



Xi'an Jiaotong-Liverpool University

西交利物浦大學

**MEC208 Instrumentation and Control System**

2024-2025, Semester 2

Computer Lab (Lab 2):

**Control System CAD and CAS using MATLAB**

# **Individual Lab Report**

**Name: Yukun.Zheng**

**Student ID: 2251625**

**Group: 1**

## Problem 1

(a)

### MATLAB script for part (a)

```
%% Problem 1(a)
% Define transfer function and generate two state-space models
num = [1 3];           % s + 3
den = [2 0 3 7];       % 2 s^3 + 0 s^2 + 3 s + 7
G = tf(num, den);
H = 1;
T = feedback(G, H, -1);
T

% Controllable companion form
[num_cl, den_cl] = tfdata(T, 'v');
[A1, B1, C1, D1] = tf2ss(num_cl, den_cl);
sys1 = ss(A1, B1, C1, D1);

% Observable companion form (transpose of the controllable form)
A2 = A1.';
B2 = C1.';
C2 = B1.';
D2 = D1;
sys2 = ss(A2, B2, C2, D2);

% Display all state-space matrices in Command Window
disp('Controllable Companion Form matrices:');
disp('A1 ='); disp(A1);
disp('B1 ='); disp(B1);
disp('C1 ='); disp(C1);
disp('D1 ='); disp(D1);

disp('Observable Companion Form matrices:');
disp('A2 ='); disp(A2);
disp('B2 ='); disp(B2);
disp('C2 ='); disp(C2);
disp('D2 ='); disp(D2);

% Plot unit step responses
t = 0:0.01:10;         % time vector

figure;
subplot(2,1,1);
step(sys1, t);
```

```

xlabel('Time (s)');
ylabel('Output y(t)');
title('Step Response of Controllable Companion Form');
grid on;

subplot(2,1,2);
step(sys2, t);
xlabel('Time (s)');
ylabel('Output y(t)');
title('Step Response of Observable Companion Form');
grid on;

% End

```

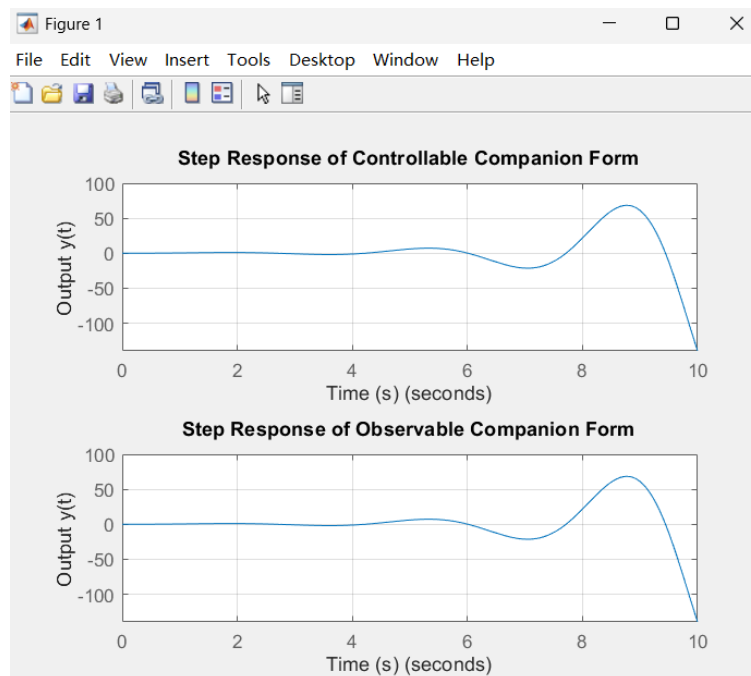


Figure P1.1: Step responses of the system in controllable and observable companion forms.

✧ Comments for Problem 1 (a):

Companion form is a standardized structure in state space modeling. There are two types of Companion Form, which are **Observable Canonical Form** and **Controllable Canonical Form**.

