Unity and Non Unity Feedback System using MATLAB

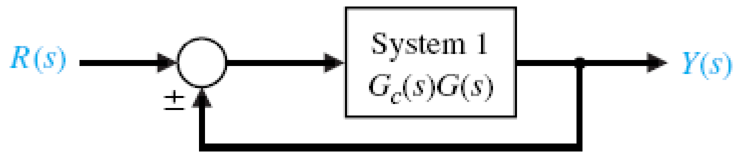
Aim: To simulation of unity and non unity feedback transfer function using MATLAB.

SOFTWARE REQUIRED: MATLAB – Personal Computer with MATLAB

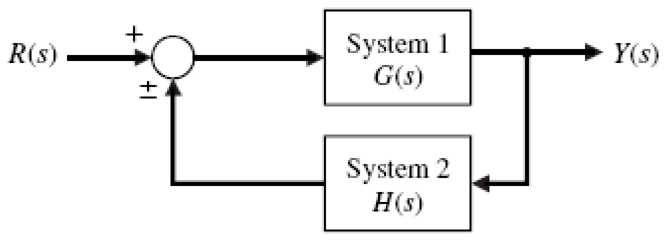
a) **simulation of unity feedback system.**

Theory:

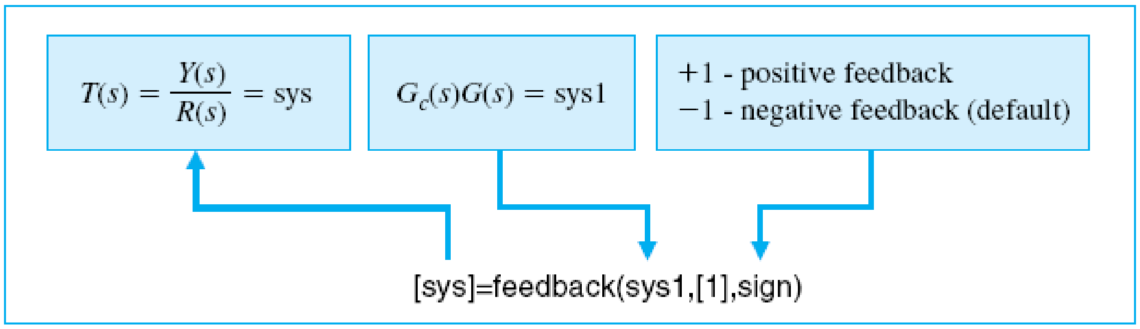
Feedback configuration: If the blocks are connected as shown below then the blocks are said to be in feedback. Notice that in the feedback there is no transfer function H(s) defined. When not specified, H(s) is unity. Such a system is said to be a unity feedback system.



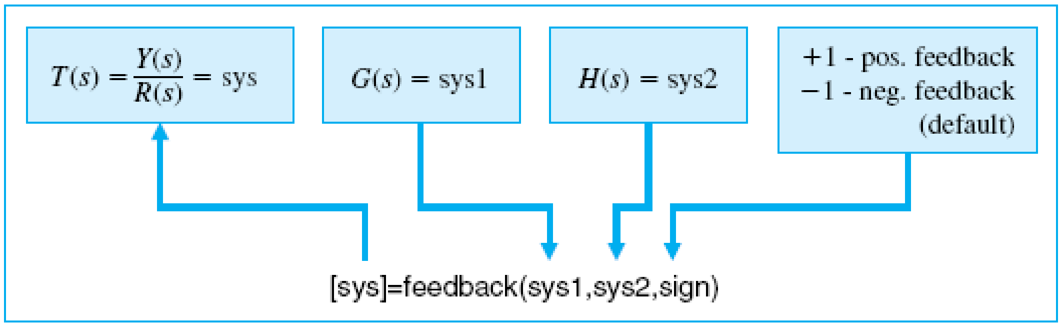
When H(s) is non-unity or specified, such a system is said to be a non-unity feedback system as shown below:



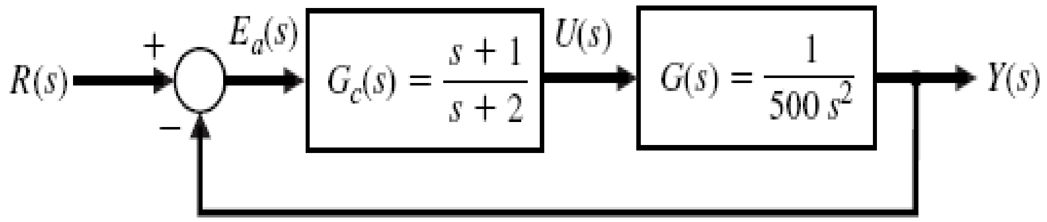
The MATLAB command for implementing a feedback system is “feedback” as shown below:


$T(s) = \frac{Y(s)}{R(s)} = \text{sys}$
$G_c(s)G(s) = \text{sys1}$
+1 - positive feedback
−1 - negative feedback (default)
[sys]=feedback(sys1,[1],sign)

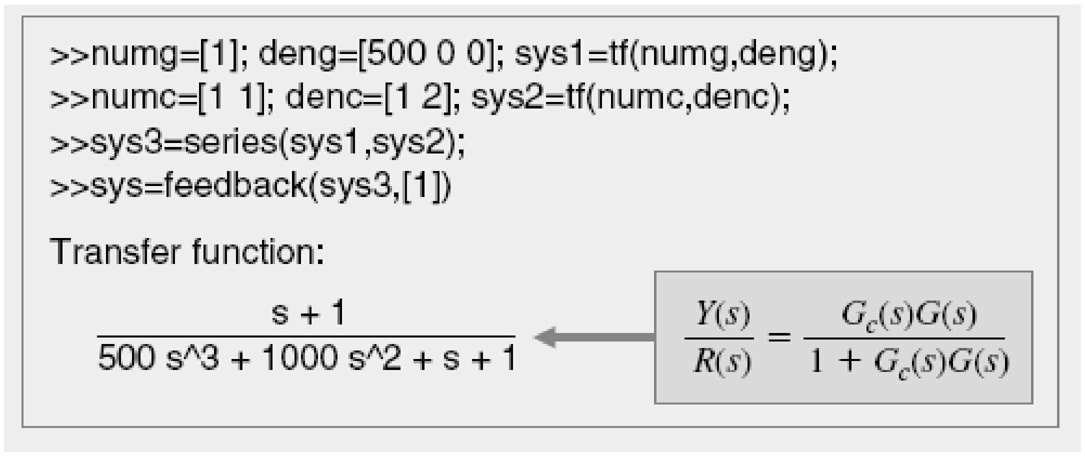
When H(s) is non-unity or specified, such a system is said to be a non-unity feedback system as shown below:


