MERCY CHEPNGENO

ENE221-0159/2022

BSc. IN TELECOMMUNICATIONS AND INFO. ENGINEERING

ETI 2310

CONTROL ENGINEERING

MATLAB SIMULATIONS REPORT

Overview

Control Engineering is a specialized discipline that integrates principles from mathematics, physics, and computer science to automate and optimize dynamic systems.

MATLAB (Matrix Laboratory) is an indispensable tool in control engineering, serving as both a computational workbench and a design platform for modeling, analyzing, and implementing control systems.

It is used for; modelling and simulation, controller design and some real-world implementation

LAB 2: POLYNOMIALS IN MATLAB

Objective: To represent polynomials in MATLAB, find roots of polynomials, create polynomials when roots are known and obtain partial fractions**.**

Exercise 1:

Consider the two polynomials $p(s) = s^2 + 2s + 1$ and $q(s) = s + 1$. Using MATLAB compute

a. $p(s) * q(s)$
b. Roots of $p(s)$ and $q(s)$
c. $p(-1)$ and $q(6)$

```
>> p=[1 2 1];
>> q=[1 1];
>> r=conv (p, q)

r =
 1    3    3    1

>> r=roots(p)

r =
   -1
   -1

>> r=roots(q)
r =
   -1

>> polyval(p,-1)
ans =
     0

>> polyval(q,6)
ans =
```

7

```
r =

    -1
    -1


r =

    -1


ans =

     0


ans =

     7
```

Exercise 2:

Use MATLAB command to find the partial fraction of the following

a. $\frac{B(s)}{A(s)} = \frac{2s^3+5s^2+3s+6}{s^3+6s^2+11s+6}$

b. $\frac{B(s)}{A(s)} = \frac{s^2+2s+3}{(s+1)^3}$

```
>> B=[2 5 3 6];
>> A=[1 6 11 6];
>> [r,p,k]=residue(B,A)
```

```
r =

  -6.0000
  -4.0000
   3.0000

p =

  -3.0000
  -2.0000
  -1.0000


k =

   2
```

```
>> B=[1 2 3];
>> A=[1 3 3 1];
>> [r,p,k]=residue(B,A)
```

```
r =

  1.0000

  0.0000

  2.0000

p =


  -1.0000

  -1.0000

  -1.0000

k =

   []
```

LAB 3: SCRIPTS, FUNCTIONS AND FLOW CONTROL IN MATLAB

Objective: To introduce the writing M-file scripts, creating MATLAB Functions

and reviewing MATLAB flow control like ‘if-elseif-end’, ‘for loops’ and ‘while loops.

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called **M-files**. There are two kinds of M-files;

1.**Scripts**, which do not accept input arguments or return output arguments. They operate on data in the workspace.

2.**Functions**, which can accept input arguments and return output arguments. Internal variables are local to the function.

Flow control

if,
else switch
Switch and case
for;
while;
  break;

Exersice 1: MATLAB M-file Script

Use MATLAB to generate the first 100 terms in the sequence **a(n)** define recursively by