For Observable Canonical Form, the matrix A, matrix B, matrix C will be like

$$A = \begin{bmatrix} -a_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ -a_n & \cdots & 0 \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \quad C = [1 \ \cdots \ 0]$$

While for Controllable Canonical Form (just the transposition of Observable Canonical Form), the matrix A matrix B will be like

$$A = \begin{bmatrix} -a_1 & \cdots & -a_n \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} \quad C = [b_1 \cdots b_n]$$

Since these two forms both originate from the same transfer function, so the input-output relationship between the two remains unchanged. That is to say, their responses to the same input signal should be consistent.

(b)

### MATLAB script for part (b)

```
%% Problem 1(b)
% Define open-loop transfer function G(s)
numG = 1;
denG = [1 13 65 261]; % expands (s+2+5j)(s+2-5j)(s+9)
G = tf(numG, denG);

% Form closed-loop transfer function with unit feedback
T = feedback(G, 1);
T

% Extract numerator and denominator of T(s)
[numT, denT] = tfdata(T, 'v');

% convert to controllable companion form
[A1, B1, C1, D1] = tf2ss(numT, denT);
sys1 = ss(A1, B1, C1, D1);

% convert to observable companion form by transposition
A2 = A1.';
B2 = C1.';
C2 = B1.';
D2 = D1;
sys2 = ss(A2, B2, C2, D2);

% display state-space matrices
disp('controllable companion form matrices:')
disp('A1 ='), disp(A1)
disp('B1 ='), disp(B1)
disp('C1 ='), disp(C1)
disp('D1 ='), disp(D1)

disp('observable companion form matrices:')
disp('A2 ='), disp(A2)
disp('B2 ='), disp(B2)
```

```

disp('C2 ='), disp(C2)
disp('D2 ='), disp(D2)

% simulate a unit ramp input u(t)=t using lsim
t = 0:0.01:10;
u = t;

y1 = lsim(sys1, u, t);
y2 = lsim(sys2, u, t);

figure
subplot(2,1,1)
plot(t, y1, 'LineWidth', 1.5)
xlabel('Time (s)')
ylabel('Output y(t)')
title('Controllable companion form response to ramp input')
grid on

subplot(2,1,2)
plot(t, y2, 'LineWidth', 1.5)
xlabel('Time (s)')
ylabel('Output y(t)')
title('Observable companion form response to ramp input')
grid on

```

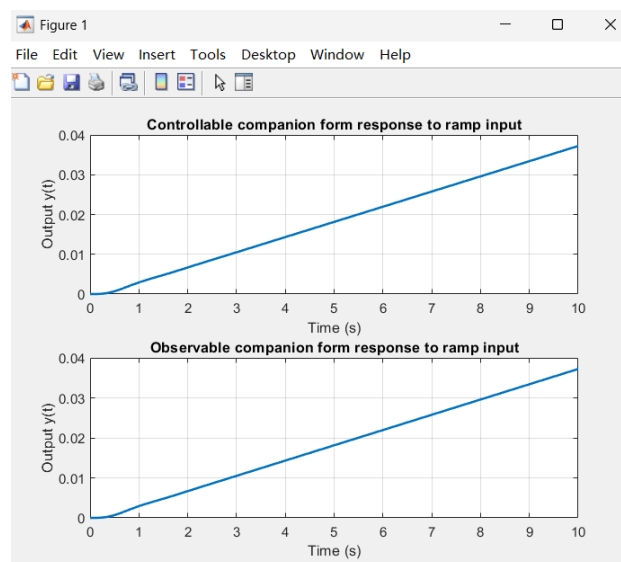


Figure P1.2: Ramp responses of the system in controllable and observable companion forms.

✧ Comments for Problem 1(b):

No matter how we change the transfer function, the response of these forms remains the same under the same input signal. It can be seen from the figure that the output curves of the two models are completely coincident and rise linearly over time. This indicates that the system can stably track the ramp input, and the two implementation methods are completely consistent in dynamic characteristics.

