CONTROL LAB
BODE PLOT &STATE SPACE

Name	sec	ID
يوسف اشرف محمد أبوطالب	4	9220972
يوسف خالد عمر محمود	4	9220984

Under supervision of

Dr. Mina

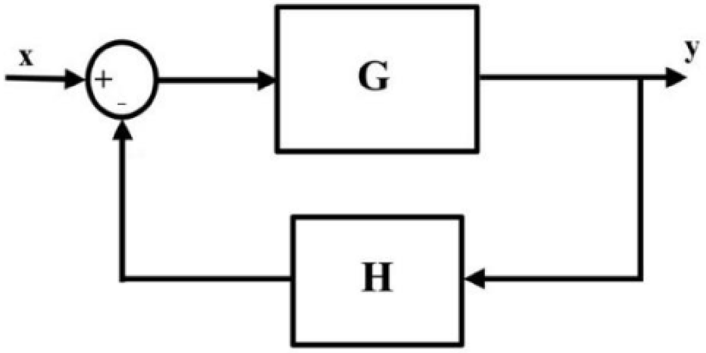
Dr. Hanan

Eng. Ahmed Amr

Part 1

Introduction

We’re going to analyze a closed loop system analytically with hand analysis and experimentally using MATLAB



Where $G(S) = \frac{1}{S(S+1)}$ & $H(S) = 1$ (unity-feedback system)

Figure 1 Closed loop System

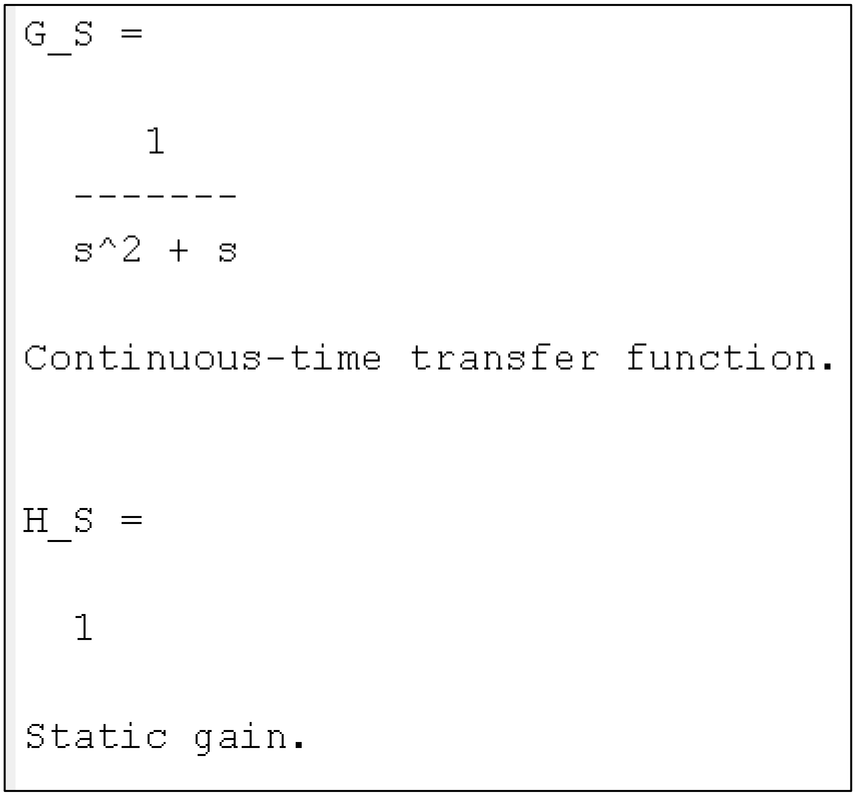
As shown in figure 1, We’re going to investigate the time domain and frequency domain characteristics of a continuous time block placed in a feedback configuration.

Q1)

Code:

```
%---------Q1---------
% Define the open-loop transfer function G(s)
num_G = 1;
den_G = [1 1 0];     % s(s+1) = s^2 + s
G_S = tf(num_G, den_G)

% Define the feedback transfer function H(s)
num_H = [1];
den_H = [1];          % Unity Feedback
H_S = tf(num_H, den_H)
```

Output:



```
G_S =

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

Continuous-time transfer function.


H_S =

 
   1

Static gain.
```

Q2)

Theoretical:

To analyze the output of the block $G(s) = \frac{1}{s(s+1)}$ for a unit step input, we will perform a hand analysis.

Step 1: Define the Input

The unit step input is defined as:

$$U(s) = \frac{1}{s}$$

Step 2: Calculate the Output

To find the output $Y(s)$ across the block $G(s)$, we can use the formula:

$$Y(s) = G(s) \cdot U(s)$$

Substituting the definitions:

$$Y(s) = \frac{1}{s(s+1)} \cdot \frac{1}{s} = \frac{1}{s^2(s+1)}$$

Step 3: Perform Inverse Laplace Transform

$$\frac{1}{s^2(s+1)} = \frac{A}{s} + \frac{B}{s^2} + \frac{C}{s+1}$$

$$\frac{1}{s^2(s+1)} = \frac{-1}{s} + \frac{1}{s^2} + \frac{1}{s+1}$$

Step 4: Finding Inverse Laplace Transform

Now we can find the inverse Laplace transforms of each term separately:

$$y(t) = -1 + t + e^{-t}$$

Interpretation

1. The term -1 indicates a negative offset in the steady state before considering the effects of the ramp and exponential decay.
2. The term t represents a linear rise in the output.
3. The term e^{-t} represents a decaying exponential that pulls the response down towards the steady-state value over time.

To analyze the stability of the system we're going to:
Step 1: Identify the Denominator

The transfer function is given by:

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

The denominator is:

$$D(s) = s(s+1) = s^2 + s$$

This gives us two equations:

4. $s = 0$
5. $s + 1 = 0 \rightarrow s = -1$

The system is **criticaly stable** as we have a pole at zero and a pole at -1 . then this process will give a ramp response.

Therefore, the overall conclusion is that the transfer function $G(s) = \frac{1}{s(s+1)}$ exhibits marginal stability due to the pole on the imaginary axis, meaning while the system does not exhibit uncontrollable growth, it does not settle to a steady state.

Code:

```
%--------Q2-------- Step Response of G(s) (Open-Loop)
% Plot step response of G(s)
draw_step(G_S, 'Open-Loop System G(s)');


function draw_step(sys, sys_name)
% Create figure
    figure;

    % Plot step response
    step(sys);
    title(['Step Response of ', sys_name]);
    grid on;

    % Display poles
    poles = pole(sys);
    disp(['Poles of ', sys_name, ':']);
    disp(poles);

    % Check stability
    if all(real(poles) < 0)
        disp('System is stable (all poles in LHP)');
    elseif any(real(poles) > 0)
        disp('System is unstable (at least one pole in RHP)');
    else
        disp('System is marginally stable (poles on imaginary axis)');
    end
end
```

Output:

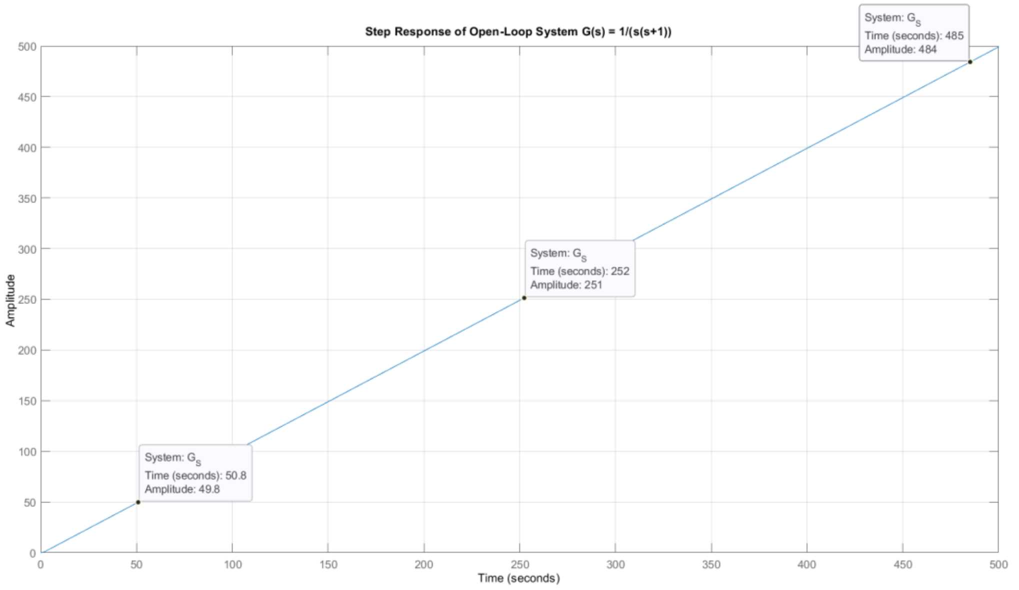
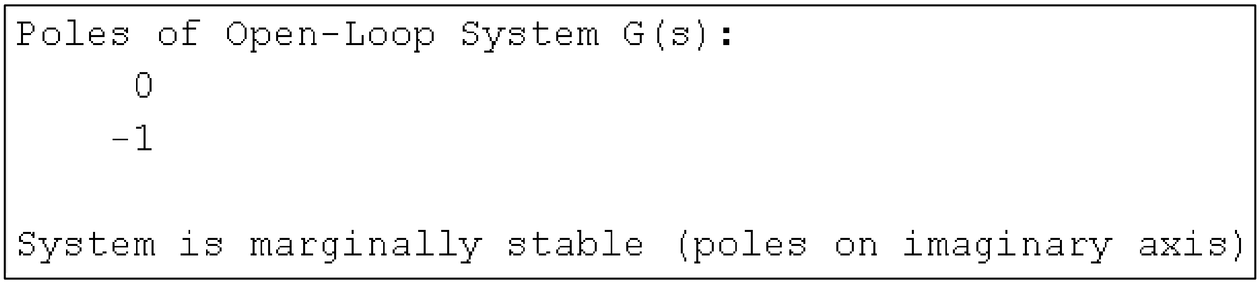


Figure 2 Gain Step Response



```
Poles of Open-Loop System G(s):
     0
    -1

System is marginally stable (poles on imaginary axis)
```

As shown in figure 2, We can say this process is not stable (critical stable) . And this expected as we have a pole in the (jw) axis .