$$a(n+1) = p * a(n) * (1 - a(n))$$

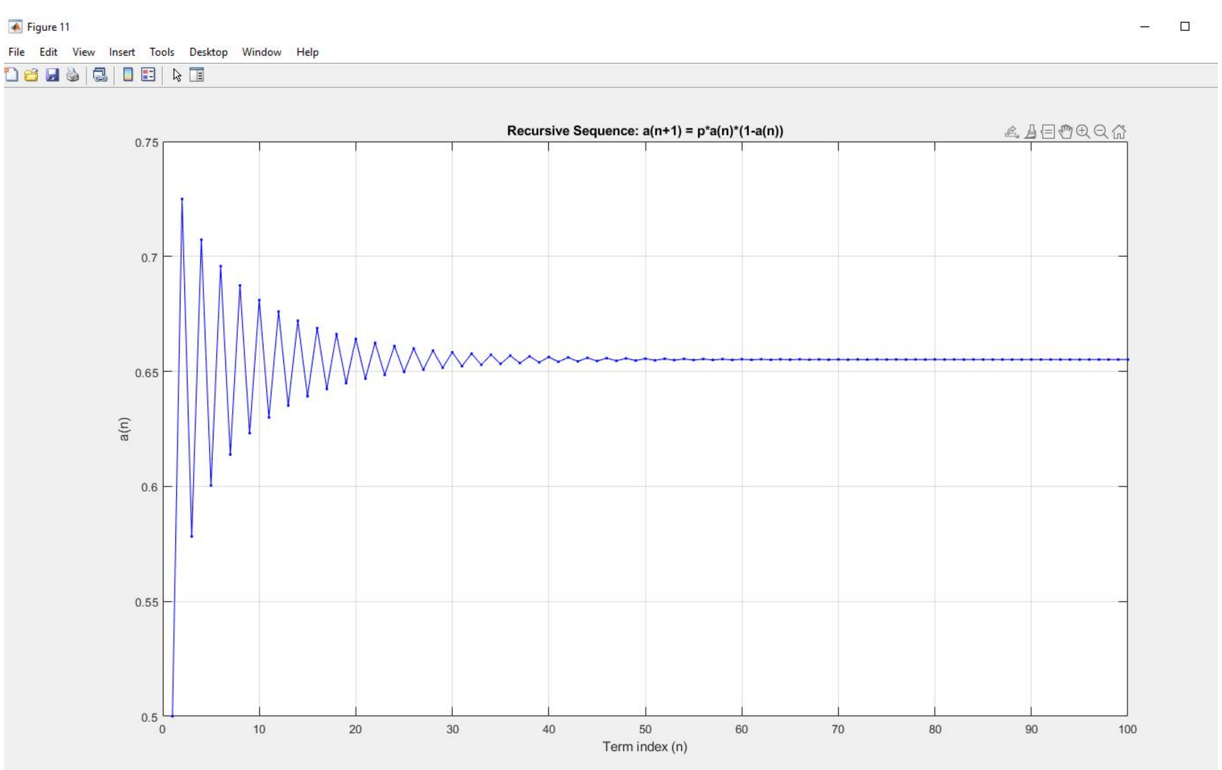
with p=2.9 and a(1) = 0.5.

```
>> p=2.9;
>>a(1)=0.5;
>>terms=100;
>>a=zeros(1, terms);
>>a(1)=a1;
>>for n=1:terms-1;
   a(n+1)=p*a(n)*(1-a(n));
end
>> disp('first 10 terms of the sequence:');
```

first 10 terms of the sequence:

```
>> disp(a(1:10));
   0.5000        0        0        0        0        0        0        0        0        0

>>figure;
plot(1:terms, a, 'b.-');
title('Recursive Sequence: a(n+1) = p*a(n)*(1-a(n))');
xlabel('Term index (n)');
ylabel('a(n)');
>>grid on
```

```
First 10 terms of the sequence:
    0.5000
    0.7250
    0.5782
    0.7073
    0.6004
    0.6958
    0.6139
    0.6874
    0.6232
    0.6810
```




```
>>disp(a(1:100));
  Columns 1 through 13

   0.5000        0        0        0        0        0        0        0        0        0        0        0        0
  Columns 14 through 26

        0        0        0        0        0        0        0        0        0        0        0        0        0
Columns 27 through 39

        0        0        0        0        0        0        0        0        0        0        0        0        0
Columns 40 through 52

        0        0        0        0        0        0        0        0        0        0        0        0        0
Columns 53 through 65
```

0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 66 through 78

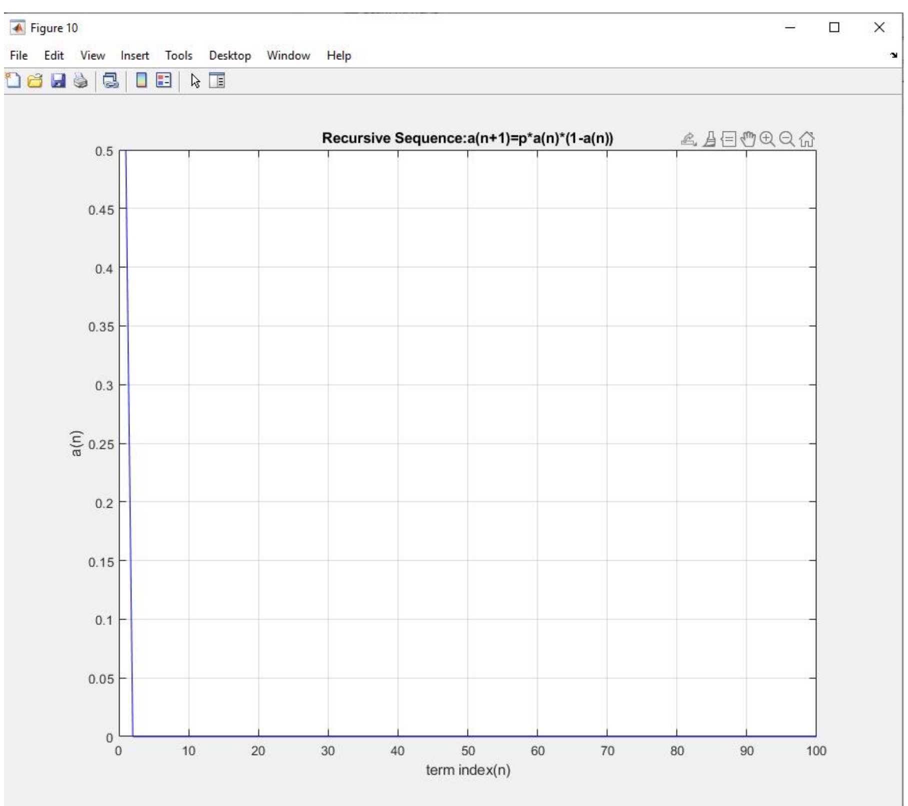
0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 79 through 91

