,1);
%plotting results
figure();
step(Gp_closed_comp_new); ylim([0 1.5]);

```

```
figure();  
lsim(Gp_closed_comp_new,t,t);
```



```
%checking if the results fit the requirements:
stepinfo(Gp_closed_comp_new)
```

```
ans = struct with fields:
    RiseTime: 0.0158
    TransientTime: 0.0252
    SettlingTime: 0.0252
    SettlingMin: 0.9009
    SettlingMax: 1.0039
    Overshoot: 0.3873
    Undershoot: 0
    Peak: 1.0039
    PeakTime: 0.0368
```

This time, all the design requirements were met. Percentage overshoot ($0.387\% < 10\%$), settling time ($0.0025 < 0.01$) and steady state error ($0.01 = 0.01$) are within the design requirements.

c) Adjusting gain K1 to 2, and checking the influence on the compensated system:

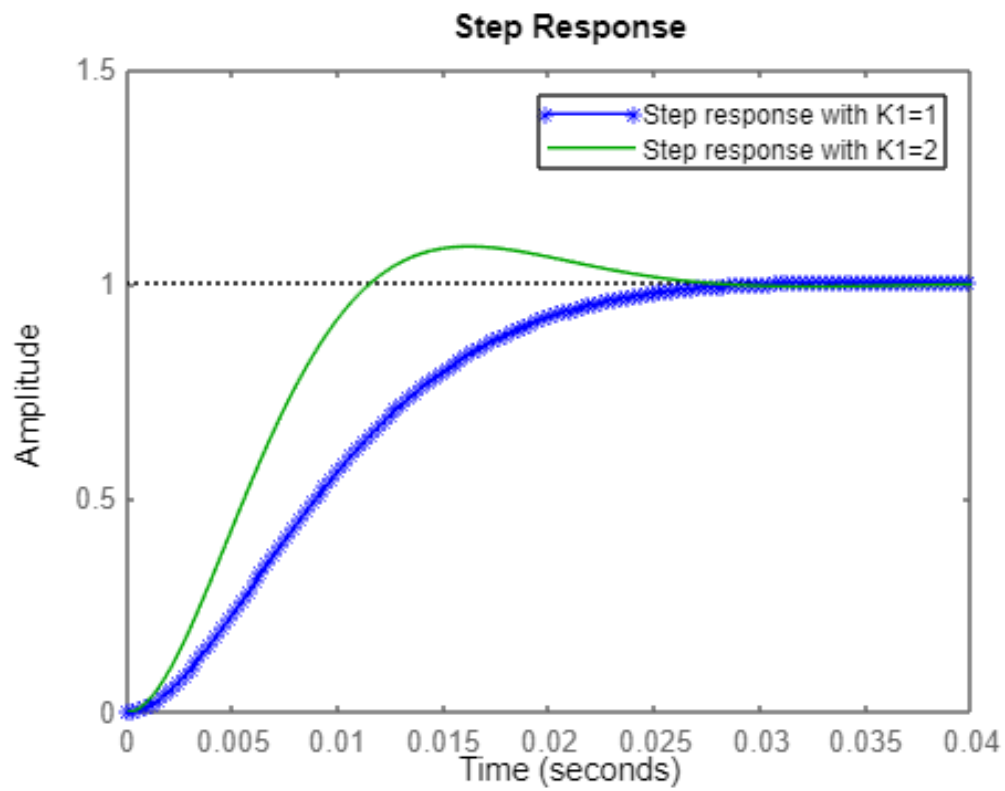
```
%creating new plant with gain K1=2
K1_new = 2;
num2=K1_new;
Gp2=tf(num2,den)
```

Gp2 =

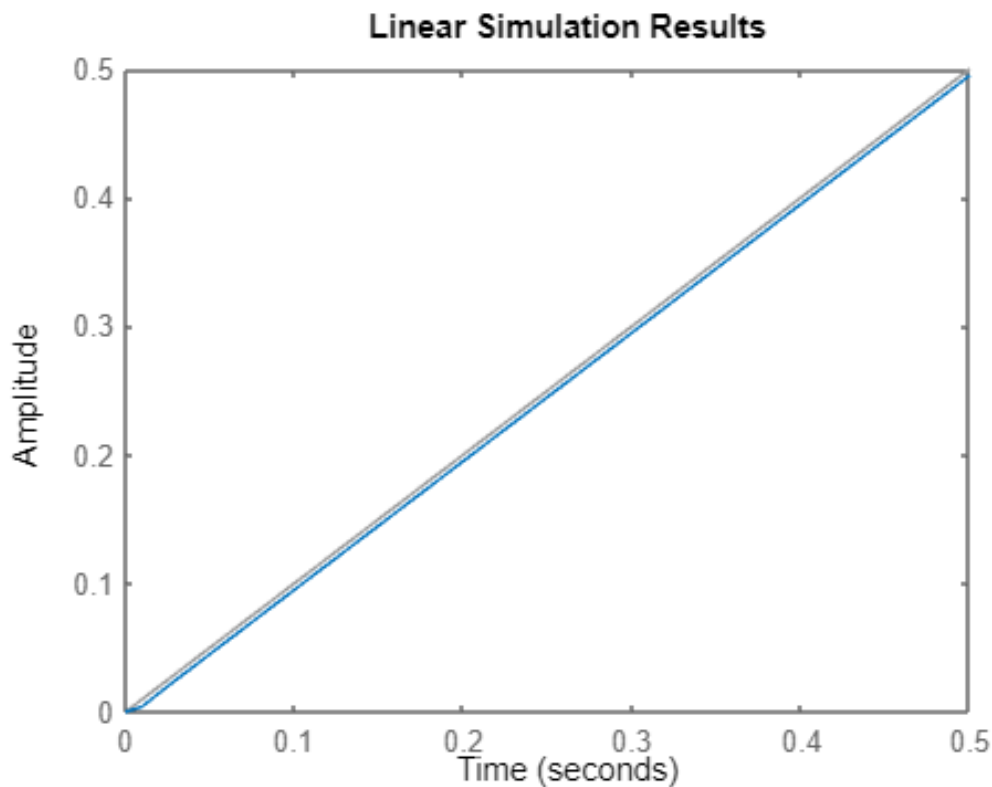
```
      2
-----
0.04 s^2 + s
```

Continuous-time transfer function.

```
%creating closed-loop system
Gp_closed_comp_new2 = feedback(Gc_new*Gp2,1);
%comparing step responses of both systems
figure();
step(Gp_closed_comp_new, 'b*-'); ylim([0 1.5]); hold on;
step(Gp_closed_comp_new2, 'g');
legend('Step response with K1=1', 'Step response with K1=2')
```



```
%checking linear simulation for the new system  
figure();  
t = 0:0.01:0.5;  
lsim(Gp_closed_comp_new2,t,t);
```



```
%collecting required data for comparison
S1=stepinfo(Gp_closed_comp_new);
S2=stepinfo(Gp_closed_comp_new2);

res=zeros(2,3);

res(1,1)=S1.Overshoot;
res(1,2)=S1.SettlingTime;
res(1,3)=0.01;