Q3)

Theoretical

Assume the open-loop transfer function $G(s)$ is:

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

The closed-loop transfer function $T(s)$ is derived using the following formula:

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

$$T(s) = \frac{\frac{1}{s(s+1)}}{1 + \frac{1}{s(s+1)}}$$

$$1 + \frac{1}{s(s+1)} = \frac{s(s+1)+1}{s(s+1)} = \frac{s^2+s+1}{s(s+1)}$$

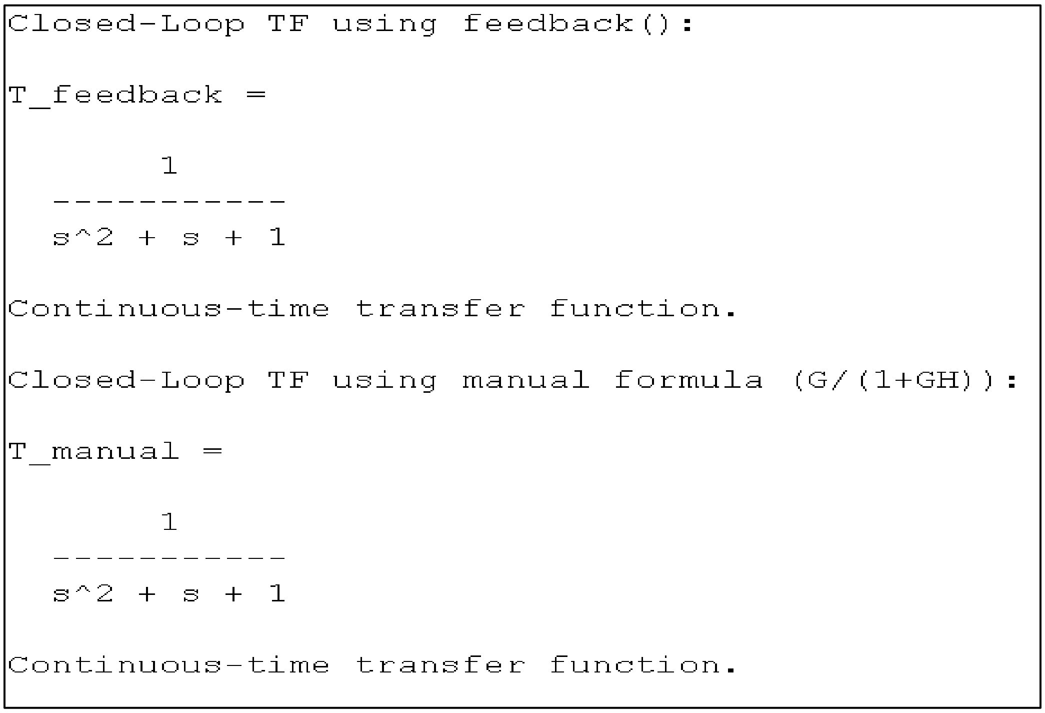
Thus, the closed-loop transfer function $T(s)$ now becomes:

$$T(s) = \frac{\frac{1}{s(s+1)}}{\frac{s^2+s+1}{s(s+1)}} = \frac{1}{s^2+s+1}$$

Code:

```
%--------Q3-------- Closed-Loop Analysis
disp('Closed-Loop TF using feedback():');
T_feedback = feedback(G_S, H_S)

disp('Closed-Loop TF using manual formula (G/(1+GH)):');
T_manual =   (1 / (1 + G_S * H_S)) * G_S;  % Equivalent to T(s) = G/(1+GH)
T_manual = minreal(T_manual)               % Cancel common terms
```

Output:



```
Closed-Loop TF using feedback():

T_feedback =

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

Continuous-time transfer function.

Closed-Loop TF using manual formula (G/(1+GH)):

T_manual =

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

Continuous-time transfer function.
```

The output and the hand analysis is the same.

Q4)

Theoretical

We derived the closed-loop transfer function:

$$T(s) = \frac{1}{s^2 + s + 1}$$

To determine stability, we need to find the poles of the transfer function $T(s)$:

$$s^2 + s + 1 = 0$$

To solve for s, we can use the quadratic formula:

$$s = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where $a = 1, b = 1, c = 1$.

Substituting these values:

$$s = \frac{-1 \pm \sqrt{1^2 - 4 \cdot 1 \cdot 1}}{2 \cdot 1} = \frac{-1 \pm \sqrt{1 - 4}}{2}$$

$$s = \frac{-1 \pm \sqrt{-3}}{2}$$

$$s = \frac{-1 \pm j\sqrt{3}}{2}$$

The poles are:

$$s = -\frac{1}{2} \pm j\frac{\sqrt{3}}{2}$$

A system is considered stable as all poles in the RHP. It is a under damped system . And also we have some damped oscillations.

Yes it is expected as the all poles in the RHP . and the poles has a imaginary part . and it is

Called a under damped system . zeta <1

Time-Domain Specifications

Starting with:

$$Y(s) = \frac{1}{s(s^2 + s + 1)}$$

We broke it down into partial fractions:

$$Y(s) = \frac{1}{s} - \frac{s+1}{s^2 + s + 1}$$

This can be rewritten as:

$$Y(s) = \frac{1}{s} - \left(\frac{s}{s^2 + s + 1} + \frac{1}{s^2 + s + 1}\right)$$

And applying laplace inverse:

$$y(t) = u(t) - \left(e^{-\frac{1}{2}t}\cos\left(\frac{\sqrt{3}}{2}t\right) + \frac{1}{\sqrt{3}}e^{-\frac{1}{2}t}\sin\left(\frac{\sqrt{3}}{2}t\right)\right)$$

Which is equal to:

$$y(t) = 1 - \frac{2}{\sqrt{3}}e^{-\frac{1}{2}t}\sin\left(\frac{\sqrt{3}}{2}t\right) \quad \text{for } t \geq 0$$

So $\zeta = 0.5$ and $\omega_n = 1$

- **Settling Time** (T_s):

$$T_s = \frac{4}{\zeta\omega_n} = \frac{4}{0.5 \cdot 1} = 8 \text{ sec}$$

- **Peak Time** $\left(T_p\right)$:

$$T_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} = \frac{\pi}{1 \cdot \sqrt{1-(0.5)^2}} = 3.6275 \text{ sec}$$

- **Rise Time** (t_r):

$$t_r = \frac{\pi - \theta}{\omega_n\sqrt{1-\zeta^2}}, \quad (\text{approx.}) \Rightarrow t_r = 2.4183 \text{ sec}$$

- **Maximum Overshoot** $\left(M_p\right)$:

$$M_p = e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}} \times 100 = 16.3\%$$

Code:

```
%--------Q4-------- Step Response of T(s)  (Closed-Loop)
% Plot step response of T(s)
draw_step(T_feedback, 'Closed-Loop System T(s)');
```

Output:

```
System: Closed-Loop System T(s)

Poles: -0.5-0.86603i          -0.5+0.86603i

Stability: stable (all poles in LHP)

Over shoot MP: 16.2929% at t = 3.592 sec

Damping ratio (ζ): 0.500

Natural frequency (ωn): 1.000 rad/s

Settling time (2%): 8.1051 sec

Rise time (10-90%): 1.6579 sec

Steady-state value: 1.0014
```

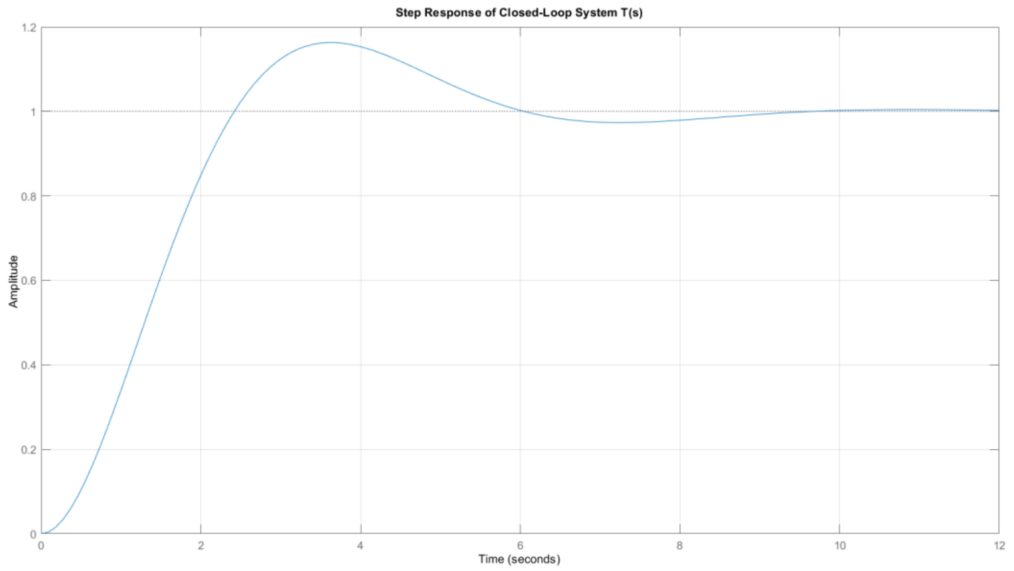


Figure 3 closed loop Step Response

As shown in figure 3, the plot shows that it has some oscillations and the location of the pole in the experiment is equal to my hand analysis. And the damped oscillations is almost equal.

Theoretical vs Experimental:

	Theoretical	Experimental
Settling Time (T_s)	8 sec	8.1051 sec
Peak Time (T_p)	3.6275 sec	3.592 sec
Rise Time (t_r)	2.4183 sec	1.6579 sec
Maximum Overshoot (M_p)	16.3%	16.2929%

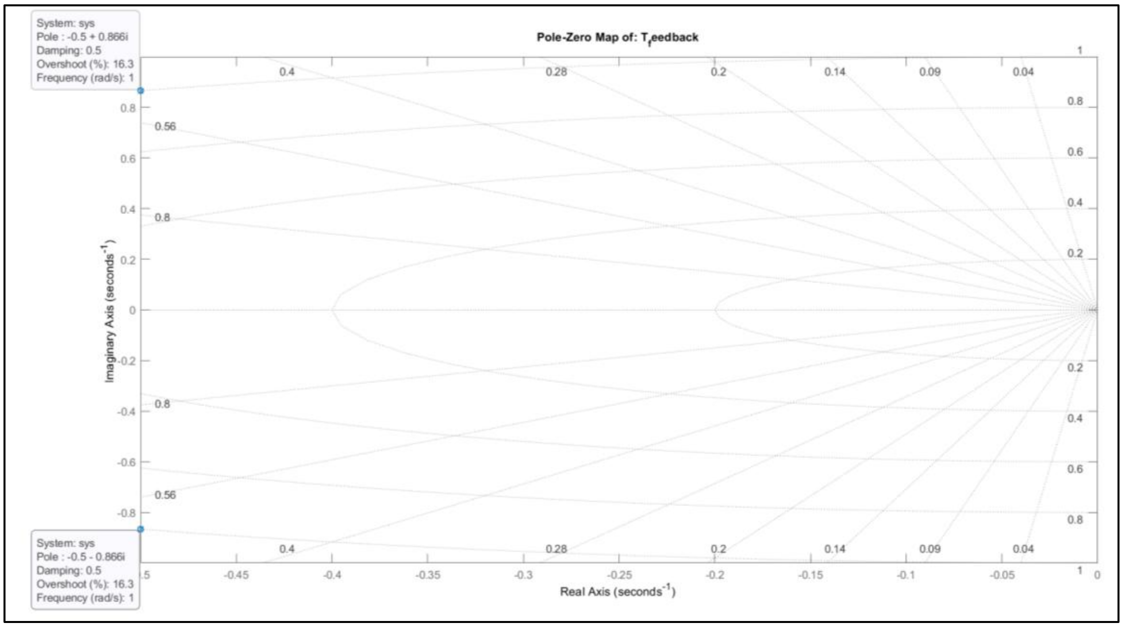
We can say the theoretical is almost equal to the Experimental.

Q5)

Code:

```
function [poles] = draw_poles(sys)

    % Create figure
    figure;

    % Plot pole-zero map
    pzmap(sys);
    title(['Pole-Zero Map of: ' inputname(1)]);
    grid on;

    % Get poles
    poles = pole(sys);

    % Display poles
    disp(['Poles of ' inputname(1) ':']);
    disp(poles);


    % Damping characteristics (for complex poles)
    if ~isreal(poles)
        [wn, zeta] = damp(sys);
        fprintf('Damping ratio (?): %.3f\n', zeta(1));
        fprintf('Natural frequency (?n): %.3f rad/s\n', wn(1));
    end

end
```

Output:



```
Poles of T_feedback:
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i


Damping ratio (ζ): 0.500
Natural frequency (ωn): 1.000 rad/s
```

Theoretical vs Experimental:

From Q4:

	Theoretical	Experimental
Poles	$-\frac{1}{2} \pm j\frac{\sqrt{3}}{2}$	-0.5000 ± 0.8660i
Natural frequency (ωn)	1 rad/s	1.000 rad/s
Damping ratio (ζ)	0.5	0.500

We can conclude that the experimental poles are ≈ theoretical ones so it agrees with the results from Q4.

