Jawaharlal Nehru Engineering College

Laboratory Manual

CONTROL SYSTEM ENGINEERING

For

Third Year EEP Students

Made by

Prof.C.B.Ingole

PREFACE

It is my great pleasure to present interactive Control System Engineering (CSE) Demonstration modules developed for second year Electrical Engineering students.

Hence, this manual consists of a ready to use set of demonstrations, illustrating the control system concepts, that can be beneficial to the students . Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

SUBJECT INDEX

Title

1. DOs and DON'Ts in Laboratory:

Do not handle different kit without reading the instructions/Instruction manuals
1. Go through through the procedure and precautions given in manual.
2. Strictly observe the instructions given by the teacher/Lab Instructor.

2.Instruction for Laboratory Teachers::

1. Lab work completed during prior session ,should be corrected during the next lab session.

2. Students should be guided and helped whenever they face difficulties.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

3. Lab Exercises :

1. To create series connection of two transfer functions.
2. To create parallel connection of two transfer functions.
3. To determine the values of transient specifications for step input.
4. To plot pole zero map and determination of stability.
5. To plot root locus and determination of stability.
6. To plot polar plot and determination of stability.
7. To plot Bode plot and calculation of Wpc and Wgc.
8. To design Proportional- Integral- Derivative Controller.
9. To study synchros transmitter and receiver.
10. To study transfer function of AC and DC servomotor.

Experiment1.

Aim : To create series connection of two transfer functions.

Program :

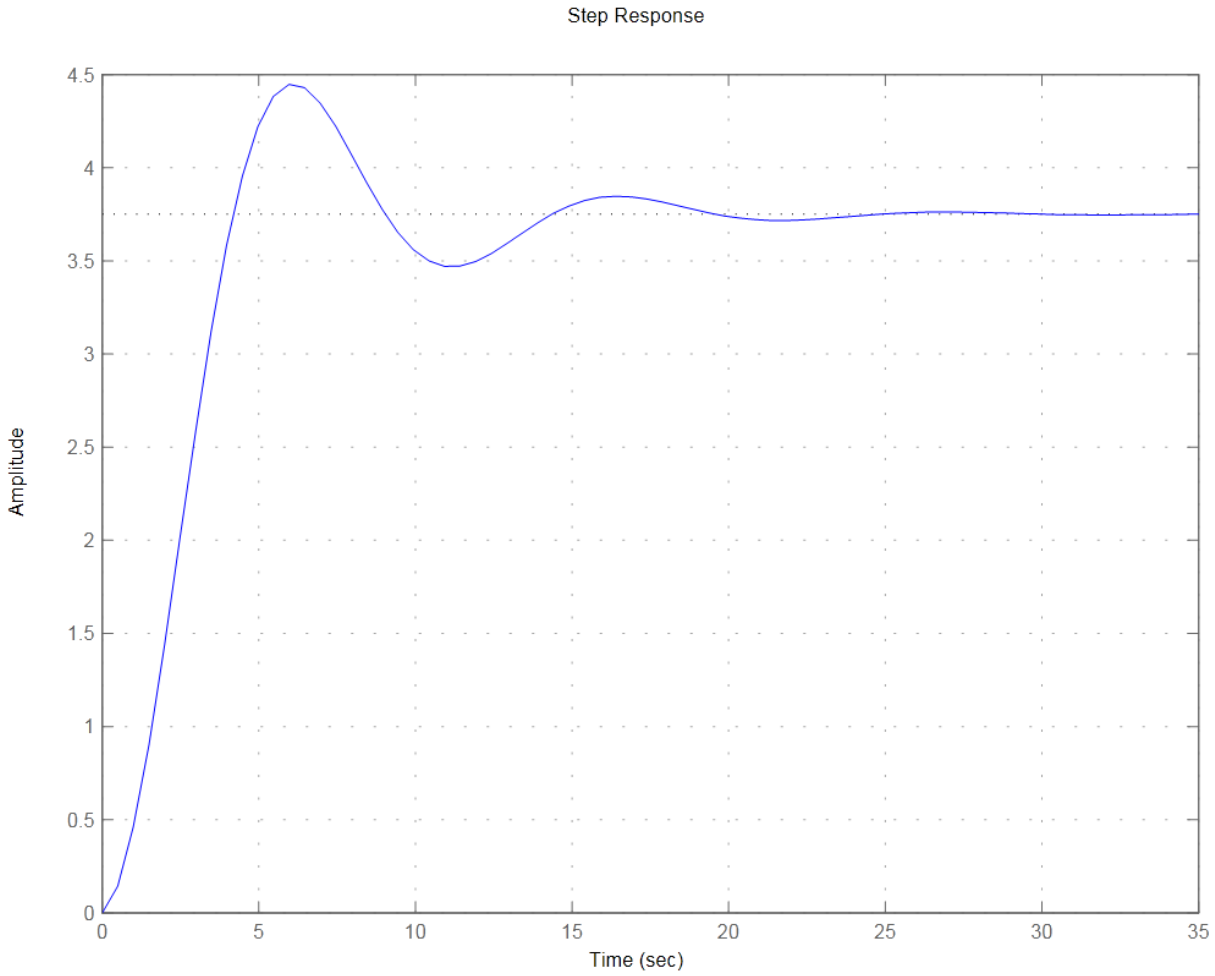
```
>> %Series connection of Two blocks

G1=tf([0 2 3],[5 2 2])

%continuous-time transfer function

Transfer function: 2 s + 3 = 5 s^2 + 2 s + 2

 >> G2=zpk(-2,[-0.5 -8],5)

%continuous-time zero/pole/gain

Zero/pole/gain: 5 (s+2) = (s+0.5) (s+8)

 >> Tseries=G1*G2 Zero/pole/gain: 2 (s+1.5) (s+2) = (s+0.5) (s+8) (s^2 + 0.4s + 0.4)

>> G1_zeros=zero(G1)

G1_zeros = -1.5000

 >> G1_poles=pole(G1)

G1_poles = -0.2000 + 0.6000i -0.2000 - 0.6000i

>> G2_zeros=zero(G2) G2_zeros = -2

 >> G2_poles=pole(G2) G2_poles = -0.5000 -8.0000

 >> [Tseries_zeros, Tseries_poles, Tseries_gain]zpkdata(Tseries,'v')


 [Tseries_zeros, Tseries_poles, Tseries_gain]zpkdata(Tseries,'v') |

>> [Tseries_zeros, Tseries_poles, Tseries_gain]=zpkdata(Tseries,'v')

Tseries_zeros = -1.5000 -2.0000

Tseries_poles = -0.2000 + 0.6000i -0.2000 - 0.6000i -0.5000 -8.0000
```

Tseries_gain = 2

>> step(Tseries)

>> hold on

>> grid on



Experiment 2.