0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 92 through 100

0 0 0 0 0 0 0 0 0



Exersice 2: MATLAB M-file Function

Consider the following equation

$$y(t) = \frac{y(0)}{\sqrt{1-\zeta}} e^{-\zeta\omega_n t} \sin(\omega_n\sqrt{1-\zeta^2} * t + \theta)$$

a) Write a MATLAB M-file function to obtain numerical values of y(t). Your function must take y(0), ζ, ω_n, t and θ as function inputs and y(t) as output argument.

b) Obtain the plot for y(t) for 0<t<10 with an increment of 0.1, by considering the following two cases

Case 1: y(0)=0.15 m, $\omega_n = \sqrt{2}$ rad/sec, $\zeta = 3/(2\sqrt{2})$ and $\theta = 0$;

Case 2: y(0)=0.15 m, $\omega_n = \sqrt{2}$ rad/sec, $\zeta = 1/(2\sqrt{2})$ and $\theta = 0$;

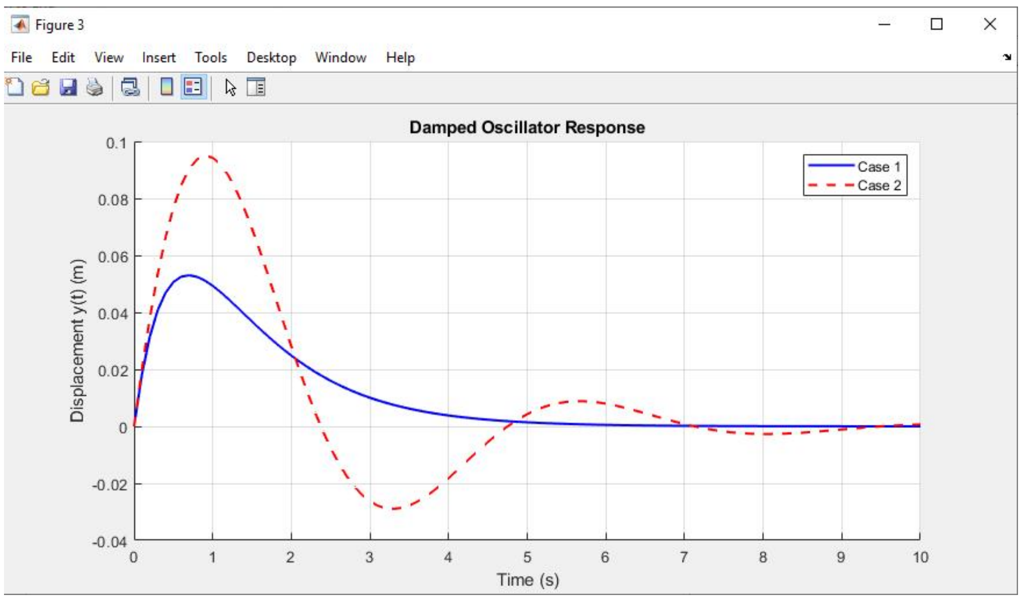
FUNCTION:DampedOscillations.m

```
function yt = dampedOscillation(y0, zeta, wn, t, theta)

   % DAMPEDOSCILLATION Calculate response of damped harmonic oscillator

   % Inputs:

   %   y0    - Initial displacement [m]

   %   zeta  - Damping ratio

   %   wn    - Natural frequency [rad/s]

   %   t     - Time vector [s]

   %   theta - Phase angle [rad]

   % Output:

   %   yt    - Displacement response [m]
```

```
  % Calculate damped natural frequency
  wd = wn * sqrt(1 - zeta^2);

  % Calculate amplitude coefficient
  amplitude = y0/ sqrt(1 - zeta^2);
   % Compute response
  yt = amplitude * exp(-zeta * wn * t) .* sin(wd * t + theta);
 end
```