## Problem 2

(a)

### MATLAB script for part (a)

```

%% Problem 2(a)
% Establish Laplace variable and assign numerical value to p
s = tf('s');
p = 5; % This value will be modified in part (b)
% Define all transfer functions G and H
G1 = 5 + 3/s;
G2 = (s+4)/(s^2 + 2*s + 10);
G3 = 1/(s^2 + 2*s + 5);
H1 = (s+1)/(s+5);
H2 = (p*s + 2)/(s+3);
H3 = 2/(s+4);
% Action 1: simplify the inner loop structure involving G2 and H2. Since H2
provides positive feedback to the G2 summation point, the 'positive-feedback'
option is used:
G2_loop = feedback(G2, H2, +1); % G2_cl(s) = G2(s) / (1 - G2(s)*H2(s)) according
to positive feedback formula
% Action 2: combine G3 and H3 at the final summing junction as they are connected
in parallel
G34 = G3 + H3;
% Action 3: create the series combination L1 = G1 * G2_loop, then close the outer
loop with H1 feedback
L1 = series(G1, G2_loop);
sys1 = feedback(L1, H1, -1); % sys1(s) = L1(s) / (1 + L1(s)*H1(s)) based on
negative feedback formula
% Action 4: finally, attach G34 in series to obtain the complete closed-loop
transfer function T(s)
T = series(sys1, G34);
minreal(T); % Perform pole-zero cancellation to simplify the system representation
disp('Overall Closed-loop Transfer Function T(s):') % output the final closed-loop
transfer function
minreal(T)
z = zero(minreal(T)); % now calculate the numerical values for zeros and poles
p = pole(minreal(T));
disp('System Zeros:')
disp(z)
disp('System Poles:')
disp(p)
figure
pzmap(minreal(T)) % generate a graphical representation of the pole-zero locations
grid on