Aim : To create parallel connection of two transfer functions.

Program :

>> %Parallel Connection of Two Block

>> G1=tf([0 2 3],[5 2 2]) Transfer function: 2 s + 3= 5 s^2 + 2 s + 2

>> G2=zpk(-2,[-0.5 -8],5)

Zero/pole/gain: 5 (s+2) = (s+0.5) (s+8)

>> Tpar=G1+G2

Zero/pole/gain: 5.4 (s+2.116) (s^2 + 0.8467s + 0.56) = (s+0.5) (s+8) (s^2 + 0.4s + 0.4)

>> G1_zeros=zero(G1) G1_zeros = -1.5000

>> G1_poles=pole(G1) G1_poles = -0.2000 + 0.6000i -0.2000 - 0.6000i

>> G2_zeros=zero(G2) G2_zeros = -2

>> G2_poles=pole(G2) G2_poles = -0.5000 -8.0000

>> [Tpar_zeros, Tpar_poles, Tpar_gain]=zpkdata(Tpar,'v')

Tpar_zeros = -0.4233 + 0.6171i -0.4233 - 0.6171i -2.1163

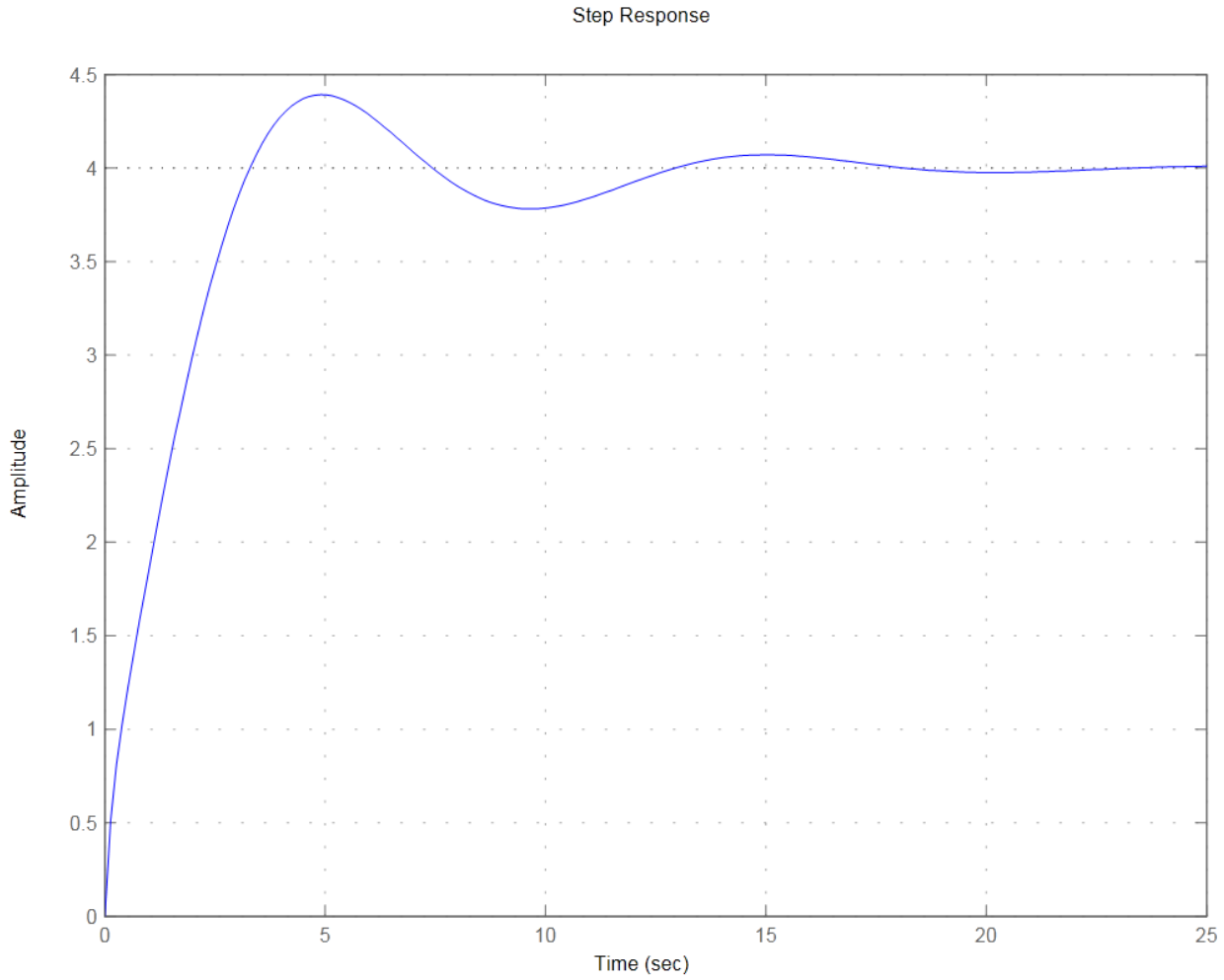
Tpar_poles = -0.2000 + 0.6000i -0.2000 - 0.6000i -0.5000 -8.0000

Tpar_gain = 5.4000

>> step(Tpar)