$T(s) = \frac{Y(s)}{R(s)} = \text{sys}$
$G(s) = \text{sys1}$
$H(s) = \text{sys2}$
+1 - pos. feedback
−1 - neg. feedback (default)
[sys]=feedback(sys1,sys2,sign)

Block diagram:



Program:



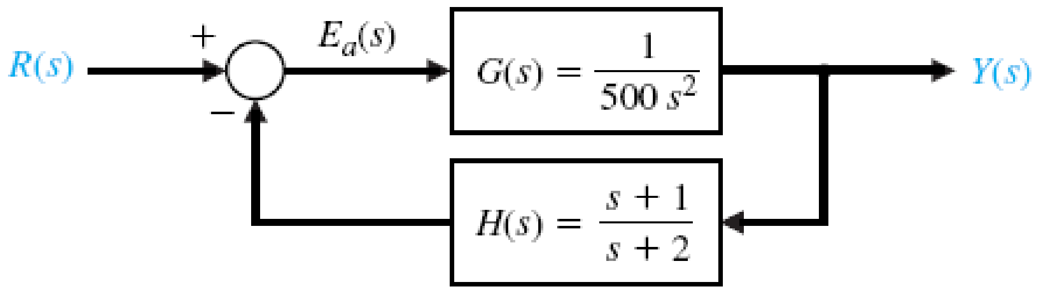
```
>>numg=[1]; deng=[500 0 0]; sys1=tf(numg,deng);
>>numc=[1 1]; denc=[1 2]; sys2=tf(numc,denc);
>>sys3=series(sys1,sys2);
>>sys=feedback(sys3,[1])

Transfer function:
           s + 1
----------------------------
500 s^3 + 1000 s^2 + s + 1
```

$$\frac{Y(s)}{R(s)} = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)}$$

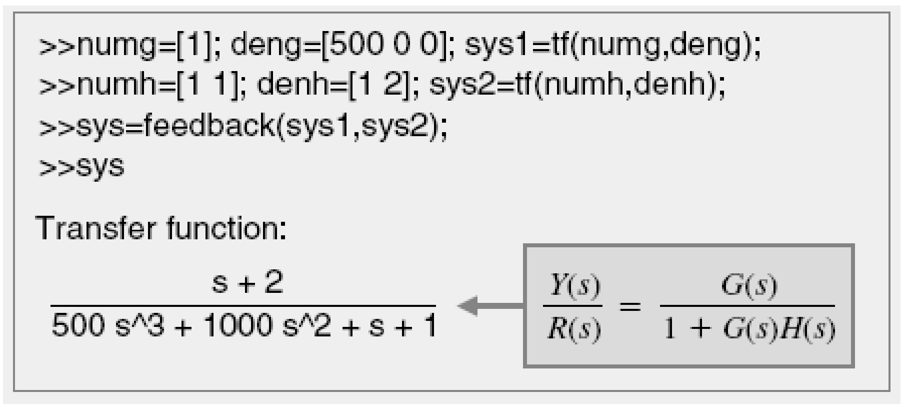
Result:

b) simulation of non unity feedback system

Block diagram



:



```
>>numg=[1]; deng=[500 0 0]; sys1=tf(numg,deng);
>>numh=[1 1]; denh=[1 2]; sys2=tf(numh,denh);
>>sys=feedback(sys1,sys2);
>>sys

Transfer function:
              s + 2
------------------------------
500 s^3 + 1000 s^2 + s + 1
```

$$\frac{Y(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)}$$

Result:

6.1. Block diagram reduction technique using MATLAB

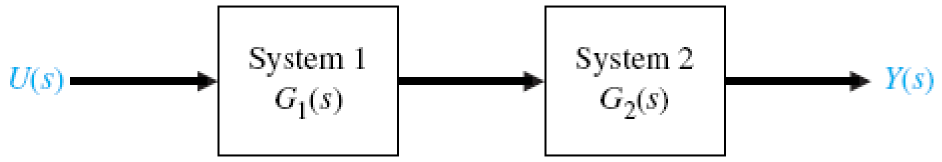
Aim: simulation of multi feedback system

<u>**SOFTWARE REQUIRED:**</u> **MATLAB – P**ersonal Computer with MATLAB

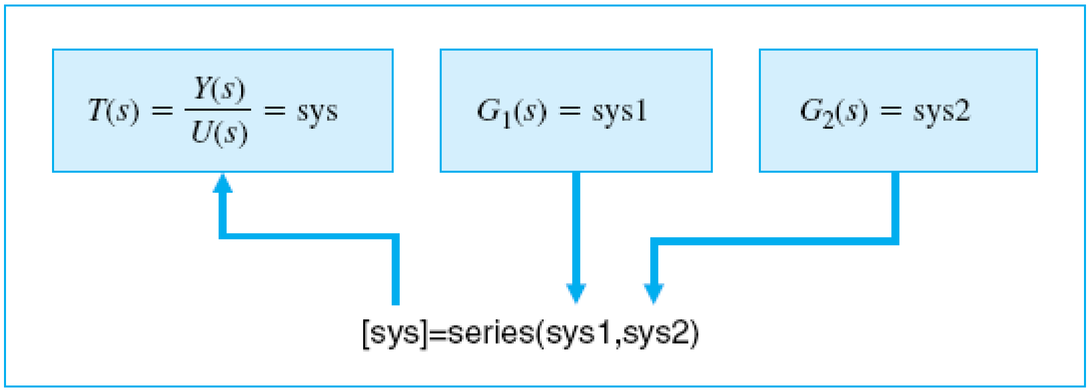
Theory:

Series configuration: If the two blocks are connected as shown below then the blocks are said

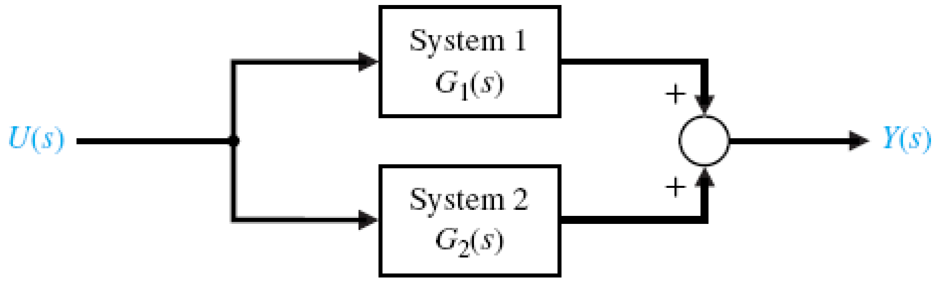
to be in series. It would like multiplying two transfer functions. The MATLAB command for the such configuration is “series”.