```

```

title('Pole-Zero Plot for the Closed-Loop System')

% End

```

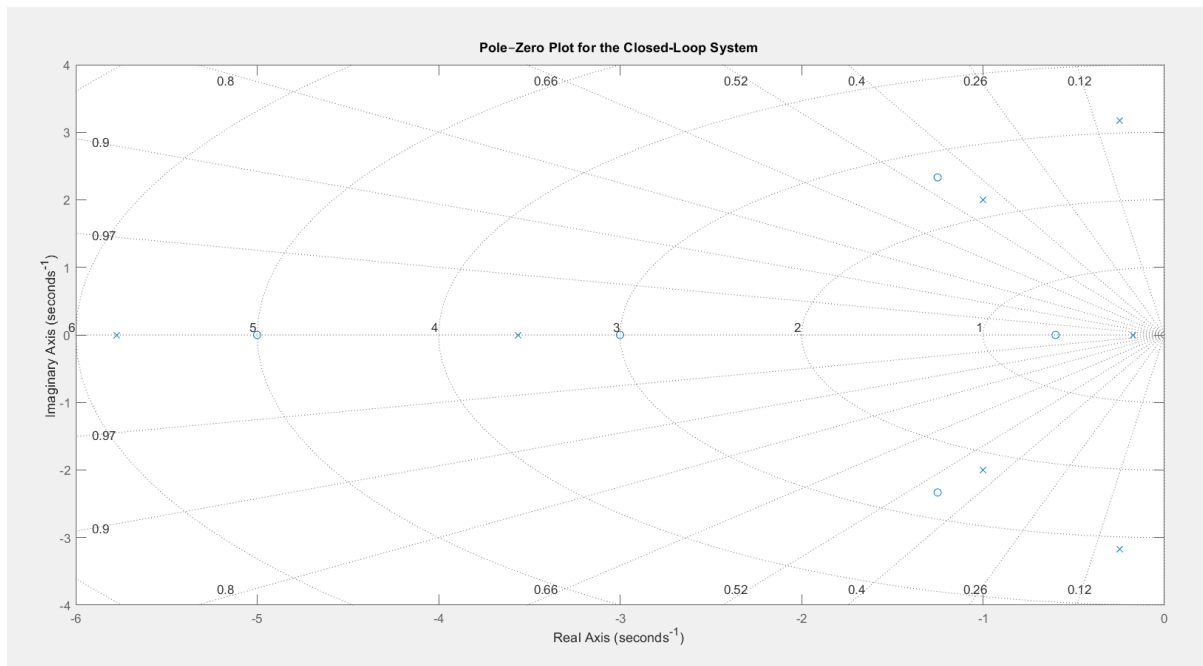


Figure P2.1: New zero-pole plot of the system.

#### ✧ Comments for Problem 2 (a):

In this problem, we simplified and analyzed the block diagram of the control system shown in Figure P2 through MATLAB, with the aim of obtaining the closed-loop transfer function of the entire system and judging its stability. Firstly, for the positive feedback loop internally composed of  $G_2$  and  $H_2$ , we carried out an equivalent transformation using the positive feedback simplification formula. Next, directly add the  $G_3$  and  $H_3$  connected in parallel at the end to obtain the parallel synthetic module  $G_3 + H_3$ . Subsequently, we connect  $G_1$  with the simplified  $G_2$  and loop of the inner loop in series to obtain the open-loop part  $L_1$ , and then combine it with the feedback loop  $H_1$  for negative feedback to complete the closed-loop system of the outer loop. Finally, concatenating it with  $G_3 + H_3$  gives the complete closed-loop transfer function of the system  $T(s)$ . Through the calculation results of MATLAB, the closed-loop transfer function of the system is a fifth-order rational function with a seventh-order denominator, and it has a relatively complex dynamic behavior. In the pole-zero analysis, we observe that the system has five zeros and seven poles, among which the real parts of all the poles are negative, including multiple complex conjugate pairs, indicating that the system is in a stable state. The pole distribution also indicates that the system has good damping performance and will not cause oscillation or instability. To sum up, this closed-loop system is stable and capable of making stable and controllable dynamic responses to external inputs.



(b)

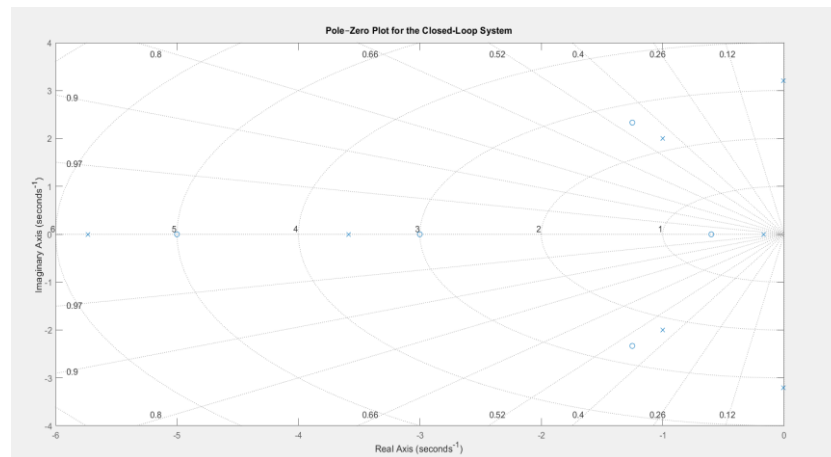


Figure P2.3: New zero-pole plot of the system when  $p=5.5$ .