SCRIPT: plot_damped_oscillations.m

```
% Time vector
t = 0:0.1:10;

% Case 1 Parameters
y0_1 = 0.15;
wn_1 = sqrt(2);
zeta_1 = 3/(2*sqrt(2));
theta_1 = 0;

% Case 2 Parameters
y0_2 = 0.15;
wn_2 = sqrt(2);
zeta_2 = 1/(2*sqrt(2));
theta_2 = 0;

% Calculate responses
y1 = dampedOscillation(y0_1, zeta_1, wn_1, t, theta_1);
y2 = dampedOscillation(y0_2, zeta_2, wn_2, t, theta_2);

% Create figure
figure;
hold on;
plot(t, y1, 'b-', 'LineWidth', 1.5);
plot(t, y2, 'r--', 'LineWidth', 1.5);
hold off;
title('Damped Oscillator Response');
xlabel('Time (s)');
ylabel('Displacement y(t) (m)');
legend('Case 1', 'Case 2');
grid on
```



Exersice 3: MATLAB Flow Control

Use 'for' or 'while' loop to convert degrees Fahrenheit (T_f) to degrees Celsius using the following equation $\boldsymbol{T_f = \frac{9}{5} * T_c + 32}$. Use any starting temperature, increment and ending temperature (example: starting temperature=0, increment=10, ending temperature = 200).

For loop implementation

```
% Initialize parameters
startTemp = 0;

increment = 10;

endTemp = 200;

% Print header
disp('Fahrenheit   Celsius');

disp('---------------------');

% For loop implementation
for tf = startTemp:increment:endTemp

   tc = (5/9)*(tf - 32);  % Conversion

   fprintf('%7.1f°F   %6.1f°C\n', tf, tc);

end
```

```
>> temp_conversion
Fahrenheit  Celsius
---------------------
   0.0°F   -17.8°C
  10.0°F   -12.2°C
  20.0°F    -6.7°C
  30.0°F    -1.1°C
  40.0°F     4.4°C
  50.0°F    10.0°C
  60.0°F    15.6°C
  70.0°F    21.1°C
  80.0°F    26.7°C
  90.0°F    32.2°C
 100.0°F    37.8°C
 110.0°F    43.3°C
 120.0°F    48.9°C
 130.0°F    54.4°C
 140.0°F    60.0°C
 150.0°F    65.6°C
 160.0°F    71.1°C
 170.0°F    76.7°C
 180.0°F    82.2°C
 190.0°F    87.8°C
 200.0°F    93.3°C
```

While_loop implementation

```
% Initialize parameters
startTemp = 0;

increment = 10;

endTemp = 200;

tf = startTemp;

 % Print header
disp('Fahrenheit   Celsius');

disp('---------------------');


% While loop implementation

while tf <= endTemp

   tc = (5/9)*(tf - 32);  % Conversion

   fprintf('%7.1f°F   %6.1f°C\n', tf, tc);

   tf = tf + increment;   % Update counter
```

```
Fahrenheit   Celsius
---------------------
    0.0°F    -17.8°C
   10.0°F    -12.2°C
   20.0°F     -6.7°C
   30.0°F     -1.1°C
   40.0°F      4.4°C
   50.0°F     10.0°C
   60.0°F     15.6°C
   70.0°F     21.1°C
   80.0°F     26.7°C
   90.0°F     32.2°C
  100.0°F     37.8°C
  110.0°F     43.3°C
  120.0°F     48.9°C
  130.0°F     54.4°C
  140.0°F     60.0°C
  150.0°F     65.6°C
  160.0°F     71.1°C
  170.0°F     76.7°C
  180.0°F     82.2°C
  190.0°F     87.8°C
  200.0°F     93.3°C
```

LAB 4: UNITY AND NON-UNITY FEEDBACK SYSTEMS USING MATLAB

1.Simulation of the unity feedback system

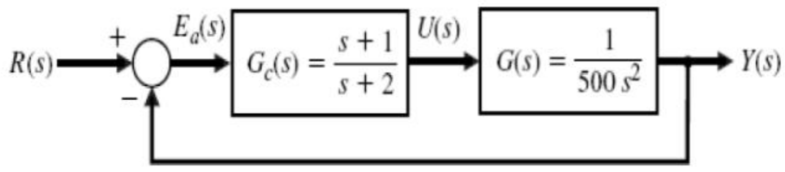
Feedback configuration-The blocks are connected as shown below and there is no transfer function H(s)defined.