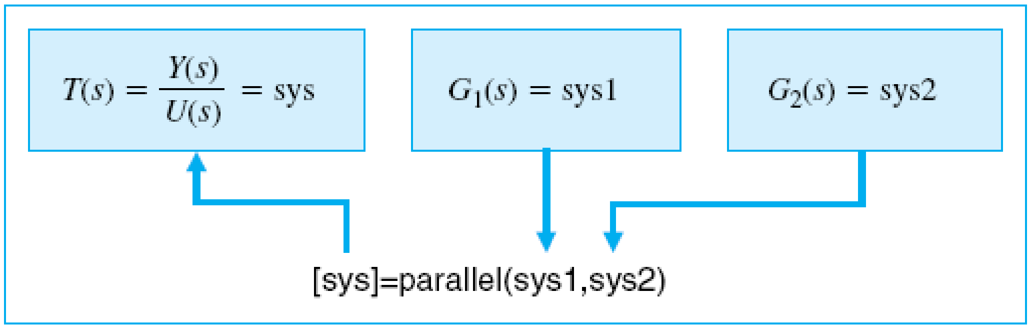
The series command is implemented as shown below



Parallel configuration: If the two blocks are connected as shown below then the blocks are said to be in parallel. It would like adding two transfer functions

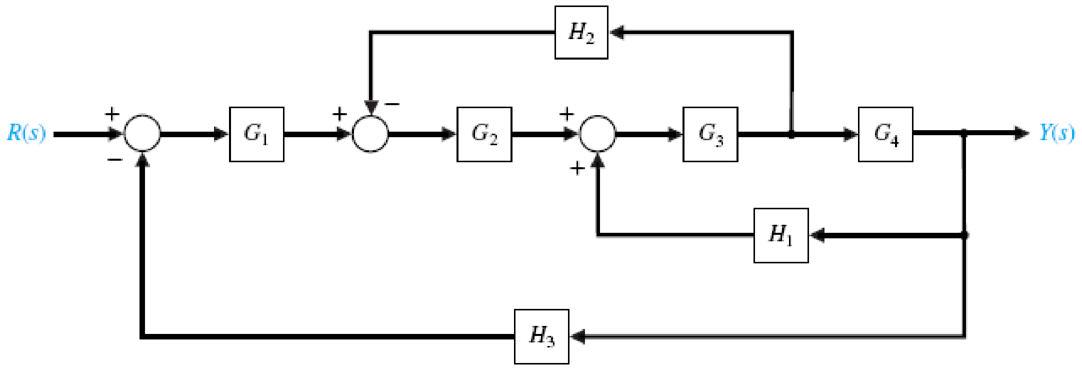


The MATLAB command for implementing a parallel configuration is “parallel” as shown below:

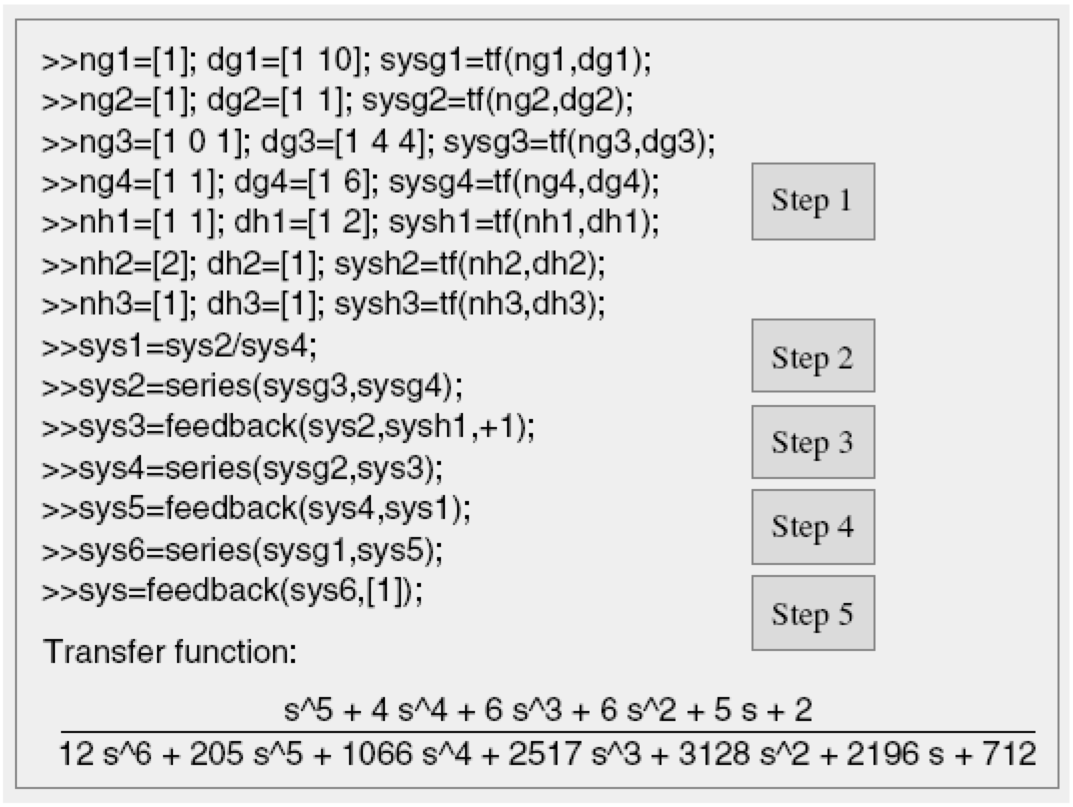


$T(s) = \frac{Y(s)}{U(s)} = \text{sys}$
$G_1(s) = \text{sys1}$
$G_2(s) = \text{sys2}$
[sys]=parallel(sys1,sys2)