>> hold on

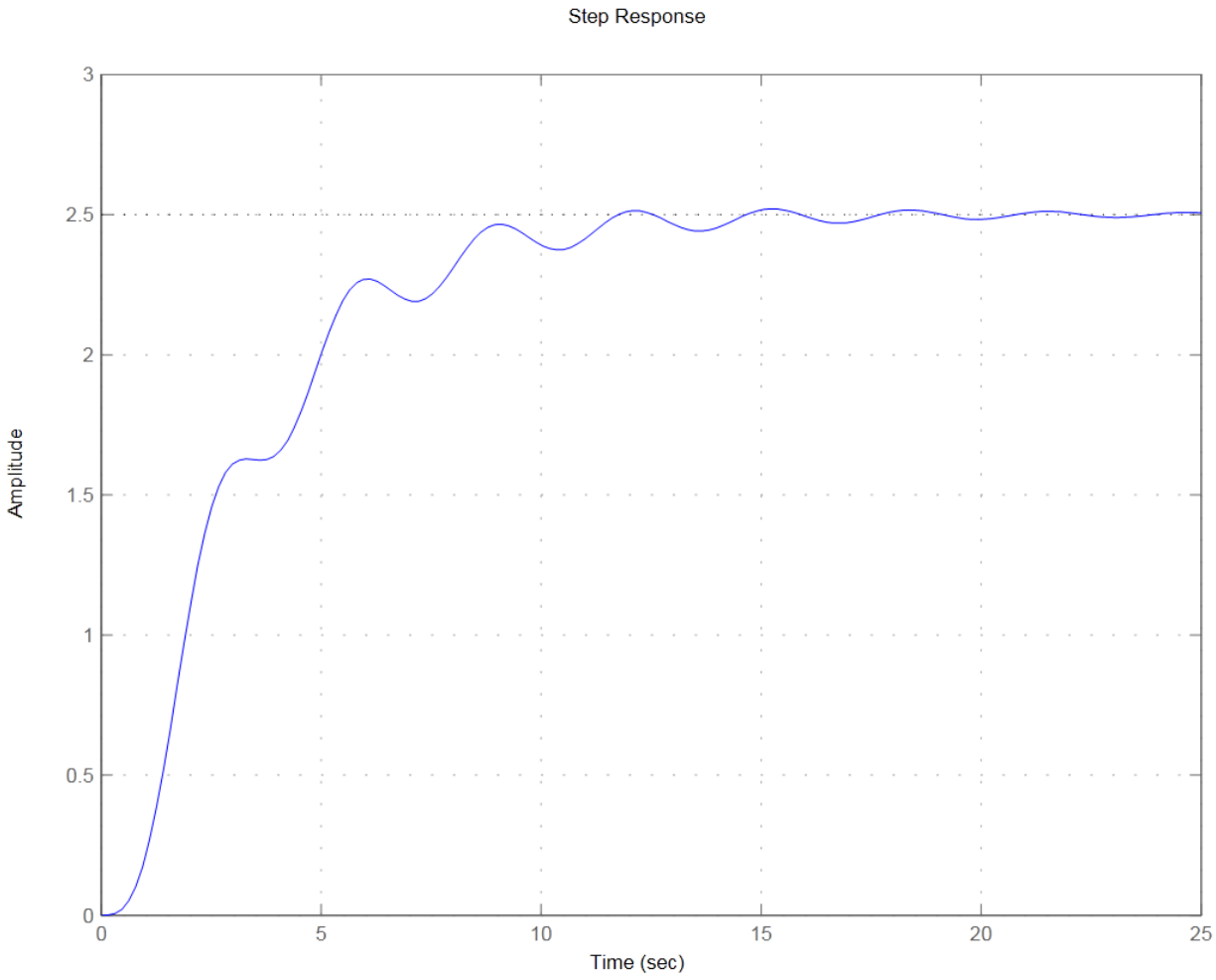
>> grid on


Step Response
Amplitude
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
5
10
15
20
25

Experiment No.3

Aim : Obtaining Rise Time, Peak Time, Maximum Overshoot, & Setting Time with MATLAB.

```
%Step response characteristics for the system

>> sys = tf([1 5],[1 2 5 7 2])

Transfer function: s + 5 = s^4 + 2 s^3 + 5 s^2 + 7 s + 2

>> S = stepinfo(sys,'RiseTimeLimits',[0.05,0.95])

S = RiseTime: 7.4514

SettlingTime: 13.9349

SettlingMin: 2.3739

SettlingMax: 2.5203

 Overshoot: 0.8108

Undershoot: 0

 Peak: 2.5203

PeakTime: 15.2243

>> step(sys);

hold on;

grid on
```


Step Response
Amplitude
Time (sec)
0
0.5
1
1.5
2
2.5
3
0
5
10
15
20
25

Experiment No. 4

Aim : To plot pole zero map and determination of stability.

>> %To draw pole-zero plot

>> H = tf([2 5 1],[1 3 5])

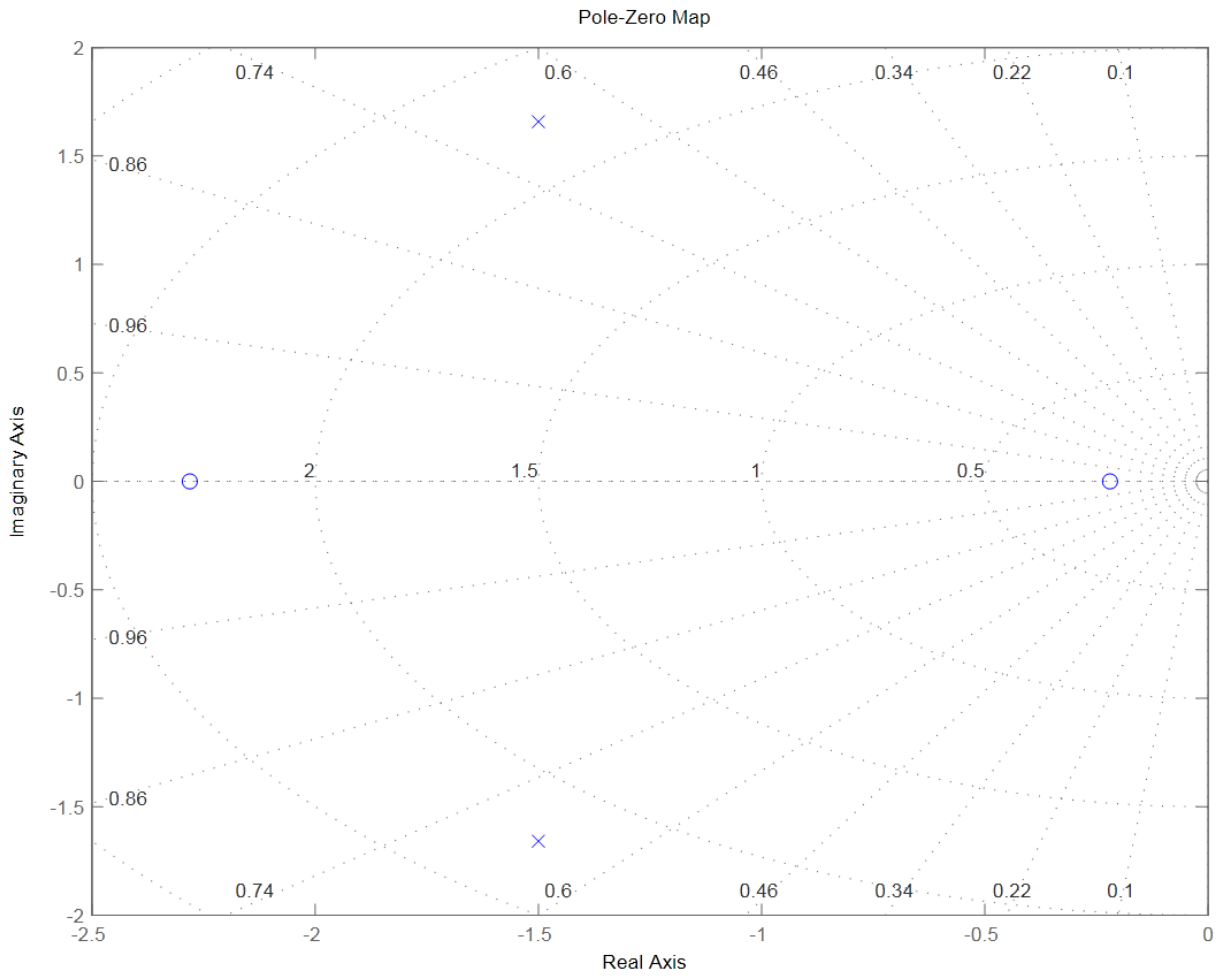
Transfer function: 2 s^2 + 5 s + 1 = s^2 + 3 s + 5