Q6)

Theoretical

From Q4 we inducted that:

$$y(t) = 1 - \frac{2}{\sqrt{3}} e^{-\frac{1}{2}t} \sin\left(\frac{\sqrt{3}}{2} t\right) \quad \text{for } t \geq 0$$

as $t \to \infty$:

$$y(t) = 1 - \frac{2}{\sqrt{3}} \cdot 0 \cdot (\text{bounded term})$$

Thus,

$$y(t) \to 1 \quad \text{as } t \to \infty$$

For the E(S)

we need to find the **error signal** $e(t)$ in the context of a control system represented by:

$$E(s) = \frac{S \cdot R(s)}{1 + G(s)H(s)} = \frac{S/S}{1 + G(s)H(s)} = \frac{1}{1 + G(s)}$$

Thus,

$$E(s) = \frac{s(s+1)}{s^2 + s + 1}$$

The steady-state error (ESS) can be found using the final value theorem in the Laplace domain:

$$\text{ESS} = \lim_{t \to \infty} e(t) = \lim_{s \to 0} sE(s)$$

This means we need to calculate $sE(s)$ and take the limit as s approaches 0.

Now we evaluate the limit:

$$\text{ESS} = \lim_{s \to 0} \frac{s^2(s+1)}{s^2 + s + 1}$$

$$\text{ESS} = \frac{0}{1} = 0$$

The steady-state error (ESS) for the system given the time response is:

$$\text{ESS} = 0$$

Output:

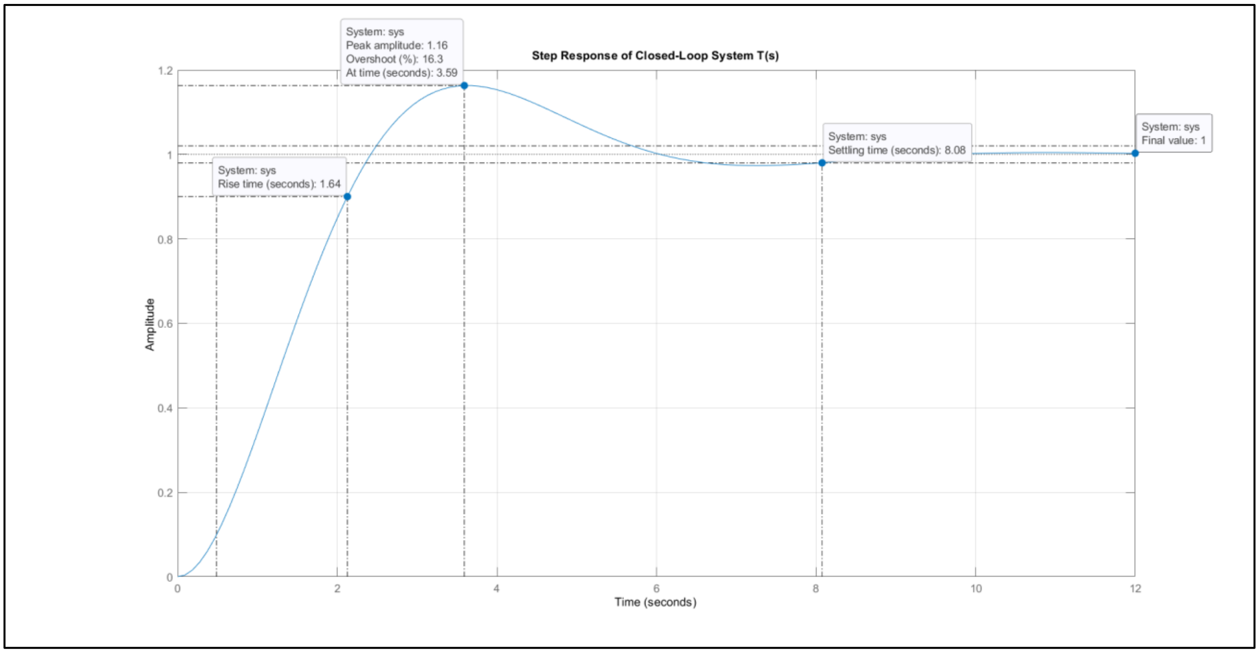


Figure 4 characteristics of the step response

From Q4)

characteristic	Value
Settling Time (T_s)	8.1051 sec
Peak Time $\left(T_p\right)$	3.592 sec
Rise Time (t_r)	1.6579 sec
Maximum Overshoot $\left(M_p\right)$	16.2929%
Ess	0.0014

This also answers Q7)

As the Ess is 0.0014, And yes it agrees with the second order system analysis.

Q8

Code:

```
function [ess, t_out, y_out] = draw_ramp(sys, t_end, zoom_time)
{
    % Set defaults if not provided
    if nargin < 2
        t_end = 100;
    end
    if nargin < 3
        zoom_time = 700;
    end

    % Create time vector
    t = 0:0.1:t_end;

    %getting the ramp
    ramp = tf(1,[1 0]);

    % Get response data
    [y_sys, t_sys] = step(sys.*ramp, t);
    [y_ideal, t_ideal] = step(ramp, t);

    % Create figure with three subplots
    figure;

    % Subplot 1: Ideal ramp input
    subplot(2,1,1);
    plot(t_ideal, y_ideal, 'b');
    hold on;
    plot(t_sys, y_sys, 'r--');
    title('Ramp Response');
    xlabel('Time (sec)');
    ylabel('Amplitude');
    legend('Ideal', 'System', 'Location', 'northwest');
    grid on;
    hold off;

    % Subplot 2: Zoomed comparison
    subplot(2,1,2);
    plot(t_ideal, y_ideal, 'b');
    hold on;
    plot(t_sys, y_sys, 'r--');
    xlim([zoom_time-50 zoom_time+50]);
    title(['Zoomed Comparison at t = ', num2str(zoom_time), ' sec']);
    xlabel('Time (sec)');
    ylabel('Amplitude');
    legend('Ideal', 'System', 'Location', 'northwest');
    grid on;
    hold off;

end
```

Output:

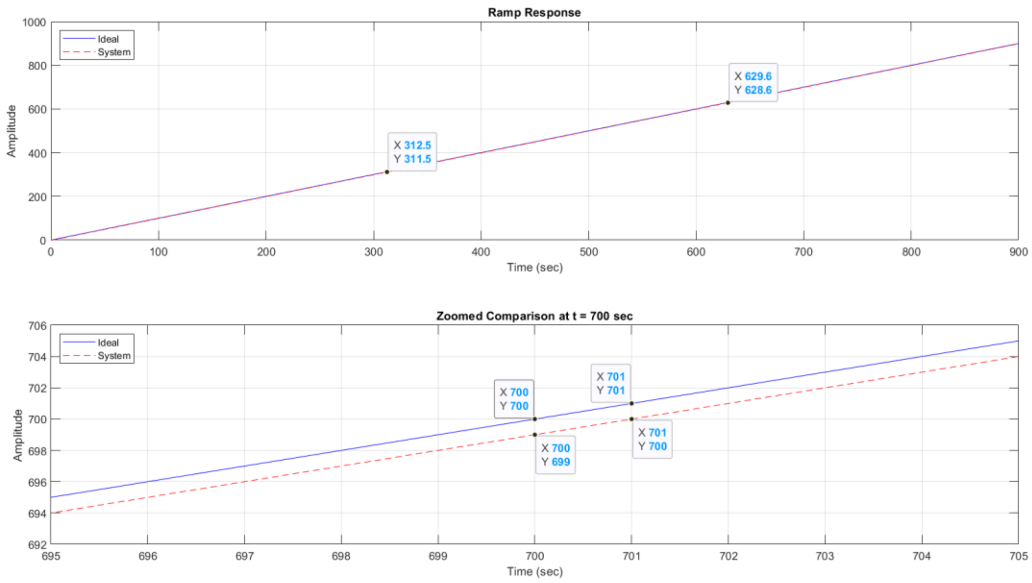


Figure 5 System Ramp Response

Steady-state error (ess): 1

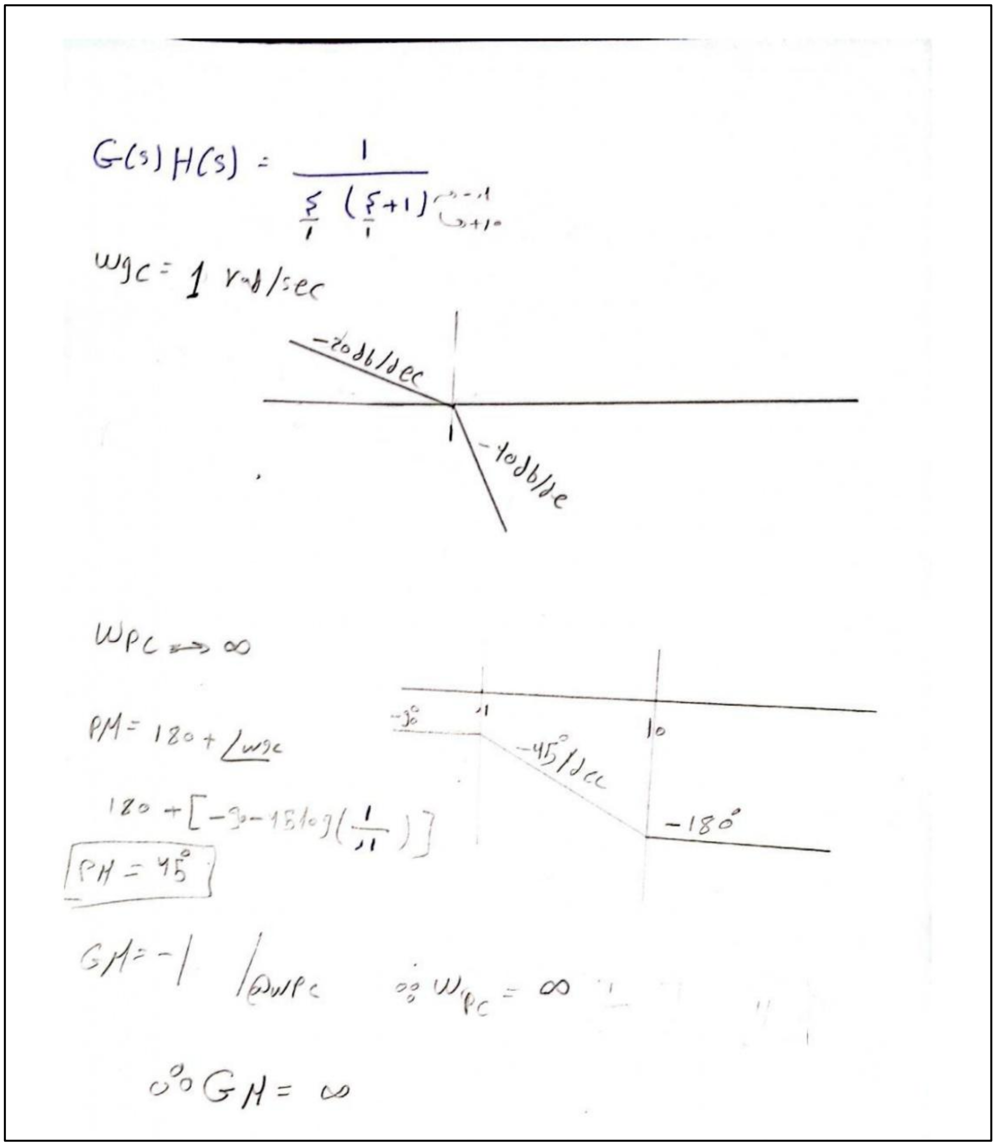
Hand analysis:

$$ess = \lim\left(\frac{\frac{1}{s(s+1)}}{\frac{s^2+s+1}{s(s+1)}}\right) = 1$$

As shown in figure 5, the theoretical agrees with the experimental.