res(2,1)=S2.Overshoot;
res(2,2)=S2.SettlingTime;
res(2,3)=0.005;
%1st row syst with gain K1=1, 2nd row K1=2.
% Order of values: overshoot, settlign time, steady state error
res
```

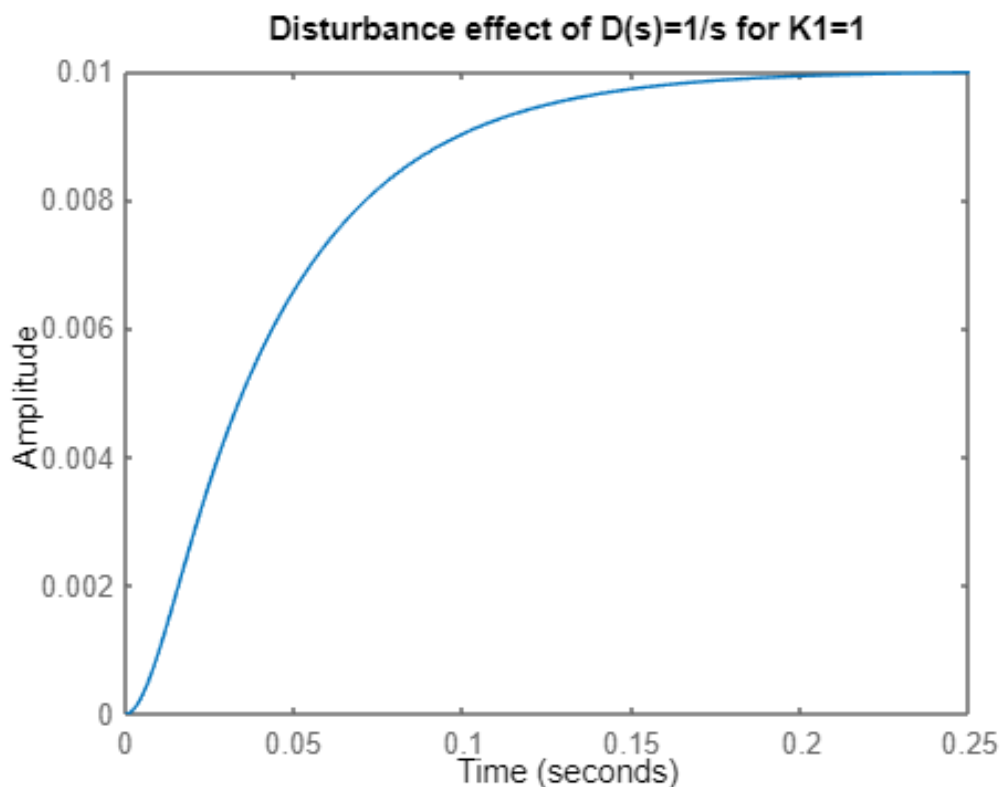
From the results presented above we can observe that both systems performance is quite good and in both cases it meets the initial requirements. The value of the steady state error, for the system with $K_1=2$, is lower than for the system with $K_1=1$. It occurs due to increas in the gain.On the other hand an increase in gain results also in an increase of percent overshoot. The value of settling time is similar although a bit better for higher gain. These results show

that the improved lead-lag compensator is robust to the change in the gain K_1 .

We can say that our system is robust because it is a closed-loop system, and therefore it is less sensitive to variations of the component. By closing the loop on the system it continuously calculates expected values to determine how much they should be increased/decreased to achieve its desired values. Thanks to this feedback our system is able to reject possible disturbances.

d) Design of a robust compensator under system disturbances:

```
%disturbance, if  $D(s)=1/s$ , then  $D(t)=1$ 
D=1;
%disturbed system
effect = Gp/(D+Gp*Gc_new);
%plotting step of the disturbed system
figure();
step(effect); xlim([0 0.25]);
title('Disturbance effect of  $D(s)=1/s$  for  $K_1=1$ ');
```



Response of the system is 100x smaller.

```
clear
clc
warning('off', 'all')
```

a)

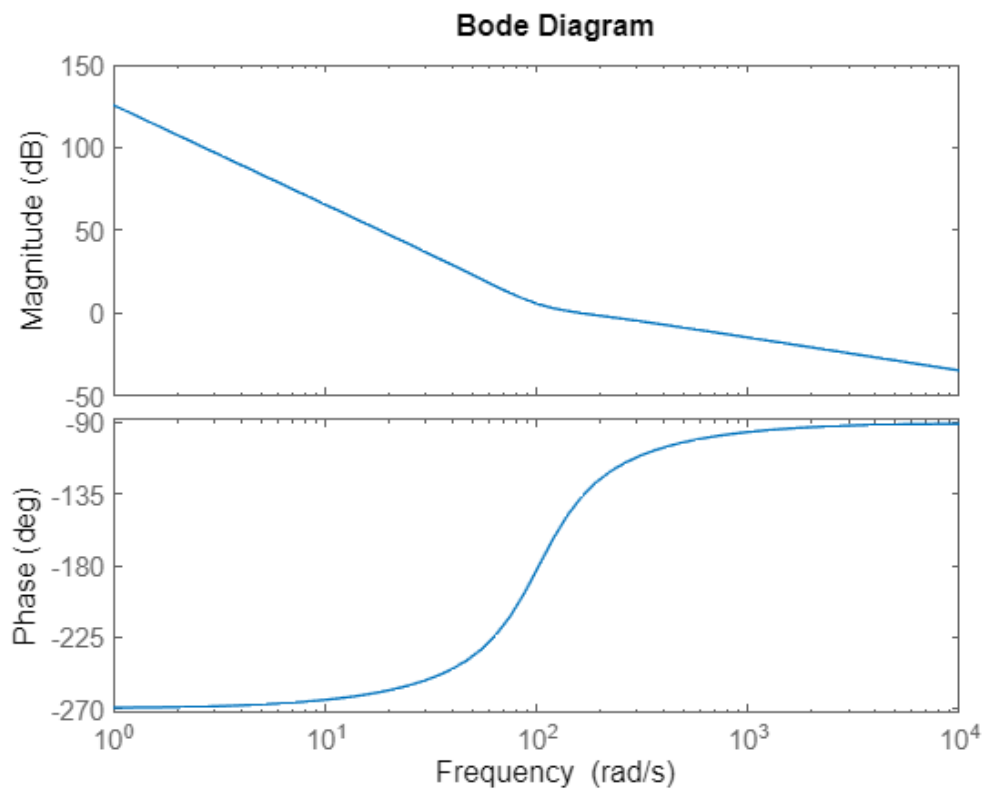
```
s = tf('s');
K1 = 5;
K2 = 500;
K3 = 0.0475;
Gc = K1 + K2/s + K3*s;
Ka = 1;
i=1;
while(1==1)
    sys = (10*Ka*Gc)/(s^2); % open loop transfer function
    [Gm,Pm] = margin(sys); % Gm is the gain margin, Pm is the phase margin.
    P_mat(i)=Pm;
    abs(42-Pm);
    if (abs(42-Pm) < 0.001)
        break;
    else
        K_mat(i)=Ka;
        i=i+1;
        Ka = Ka + 0.05;
    end
end
%displaying results
Gm_dB = 20*log10(Gm) %result in dB
```