- **Unity Feedback System**:
- No transfer function H(s)H(s)H(s) is defined in the feedback path (i.e., H(s)=1H(s) = 1H(s)=1).

- **Non-Unity Feedback System**:
- A specific transfer function H(s)H(s)H(s) is included in the feedback loop.

- **MATLAB Command feedback**:
- Used to compute the closed-loop transfer function.
-
- The sign parameter indicates feedback type:
- -1 for negative feedback (default)

- +1 for positive feedback



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
```

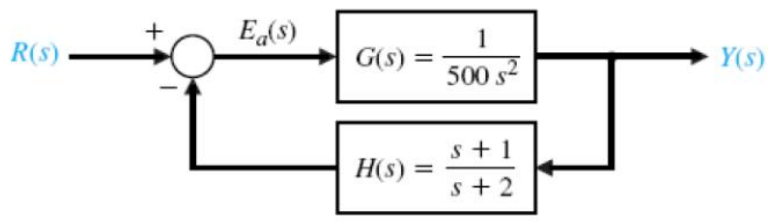
$$\frac{Y(s)}{R(s)} = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)}$$

Result:

```
%%define the plant transfer function G(s)

%G(s) = 1/(500s^2)

numg = [1];

deng = [500 0 0];

 sys1 = tf(numg, deng);


%%define the Controller Transfer Function Gc(s)

%Gc(s) = (s+1)/(s+2)

numc = [1 1];

denc = [1 2];

 sys2 = tf(numc, denc);


%Combine Plant and Controller in Series

sys3 = series(sys1, sys2);


% Negative feedback closed-loop with unity gain

sys = feedback(sys3, [1])
```

```
Transfer function:
     s + 1
-------------------------
500 s^3 + 1000 s^2 + s + 1
sys =
   s + 1
 -------------------------
 500 s^3 + 1000 s^2 + s + 1
 Continuous-time transfer function.
```

2.Simulation of non-unity feedback system-In this, feedback H(s) is defined**.**




```
>>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

```
%% Define the Plant Transfer Function G(s)
% G(s) = 1/(500s^2)
numg = [1];
deng = [500 0 0];
sysG = tf(numg, deng);

%% Define the Feedback Compensator H(s)
% H(s) = (s+1)/(s+2)
numh = [1 1];
denh = [1 2];
sysH = tf(numh, denh);

%% Calculate Closed-Loop Transfer Function for the negative feedback configuration
% Y(s)/R(s) = G(s)/(1 + G(s)H(s))
sysCL = feedback(sysG, sysH);

% Display Results
disp('Closed-loop Transfer Function Y(s)/R(s):');
sysCL
```

```
Closed-loop Transfer Function Y(s)/R(s):

sysCL =


      s + 2

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

  500 s^3 + 1000 s^2 + s + 1


Continuous-time transfer function.
```

LAB 5: BLOCK DIAGRAM REDUCTION TECHNIQUE USING MATLAB

Simulation of multi-feedback systems

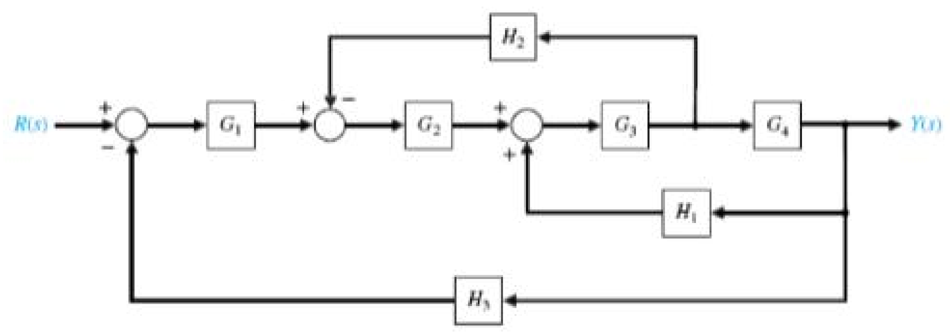
- **Series Configuration**:
- Two systems are connected end-to-end: T(s)=G1(s)G2(s)
- T(s) = G1(s)G_2(s)