Blockdiagram:



Program:



```
>>ng1=[1]; dg1=[1 10]; sysg1=tf(ng1,dg1);
>>ng2=[1]; dg2=[1 1]; sysg2=tf(ng2,dg2);
>>ng3=[1 0 1]; dg3=[1 4 4]; sysg3=tf(ng3,dg3);
>>ng4=[1 1]; dg4=[1 6]; sysg4=tf(ng4,dg4);          Step 1
>>nh1=[1 1]; dh1=[1 2]; sysh1=tf(nh1,dh1);
>>nh2=[2]; dh2=[1]; sysh2=tf(nh2,dh2);
>>nh3=[1]; dh3=[1]; sysh3=tf(nh3,dh3);
>>sys1=sys2/sys4;                                   Step 2
>>sys2=series(sysg3,sysg4);
>>sys3=feedback(sys2,sysh1,+1);                     Step 3
>>sys4=series(sysg2,sys3);
>>sys5=feedback(sys4,sys1);                         Step 4
>>sys6=series(sysg1,sys5);
>>sys=feedback(sys6,[1]);                           Step 5

Transfer function:

                 s^5 + 4 s^4 + 6 s^3 + 6 s^2 + 5 s + 2
 ----------------------------------------------------------------
 12 s^6 + 205 s^5 + 1066 s^4 + 2517 s^3 + 3128 s^2 + 2196 s + 712
```

Result:

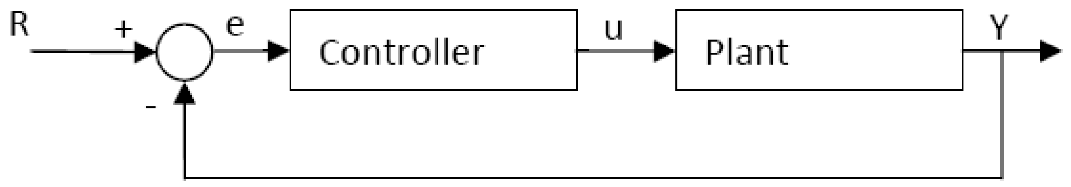
Simulation of P, PD, PI, PID controller

Aim: To simulate the P, PD, PI, PID controller for unit step input.

SOFTWARE REQUIRED: MATLAB – Personal Computer with MATLAB.

Theory:

Consider the following unity feedback system:



Plant: A system to be controlled.

Controller: Provides excitation for the plant; Designed to control the overall system behavior.

The three-term controller: The transfer function of the PID controller looks like the following

$$K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s}$$

KP = Proportional gain

KI = Integral gain

KD = Derivative gain

First, let's take a look at how the PID controller works in a closed-loop system using the

schematic shown above. The variable (e) represents the tracking error, the difference between the

desired input value (R) and the actual output (Y). This error signal (e) will be sent to the PIDcontroller, and the controller computes both the derivative and the integral of this error signal.The signal (u) just past the controller is now equal to the proportional gain (KP) times themagnitude of the error plus the integral gain (KI) times the integral of the error plus thederivative gain (KD) times the derivative of the error.

$$u = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt}$$

Plug these values into the above transfer function

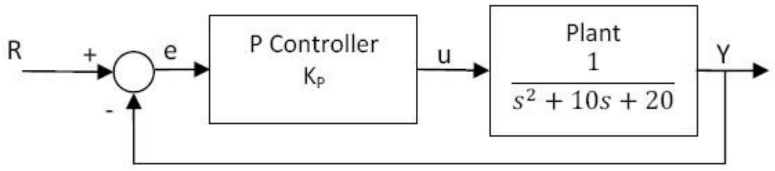
$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20}$$

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

Proportional control:

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

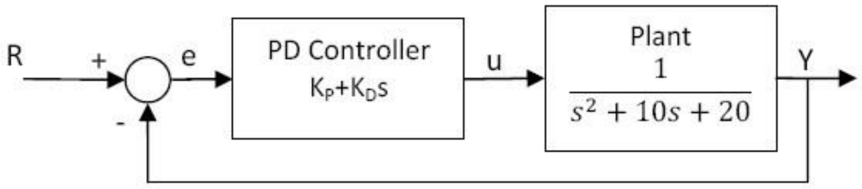


$$\frac{X(s)}{F(s)} = \frac{K_P}{s^2 + 10s + (20 + K_P)}$$

Let the proportional gain (K_P) equal 300:

Proportional-Derivative control:

The closed-loop transfer function of the given system with a PD controller is:

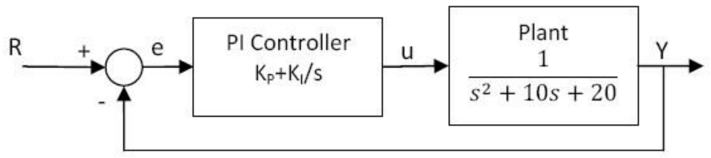


$$\frac{X(s)}{F(s)} = \frac{K_D s + K_P}{s^2 + (10 + K_D)s + (20 + K_P)}$$

Let K_P equal 300 as before and let K_D equal 10.

Proportional-Integral control:

Before going into a PID control, let's take a look at a PI control. For the given system, the closed-loop transfer function with a PI control is:

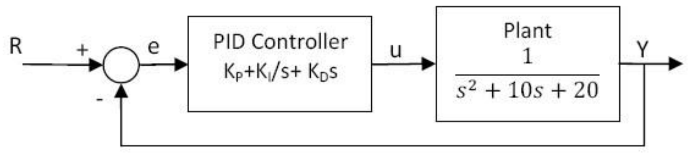


$$\frac{X(s)}{F(s)} = \frac{K_P s + K_I}{s^3 + 10s^2 + (20 + K_P)s + K_I}$$

Let's reduce the K_P to 30, and let K_I equal 70.

Proportional-Integral-Derivative control:

Now, let's take a look at a PID controller. The closed-loop transfer function of the given system with a PID controller is:



$$\frac{X(s)}{F(s)} = \frac{K_D s^2 + K_P s + K_I}{s^3 + (10 + K_D)s^2 + (20 + K_P)s + K_I}$$

After several trial and error runs, the gains Kp=350, Ki=300, and Kd=50 provided the desired response. To confirm, enter the following commands to an m-file and run it in the command window. You should get the following step response.