Q9)

Theo


-1
+10
-20db/dec
1
-40db/de
-90°
.1
10
-45°/dec
-180°

$$G(s)H(s) = \frac{1}{\frac{s}{1}\left(\frac{s}{1}+1\right)}$$

$\omega_{gc} = 1$ rad/sec

$\omega_{pc} \Rightarrow \infty$

$PM = 180 + \angle\omega_{gc}$

$180 + \left[-90 - 45\log\left(\frac{1}{.1}\right)\right]$

$\boxed{PM = 45^\circ}$

$GM = -1 \Big/_{@\omega_{pc}}$ $\therefore \omega_{pc} = \infty$

$\therefore GM = \infty$

Code:

```matlab
function [Gm, Pm, Wgc, Wpc] = draw_Bode_Plot(sys)
% BODE_PLOT Analyzes system stability margins and compares margin()
%   Bode_Plot(sys)
%
%   Input:
%       sys - Transfer function (tf object or state-space model)
%   Outputs:
%       Gm - Gain margin (dB)
%       Pm - Phase margin (degrees)
%       Wgc - Gain crossover frequency (rad/sec)
%       Wpc - Phase crossover frequency (rad/sec)

    % Create margin plot
    figure;
    margin(sys);
    grid on;

    % Get stability margins
    [Gm, Pm, Wgc, Wpc] = margin(sys);

    % Display results
    disp(['=== Stability Margins for ' inputname(1) ' ===']);
    disp(['Gain Margin: ', num2str(Gm), ' dB at ', num2str(Wgc), ' rad/s']);
    disp(['Phase Margin: ', num2str(Pm), '° at ', num2str(Wpc), ' rad/s']);

end
```

Output:

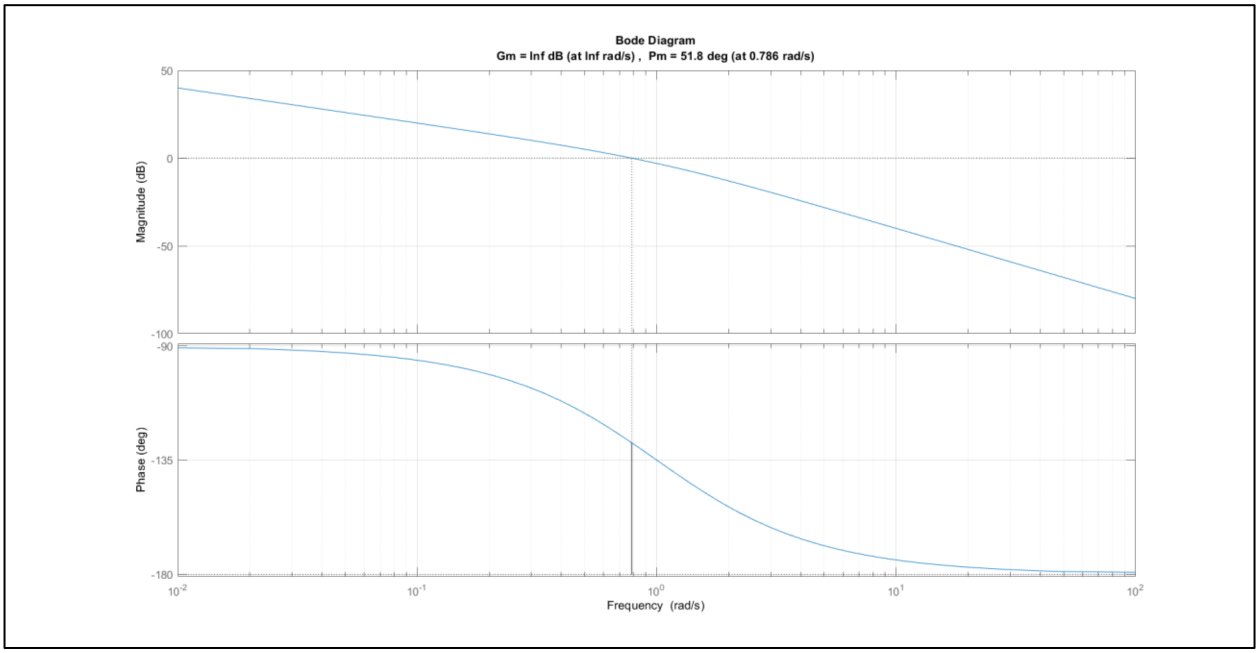


Figure 6 Systme Bode Plot

```
=== Stability Margins for  ===

Gain Margin: Inf dB at Inf rad/s

Phase Margin: 51.8273° at 0.78615 rad/s
```

As shown in figure 6, We notices that the wps at infinity so the gain margin will be equal to infinity, but the phase margin we had small diffrence between the hand analysis and the experinital by almost (5 degree)

Part2

Introduction:

We’re going to investigate the properties of the system described by a state space representation analytically with hand analysis and experimentally using MATLAB.

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \qquad y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

Figure 7 System State Space Representation

As shown in figure 7, We’re given the state space representation of the system.

Q1)

Theoritical

We have the following state-space equations:

State Equation:

$$\dot{x} = Ax + Bu$$

Where:

$$A = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Output Equation:

$$y = Cx + Du$$

Where:

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = 0$$

$$H(s) = C(sI - A)^{-1}B + D$$

First, calculate $sI - A$:

$$sI = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

Thus, we have:

$$sI - A = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix} = \begin{bmatrix} s & -1 \\ 6 & s+5 \end{bmatrix}$$

Compute the Inverse $(sI - A)^{-1}$

The determinant $ad - bc$ of this matrix is:

$$s(s+5) - (-1)(6) = s^2 + 5s + 6$$

Thus, the inverse is:

$$(sI - A)^{-1} = \frac{1}{s^2 + 5s + 6} \begin{bmatrix} s+5 & 1 \\ -6 & s \end{bmatrix}$$

Multiply by B

Now we multiply by B:

$$(sI - A)^{-1}B = \frac{1}{s^2 + 5s + 6} \begin{bmatrix} s+5 & 1 \\ -6 & s \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{s^2 + 5s + 6} \begin{bmatrix} 1 \\ s \end{bmatrix}$$

Compute $C(sI - A)^{-1}B$

Next, we find:

$$C(sI - A)^{-1}B = \begin{bmatrix} 1 & 0 \end{bmatrix} \frac{1}{s^2 + 5s + 6} \begin{bmatrix} 1 \\ s \end{bmatrix} = \frac{1}{s^2 + 5s + 6}$$

$$H(s) = C(sI - A)^{-1}B + D = \frac{1}{s^2 + 5s + 6}$$

Final Transfer Function

Therefore, the transfer function of the system is:

$$H(s) = \frac{1}{s^2 + 5s + 6}$$

Q2)

Code:

```
% Given system matrices
A = [0 1; -6 -5];
B = [0; 1];
C = [1 0];
D = [0];
n = 2;                   % System order
sys = ss(A,B,C,D);       % State Space model
x0 = [0; 1];             % Initial condition

% Q2: Transfer function conversion
[num, den] = ss2tf(A,B,C,D);
syms s
TF_Manual = C*inv(s*eye(n)-A)*B + D;
TF_builtin = tf(num,den);
```

Output:

```
TF_Manual =


1/(s^2 + 5*s + 6)



TF_builtin =


      1
  -------------
  s^2 + 5 s + 6


Continuous-time transfer function.
```

Q3:

Theoretical:

$$\Phi(s) = \frac{1}{(s+2)(s+3)}\begin{bmatrix} s+5 & 1 \\ -6 & s \end{bmatrix}$$

$$\Phi(s) = \begin{bmatrix} \dfrac{s+5}{(s+2)(s+3)} & \dfrac{1}{(s+2)(s+3)} \\ \dfrac{-6}{(s+2)(s+3)} & \dfrac{s}{(s+2)(s+3)} \end{bmatrix}$$

First Entry: $\frac{s+5}{(s+2)(s+3)}$

$$\frac{s+5}{(s+2)(s+3)} = \frac{3}{s+2} + \frac{-2}{s+3}$$

Second Entry: $\frac{1}{(s+2)(s+3)}$

$$\frac{1}{(s+2)(s+3)} = \frac{1}{s+2} - \frac{1}{s+3}$$

Third Entry: $\frac{-6}{(s+2)(s+3)}$

$$\frac{-6}{(s+2)(s+3)} = -6\left(\frac{1}{s+2} - \frac{1}{s+3}\right)$$

Fourth Entry: $\frac{s}{(s+2)(s+3)}$

$$\frac{s}{(s+2)(s+3)} = \frac{-2}{s+2} + \frac{3}{s+3}$$

Now, substitute back into $\Phi(s)$:

$$\Phi(s) = \begin{bmatrix} \dfrac{3}{s+2} + \dfrac{-2}{s+3} & \dfrac{1}{s+2} - \dfrac{1}{s+3} \\ -6\left(\dfrac{1}{s+2} - \dfrac{1}{s+3}\right) & \dfrac{-2}{s+2} + \dfrac{3}{s+3} \end{bmatrix}$$

Inverse Laplace Transforms**:**

$$\Phi(t) = \begin{bmatrix} 3e^{-2t} - 2e^{-3t} & e^{-2t} - e^{-3t} \\ -6(e^{-2t} - e^{-3t}) & -2e^{-2t} + 3e^{-3t} \end{bmatrix}$$

Verify $\boldsymbol{\Phi(0) = I}$

When we evaluate $\Phi(0)$:

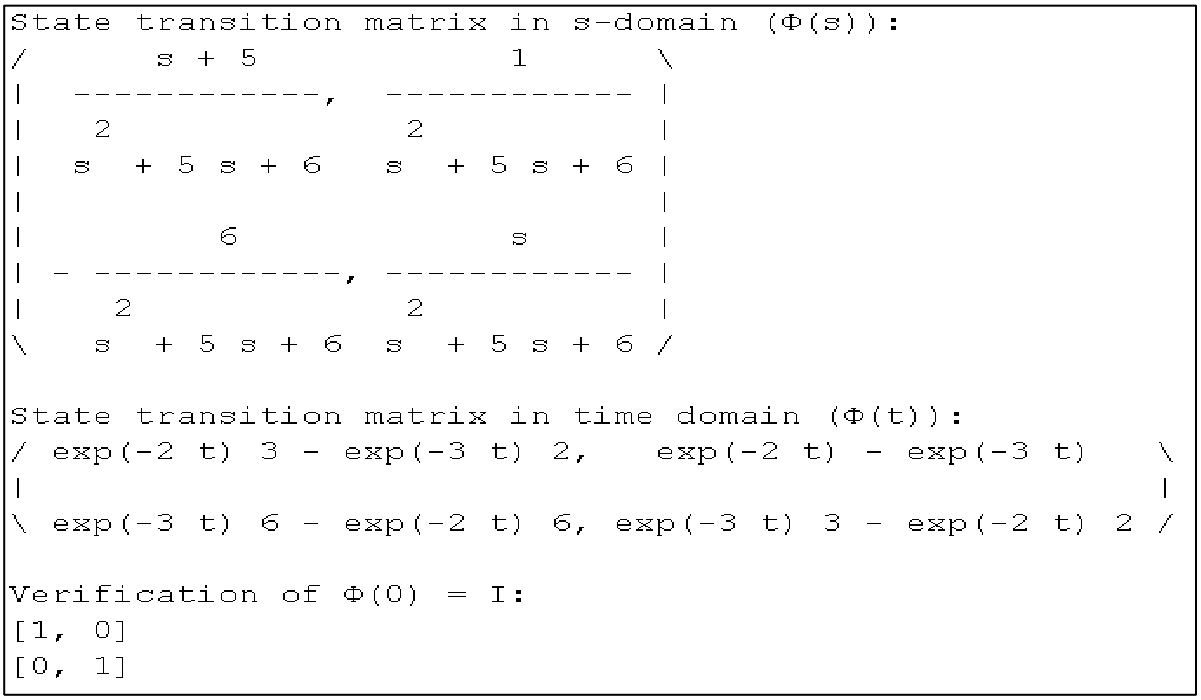
$$\Phi(0) = \begin{bmatrix} 3-2 & 1-1 \\ -6(1-1) & -2+3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

it to yield I directly.