- T(s)=G1 (s)G2 (s)

- **Parallel Configuration**:
- Two systems operate in parallel and their outputs are added:

  T(s)=G1(s)+G2(s)
  T(s) = G_1(s) + G_2(s)
  T(s)=G1 (s)+G2 (s)

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

```
%% Define all individual transfer functions

% Forward path transfer functions

ng1 = [1]; dg1 = [1 10]; sysg1 = tf (ng1, dg1);          % G1 = 1/(s+10)

ng2 = [1]; dg2 = [1 1]; sysg2 = tf (ng2, dg2);           % G2 = 1/(s+1)

ng3 = [1 0 1]; dg3 = [1 4 4]; sysg3 = tf (ng3, dg3);    % G3 = (s^2+1)/(s^2+4s+4)

ng4 = [1 1]; dg4 = [1 6]; sysg4 = tf (ng4, dg4);         % G4 = (s+1)/(s+6)


% Feedback path transfer functions

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);


%% Combine blocks according to block diagram algebra

% Series connection of G3 and G4

sys2 = series(sysg3, sysg4);
```

```
% Positive feedback loop with H1
sys3 = feedback (sys2, sysh1, +1);

% Series connection of G2 with previous result
sys4 = series (sysg2, sys3);

% Create the feedback path for the main loop (H2/H3)
sys1 = sysh2/sysh3;

% Main feedback loop (negative feedback by default)
sys5 = feedback(sys4, sys1);

% Series connection with G1
sys6 = series(sysg1, sys5);

% Final unity feedback loop
sys = feedback(sys6, [1]);

% Display the final transfer function
disp('Final Closed-Loop Transfer Function Y(s)/R(s):');
sys

% Display numerator and denominator separately
[num, den] = tfdata(sys, 'v');
disp('Numerator coefficients:');
disp(num')
disp('Denominator coefficients:');
disp(den')
```

```
Final Closed-Loop Transfer Function Y(s)/R(s):

sys =


        s^4 + 3 s^3 + 3 s^2 + 3 s + 2
  ---------------------------------------------------
  12 s^5 + 183 s^4 + 753 s^3 + 1434 s^2 + 1364 s + 512

Continuous-time transfer function.

Numerator coefficients:

    0
    1
    3
    3
    3
    2

Denominator coefficients:

      12
     183
     753
    1434
    1364
     512
```

LAB 6: SIMULATION OF P, PD, PI, PID CONTROLLERS

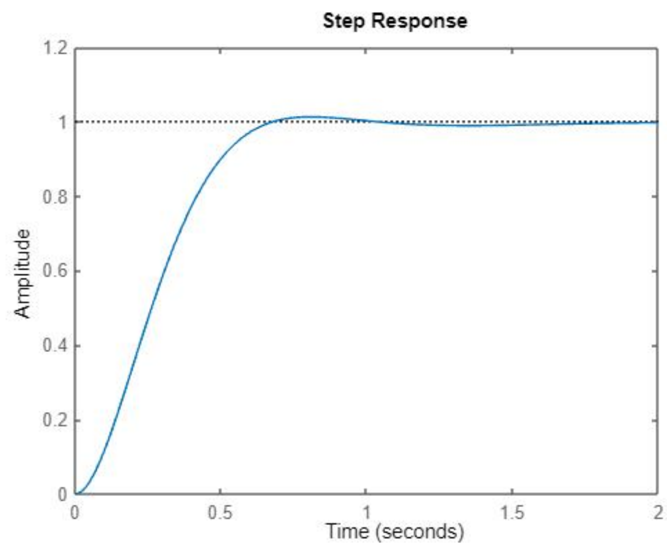
- **Control System Overview**: A feedback system where the **controller** regulates the behavior of the **plant**.

- **Controller Types**:
- **P (Proportional) Controller**: Adjusts based on the current error.
- **PI (Proportional-Integral) Controller**: Combines current error and accumulated past errors.
- **PD (Proportional-Derivative) Controller**: Combines current error and rate of error change.
- **PID Controller**: Uses all three— proportional, integral, and derivative actions.

- **MATLAB Transfer Functions**:
- Transfer functions are shown for each controller type in closed-loop form.
- Different values for KP, KI, KD, are tested to observe their effects. **Controller Effects Summary**:
- **Proportional**: Reduces rise time.
- **Integral**: Eliminates steady-state error.
- **Derivative**: Improves stability and reduces overshoot.