```
Gm_dB = -5.1842
```

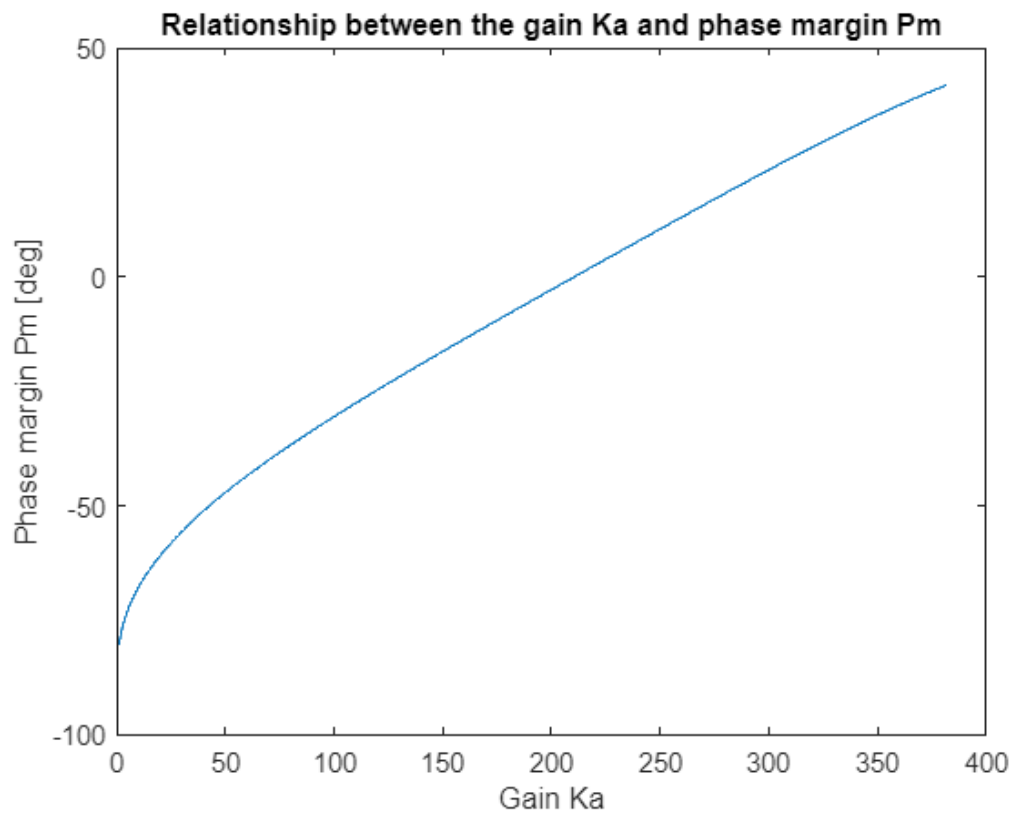
```
Ka
```

```
Ka = 382.4000
```

```
%plotting bode
figure(1)
bode(sys);
```

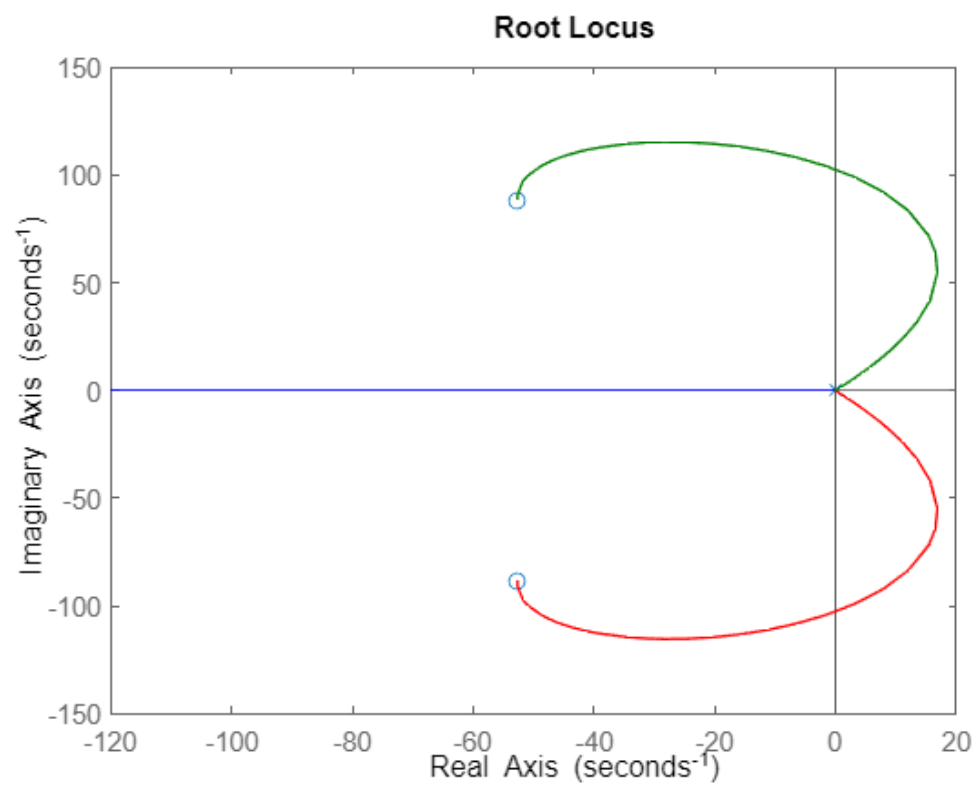


```
figure()
plot(K_mat,P_mat(:,1:7628))
title("Relationship between the gain Ka and phase margin Pm")
xlabel("Gain Ka")
ylabel("Phase margin Pm [deg]")
```

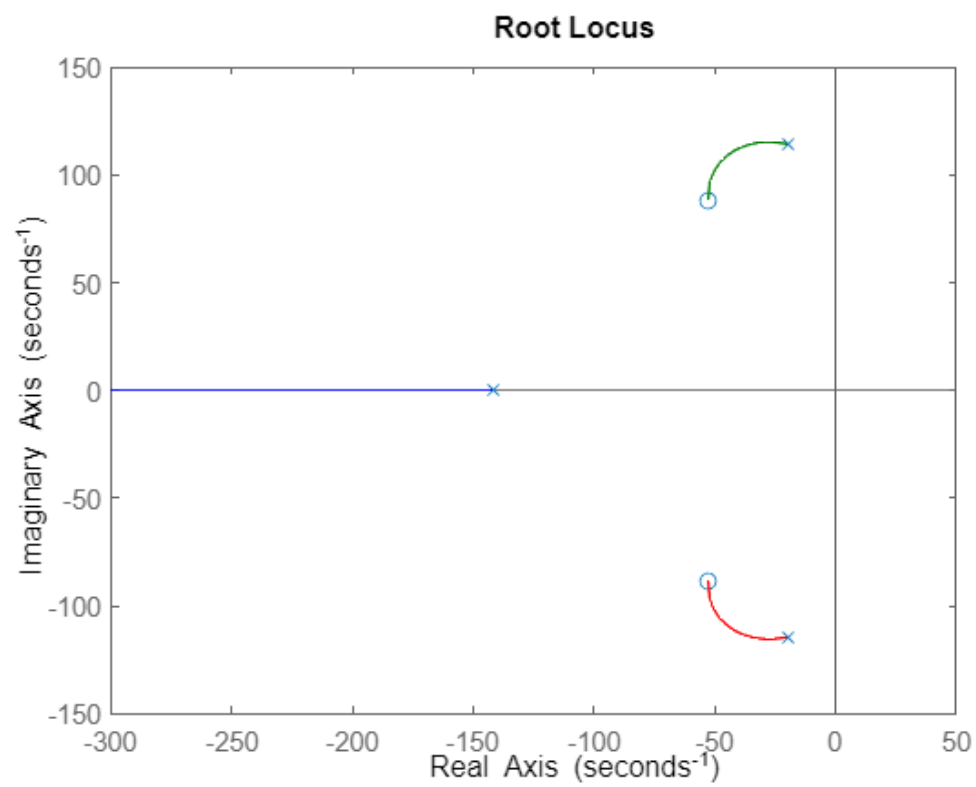


b)

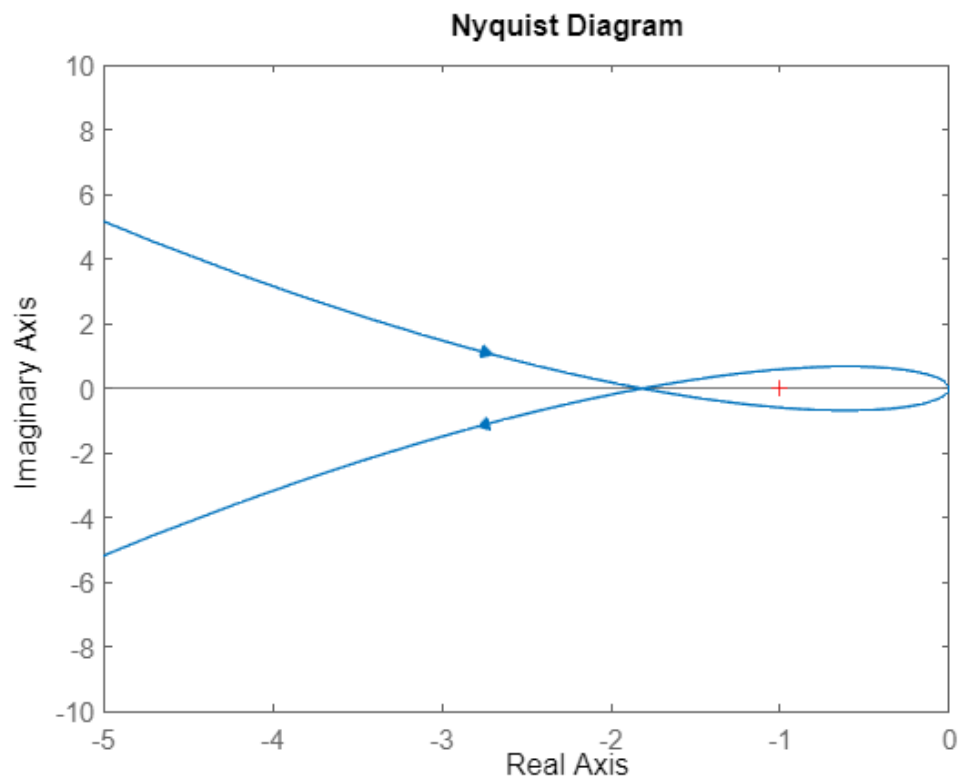
```
%defining values
K1 = 5;
K2 = 500;
K3 = 0.0475;
Ka=382.400;
Gc = K1 + K2/s + K3*s;
%creating syst
sys = (10*Ka*Gc/(s^2));
[Gm,Pm] = margin(sys);
sys_closed = feedback(sys,1);
%rlocus open sys
figure()
rlocus(sys);
```

```
%rlocus closed sys  
figure()  
rlocus(sys_closed);
```



```
roots = eig(sys_closed);  
% Nyquist plot  
figure()  
nyquist(sys)  
xlim([-5 0]);  
ylim([-10 10]);
```