Code

```
% Q3: State transition matrix calculation
% Compute ?(s) = [sI - A]^-1
Phi_s = inv(s*eye(n) - A);

% Compute ?(t) by inverse Laplace transform
syms t
Phi_t = ilaplace(Phi_s);

% Verify ?(0) = I
Phi_0 = subs(Phi_t, t, 0);

% Display results
disp('State transition matrix in s-domain (?(s)):');
pretty(Phi_s)

disp('State transition matrix in time domain (?(t)):');
pretty(Phi_t)

disp('Verification of ?(0) = I:');
disp(Phi_0);
```

Output



```
State transition matrix in s-domain (Φ(s)):
/      s + 5            1      \
|  ------------,  ------------ |
|   2              2           |
|  s  + 5 s + 6   s  + 5 s + 6 |
|                              |
|         6             s      |
| - ------------, ------------ |
|    2             2           |
\   s  + 5 s + 6  s  + 5 s + 6 /

State transition matrix in time domain (Φ(t)):
/ exp(-2 t) 3 - exp(-3 t) 2,   exp(-2 t) - exp(-3 t)   \
|                                                      |
\ exp(-3 t) 6 - exp(-2 t) 6, exp(-3 t) 3 - exp(-2 t) 2 /

Verification of Φ(0) = I:
[1, 0]
[0, 1]
```

it agrees with the theoretical analysis.

Q4)

Theoretical

$$\Phi(t) = \begin{bmatrix} 3e^{-2t} - 2e^{-3t} & e^{-2t} - e^{-3t} \\ -6(e^{-2t} - e^{-3t}) & -2e^{-2t} + 3e^{-3t} \end{bmatrix}$$

Hence, we have:

$$\dot{\Phi}(t) = \begin{bmatrix} -6e^{-2t} + 6e^{-3t} & -2e^{-2t} + 3e^{-3t} \\ 12e^{-2t} - 18e^{-3t} & 4e^{-2t} - 9e^{-3t} \end{bmatrix}$$

Now let's calculate $A\Phi(t)$:

$$A = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix}$$

Using the previously defined $\Phi(t)$:

$$\Phi(t) = \begin{bmatrix} 3e^{-2t} - 2e^{-3t} & e^{-2t} - e^{-3t} \\ -6(e^{-2t} - e^{-3t}) & -2e^{-2t} + 3e^{-3t} \end{bmatrix}$$

Let's compute $A\Phi(t)$:

6. **First Row** of $A\Phi(t)$:
   - First Column:

$$0 \cdot (3e^{-2t} - 2e^{-3t}) + 1 \cdot \left(-6(e^{-2t} - e^{-3t})\right) = -6(e^{-2t} - e^{-3t}) = -6e^{-2t} + 6e^{-3t}$$

And so on

Now we can assemble $A\Phi(t)$:

$$A\Phi(t) = \begin{bmatrix} -6e^{-2t} + 6e^{-3t} & -2e^{-2t} + 3e^{-3t} \\ 12e^{-2t} - 18e^{-3t} & 4e^{-2t} - 9e^{-3t} \end{bmatrix}$$

Comparing $\dot{\Phi}(t)$ and $A\Phi(t)$:

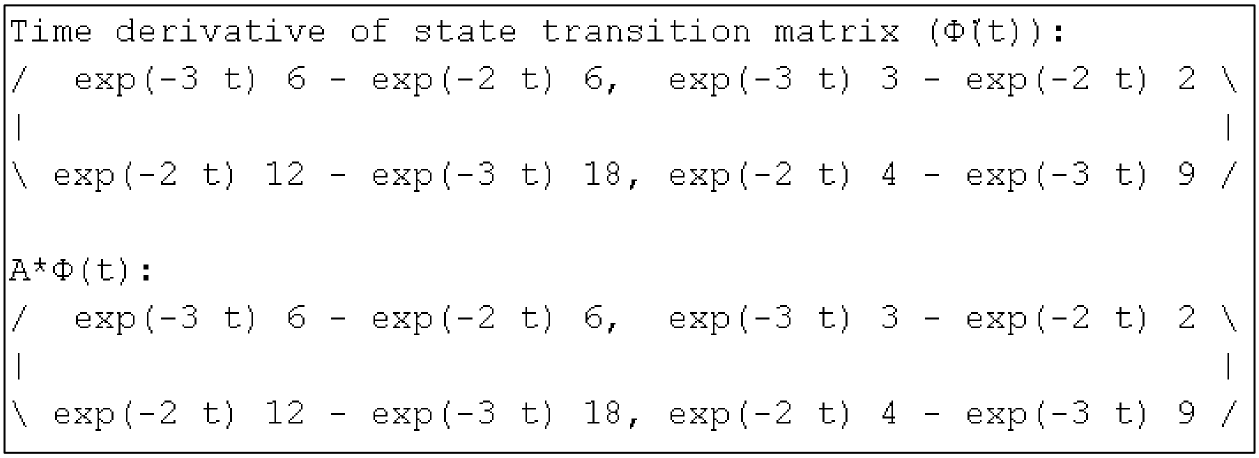
$$\dot{\Phi}(t) = A\Phi(t) = \begin{bmatrix} -6e^{-2t} + 6e^{-3t} & -2e^{-2t} + 3e^{-3t} \\ 12e^{-2t} - 18e^{-3t} & 4e^{-2t} - 9e^{-3t} \end{bmatrix}$$

we verify that $\dot{\boldsymbol{\Phi}}(\boldsymbol{t}) = \boldsymbol{A\Phi}(\boldsymbol{t})$.

Code:

```
% Q4: Verify that ??(t) = A?(t)
Phi_dot = diff(Phi_t, t);  % Take time derivative of ?(t)
A_Phi = A*Phi_t;           % Multiply A with ?(t)

disp('Time derivative of state transition matrix (??(t)):');
pretty(Phi_dot)

disp('A*?(t):');
pretty(A_Phi)

disp('Verification successful: ??(t) = A?(t)');
```

Output:



```
Time derivative of state transition matrix (Φ̇(t)):
/   exp(-3 t) 6 - exp(-2 t) 6,  exp(-3 t) 3 - exp(-2 t) 2 \
|                                                         |
\ exp(-2 t) 12 - exp(-3 t) 18, exp(-2 t) 4 - exp(-3 t) 9 /

A*Φ(t):
/   exp(-3 t) 6 - exp(-2 t) 6,  exp(-3 t) 3 - exp(-2 t) 2 \
|                                                         |
\ exp(-2 t) 12 - exp(-3 t) 18, exp(-2 t) 4 - exp(-3 t) 9 /
```

So we can see that it agrees with the theoretical analysis.

Q5)

Theoretical

Step 1: Controllability

the controllability matrix $\mathcal{C}$:

$$\mathcal{C} = [B \quad AB]$$

1. First, compute AB:

$$AB = A \cdot B = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -5 \end{bmatrix}$$

So, we have:

$$\mathcal{C} = [B \quad AB] = \begin{bmatrix} 0 & 1 \\ 1 & -5 \end{bmatrix}$$

Rank of $\boldsymbol{\mathcal{C}}$:

The determinant ($\det(\mathcal{C})$) is:

$$\det \begin{bmatrix} 0 & 1 \\ 1 & -5 \end{bmatrix} = (0 \cdot -5) - (1 \cdot 1) = -1$$

Since the determinant is non-zero, $\mathcal{C}$ has full rank. Therefore, the system is controllable.

Step 2: Observability

the observability matrix $\mathcal{O}$:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \end{bmatrix}$$

1. compute CA:

$$CA = C \cdot A = [1 \quad 0] \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix} = [0 \quad -5]$$

Thus, we have:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -5 \end{bmatrix}$$

Rank of $\boldsymbol{\mathcal{O}}$:

The determinant of this matrix is non-zero. Thus, indicating that the rank is 2.

Conclusion

The system is controllable and observable

Code:

```matlab
% Q5 Check Controllability and Observability
% Check Controllability
Co = ctrb(A, B);  % Controllability matrix
rank_Co = rank(Co);
disp('Controllability Matrix:');
disp(Co);
disp(['Rank of Controllability Matrix: ', num2str(rank_Co)]);

if rank_Co == n
    disp('System is Controllable (as expected)');
else
    disp('System is Not Controllable (unexpected for this system)');
end

% Check Observability
Ob = obsv(A, C);  % Observability matrix
rank_Ob = rank(Ob);
disp('Observability Matrix:');
disp(Ob);
disp(['Rank of Observability Matrix: ', num2str(rank_Ob)]);

if rank_Ob == n
    disp('System is Observable (as expected)');
else
    disp('System is Not Observable (unexpected for this system)');
end
```

Output:

```
Controllability Matrix:

     0     1

     1    -5


Rank of Controllability Matrix: 2

System is Controllable (as expected)

Observability Matrix:

     1     0

     0     1


Rank of Observability Matrix: 2

System is Observable (as expected)
```

Q6)

Theoretical

The state equation is given by:

$$x(t) = \Phi(t)x(0)$$

$$\Phi(t) = \begin{bmatrix} 3e^{-2t} - 2e^{-3t} & e^{-2t} - e^{-3t} \\ -6(e^{-2t} - e^{-3t}) & -2e^{-2t} + 3e^{-3t} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Final Answers

1. **Homogeneous state solution**:

$$x(t) = \begin{bmatrix} e^{-2t} - e^{-3t} \\ -2e^{-2t} + 3e^{-3t} \end{bmatrix}$$

2. **Output response**:

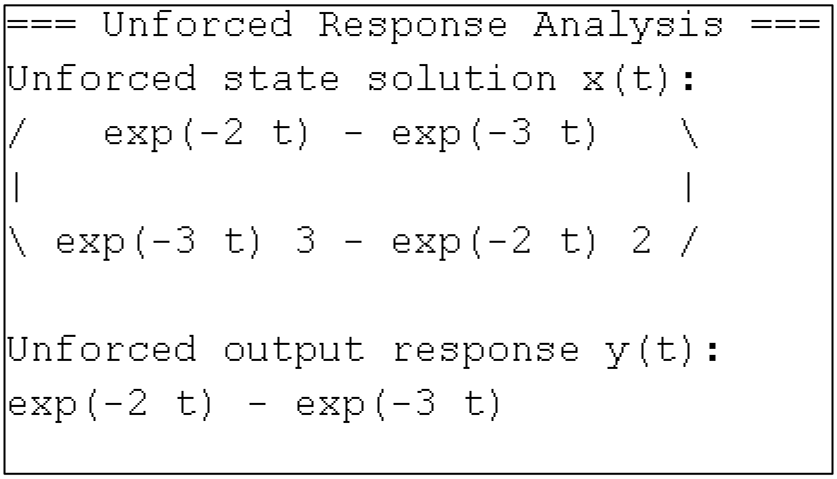
$$x(t) = \mathrm{C}\Phi(t)x(0)$$

$$y(t) = e^{-2t} - e^{-3t}$$

Code:

```
% Q6: Unforced (Homogeneous) Response
disp('=== Unforced Response Analysis ===');

% Compute state solution x(t) = ?(t)*x0
x_t = Phi_t * x0;

disp('Unforced state solution x(t):');
pretty(x_t)

% Compute output solution y(t) = C*x(t) + D*u(t)
% Since u(t)=0 for unforced response:
y_t = C*x_t + D*0;

disp('Unforced output response y(t):');
pretty(y_t)

% Plot the results
t_vals = linspace(0, 5, 500);  % Time vector from 0 to 5 seconds

% Convert symbolic expressions to numeric functions
x1_func = matlabFunction(x_t(1));
x2_func = matlabFunction(x_t(2));
y_func = matlabFunction(y_t);

% Evaluate solutions
x1_vals = arrayfun(x1_func, t_vals);
x2_vals = arrayfun(x2_func, t_vals);
y_vals = arrayfun(y_func, t_vals);

% Plot state responses
figure;
subplot(2,1,1);
plot(t_vals, x1_vals, 'b', 'LineWidth', 2);
hold on;
plot(t_vals, x2_vals, 'r--', 'LineWidth', 2);
title('Unforced State Response');
xlabel('Time (s)');
ylabel('State Values');
legend('x_1(t)', 'x_2(t)');
grid on;

% Plot output response
subplot(2,1,2);
plot(t_vals, y_vals, 'm', 'LineWidth', 2);
title('Unforced Output Response y(t)');
xlabel('Time (s)');
ylabel('Output y(t)');
grid on;

% Compare with MATLAB's built-in initial() function
[~,t_num,x_num] = initial(sys,x0,t_vals(end));
y_num = x_num*C';  % Equivalent to C*x since D=0

% Display symbolic solutions
disp(' ');
disp('Analytic Solutions:');
disp('x1(t) = '); pretty(x_t(1))
disp('x2(t) = '); pretty(x_t(2))
disp('y(t) = '); pretty(y_t)
```