OPEN LOOP-STEP RESPONSE

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



PROPORTIONAL CONTROL

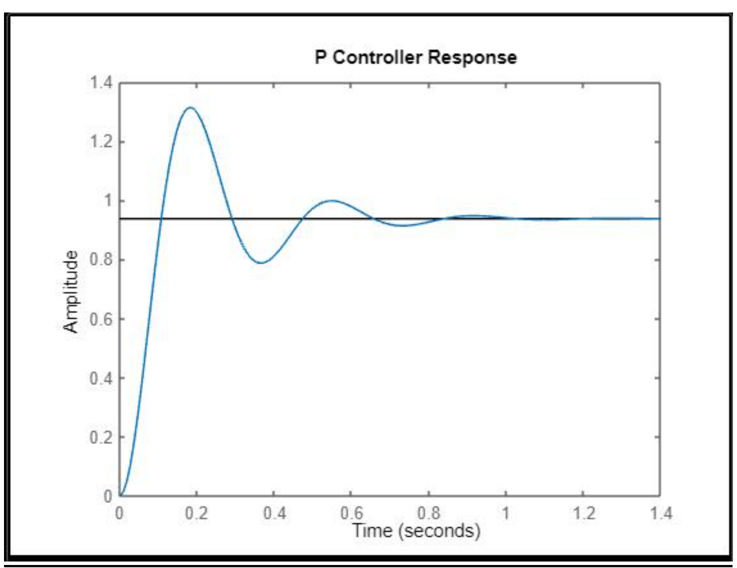
>>Kp=300;

contr=Kp;

sys_cl=feedback(contr*plant,1);

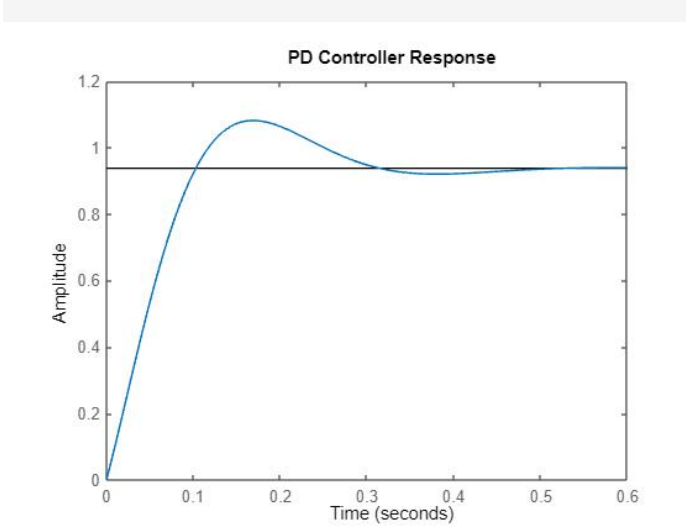
t=0:0.01:2;

step(sys_cl,t)



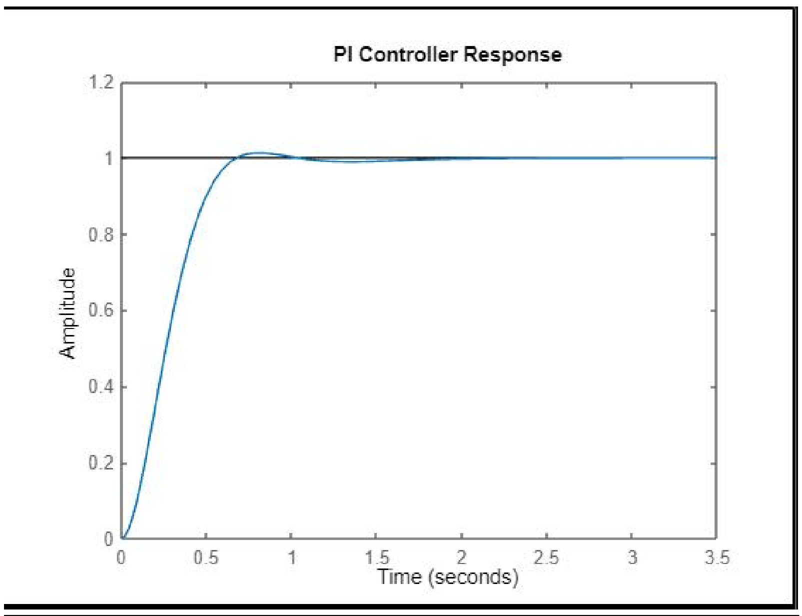
PROPORTIONAL -DERIVATIVE CONTROL

```
Kp = 300; Kd = 10;
PD_controller = tf([Kd Kp], 1);
sys_pd = feedback(PD_controller * plant, 1);
step(sys_pd);
title('PD Controller Response');
```