```
%confirming results are correct
eig(sys_closed)
```

```
ans = 3x1 complex
102 x
-1.4187 + 0.0000i
-0.1989 + 1.1438i
-0.1989 - 1.1438i
```

c)

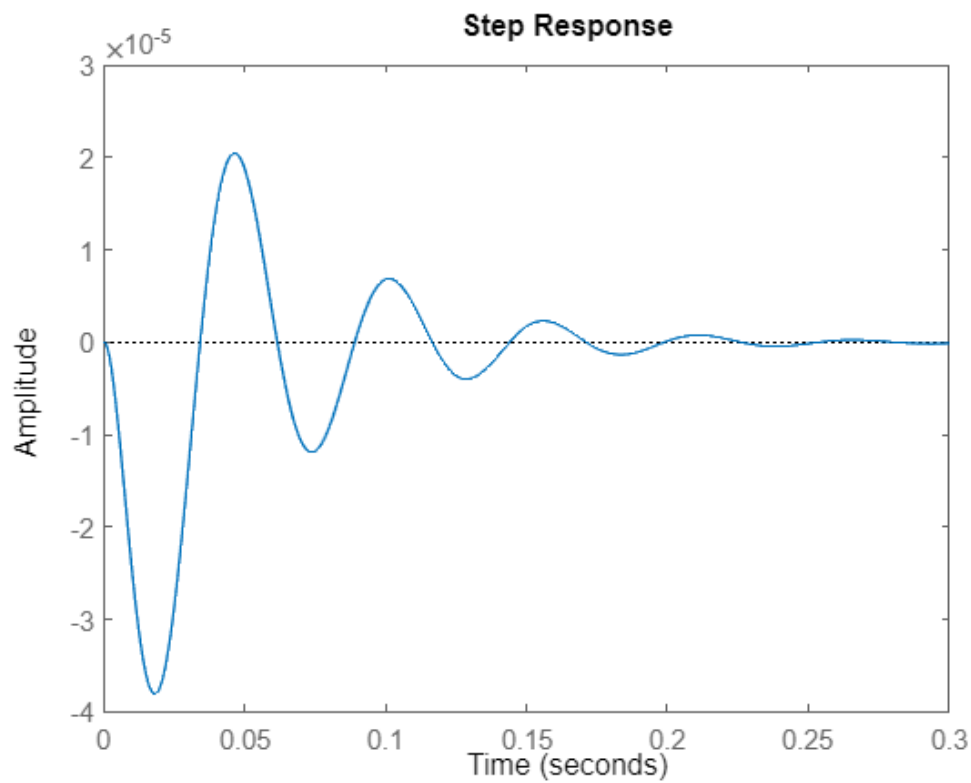
```
%creating system
YD = (-s)/(s^3 + 181.6*s^2 + 19120*s + 1912000)
```

```
YD =
```

$$\frac{-s}{s^3 + 181.6 s^2 + 19120 s + 1.912e06}$$

Continuous-time transfer function.

```
figure()
step(YD)
```

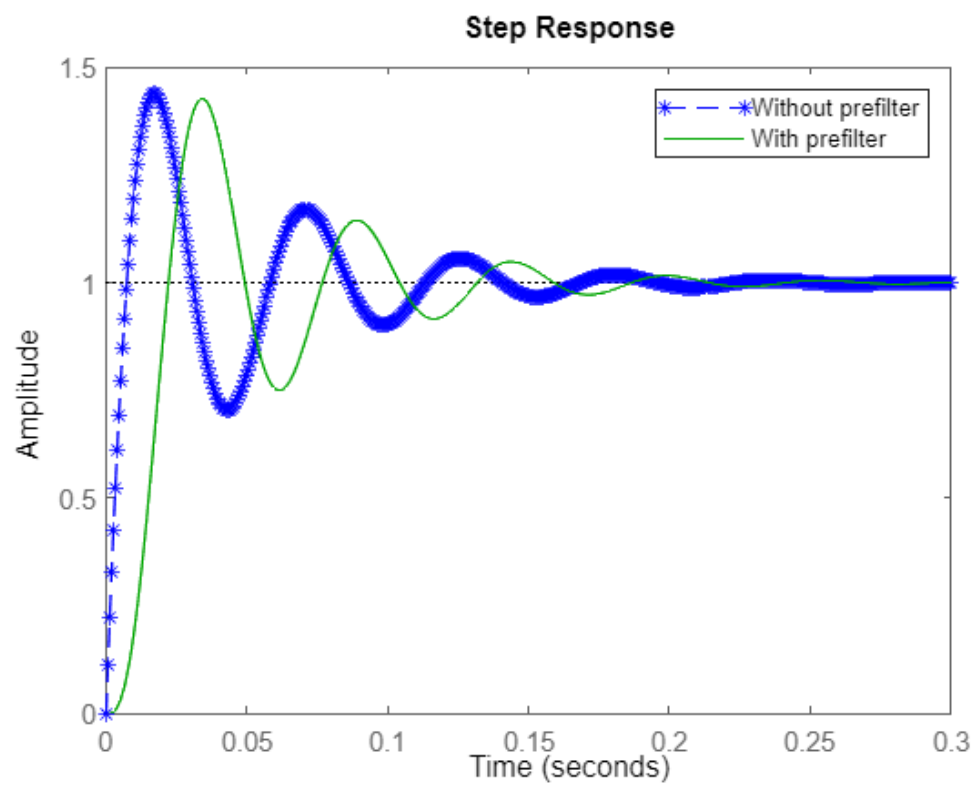


```
%checking what is max of Y(t)
SI=stepinfo(YD);
Max_Yt=SI.Peak
```

```
Max_Yt = 3.8054e-05
```

d)

```
s = tf('s');
K1 = 5;
K2 = 500;
K3 = 0.0475;
Gc = K1 + K2/s + K3*s;
Ka = 382.4000;
Gp = 1912000/(181.6*s^2+19120*s+1912000);
sys = (10*Ka*Gc/(s^2));
closed_sys = feedback(sys,1);
closed_sys_filter = Gp*closed_sys;
figure(1)
step(closed_sys,'b--*');
hold on
step(closed_sys_filter,'g');
legend("Without prefilter", "With prefilter");
```



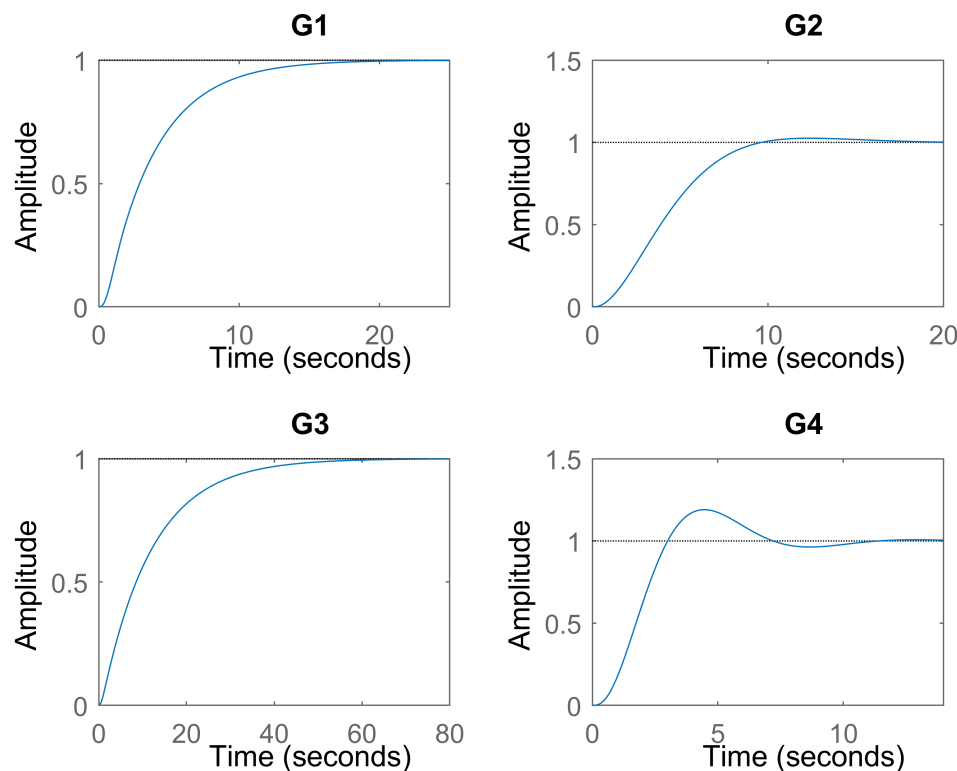
```
clear
clc
```

Step.1 Create systems

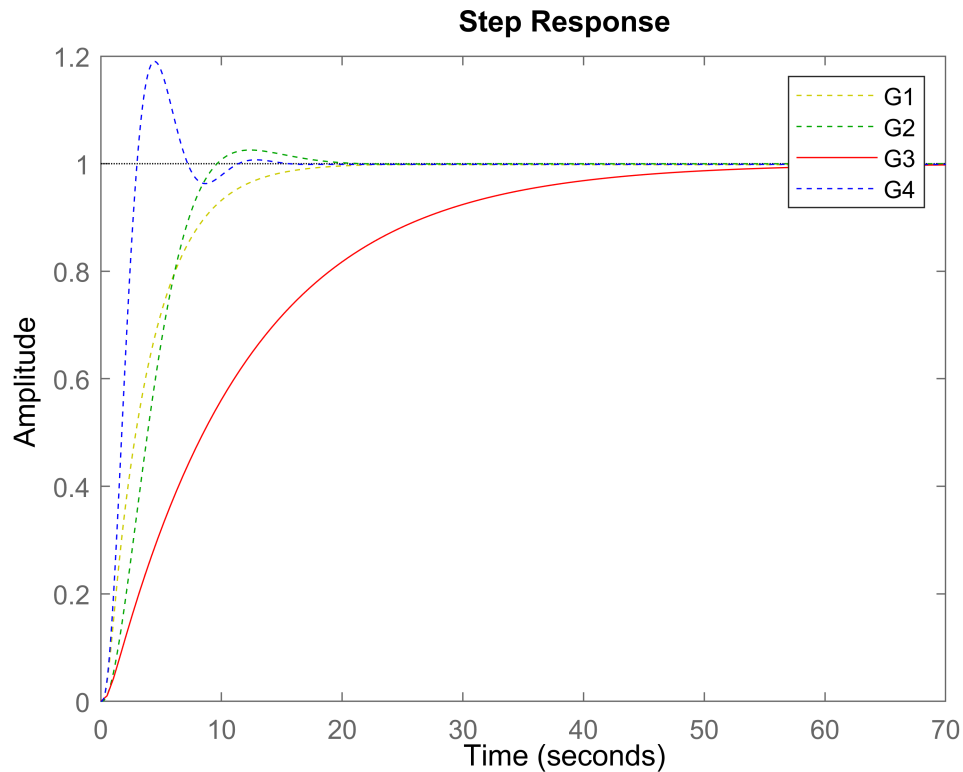
```
%transfer functions
s = tf('s');
G1 = 3/(s^3 + 5*s^2 + 12*s);
G2 = 1/(s^3 + 7*s^2 + 4*s);
G3 = 1/(s^3 + 7*s^2 + 12*s);
G4 = 3/(s^3 + 5*s^2 + 4*s);
```

Step.2 Determining polynomials representing worst case