Output:



```
=== Unforced Response Analysis ===
Unforced state solution x(t):
/    exp(-2 t) - exp(-3 t)    \
|                             |
\ exp(-3 t) 3 - exp(-2 t) 2 /

Unforced output response y(t):
exp(-2 t) - exp(-3 t)
```

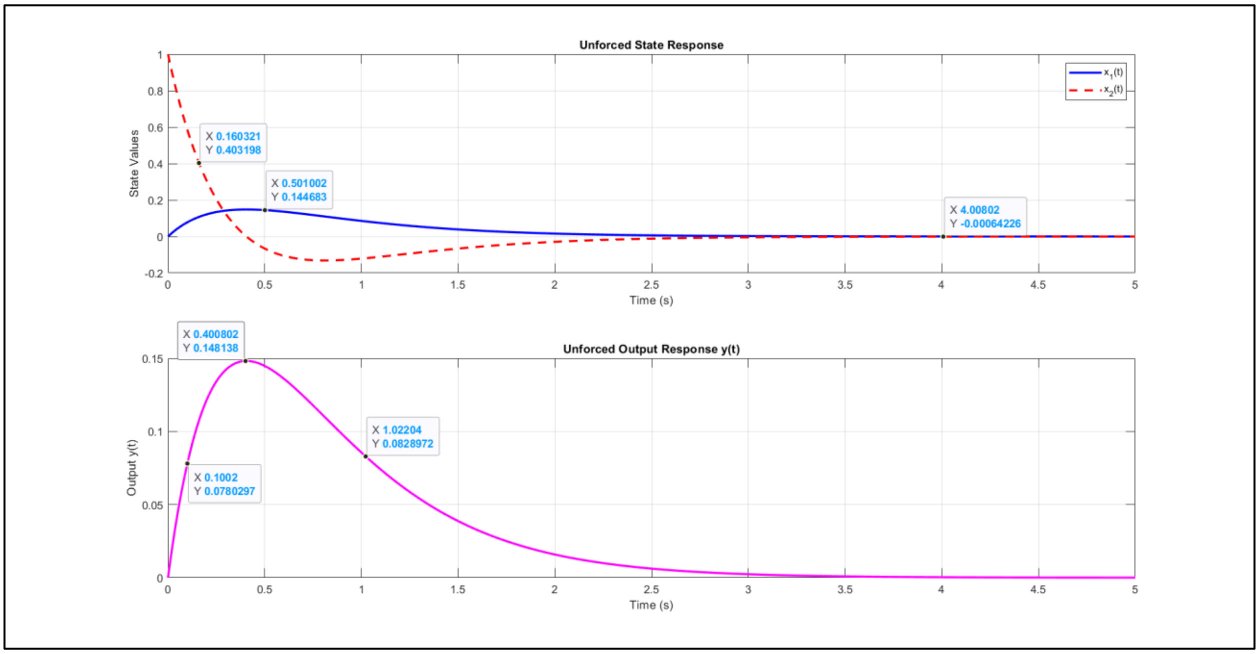


Figure 8 Unforced Output Plot

As shown in figure 8, The **experimental** unforced output agrees with the theoretical one.

Q7)

$$X(t) = \Phi(t)X(0) + \int_0^t \Phi\,(\tau)B\,u(t-\tau)\,d\tau$$

$$\Phi(\tau)B = \begin{bmatrix} e^{-2\tau} - e^{-3\tau} \\ -2e^{-2\tau} + 3e^{-3\tau} \end{bmatrix}$$

$$\int_0^t \Phi\,(\tau)B\,u(t-\tau)\,d\tau = \begin{bmatrix} \displaystyle\int_0^t e^{-2\tau} - e^{-3\tau}\,d\tau \\ \displaystyle\int_0^t -2e^{-2\tau} + 3e^{-3\tau}\,d\tau \end{bmatrix} = \begin{bmatrix} \frac{1}{2}e^{-2t} - \frac{1}{3}e^{-3t} + \frac{5}{6} \\ -e^{-2t} + e^{-3t} + t \end{bmatrix}$$

$$\Rightarrow X(t) = \begin{bmatrix} e^{-2t} - st \\ -2e^{-2t} + 3e^{-3t} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}e^{-2t} - \frac{1}{3}e^{-3t} + \frac{5}{6} \\ -e^{-2t} + e^{-3t} + t \end{bmatrix} = \begin{bmatrix} \frac{3}{2}e^{-2t} - \frac{4}{3}e^{-3t} + \frac{5}{6} \\ -3e^{-2t} + 4e^{-3t} \end{bmatrix}$$

$$y(t) = C\Phi(t)X(0) + \int_0^t C\,\Phi(\tau)B\,u(t-\tau)\,d\tau$$

$$C\Phi(\tau)B = [e^{-2\tau} - e^{-3\tau}]$$

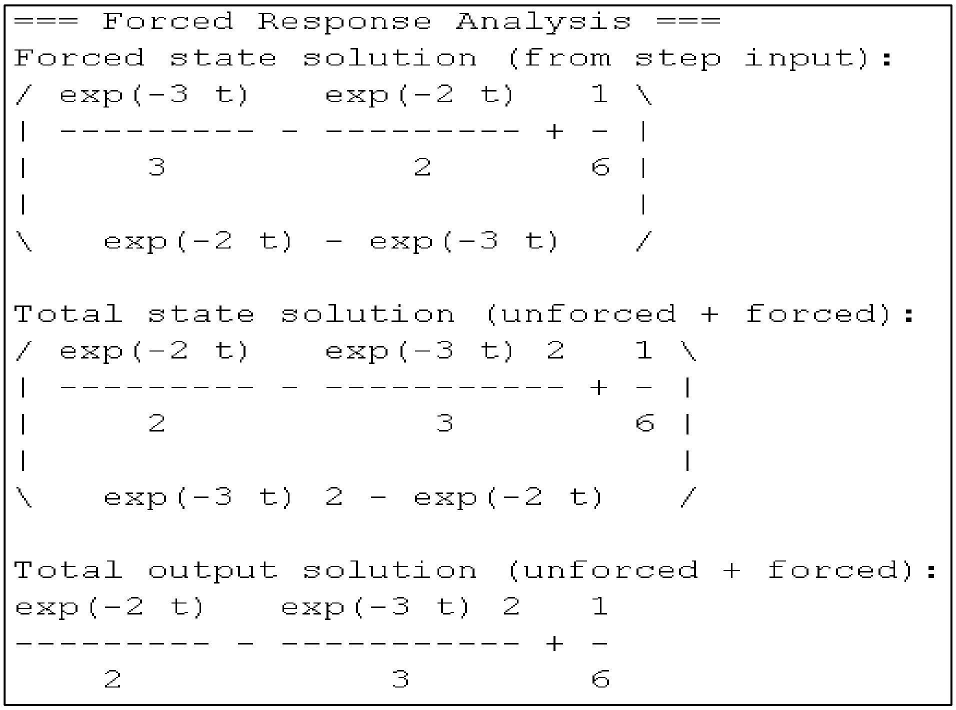
$$\int_0^t (e^{-2\tau} - e^{-3\tau})\,d\tau = \left[-\frac{1}{2}e^{-2\tau} + \frac{1}{3}e^{-3\tau} + \frac{1}{6}\right]_0^t$$

$$\Rightarrow y(t) = [e^{-2t} - e^{-3t}] + \left[-\frac{1}{2}e^{-2t} + \frac{1}{3}e^{-3t} + \frac{1}{6}\right]$$

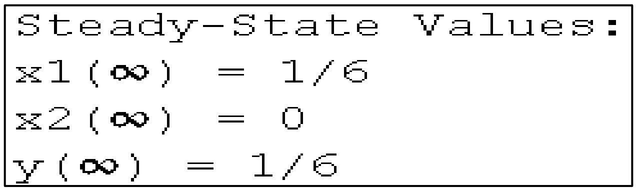
Code:

```matlab
% Q7: Forced Response Analysis (Unit Step Input)
disp('=== Forced Response Analysis ===');
% Using Frequency Domain Approach
U_s = 1/s;  % Laplace transform of unit step
U_t =ilaplace(U_s);

% Compute forced component in frequency domain
X_forced_s = Phi_s * B * U_s;

% Convert to time domain
x_forced_t = ilaplace(X_forced_s);

% Total solution (homogeneous + forced)
x_total_t = x_t + x_forced_t;

% Output solution
y_total_t = C*x_total_t + D*U_t;  % D*u(t) where u(t)=1 for t>0

disp('Forced state solution (from step input):');
pretty(x_forced_t)

disp('Total state solution (unforced + forced):');
pretty(x_total_t)

% Direct evaluation using subs()
x1_vals = double(subs(x_total_t(1), t, t_vals));
x2_vals = double(subs(x_total_t(2), t, t_vals));
y_vals = double(subs(y_total_t, t, t_vals));

% Plot results
figure;

% State responses
subplot(2,1,1);
plot(t_vals, x1_vals, 'b', 'LineWidth', 2);
hold on;
plot(t_vals, x2_vals, 'r--', 'LineWidth', 2);
title('Total State Response (Step Input)');
xlabel('Time (s)');
ylabel('State Values');
legend('Analytic x_1(t)', 'Analytic x_2(t)');
grid on;

% Output response
subplot(2,1,2);
plot(t_vals, y_vals, 'm', 'LineWidth', 2);
hold on;
title('Total Output Response y(t) (Step Input)');
xlabel('Time (s)');
ylabel('Output y(t)');
legend('Analytic y(t)');
grid on;

disp(' ');
disp('Steady-State Values:');
disp(['x1(?) = ' char(ss_x1)]);
disp(['x2(?) = ' char(ss_x2)]);
disp(['y(?) = ' char(ss_y)]);
```

Output:



```
=== Forced Response Analysis ===
Forced state solution (from step input):
/ exp(-3 t)    exp(-2 t)    1 \
| ---------- - ---------- + - |
|     3            2        6 |
|                             |
\   exp(-2 t) - exp(-3 t)    /

Total state solution (unforced + forced):
/ exp(-2 t)    exp(-3 t) 2    1 \
| ---------- - ------------ + - |
|     2             3         6 |
|                               |
\   exp(-3 t) 2 - exp(-2 t)    /

Total output solution (unforced + forced):
exp(-2 t)    exp(-3 t) 2    1
---------- - ------------ + -
    2             3         6
```



```
Steady-State Values:
x1(∞)  = 1/6
x2(∞)  = 0
y(∞)  = 1/6
```

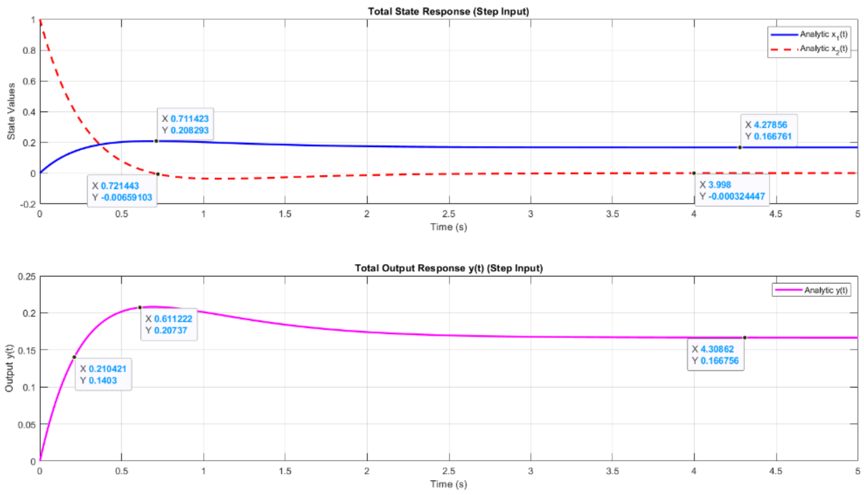


Figure 9 Forced Output Plot

As shown in figure 9,

The result of the hand analysis is equal to the result of the experiment

Q8)