>> Z=zero(H) Z = -2.2808 -0.2192

>> P=pole(H) P = -1.5000 + 1.6583i -1.5000 - 1.6583i

>> pzmap(H)

>> grid on



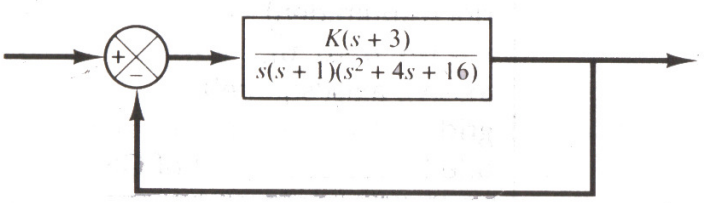
Experiment No.5

Aim : Plotting root locus using MATLAB

Consider the system shown in figure 6. plot root loci with a square aspect ratio so that a line with slope 1 is a true 45 line. Choose the region of root-locus plot to be

$$-6 \le x \le 6, \qquad -6 \le x \le 6$$

where x & y are the real-axis coordinate & imaginary-axis coordinate, respectively.



To set the given plot region on the screen to be square, enter the command

V = [-6 6 -6 6]; axis (v); axis('square')

With this command, the region of the plot is as specified & the line with slope 1 is at a true 45^0, not skewed by the irregular shape of the screen.

For this problem, the dominator is given as a product as first and second order terms. So we must multiply this terms to get polynomial in *s*. the multiplication of this terms can be done easily by use of the convolution command as shown next.

Define :

A = s (s + 1); a = [1 1 0]

B = s^2 + 4s + 16; b = [1 4 16]

Than we use the following command:

C = conv(a, b)

Note the conv(a, b) gives the product of two polynomial a & b. see the following computer output:

```
a = [1   1   0];
b = [1   4   16];
c = conv (a, b)
c =
    1 5  20  16  0
```

The denominator polynomials is thus found to be

den = [1 5 20 16 0]

to find the complex-conjugate open-loop poles (the root of s^2 + 4s. + 16 = 0), we may enter the root command as follows:

```
R = root(b)
R =
     -2.0000 + 3.464li
     -2.0000 - 3.464li
```

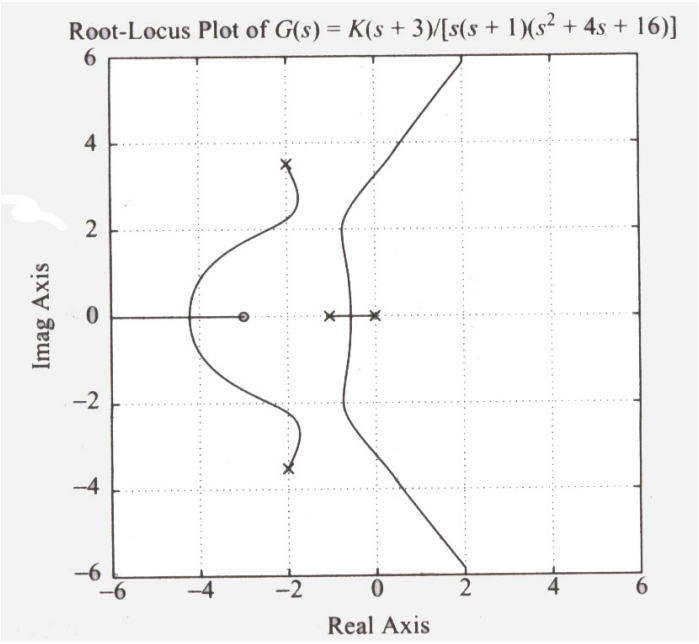
Thus, the system has the following open loop zero & open loop poles:

Open loop zero: $s = -3$

Open loop zero: $s = 0, \quad s = -1, \quad s = -2 \pm j3.4641$

%-------- Root-locus plot -------

```
num = [0  0  0  1  3];
den = [1  5  20  16  0];
rlocus (num, den)
v = [-6  6  -6  6];
axis(v); axis('square')
grid;
title ('Root-Locus Plot of G(s) = K(s + 3)/[s(s + 1)(s^2 + 4s + 16)]')
```

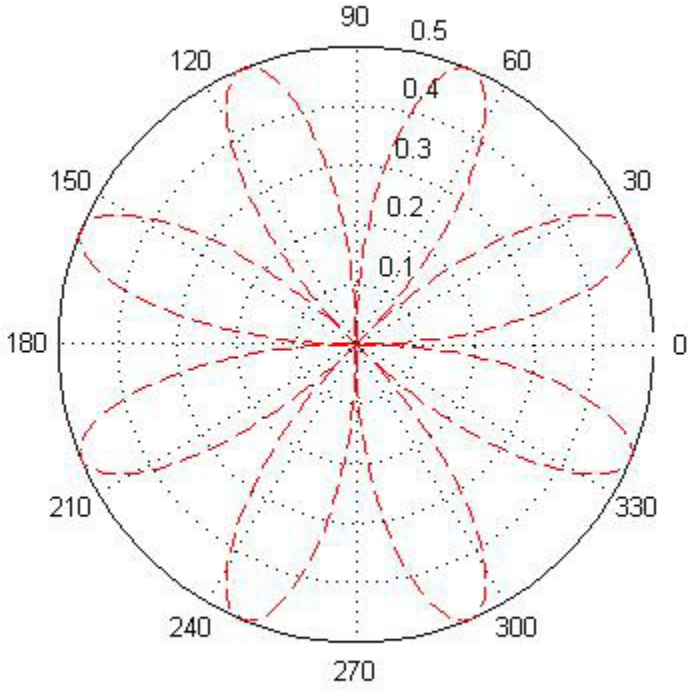


Experiment 6

Aim : To obtain polar plot and determination of stability.