```
%step responses separetly
figure();
subplot(2,2,1);
step(feedback(G1,1)); title('G1'); xlim([0 25]);
subplot(2,2,2);
step(feedback(G2,1)); title('G2'); xlim([0 20]); ylim([0 1.5]);
subplot(2,2,3);
step(feedback(G3,1)); title('G3'); xlim([0 80]);
subplot(2,2,4);
step(feedback(G4,1)); title('G4'); ylim([0 1.5]);
```



```
%plotting responses on one plot
figure();
step(feedback(G1,1),'y--'); hold on;
step(feedback(G2,1),'g--'); hold on;
step(feedback(G3,1),'r'); hold on;
step(feedback(G4,1),'b--'); hold on;
legend('G1', 'G2', 'G3', 'G4');
```



```
%saving characteristics of each plot
S1 = stepinfo(feedback(G1,1));
S2 = stepinfo(feedback(G2,1));
S3 = stepinfo(feedback(G3,1));
S4 = stepinfo(feedback(G4,1));
%preparing result table
Res=zeros(4);
Res(1,1)=S1.PeakTime;
Res(1,2)=S1.Overshoot;
Res(1,3)=S1.RiseTime;
Res(1,4)=S1.SettlingTime;

Res(2,1)=S2.PeakTime;
Res(2,2)=S2.Overshoot;
Res(2,3)=S2.RiseTime;
Res(2,4)=S2.SettlingTime;

Res(3,1)=S3.PeakTime;
```

```

Res(3,2)=S3.Overshoot;
Res(3,3)=S3.RiseTime;
Res(3,4)=S3.SettlingTime;

Res(4,1)=S4.PeakTime;
Res(4,2)=S4.Overshoot;
Res(4,3)=S4.RiseTime;
Res(4,4)=S4.SettlingTime;
%comparaision of the results (Peak time, Overshoot, RiseTime, SettlingTime
%from left to right
Res

```

```

Res = 4x4
    22.9920         0     7.7928    14.3607
    12.4206     2.5409     5.8843    14.4200
    83.4243         0    25.0431    45.1875
     4.4911    19.0146     1.9164    10.0848

```

Step.3 Designing a robust PID controller for the worst-case plant

a) Selecting natural frequency

```

%selecting natural frequency based on the equations
%beacouse of the condition that PeakTime response is max 2.5s, we choose omega = 2.5 rad/s
%additionally the condition is that omega >= 2.2 rad/s
wn = 2.5;

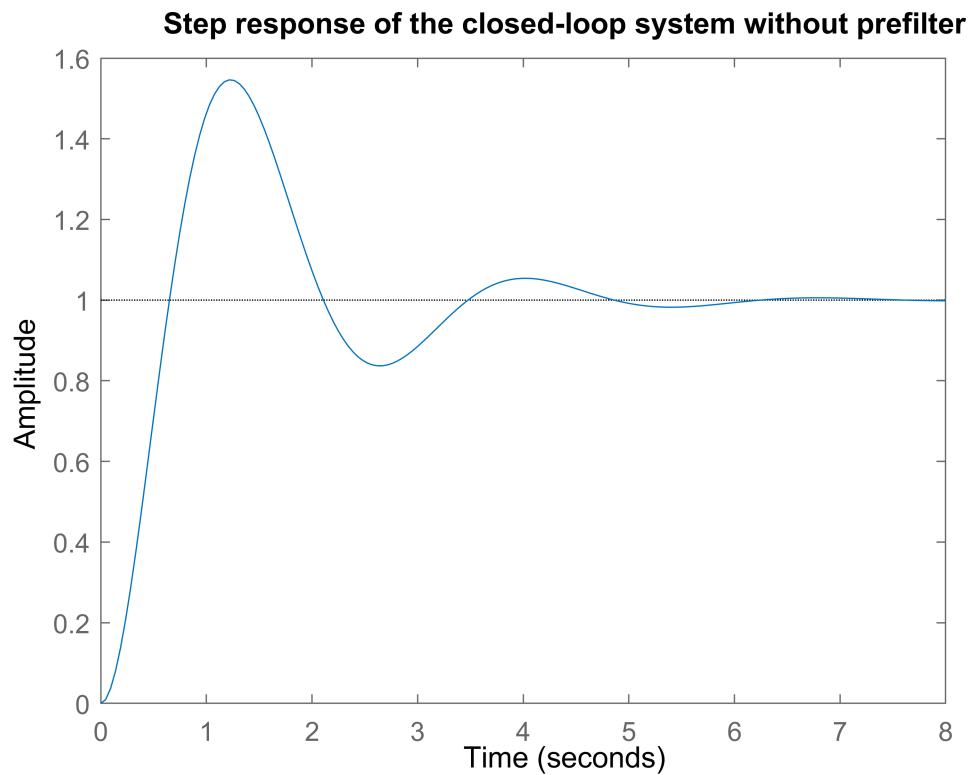
```

b) Determining PID controller parameters

```

%Gains of the PID controller from the table
Kp = 2.7*wn^3;
Ki = wn^4;
Kd = 3.4*wn^2 - 12;
Gc = Kp + Ki/s + Kd*s;
T1 = feedback(Gc*G3,1);
figure();
step(T1);
title('Step response of the closed-loop system without prefilter');

```

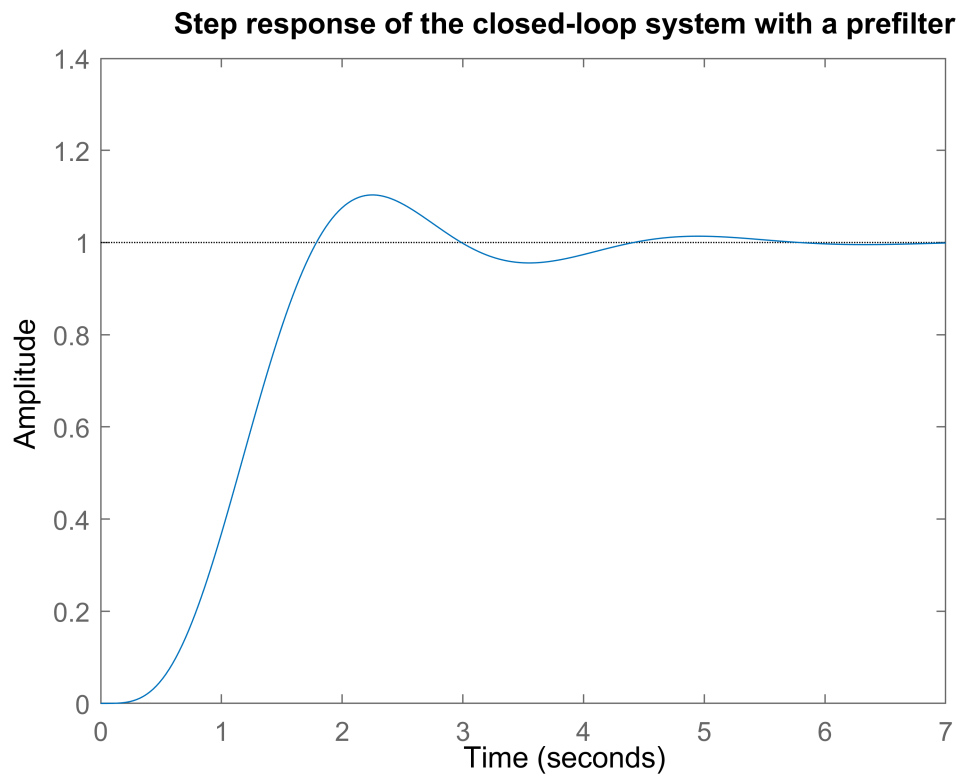
```
stepinfo(T1)
```