Theoretical

Given:

$$T_s = 1 \quad \text{and} \quad \zeta = 0.7 \quad \Rightarrow \quad \omega_n = \frac{4}{T_s} = \frac{5.714}{1} = 5.714$$

Desired characteristic equation:

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = s^2 + 5.6s + 16 \approx s^2 + 8s + 32.653 = 0$$

We want:

$$| sI - A + BK |= 0$$

Let:

$$BK = \begin{bmatrix} 0 & 0 \\ k_1 & k_2 \end{bmatrix} \Rightarrow BK = [k_1 \quad k_2]$$

$$sI - A + BK = \begin{bmatrix} s & -1 \\ 6 + k_1 & s + 5 + k_2 \end{bmatrix}$$

$$| sI - A + BK |= s(s + 5 + k_2) + 6 + k_1 = s^2 + s(5 + k_2) + 6 + k_1$$

Compare with:

$$s^2 + 8s + 32.653$$

Matching coefficients:

- $5 + k_2 = 8 \Rightarrow k_2 = 3$
- $6 + k_1 = 32.653 \Rightarrow k_1 = 26.653$

Final result:

$$K = [26.653 \quad 3]$$

Code:

```
% Q8: State Feedback Design
disp('=== State Feedback Design ===');

% Original system step response
figure;
step(TF_builtin);
title('Original System Step Response');
grid on;

% Design specifications
zeta_desired = 0.7;      % Desired damping ratio
ts_desired = 1;          % Desired settling time (sec)

% Hand analysis to determine desired poles
wn = 4/(zeta_desired*ts_desired);  % Natural frequency from settling time
sigma = zeta_desired*wn;           % Real part of poles
wd = wn*sqrt(1-zeta_desired^2);    % Imaginary part

% Desired characteristic polynomial
desired_poly = (s + sigma + 1i*wd)*(s + sigma - 1i*wd);
desired_poly = expand(desired_poly);

% Convert to numerical polynomial
desired_coeffs = sym2poly(desired_poly);

% Hand calculation of K matrix
% Characteristic polynomial of A-BK: s^2 + (5+K2)s + (6+K1)
% Compare with desired polynomial: s^2 + 2*zeta*wn*s + wn^2

K1 = desired_coeffs(3) - 6;  % From constant term
K2 = desired_coeffs(2) - 5;  % From s term
K = [K1 K2];

disp('Desired closed-loop poles:');
disp([-sigma+1i*wd, -sigma-1i*wd]);

disp('Feedback gain matarix K:');
disp(K);

% Verification
Ac = A - B*K;
[num_2, denum_2] = ss2tf(Ac,B,C,D);
TF_state_feedback = tf(num_2, denum_2);

% Step response analysis
figure;
step_info = stepinfo(TF_state_feedback);
step(TF_state_feedback);
title('System with State Feedback');
grid on;

disp('Closed-loop system performance:');
disp(['Settling Time: ', num2str(step_info.SettlingTime), ' sec']);
disp(['Overshoot: ', num2str(step_info.Overshoot), '%']);
```

Output:

```
=== State Feedback Design ===
Desired closed-loop poles:
  -4.0000 + 4.0808i  -4.0000 - 4.0808i


Feedback gain matrix K:
   26.6531    3.0000


Closed-loop system performance:
Settling Time: 1.0463 sec
Overshoot: 4.5986%
```

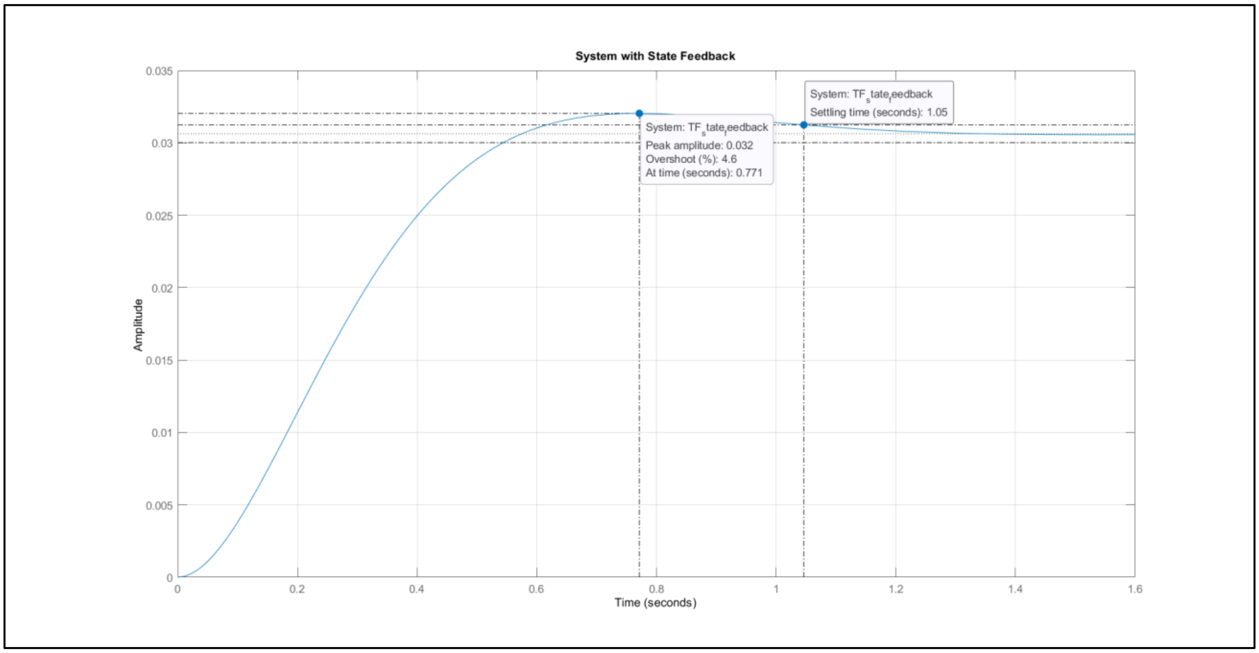


Figure 10 Desired System Response

As shown in figure 10,

characteristic	Value
Settling Time (T_s)	1.05 sec
Maximum Overshoot (M_p)	4.6%

Appendix

Bode Plot Code:

```
clc;
clear;
close all;

%--------Q1--------  Define G(s) and H(s)
% Define the open-loop transfer function G(s)
num_G = 1;
den_G = [1 1 0];     % s(s+1) = s^2 + s

% Define the feedback transfer function H(s)
num_H = [1];
den_H = [1];         % Unity Feedback
[G_S, H_S] = create_system(num_G, den_G, num_H, den_H)

%--------Q2-------- Step Response of G(s) (Open-Loop)
% Plot step response of G(s)
draw_step(G_S, 'Open-Loop System G(s)');

%--------Q3-------- Closed-Loop Analysis
disp('Closed-Loop TF using feedback():');
T_feedback = feedback(G_S, H_S)

disp('Closed-Loop TF using manual formula (G/(1+GH)):');
T_manual =   (1 / (1 + G_S * H_S)) * G_S;  % Equivalent to T(s) = G/(1+GH)
T_manual = minreal(T_manual)               % Cancel common terms

%--------Q4-------- Step Response of T(s) (Closed-Loop)
% Plot step response of T(s)
draw_step(T_feedback, 'Closed-Loop System T(s)');

%--------Q5-------- locations of the poles
draw_poles(T_feedback);

%--------Q8-------- Ramp Response
[ess, r_t_out, r_y_out] = draw_ramp(T_feedback, 700+200, 700);

%--------Q9-------- Frequency Response
[Gm, Pm, Wgc, Wpc] = draw_Bode_Plot(G_S*H_S);

%-----------Functions------

function [G_S, H_S] = create_system(num_G, den_G, num_H, den_H)
% CREATE_SYSTEM Creates open-loop and feedback transfer functions
%   [G_S, H_S] = create_system(num_G, den_G, num_H, den_H)
%
%   Inputs:
%       num_G - Numerator coefficients of G(s)
%       den_G - Denominator coefficients of G(s)
%       num_H - Numerator coefficients of H(s) (default: 1)
%       den_H - Denominator coefficients of H(s) (default: 1)
%
%   Outputs:
%       G_S - Open-loop transfer function
%       H_S - Feedback transfer function

    % Set default unity feedback if not specified
    if nargin < 3
```

```
        num_H = 1;
        den_H = 1;
    end

    % Create transfer functions
    G_S = tf(num_G, den_G)
    H_S = tf(num_H, den_H)
end

function [wn, zeta,response_info] = draw_step(sys, sys_name)
% DRAW_STEP Plots step response and returns key performance metrics
%   [response_info] = draw_step(sys, sys_name)
%
%   Inputs:
%       sys - Transfer function (tf object)
%       sys_name - Name of the system for title (string)
%
%   Outputs:
%       response_info - Structure containing:
%           .poles - System poles
%           .stability - Stability classification
%           .peak_response - Peak response value and time
%           .settling_time - Time to settle within 2% of final value
%           .rise_time - 10-90% rise time
%           .steady_state - Final steady-state value
%       Figure with step response

    % Create figure
    figure;

    % Get step response data
    [y, t] = step(sys);

    % Plot step response
    step(sys);
    title(['Step Response of ', sys_name]);
    grid on;

    % Calculate response characteristics
    response_info = struct();
    response_info.poles = pole(sys);

    % Stability determination
    if all(real(response_info.poles) < 0)
        response_info.stability = 'stable (all poles in LHP)';
    elseif any(real(response_info.poles) > 0)
        response_info.stability = 'unstable (at least one pole in RHP)';
    else
        response_info.stability = 'marginally stable (poles on imaginary
axis)';
    end

    % Peak response (overshoot)
    [response_info.peak_response.value, peak_idx] = max(y);
    response_info.peak_response.time = t(peak_idx);

    % Steady-state value (last 10% of response)
    steady_state_val = mean(y(end-round(length(y)*0.1):end));
    response_info.steady_state = steady_state_val;

    % Settling time (within 2% of steady-state)
    settled_idx = find(abs(y - steady_state_val) > 0.02*steady_state_val, 1,
'last');
    if isempty(settled_idx)
        response_info.settling_time = 0;
    else
        response_info.settling_time = t(settled_idx);
    end

    % Rise time (10% to 90% of steady-state)
    rise_start = find(y >= 0.1*steady_state_val, 1);
    rise_end = find(y >= 0.9*steady_state_val, 1);
    if ~isempty(rise_start) && ~isempty(rise_end)
        response_info.rise_time = t(rise_end) - t(rise_start);
    else
        response_info.rise_time = NaN;
    end

    % Display results in command window
    disp(['System: ', sys_name]);
    disp(['Poles: ', num2str(response_info.poles')]);
    disp(['Stability: ', response_info.stability]);
    disp(['Over shoot MP: ', num2str(100*(response_info.peak_response.value-
1)), ...
          '% at t = ', num2str(response_info.peak_response.time), ' sec']);
    % Damping characteristics (for complex poles)
    if ~isreal(response_info.poles)
        [wn, zeta] = damp(sys);
        fprintf('Damping ratio (?): %.3f\n', zeta(1));
        fprintf('Natural frequency (?n): %.3f rad/s\n', wn(1));
    end

    disp(['Settling time (2%): ', num2str(response_info.settling_time), '
sec']);
    disp(['Rise time (10-90%): ', num2str(response_info.rise_time), ' sec']);
    disp(['Steady-state value: ', num2str(response_info.steady_state)]);
end

function [poles] = draw_poles(sys)
% DRAW_POLES Plots pole-zero map and returns system poles
%   [poles] = draw_poles(sys)
%
%   Input:
%       sys - Transfer function (tf object) or state-space model
%
%   Output:
%       poles - Array of system poles
%
%   Displays:
%       - Pole-zero plot
%       - Pole locations in command window
%       - Stability information

    % Create figure
    figure;

    % Plot pole-zero map
    pzmap(sys);
    title(['Pole-Zero Map of: ' inputname(1)]);
    grid on;

    % Get poles
    poles = pole(sys);

    % Display poles
    disp(['Poles of ' inputname(1) ':']);
    disp(poles);


    % Damping characteristics (for complex poles)
    if ~isreal(poles)
        [wn, zeta] = damp(sys);
        fprintf('Damping ratio (?): %.3f\n', zeta(1));
        fprintf('Natural frequency (?n): %.3f rad/s\n', wn(1));
    end

end

function [ess, t_out, y_out] = draw_ramp(sys, t_end, zoom_time)
% DRAW_RAMP Plots ramp response in three subplots
%   [ess, t_out, y_out] = draw_ramp(sys, t_end, zoom_time)
%
%   Inputs:
%       sys - Closed-loop transfer function (tf object)
%       t_end - End time for simulation (default: 100 sec)
%       zoom_time - Time to zoom in (default: 700 sec)
%
%   Outputs:
%       ess - Steady-state error
%       t_out - Time vector
%       y_out - System response vector
%
%   Generates figure with three subplots:
%       1. Ideal ramp input
%       2. System response
%       3. Zoomed comparison at specified time

    % Set defaults if not provided
    if nargin < 2
        t_end = 100;
    end
    if nargin < 3
        zoom_time = 700;
    end

    % Create time vector
    t = 0:0.1:t_end;

    %getting the ramp
    ramp = tf(1,[1 0]);

    % Get response data
    [y_sys, t_sys] = step(sys.*ramp, t);
    [y_ideal, t_ideal] = step(ramp, t);

    % Create figure with three subplots
    figure;

    % Subplot 1: Ideal ramp input
    subplot(2,1,1);
    plot(t_ideal, y_ideal, 'b');
    hold on;
    plot(t_sys, y_sys, 'r--');
    title('Ramp Response');
    xlabel('Time (sec)');
    ylabel('Amplitude');
    legend('Ideal', 'System', 'Location', 'northwest');
    grid on;
    hold off;

    % Subplot 2: Zoomed comparison
    subplot(2,1,2);
    plot(t_ideal, y_ideal, 'b');
    hold on;
    plot(t_sys, y_sys, 'r--');
    xlim([zoom_time-50 zoom_time+50]);
    title(['Zoomed Comparison at t = ', num2str(zoom_time), ' sec']);
    xlabel('Time (sec)');
    ylabel('Amplitude');
    legend('Ideal', 'System', 'Location', 'northwest');
    grid on;
    hold off;

    % Calculate steady-state error (use last 10% of simulation)
    final_idx = round(0.9*length(t_sys)):length(t_sys);
    ess = mean(y_ideal(final_idx) - y_sys(final_idx));

    % Display results
    disp(['Steady-state error (ess): ', num2str(ess)]);

    % Return output data if requested
    if nargout > 1
        t_out = t_sys;
        y_out = y_sys;
    end
end

function [Gm, Pm, Wgc, Wpc] = draw_Bode_Plot(sys)
% BODE_PLOT Analyzes system stability margins and compares margin()
%   Bode_Plot(sys)
%
%   Input:
%       sys - Transfer function (tf object or state-space model)
%   Outputs:
%       Gm - Gain margin (dB)
%       Pm - Phase margin (degrees)
%       Wgc - Gain crossover frequency (rad/sec)
%       Wpc - Phase crossover frequency (rad/sec)

    % Create margin plot
    figure;
    margin(sys);
    grid on;

    % Get stability margins
    [Gm, Pm, Wgc, Wpc] = margin(sys);

    % Display results
    disp(['=== Stability Margins for ' inputname(1) ' ===']);
    disp(['Gain Margin: ', num2str(Gm), ' dB at ', num2str(Wgc), ' rad/s']);
    disp(['Phase Margin: ', num2str(Pm), '° at ', num2str(Wpc), ' rad/s']);

end
```