Create a simple polar plot using a dashed red line:

```
t = 0:.01:2*pi;
polar(t,sin(2*t).*cos(2*t),'--r')
```

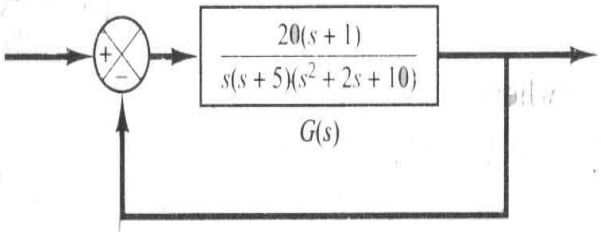


Experiment : 7

Aim : To plot bode plot and determination of Wgc and Wpc.

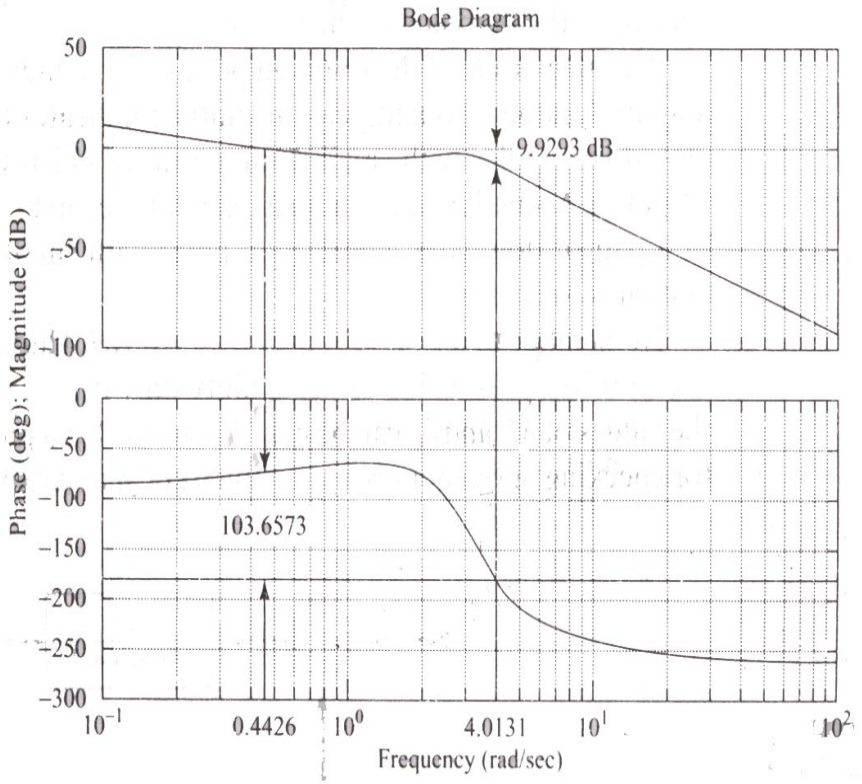
Draw a Bode diagram of the open-loop transfer function $G(s)$ of the closed-loop system shown in Figure 13 Determine the gain margin, phase margin, phase-crossover frequency, and gain-crossover frequency with MATLAB.

A MATLAB program to plot a Bode diagram and to obtain the gain margin, phase margin, phase-crossover frequency, and gain-crossover frequency is shown in MATLAB Program 13 The Bode diagram of $G(s)$ is shown in Figure



MATLAB Program 13

```
num = [0  0  0  20  20];
den = conv([1  5  0],[1  2  10]);
sys = tf(num,den);
w = logspace(-1,2,100);
bode(sys,w)
[Gm,pm,wcp,wcg] = margin(sys);
GmdB = 20*log10(Gm);
[GmdB  pm  wcp  wcg]

ans =

   9.9293  103.6573  4.0131  0.4426
```

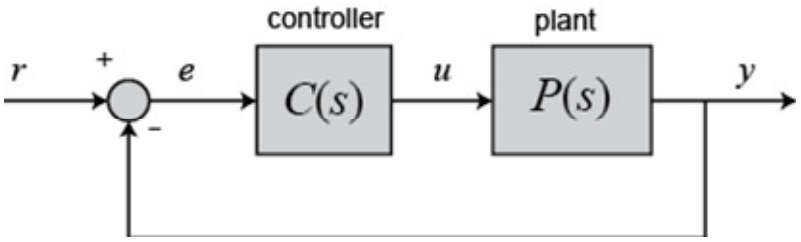

Bode Diagram
50
0
−50
−100
9.9293 dB
0
−50
−100
−150
−200
−250
−300
103.6573
Phase (deg); Magnitude (dB)
10^{-1}
0.4426
10^{0}
4.0131
10^{1}
10^{2}
Frequency (rad/sec)

Experiment 8.

Aim : To design Proportional- Integral- Derivative Controller.

Key MATLAB commands used are: `tf` , `step` , `pid` , `feedback` , `pidtool` , `pidtune`

In this , we will consider the following unity feedback system:



The transfer function of a PID controller is found by taking the Laplace transform of Eq.(1).

(2) $$K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

K_p= Proportional gain K_i= Integral gain K_d= Derivative gain

We can define a PID controller in MATLAB using the transfer function directly, for example:

```
Kp = 1;
Ki = 1;
Kd = 1;

s = tf('s');
C = Kp + Ki/s + Kd*s
C =

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

Continuous-time transfer function.
```

Alternatively, we may use MATLAB's **pid controller object** to generate an equivalent contin

```
C = pid(Kp,Ki,Kd)
C =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = 1, Ki = 1, Kd = 1
```

The control signal (u) to the plant is equal to the proportional gain (K_p) times the magnitude of the error plus the integral gain (K_i) times the integral of the error plus the derivative gain (K_d) times the derivative of the error.