```
ans = struct with fields:
    RiseTime: 0.4408
    TransientTime: 4.6016
    SettlingTime: 4.6016
    SettlingMin: 0.8371
    SettlingMax: 1.5463
    Overshoot: 54.6283
    Undershoot: 0
    Peak: 1.5463
    PeakTime: 1.2178
```

%overshoot is too big so we add filter

c) Determining a prefilter $G_p(s)$ to remove all the zeros

```
Gp = Ki/(Kd*s^2 + Kp*s + Ki);
T1 = feedback(Gc*G3,1);
T = T1*Gp;
figure();
step(T); ylim([0 1.4]);
title('Step response of the closed-loop system with a prefilter');
```



```
stepinfo(T)
```

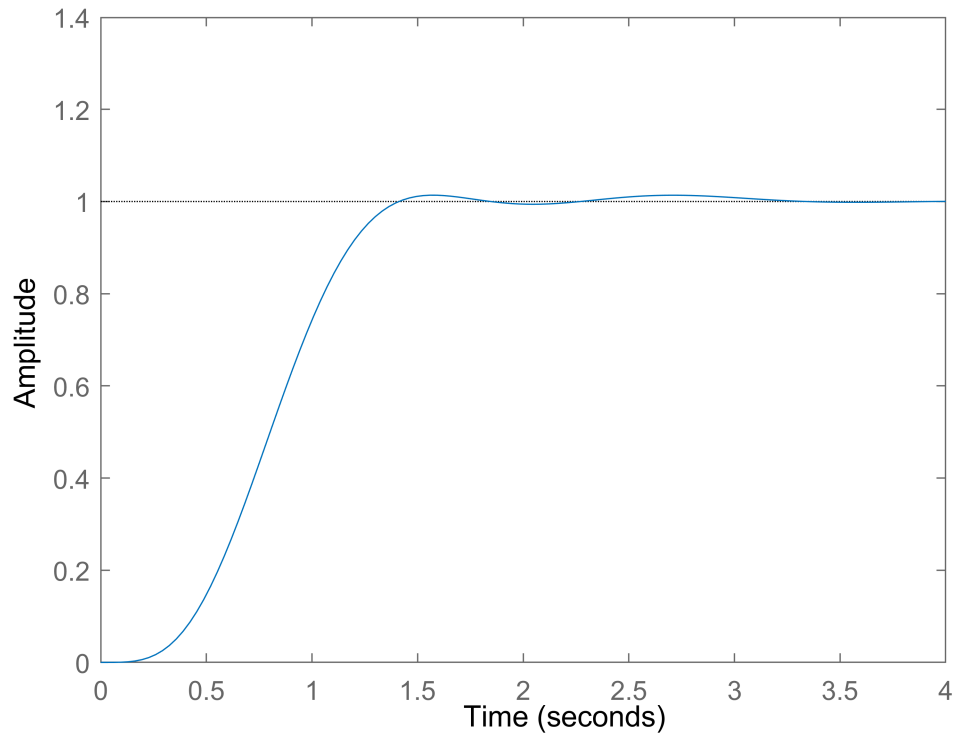
```
ans = struct with fields:
    RiseTime: 0.9897
    TransientTime: 4.0911
    SettlingTime: 4.0911
    SettlingMin: 0.9039
    SettlingMax: 1.1032
    Overshoot: 10.3238
    Undershoot: 0
    Peak: 1.1032
    PeakTime: 2.2399
```

ALL of this is the INITIAL assumption ($\omega_n = 2.5$), the optimal solution is found for $\omega_n = 3.4$.

Below we can see the plots of the step in such a case

```
wn = 3.4;
Kp = 2.7*wn^3;
Ki = wn^4;
Kd = 3.4*wn^2 - 12;
Gc = Kp + Ki/s + Kd*s;
Gp = Ki/(Kd*s^2 + Kp*s + Ki);
T1 = feedback(Gc*G3,1);
T = T1*Gp;
figure();
step(T); ylim([0 1.4]);
title('Step response of the closed-loop system with correct prefilter');
```

Step response of the closed-loop system with correct prefilter

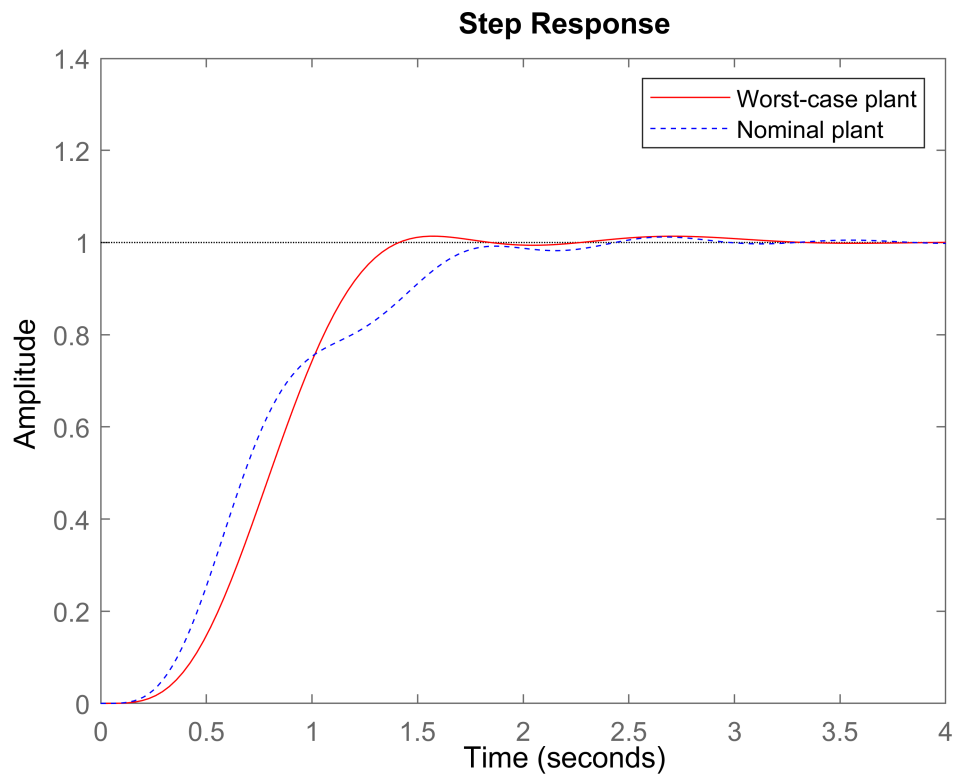


```
stepinfo(T)
```

```
ans = struct with fields:
    RiseTime: 0.7362
    TransientTime: 1.3349
    SettlingTime: 1.3349
    SettlingMin: 0.9168
    SettlingMax: 1.0137
    Overshoot: 1.3704
    Undershoot: 0
    Peak: 1.0137
    PeakTime: 1.5591
```

Now we have to compare step responses for the closed loop prefiltered system obtained from $G_3(s)$ and from nominal plant ($K=2$, $p=2$).