The characteristics of P, I, and D controllers:

The 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 controller (K_I) will have the effect of eliminating the steady state error, but it may make the transient response worse. A derivative control (K_D) will have the effect of increasing the stability of the system, reducing the overshoot and improving the transient response.

Effect of each controller K_P, K_I and K_D on the closed-loop system are summarized below

CL Response	**Rise Time**	**Overshoot**	**Settling Time**	**S-S Error**
K_P	Decrease	Increase	Small Change	Decrease
K_I	Decrease	Increase	Increases	Eliminate
K_D	Small Change	Decreases	Decreases	Small Change

Program:

Open-loop step response:

```
num=1;
den=[1 10 20];
plant=tf(num,den);
step(plant)
```

Proportional control:

```
Kp=300;
contr=Kp;
sys_cl=feedback(contr*plant,1);
t=0:0.01:2;
step(sys_cl,t)
```

Proportional-Derivative control:

```
Kp=300;
Kd=10;
contr=tf([Kd Kp],1);
sys_cl=feedback(contr*plant,1);
t=0:0.01:2;
step(sys_cl,t)
```

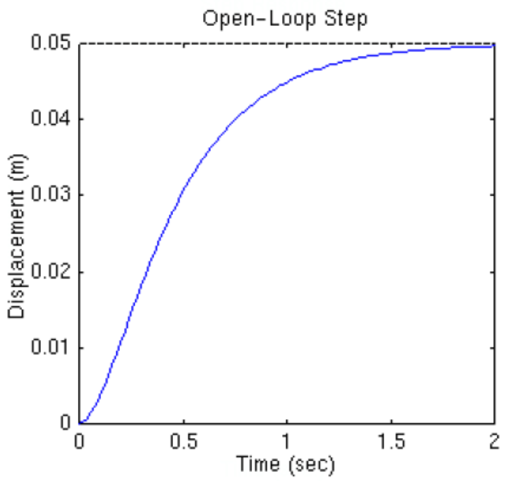
Proportional-Integral control:

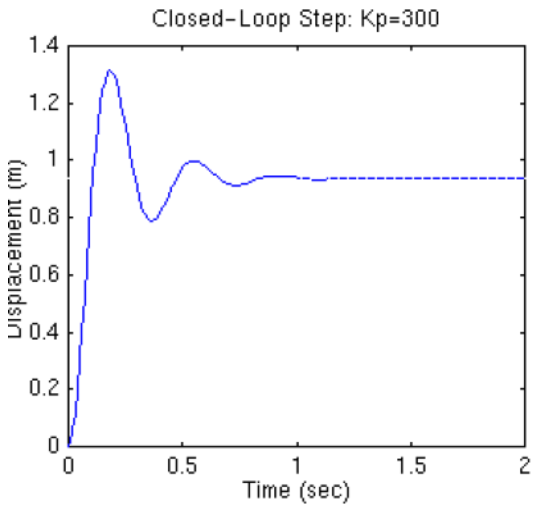
```
Kp=30;
Ki=70;
contr=tf([Kp Ki],[1 0]);
sys_cl=feedback(contr*plant,1);
t=0:0.01:2;
step(sys_cl,t)
```

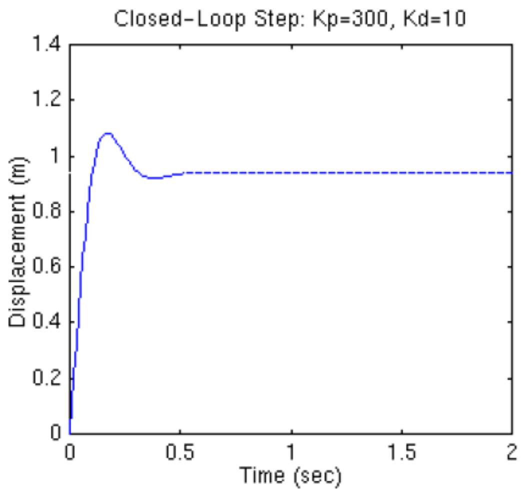
Proportional-Integral-Derivative control:

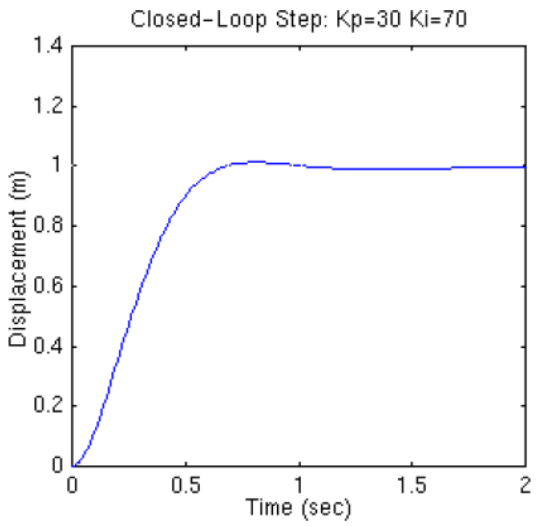
```
Kp=350;
Ki=300;
Kd=50;
contr=tf([Kd Kp Ki],[1 0]);
sys_cl=feedback(contr*plant,1);
t=0:0.01:2;
step(sys_cl,t)
```

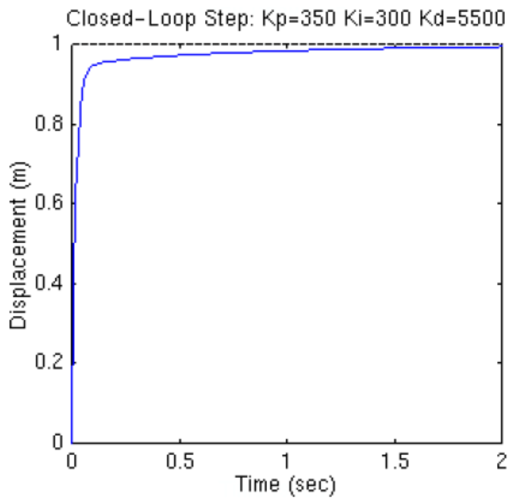
Model graph;


Open-Loop Step
Displacement (m)
Time (sec)
0.05
0.04
0.03
0.02
0.01
0
0.5
1
1.5
2