This control signal (u) is sent to the plant, and the new output (y) is obtained. The new output (y) is then fed back and compared to the reference to find the new error signal (e). The controller takes this new error signal and computes its derivative and its integral again, ad infinitum.

The transfer function of a PID controller is found by taking the Laplace transform of Eq.(1).

(2) $$K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

K_p= Proportional gain K_i= Integral gain K_d= Derivative gain

We can define a PID controller in MATLAB using the transfer function directly, for example:

```
Kp = 1;
Ki = 1;
Kd = 1;

s = tf('s');
C = Kp + Ki/s + Kd*s
C =

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

Continuous-time transfer function.
```

Alternatively, we may use MATLAB's **pid controller object** to generate an equivalent contin

```
C = pid(Kp,Ki,Kd)
C =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = 1, Ki = 1, Kd = 1

Continuous-time PID controller in parallel form.
```

Let's convert the pid object to a transfer function to see that it yields the same result as above:

```
tf(C)
ans =

  s^2 + s + 1  -----------
```

```
       s

Continuous-time transfer function.
```

The Characteristics of P, I, and D Controllers

A proportional controller (K_p) will have the effect of reducing the rise time and will reduce but never eliminate the **steady-state error**. An integral control (K_i) will have the effect of eliminating the steady-state error for a constant or step input, but it may make the transient response slower. A derivative control (K_d) will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response.

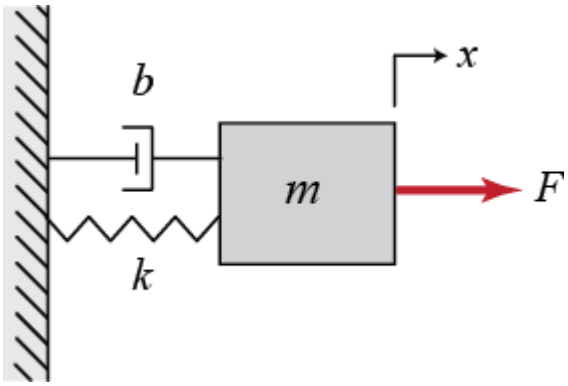
The effects of each of controller parameters, K_p, K_d, and $K_i$on a closed-loop system are summarized in the table below.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	No Change

Note that these correlations may not be exactly accurate, because K_p, K_i, and $K_d$are dependent on each other. In fact, changing one of these variables can change the effect of the other two. For this reason, the table should only be used as a reference when you are determining the values for K_i, $K_p$and K_d.

Example Problem

Suppose we have a simple mass, spring, and damper problem.



The modeling equation of this system is

(3)$M\ddot{x} + b\dot{x} + kx = F$

Taking the Laplace transform of the modeling equation, we get

(4) $Ms^2X(s) + bsX(s) + kX(s) = F(s)$

The transfer function between the displacement $X(s)$ and the input $F(s)$ then becomes

(5) $$\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + bs + k}$$

Let

```
M = 1 kg
b = 10 N s/m
k = 20 N/m
F = 1 N
```

Plug these values into the above transfer function

(6) $$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20}$$

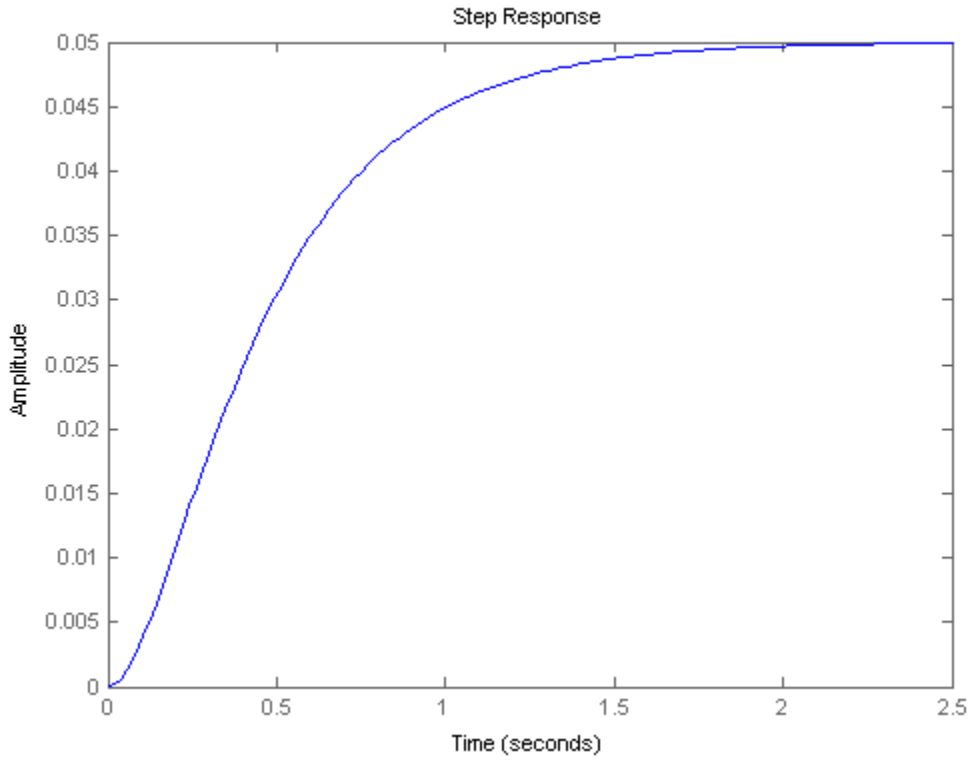
The goal of this problem is to show you how each of K_p, K_i and K_d contributes to obtain

```
Fast rise time
Minimum overshoot
No steady-state error
```

Open-Loop Step Response

Let's first view the open-loop step response. Create a new m-file and run the following code:

```
s = tf('s');
P = 1/(s^2 + 10*s + 20);
step(P)
```



The DC gain of the plant transfer function is 1/20, so 0.05 is the final value of the output to an unit step input. This corresponds to the steady-state error of 0.95, quite large indeed. Furthermore, the rise time is about one second, and the settling time is about 1.5 seconds. Let's design a controller that will reduce the rise time, reduce the settling time, and eliminate the steady-state error.

Proportional Control

From the table shown above, we see that the proportional controller (Kp) reduces the rise time, increases the overshoot, and reduces the steady-state error.