```
K = 2;
p = 2;
G_nominal = K/(s*(s+p)*(s+4));
T1_nominal = feedback(Gc*G_nominal,1);
T_nominal = T1_nominal*Gp;
figure();
step(T, 'r'); ylim([0 1.4]); hold on;
step(T_nominal, 'b--');
legend('Worst-case plant', 'Nominal plant');
```



```
stepinfo(T_nominal)
```

```
ans = struct with fields:
    RiseTime: 1.1125
    TransientTime: 1.7210
    SettlingTime: 1.7210
    SettlingMin: 0.9052
    SettlingMax: 1.0120
    Overshoot: 1.2030
    Undershoot: 0
    Peak: 1.0120
    PeakTime: 2.6582
```

Step.4 Test the designed robust PID controller with the prefilter $G_p(s)$ for the four special conditions.

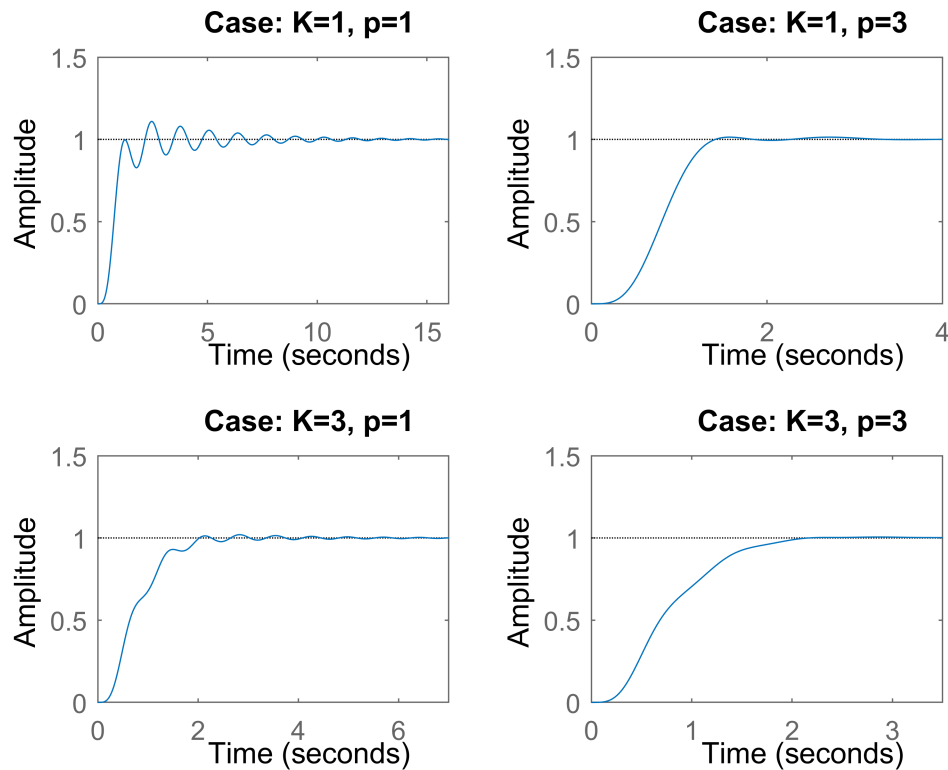
a) $K=1$ $p=1$ b) $K=1$ $p=3$ c) $K=3$ $p=1$ d) $K=3$ $p=3$

```
%First case K=1, p=1
C1 = 1/(s*(s+1)*(s+4));
T_C1 = feedback(Gc*C1,1);
T1 = T_C1*Gp;
%Second case K=1, p=3
C2 = 1/(s*(s+3)*(s+4));
T_C2 = feedback(Gc*C2,1);
T2 = T_C2*Gp;
%Third case K=3, p=1
C3 = 3/(s*(s+1)*(s+4));
T_C3 = feedback(Gc*C3,1);
```

```

T3 = T_C3*Gp;
%Fourth case K=3, p=3
C4 = 3/(s*(s+3)*(s+4));
T_C4 = feedback(Gc*C4,1);
T4 = T_C4*Gp;
%plotting results
figure();
subplot(2,2,1); step(T1); title('Case: K=1, p=1'); ylim([0 1.5]);
subplot(2,2,2); step(T2); title('Case: K=1, p=3'); ylim([0 1.5]);
subplot(2,2,3); step(T3); title('Case: K=3, p=1'); ylim([0 1.5]);
subplot(2,2,4); step(T4); title('Case: K=3, p=3'); ylim([0 1.5]);

```



```

%saving results
S1 = stepinfo(T1);
S2 = stepinfo(T2);
S3 = stepinfo(T3);
S4 = stepinfo(T4);
%preparing result table
Res2=zeros(4);
Res2(1,1)=S1.PeakTime;
Res2(1,2)=S1.Overshoot;
Res2(1,3)=S1.RiseTime;
Res2(1,4)=S1.SettlingTime;

Res2(2,1)=S2.PeakTime;
Res2(2,2)=S2.Overshoot;
Res2(2,3)=S2.RiseTime;

```

```

Res2(2,4)=S2.SettlingTime;

Res2(3,1)=S3.PeakTime;
Res2(3,2)=S3.Overshoot;
Res2(3,3)=S3.RiseTime;
Res2(3,4)=S3.SettlingTime;

Res2(4,1)=S4.PeakTime;
Res2(4,2)=S4.Overshoot;
Res2(4,3)=S4.RiseTime;
Res2(4,4)=S4.SettlingTime;
%comparaision of the results (Peak time, Overshoot, RiseTime, SettlingTime
%from left to right
Res2

```

```

Res2 = 4x4
    2.4587    10.9336    0.6056    8.4515
    1.5591     1.3704    0.7362    1.3349
    2.8249     1.9885    1.0267    2.5166
    2.8475     0.5572    1.0694    1.9197

```

```

%1st system
num1=[3.333 0 0];
den1=[1 0 -42.51 0 0];
G1s=tf(num1,den1)

```

```

G1s =

      3.333 s^2
      -----
      s^4 - 42.51 s^2

```

Continuous-time transfer function.

```

%2nd system
num2=[1 0 -32.7];
den2=[1 0 -42.51 0 0];
G2s=tf(num2,den2)

```

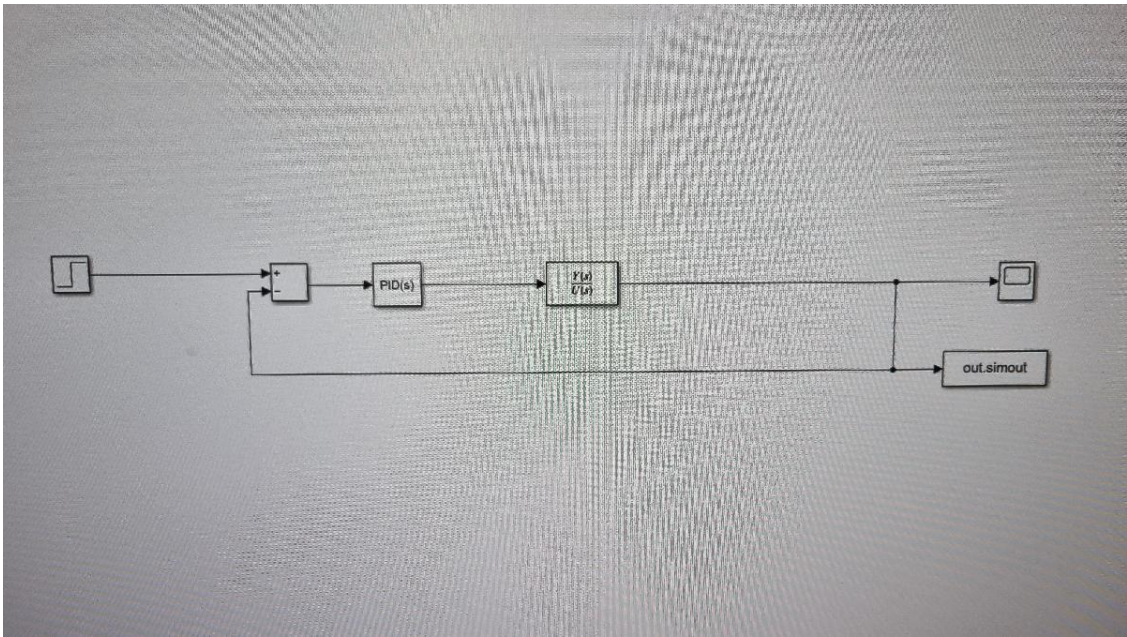
```