Closed-Loop Step: Kp=300
Displacement (m)
Time (sec)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.5
1.5
2


Closed-Loop Step: Kp=300, Kd=10
Displacement (m)
Time (sec)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.5
1.5
2


Closed-Loop Step: Kp=30 Ki=70
Displacement (m)
Time (sec)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.5
1.5
2



Observation:

Effect of each controller K_P, K_I and K_D on the closed-loop system are summarized below

CL Response	**Rise Time**	**Overshoot**	**Settling Time**	**S-S Error**
$\mathbf{K_P}$	Decrease	Increase	Small Change	Decrease
$\mathbf{K_I}$	Decrease	Increase	Increases	Eliminate
$\mathbf{K_D}$	Small Change	Decreases	Decreases	Small Change

Result:

6.2. Simulation of DC Motor characteristics using MATLAB

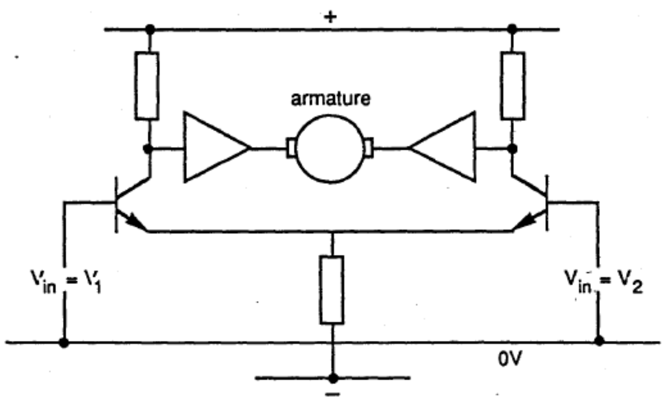
AIM: To Simulate the Dc Motor Characteristics Using MATLAB.

SOFTWARE REQUIRED: **MATLAB – P**ersonal Computer with MATLAB.

THEORY:

This experiment will illustrate the characteristics of the D.C. motor used in the Modular Servo and show how it can be controlled by the Servo Amplifier.

The motor is a permanent magnet type and has a single armature winding. Current flow through the armature is controlled by power amplifiers as in figure so that rotation in both directions is possible by using one, or both of the inputs. In most of the later assignments the necessary input signals are provided by a specialized Pre-Amplifier Unit PA150C, which connected to Inputs 1 and 2 on SA150D



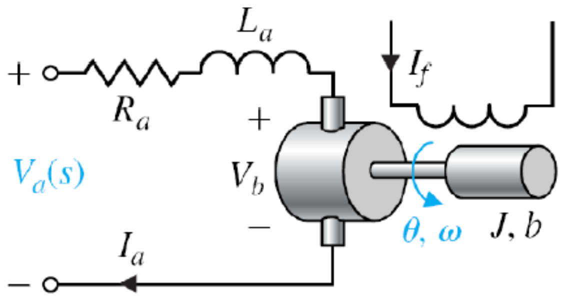
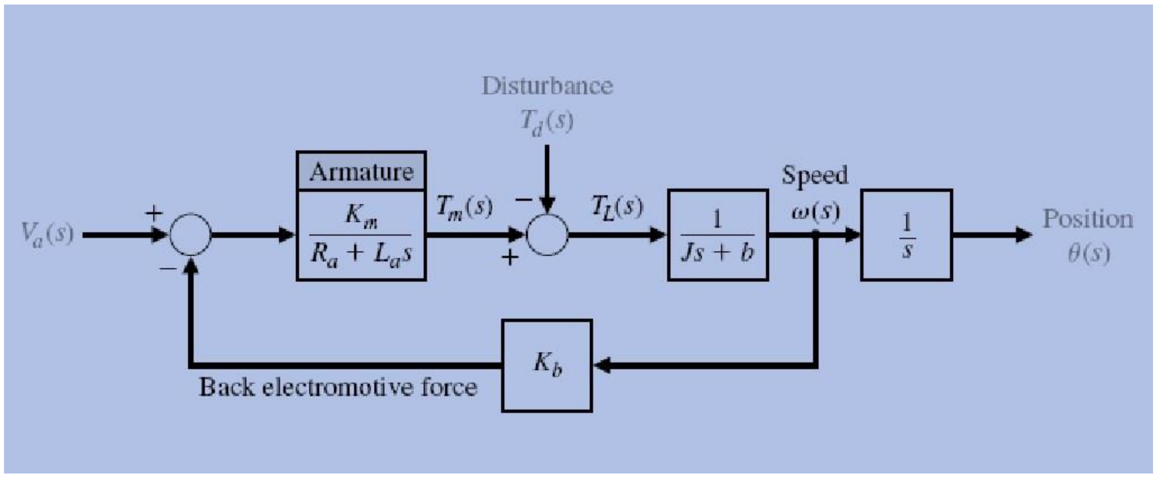


Figure: DC motor armature-controlled rotational actuator

As the motor accelerates the armature generates an increasing 'back-emf' V_a tending to oppose the driving voltage V_{in}. The armature current is thus roughly proportional to (V_{in} - V_a). If the speed drops (due to loading) V_a reduces, the current increases and thus so does the motor torque. This tends to oppose the speed drop. This mode of control is called 'armature-control' and gives a speed proportional to V_{in} as in figure.

The final block diagram is as follows:

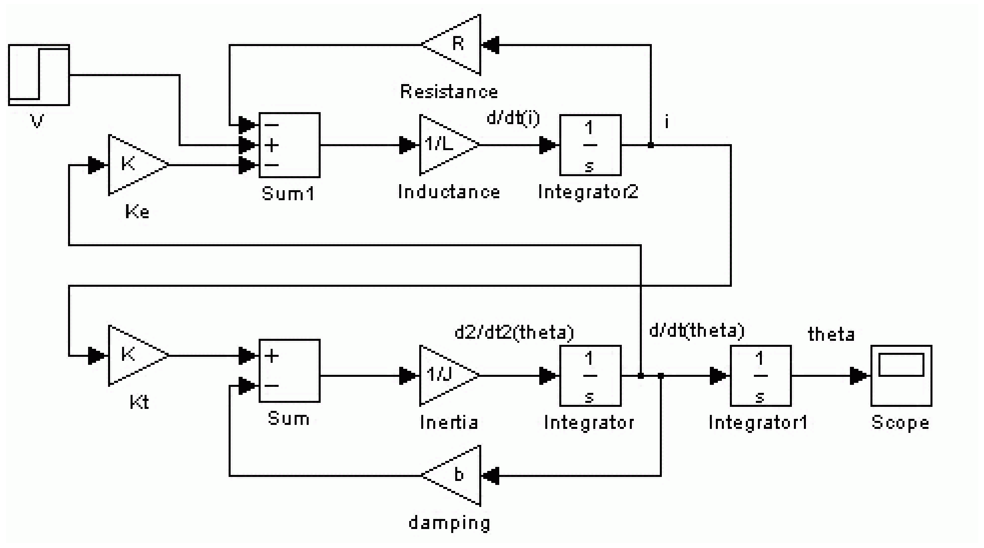


2. DC motor nominal values

- moment of inertia of the rotor (J) = 3.2284E-6 kg.m^2/s^2
- damping ratio of the mechanical system (b) = 3.5077E-6 Nms
- electromotive force constant (K=Ke=Kt) = 0.0274 Nm/Amp

- electric resistance (R) = 4 ohm
- electric inductance (L) = 2.75E-6 H
- input (V): Source Voltage
- output (theta): position of shaft

SIMULINK DIAGRAM:



OUTPUT WAVE FORMS:

RESULT:

6.3. Simulation of poles and zeros of a transfer function

Aim: To simulate the Poles and zeros of the given transfer function.

<u>SOFTWARE REQUIRED:</u> MATLAB – Personal Computer with MATLAB.

Theory:

To obtain the poles and zeros of the system use the MATLABcommand “pole” and“zero” respectively as shown in example 5. You can also use MATLABcommand “pzmap” to obtain the same.

Transfer function:

$$\text{Given } G_1 = \frac{1}{(s+10)};\ G_2 = \frac{1}{(s+1)};\ G_3 = \frac{s^2+1}{(s^2+4s+4)};$$

$H = 1$

Program:

```
ng1=[1];

dg1=[1 10];
```

```
sysg1=tf(ng1,dg1);

ng2=[1];

dg2=[1 1];

sysg2=tf(ng2,dg2);

ng3=[1 0 1];

dg3=[1 4 4];

sysg3=tf(ng3,dg3);

p1=pole(sysg1)

z1=zero(sysg1)

p2=pole(sysg2)

z2=zero(sysg2)

p3=pole(sysg3)

z3=zero(sysg3)
```

Output:

6.4. STATE MODEL FOR CLASSICAL TRANSFER FUNCTION & VICE VERSA USING MATLAB

AIM:

To find state model for classical transfer function and transfer function from state model using MATLAB.

APPARATUS: *Computer with MATLAB software*

THEORY:

COMMAND 1:	CLC: It clears the MATLAB command window
COMMAND 2:	CLEAR: it clears the MATLAB work shop variables.
COMMAND 3:	DISP: Syntax – disp (variable): It displays the variable specified on command window.
COMMAND 4:	PAUSE: With this command the execution will be stopped and it waits for the enter key.
COMMAND 5:	INPUT: Syntax: Variable = Input (‘Comment’);
COMMAND 6:	PERCENTAGE: It is used at the beginning of any statement to make it as a comment in the program.
COMMAND 7:	R-LOCUS: Syntax: r locus (Variable): With this we can plot the root locus of any transfer function. That means in the above syntax the variable is nothing but a transfer function.
COMMAND 8:	BODE: Syntax: Bode (Variable): With this command we can get bode plot of the given transfer function.
COMMAND 9:	MARGIN: Syntax: Margin (Variable): With this command we can get gain and phase margin of a bode plot of the given transfer function.

COMMAND10: SS: Syntax: Variable1= SS(Variable2): With this command we can get state space model for the given transfer function. Variable 2 is a transfer function and variable 1 holds the SS model.

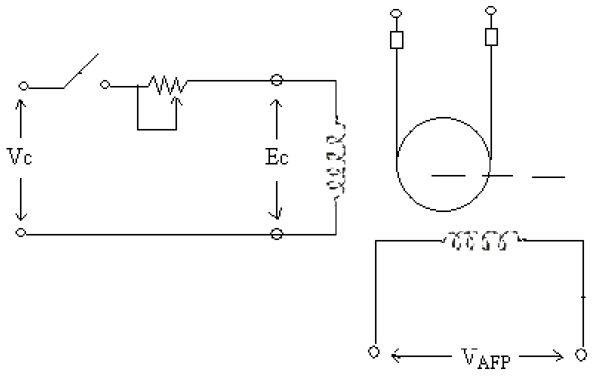
COMMAND11: SS DATA:Syntax: [a,b,c,d]=SSdata (Variable):

With this command we can retrieve the a,b,c,d matrices of a state space model. Variable holds the state space model.

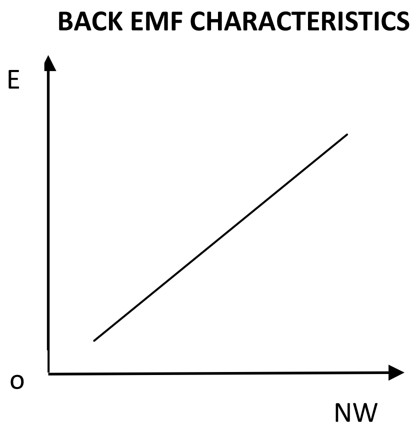
PROCEDURE:

1. Write the programme in MATLAB text editor using mat lab instructions for state model of classical transfer function and for transfer function from state model.
2. Run the programs.
3. Note down the outputs.