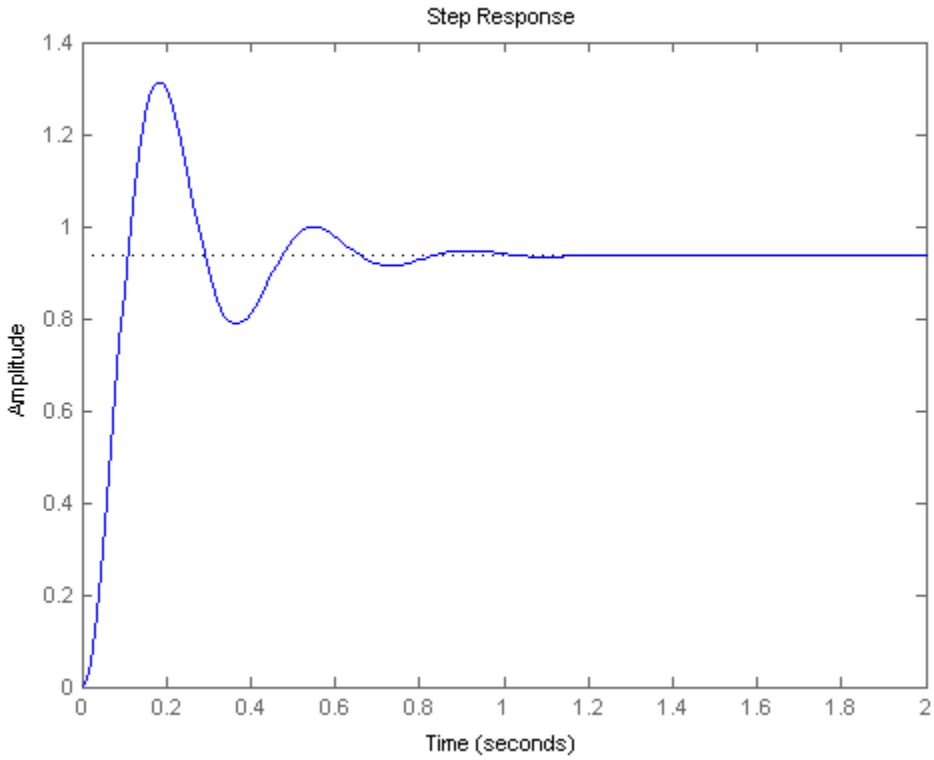
The closed-loop transfer function of the above system with a proportional controller is:

$$\frac{X(s)}{F(s)} = \frac{K_p}{s^2 + 10s + (20 + K_p)} \quad (7)$$

Let the proportional gain (K_p) equal 300 and change the m-file to the following:

```
Kp = 300;
C = pid(Kp)
T = feedback(C*P,1)

t = 0:0.01:2;
```

```
step(T,t)
C =

  Kp = 300

P-only controller.


T =

        300
  ----------------
  s^2 + 10 s + 320

Continuous-time transfer function.
```



The above plot shows that the proportional controller reduced both the rise time and the steady-state error, increased the overshoot, and decreased the settling time by small amount

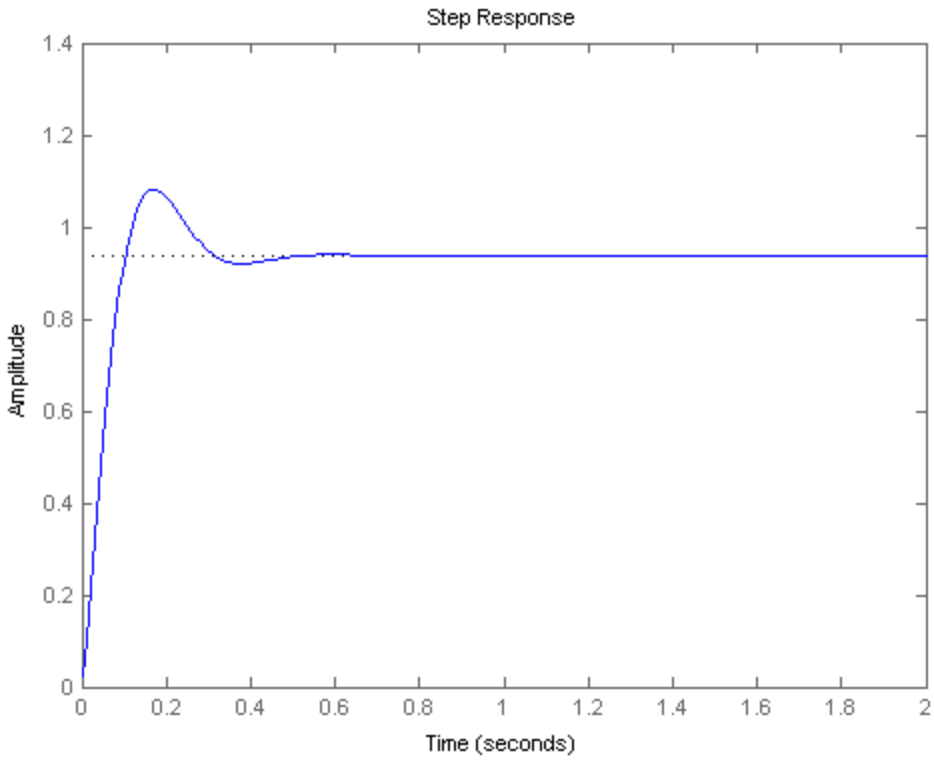
Proportional-Derivative Control

Now, let's take a look at a PD control. From the table shown above, we see that the derivative controller (Kd) reduces both the overshoot and the settling time. The closed-loop transfer function of the given system with a PD controller is:

(8) $$\frac{X(s)}{F(s)} = \frac{K_d s + K_p}{s^2 + (10 + K_d)s + (20 + K_p)}$$

Let K_p equal 300 as before and let K_d equal 10. Enter the following commands into an m-file and run it in the MATLAB command window.

```
Kp = 300;
Kd = 10;
C = pid(Kp,0,Kd)
T = feedback(C*P,1)

t = 0:0.01:2;
step(T,t)
C =


  Kp + Kd * s


  with Kp = 300, Kd = 10

Continuous-time PD controller in parallel form.


T =

     10 s + 300
  ----------------
  s^2 + 20 s + 320

Continuous-time transfer function.
```



This plot shows that the derivative controller reduced both the overshoot and the settling time, and had a small effect on the rise time and the steady-state error.