G2s =

      s^2 - 32.7
      -----
      s^4 - 42.51 s^2

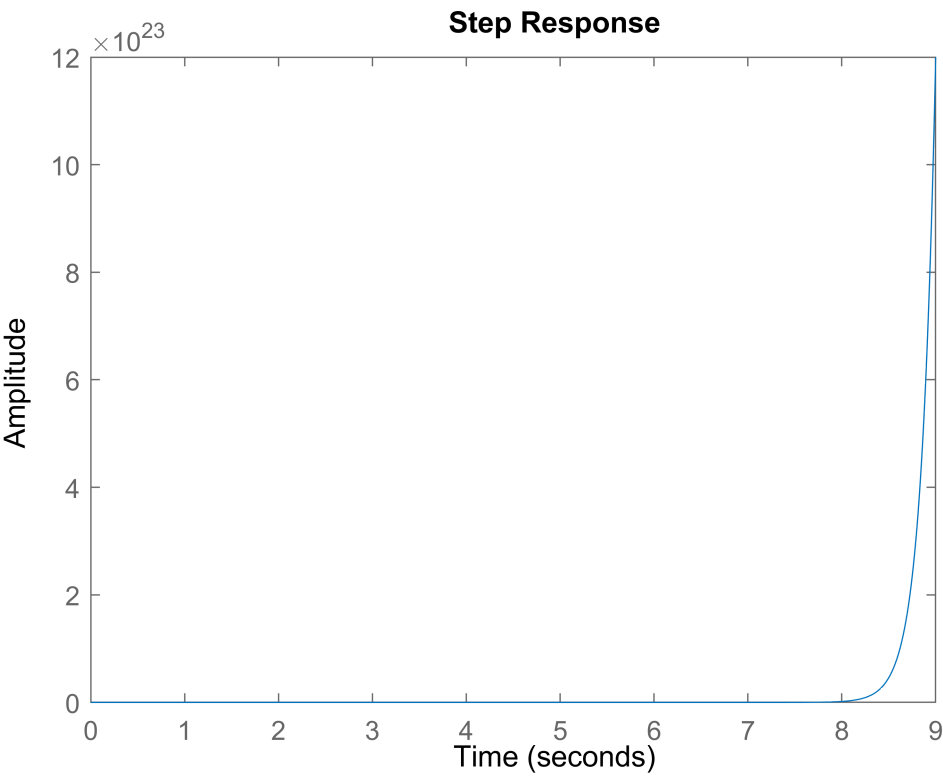
```

Continuous-time transfer function.

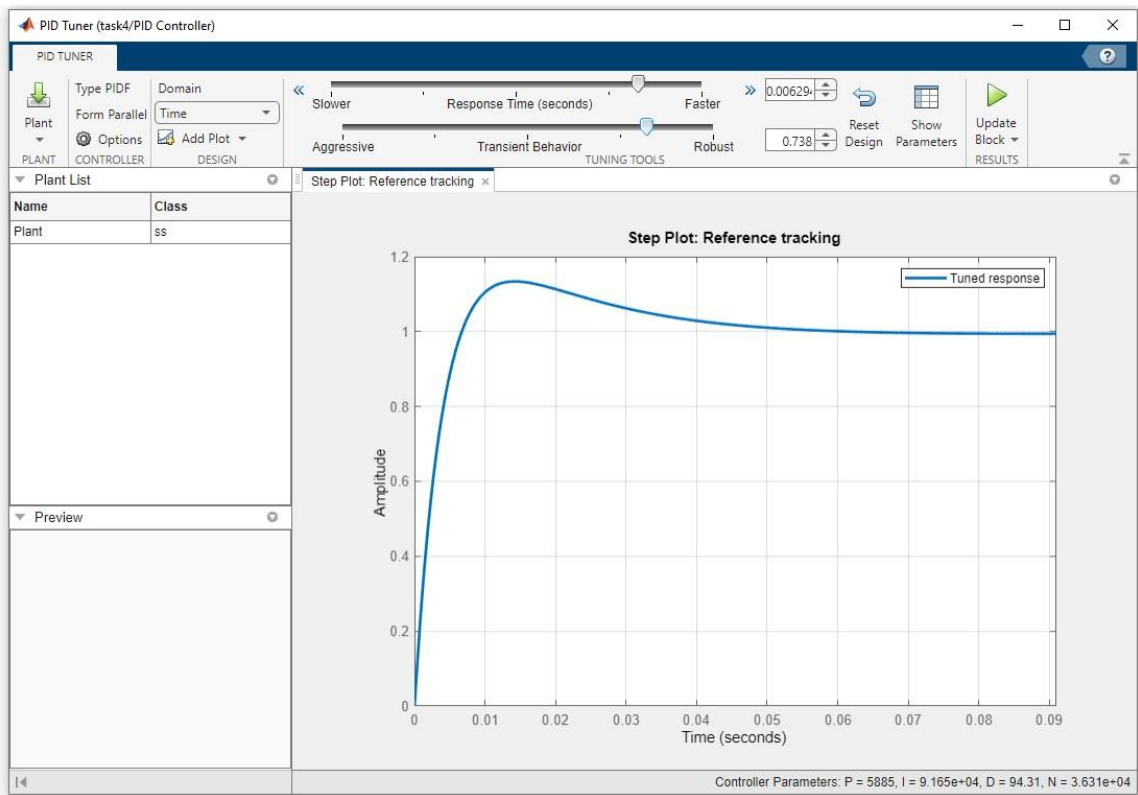


The step response of system before PID tuning:

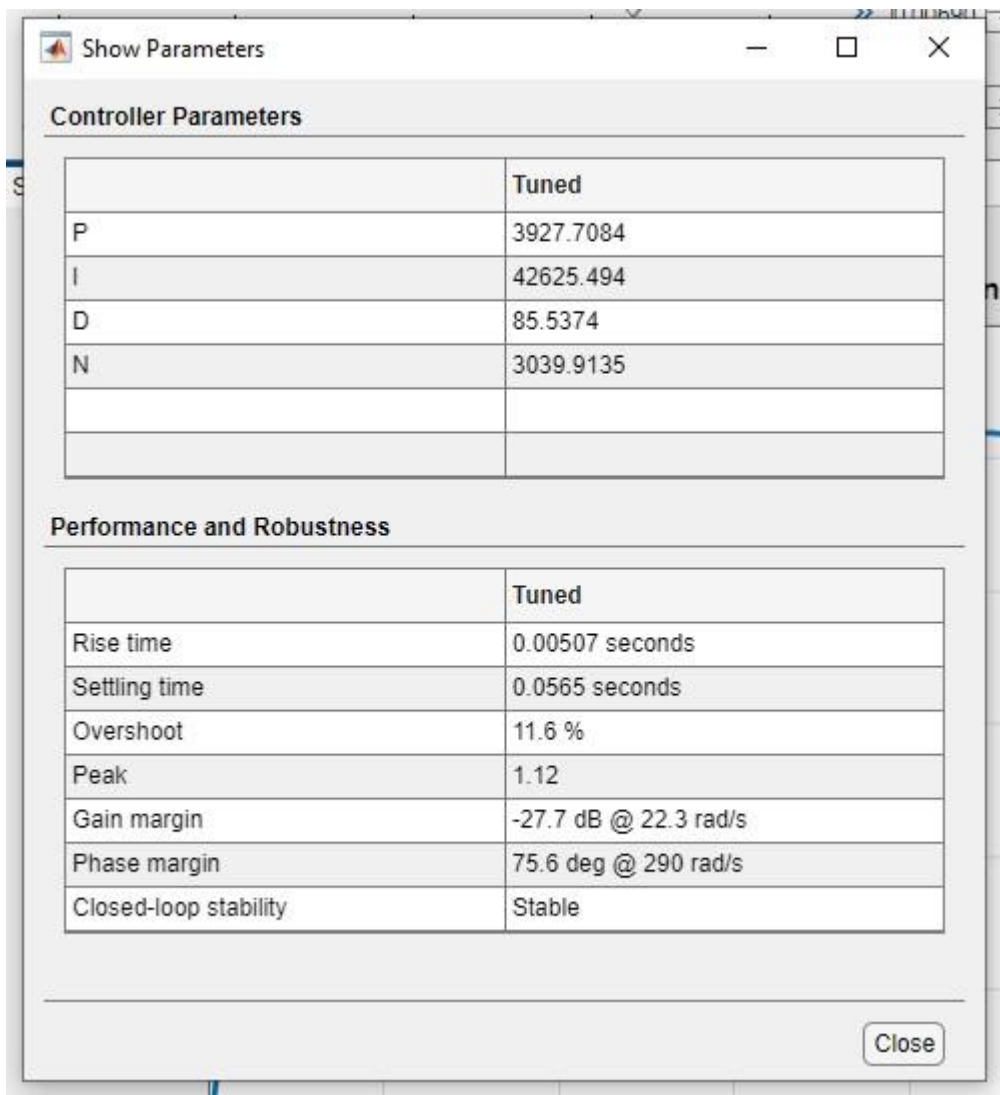
```
figure();
step(G1s)
```



PID tuner setup:



PID parameters and step response characteristics for step response of the tuned system:



PID tuner is a great tool used for stabilizing system by a PID controller. We are given a real time step plot and step characteristics for changing two parameters: Response time and Transient behavior.