State Space Code:

```
clc
clear all
close all

% Given system matrices
A = [0 1; -6 -5];
B = [0; 1];
C = [1 0];
D = [0];
n = 2;                % System order
sys = ss(A,B,C,D);    % State Space model
x0 = [0; 1];          % Initial condition

% Q2: Transfer function conversion
[num, den] = ss2tf(A,B,C,D);
syms s
TF_Manual = C*inv(s*eye(n)-A)*B + D
TF_builtin = tf(num,den)

% Q3: State transition matrix calculation
% Compute ?(s) = [sI - A]^-1
Phi_s = inv(s*eye(n) - A);

% Compute ?(t) by inverse Laplace transform
syms t
Phi_t = ilaplace(Phi_s);

% Verify ?(0) = I
Phi_0 = subs(Phi_t, t, 0);

% Display results
disp('State transition matrix in s-domain (?(s)):');
pretty(Phi_s)

disp('State transition matrix in time domain (?(t)):');
pretty(Phi_t)

disp('Verification of ?(0) = I:');
disp(Phi_0);


% Q4: Verify that ??(t) = A?(t)
Phi_dot = diff(Phi_t, t);  % Take time derivative of ?(t)
A_Phi = A*Phi_t;           % Multiply A with ?(t)

disp('Time derivative of state transition matrix (??(t)):');
pretty(Phi_dot)

disp('A*?(t):');
pretty(A_Phi)

disp('Verification successful: ??(t) = A?(t)');

% Q5 Check Controllability and Observability
% Check Controllability
Co = ctrb(A, B);  % Controllability matrix
rank_Co = rank(Co);
```

```
disp('Controllability Matrix:');
disp(Co);
disp(['Rank of Controllability Matrix: ', num2str(rank_Co)]);

if rank_Co == n
    disp('System is Controllable (as expected)');
else
    disp('System is Not Controllable (unexpected for this system)');
end

% Check Observability
Ob = obsv(A, C);  % Observability matrix
rank_Ob = rank(Ob);
disp('Observability Matrix:');
disp(Ob);
disp(['Rank of Observability Matrix: ', num2str(rank_Ob)]);

if rank_Ob == n
    disp('System is Observable (as expected)');
else
    disp('System is Not Observable (unexpected for this system)');
end

% Q6: Unforced (Homogeneous) Response
disp('=== Unforced Response Analysis ===');

% Compute state solution x(t) = ?(t)*x0
x_t = Phi_t * x0;

disp('Unforced state solution x(t):');
pretty(x_t)

% Compute output solution y(t) = C*x(t) + D*u(t)
% Since u(t)=0 for unforced response:
y_t = C*x_t + D*0;

disp('Unforced output response y(t):');
pretty(y_t)

% Plot the results
t_vals = linspace(0, 5, 500);  % Time vector from 0 to 5 seconds

% Convert symbolic expressions to numeric functions
x1_func = matlabFunction(x_t(1));
x2_func = matlabFunction(x_t(2));
y_func = matlabFunction(y_t);

% Evaluate solutions
x1_vals = arrayfun(x1_func, t_vals);
x2_vals = arrayfun(x2_func, t_vals);
y_vals = arrayfun(y_func, t_vals);

% Plot state responses
figure;
subplot(2,1,1);
plot(t_vals, x1_vals, 'b', 'LineWidth', 2);
hold on;
plot(t_vals, x2_vals, 'r--', 'LineWidth', 2);
title('Unforced State Response');
xlabel('Time (s)');
ylabel('State Values');
```

```
legend('x_1(t)', 'x_2(t)');
grid on;

% Plot output response
subplot(2,1,2);
plot(t_vals, y_vals, 'm', 'LineWidth', 2);
title('Unforced Output Response y(t)');
xlabel('Time (s)');
ylabel('Output y(t)');
grid on;

% Compare with MATLAB's built-in initial() function
[~,t_num,x_num] = initial(sys,x0,t_vals(end));
y_num = x_num*C';  % Equivalent to C*x since D=0

% Display symbolic solutions
disp(' ');
disp('Analytic Solutions:');
disp('x1(t) = '); pretty(x_t(1))
disp('x2(t) = '); pretty(x_t(2))
disp('y(t) = '); pretty(y_t)

% Q7: Forced Response Analysis (Unit Step Input)
disp('=== Forced Response Analysis ===');

% Using Frequency Domain Approach
U_s = 1/s;  % Laplace transform of unit step
U_t =ilaplace(U_s);

% Compute forced component in frequency domain
X_forced_s = Phi_s * B * U_s;

% Convert to time domain
x_forced_t = ilaplace(X_forced_s);

% Total solution (homogeneous + forced)
x_total_t = x_t + x_forced_t;

% Output solution
y_total_t = C*x_total_t + D*U_t;  % D*u(t) where u(t)=1 for t>0

disp('Forced state solution (from step input):');
pretty(x_forced_t)

disp('Total state solution (unforced + forced):');
pretty(x_total_t)

% Direct evaluation using subs()
x1_vals = double(subs(x_total_t(1), t, t_vals));
x2_vals = double(subs(x_total_t(2), t, t_vals));
y_vals = double(subs(y_total_t, t, t_vals));

% Plot results
figure;

% State responses
subplot(2,1,1);
plot(t_vals, x1_vals, 'b', 'LineWidth', 2);
hold on;
plot(t_vals, x2_vals, 'r--', 'LineWidth', 2);
title('Total State Response (Step Input)');
```

```
xlabel('Time (s)');
ylabel('State Values');
legend('Analytic x_1(t)', 'Analytic x_2(t)');
grid on;

% Output response
subplot(2,1,2);
plot(t_vals, y_vals, 'm', 'LineWidth', 2);
hold on;
title('Total Output Response y(t) (Step Input)');
xlabel('Time (s)');
ylabel('Output y(t)');
legend('Analytic y(t)');
grid on;

% Display final steady-state values
ss_x1 = limit(x_total_t(1), t, inf);
ss_x2 = limit(x_total_t(2), t, inf);
ss_y = limit(y_total_t, t, inf);

disp(' ');
disp('Steady-State Values:');
disp(['x1(?) = ' char(ss_x1)]);
disp(['x2(?) = ' char(ss_x2)]);
disp(['y(?) = ' char(ss_y)]);


% Q8: State Feedback Design
disp('=== State Feedback Design ===');

% Original system step response
figure;
step(TF_builtin);
title('Original System Step Response');
grid on;

% Design specifications
zeta_desired = 0.7;      % Desired damping ratio
ts_desired = 1;          % Desired settling time (sec)

% Hand analysis to determine desired poles
wn = 4/(zeta_desired*ts_desired);  % Natural frequency from settling time
sigma = zeta_desired*wn;           % Real part of poles
wd = wn*sqrt(1-zeta_desired^2);    % Imaginary part

% Desired characteristic polynomial
desired_poly = (s + sigma + 1i*wd)*(s + sigma - 1i*wd);
desired_poly = expand(desired_poly);

% Convert to numerical polynomial
desired_coeffs = sym2poly(desired_poly);

% Hand calculation of K matrix
% Characteristic polynomial of A-BK: s^2 + (5+K2)s + (6+K1)
% Compare with desired polynomial: s^2 + 2*zeta*wn*s + wn^2

K1 = desired_coeffs(3) - 6;  % From constant term
K2 = desired_coeffs(2) - 5;  % From s term
K = [K1 K2];

disp('Desired closed-loop poles:');
disp([-sigma+1i*wd, -sigma-1i*wd]);

disp('Feedback gain matrix K:');
disp(K);

% Verification
Ac = A - B*K;
[num_2, denum_2] = ss2tf(Ac,B,C,D);
TF_state_feedback = tf(num_2, denum_2);

% Step response analysis
figure;
step_info = stepinfo(TF_state_feedback);
step(TF_state_feedback);
title('System with State Feedback');
grid on;

disp('Closed-loop system performance:');
disp(['Settling Time: ', num2str(step_info.SettlingTime), ' sec']);
disp(['Overshoot: ', num2str(step_info.Overshoot), '%']);
```