Proportional-Integral Control

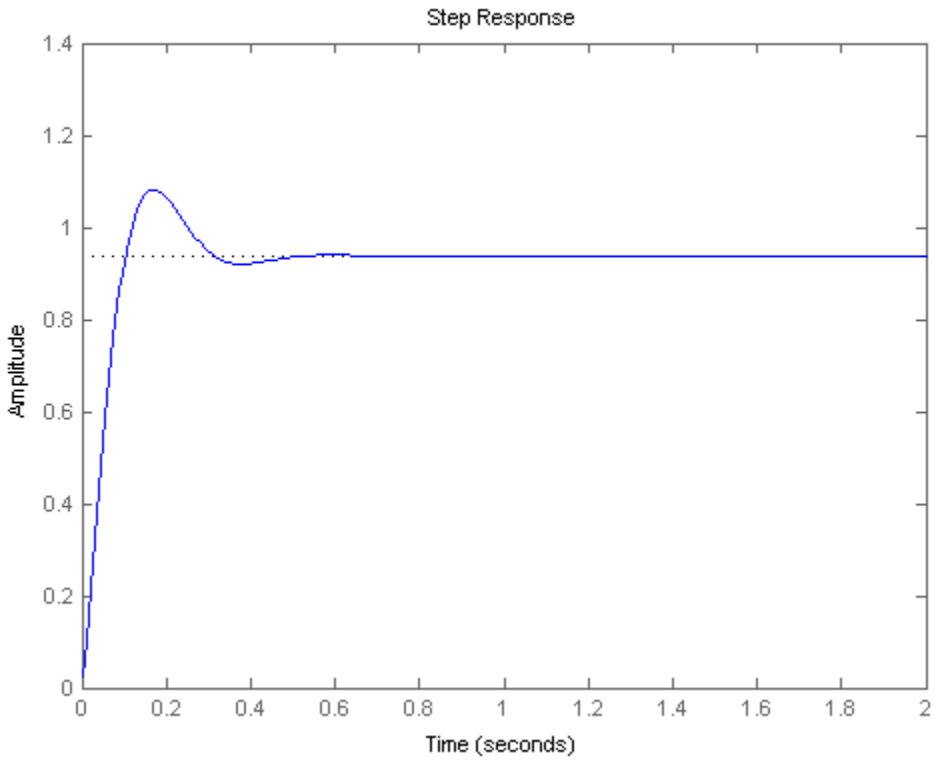
Now, let's take a look at a PD control. From the table shown above, we see that the derivative controller (Kd) reduces both the overshoot and the settling time. The closed-loop transfer function of the given system with a PD controller is:

$$\frac{X(s)}{F(s)} = \frac{K_d s + K_p}{s^2 + (10 + K_d)s + (20 + K_p)} \quad (8)$$

Let $K_p$equal 300 as before and let $K_d$equal 10. Enter the following commands into an m-file and run it in the MATLAB command window.

```
Kp = 300;
Kd = 10;
C = pid(Kp,0,Kd)
T = feedback(C*P,1)

t = 0:0.01:2;
step(T,t)
C =
```

```
  Kp + Kd * s


  with Kp = 300, Kd = 10

Continuous-time PD controller in parallel form.


T =

     10 s + 300
  ----------------
  s^2 + 20 s + 320

Continuous-time transfer function.
```



This plot shows that the derivative controller reduced both the overshoot and the settling time, and had a small effect on the rise time and the steady-state error

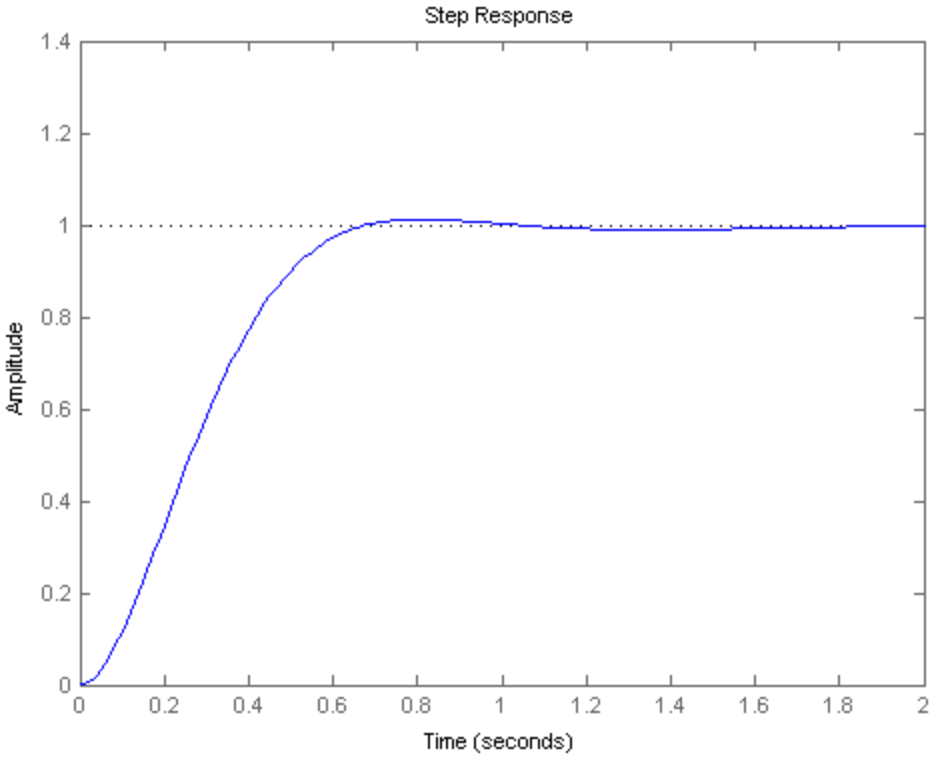
Proportional-Integral-Derivative Control

v Before going into a PID control, let's take a look at a PI control. From the table, we see that an integral controller (Ki) decreases the rise time, increases both the overshoot and the settling time, and eliminates the steady-state error. For the given system, the closed-loop transfer function with a PI control is:

(9) $$\frac{X(s)}{F(s)} = \frac{K_p s + K_i}{s^3 + 10s^2 + (20 + K_p s + K_i)}$$

Let's reduce the K_p to 30, and let K_i equal 70. Create an new m-file and enter the following commands.

```
Kp = 30;
Ki = 70;
C = pid(Kp,Ki)
T = feedback(C*P,1)
t = 0:0.01:2;
step(T,t)
C =

             1
  Kp + Ki * ---
             s

  with Kp = 30, Ki = 70

Continuous-time PI controller in parallel form.


T =

         30 s + 70
  -----------------------
  s^3 + 10 s^2 + 50 s + 70

Continuous-time transfer function.
```


Step Response
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2