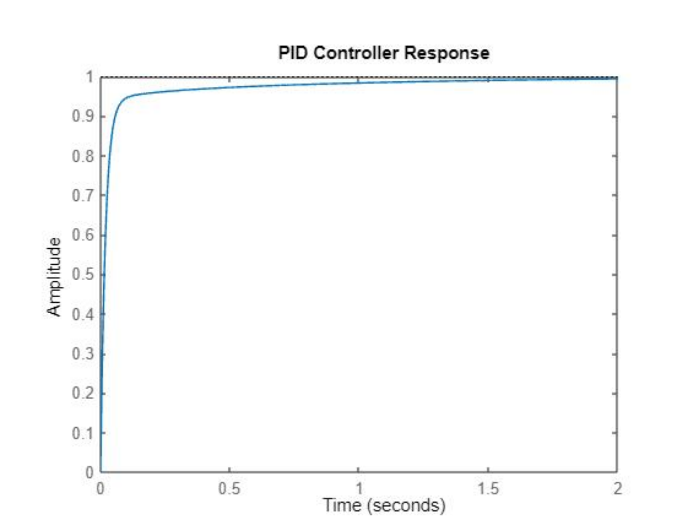
PROPORTIONAL=INTERAL CONTROL

```
Kp = 30; Ki = 70;
PI_controller = tf([Kp Ki], [1 0]);
sys_pi = feedback(PI_controller * plant, 1);
step(sys_pi);
title('PI Controller Response');
```



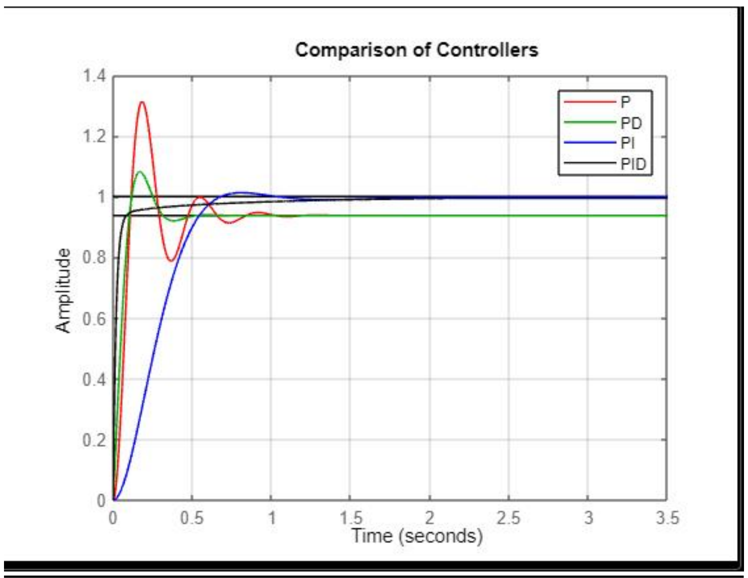
PROPORTIONAL-INTEGRAL -DERIVATIVE CONTROL

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



SYSTEM COMBINATION

```
figure;
step(sys_p, 'r'); hold on;   % P (Red)
step(sys_pd, 'g');        % PD (Green)
step(sys_pi, 'b');        % PI (Blue)
step(sys_pid, 'k');           % PID (Black)
legend('P', 'PD', 'PI', 'PID');
title('Comparison of Controllers');
grid on;
```



CONCLUSION

This series of labs provided a practical understanding of control system configurations and controller design using MATLAB. In Lab 4, students learned how to model unity and non-unity feedback **systems**, observing how feedback affects the stability and performance of control systems. Lab 5 introduced block diagram reduction techniques using series and parallel connections, enabling simplification of complex multi-loop systems into manageable forms. Finally, Lab 6 focused on simulating and analyzing the effects of P, PI, PD, and PID controllers on a closed-loop system. By adjusting the controller gains, students gained insight into how each control action (proportional, integral, derivative) influences system behavior in terms of rise time, overshoot, steady-state error, and stability.