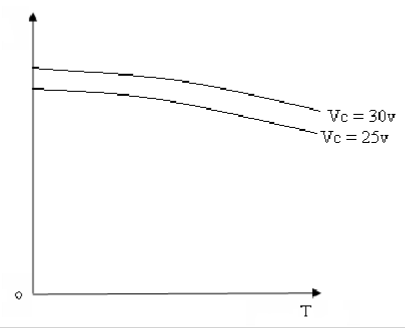
CIRCUIT DIAGRAM:



GRAPHS



SPEED-TORQUE CHAEACTERISTIC



PROGRAM 1:

```
a= input ( “Enter the values of a matrix” );
b= input ( “Enter the values of b matrix” );
c= input ( “Enter the values of c matrix” );
d= input ( “Enter the values of d matrix” );

[num , den] = SS2 tf  (a,b,c,d,1)
S1=tf  (num(1, : ) , den );
S2=tf  (num(2, : ) , den );
 [num1 , den1 ] = SS2 tf  (a,b,c,d,2);
S3=tf (num1 (1, : ) , den1 );
```

```
S4=tf (num1 (2, : ) , den1 );

DISP [S1,S2,S3,S4];
```

PROGRAM 2:

```
Num = input ("Enter numerator polynomial values in the form of matrix array" );

den1 = input ("Enter denominator 1 values" );

den2 = input ("Enter denominator 2 values" );

den = conv (den1,den2);

H = tf (num,den);

P = SS(H);

[a,b,c,d] = SS data(P);
```

RESULT:

The state model for classical transfer function and transfer function from state model are obtained using MATLAB software.

6.5. STABILITY ANALYSIS USING BODE PLOTE USING MATLAB

AIM: To find out the Simulation of stability analysis of Linear Time Invariant using MATLAB.

SOFTWARE REQUIRED: **MATLAB – Mat**rix **Lab**oratory.

BODE PLOT for 2nd , 3rd ,4th & 5th order systems:

2nd order system:

The transfer function is $G(s) = \dfrac{25}{s^2 + 4s + 25}$. Plot the bode plot.(p.517)

Program:

```
num=[0 0 25];
den=[1 4 25];
bode(num,den)
title('Bode diagram of G(s)= 25/s^2+4s+25)')
```

3rd order system:

The transfer function is G(s) = $\dfrac{9\left(s^2 + 0.2s + 1\right)}{s(s^2 + 4s + 25)}$. Plot the bode plot.(p.518)

Program:

```
num=[0 9 1.8 9];

den=[1 1.2 9 0];

bode(num,den)

title('Bode diagram of G(s)= 9(s^2+0.2s+1)/s(s^2+1.2s+9)')
```

4th order system:

The transfer function is G(s)= $\dfrac{4(2s+1)}{s^2(s^2 + 0.4s + 4)}$. obtain the Bode plot(p.611)

Program:

```
num=[0 0 0 8 4];

den=[1 0.4 4 0 0];

bode(num,den)

title('Bode diagram of G(s)= 4(2s+1)/(s^2(s^2+0.4s+4)')
```

5th order system:

The transfer function is G(s) = $\dfrac{40(s+0.4)(s+0.2)}{s(s+4)(s+0.02)(s+1)(s+2)}$

Plot the bode plot.(p.670)

Program:

```
num=[0 0 0 40 24 3.2];

den=[1 9.02 24.18 16.48 0.32 0];

bode(num,den)

title('Bode diagram of G(s)')
```

Result: Bode plot is plotted for the given transfer function is plotted using MATLAB has been successfully completed.

6.6. STABILITY ANALYSIS USING ROOT LOCUS USING MATLAB

ROOT LOCUS for 2nd , 3rd ,4th & 5th order systems:

2nd order system:

The transfer function is G(s) = $\frac{K(s+2)}{(s+3)(s+4)}$. Obtain the Root Locus plot.

Program:

```
num=[0 1 2 ];

den=[1 7 12];

rlocus(num,den)

v=[-6 6 6 -6];

axis(v);

grid;

title('Root Locus Plot of G(s)=K(s+2)/[(s+3)(s+4)]')
```

3rd order system:

The transfer function is G(s) = $\frac{K(s+0.4)}{s^2(s+3.6)}$. Obtain the Root Locus plot. (p.400)

Program:

```
num=[0 0 1 0.4];

den=[1 3.6 0 0];

rlocus(num,den)

v=[-5 1 –3 3]; axis(v);

grid;

title('Root Locus Plot of G(s)=K(s+0.4)/[s^2(s+3.6)]')
```

4th order system:

The transfer function is G(s) = $\frac{K(s+3)}{s(s+1)(s^2+4s+16)}$ .. Obtain Root locus plot. (p.360)

Program:

```
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)]')
```

5th order system:

The transfer function is G(s) = $\dfrac{K\,(s^2 + 2s + 4)}{s(s+4)(s+6)(s^2 + 1.4s + 1)}$. Obtain the Root locus Plot. (p.378)

Program:

```
num=[0 0 0 1 2 4];

den=[1 11.4 39 43.6 24 0];

rlocus(num,den)

v=[-7 3 -5 5];

axis(v); axis(‘square’);

grid;

title(‘Root Locus Plot of G(s) = K(s^2+2s+4)/[s(s+4)(s+6)(s^2+1.4s+1)]’)
```

Result: Root Locus Plot is plotted for the given transfer function is plotted using MATLAB has been successfully completed.

6.7. STABILITY ANALYSIS USING NYQUIST PLOT USING MATLAB

NYQUIST PLOT for 2nd , 3rd ,4th & 5th order systems:

2nd order system:

The transfer function is G(s) = $\frac{1}{s^2 + 0.8s + 1}$. Obtain the Root Locus plot. (p.532)

Program:

```
num=[0 0 1];

den=[1 0.8 1];

nyquist(num,den)

v=[-2 2 -2 2];

axis(v);

grid;

title('Nyquist Plot of G(s)=1/s^2+0.8s+1)')
```

3rd order system:

The transfer function is G(s) = $\frac{20(s^2 + s + 0.5)}{s(s+1)(s+10)}$. Draw a Nyquist plot using MATLAB. (p.600)

Program:

```
num=[0 20 20 10];

den=[1 11 10 0];

nyquist(num,den)

v=[-2 3 –3 3]; axis(v);

grid;

title('Nyquist Plot of G(s)=20(s^2+s+0.5)/[s(s+1)(s+10)]')
```

4TH order system:

The transfer function is G(s) = $\frac{100(s+0.1)}{(s+10)(s^3+3S^2+2S+10)}$. Draw a Nyquist plot using MATLAB. (p.600)

Program:

```
num=[0 0 0 100 10];

den=[1 13 32 26 60];

nyquist(num,den)

v=[-2 3 –3 3]; axis(v);

grid;

title('Nyquist Plot of G(s)=100(s+0.1)/[(s+10)(s^3+3S^2+2S+10)]')
```

Result: Nyquist Plot is plotted for the given transfer function is plotted using MATLAB has been successfully completed.