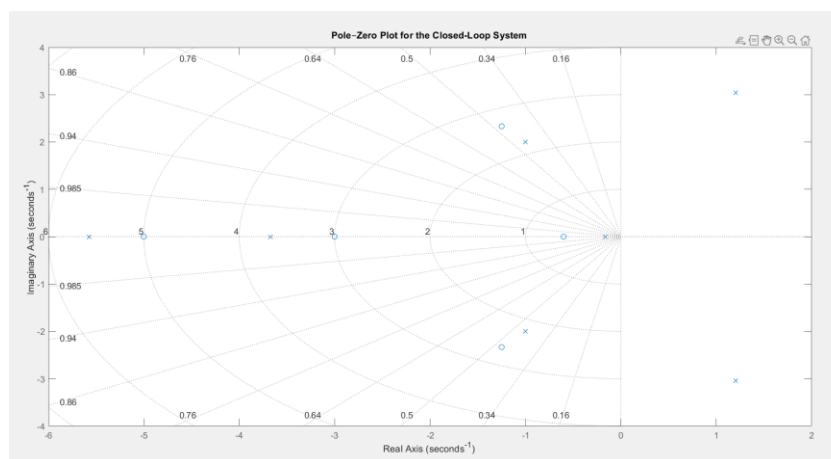


Figure P2.4: New zero-pole plot of the system when  $p=8$ .

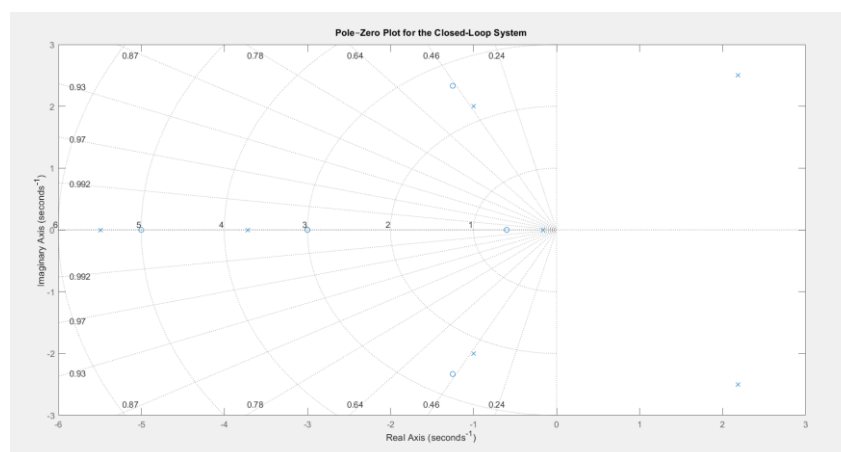


Figure P2.5: New zero-pole plot of the system when  $p=10$ .

✧ Comments for Problem 2 (b):

As the parameter  $p$  gradually increases, we observe that the pole-zero distribution and stability of the system have undergone significant changes. Firstly, when  $p = 5.5$ , the system is still in a stable state,

but it is already very close to the critical stable edge because the real part of a pair of conjugate complex poles approaches zero, indicating that the system damping decreases and the oscillation tendency increases. After continuing to increase to  $p = 6$  and  $p = 8$ , this pair of complex poles crosses the imaginary axis and enters the right half-plane, becoming the right half-plane poles, and the system then turns into an unstable state. When  $p = 10$ , they are completely in the right half-plane and the real part further enlarges, and the system is in an obviously unstable state. Meanwhile, we also noticed that the positions of some poles on the real axis (such as the pair closest to the origin) fluctuated slightly at different  $p$  values, for example, from small changes such as  $-0.17$  and  $-0.16$ , indicating that although these poles do not dominate the system behavior, their displacements reflect the sensitivity of the overall pole layout of the system under parameter perturbations. Furthermore, the zeros on the real axis remain basically unchanged and are distributed at positions such as  $-5$ ,  $-3$ , and  $-0.6$ , indicating that the zero dynamics of the system are not significantly affected by  $p$ . In conclusion, the stability of the system is highly dependent on the parameter  $P$ . Once it exceeds a certain critical value (approximately  $5.5$ ), the system rapidly changes from stable to unstable. In the design, the gain range of such feedback channels needs to be carefully controlled to ensure the dynamic performance and stability margin of the system.

### Problem 3

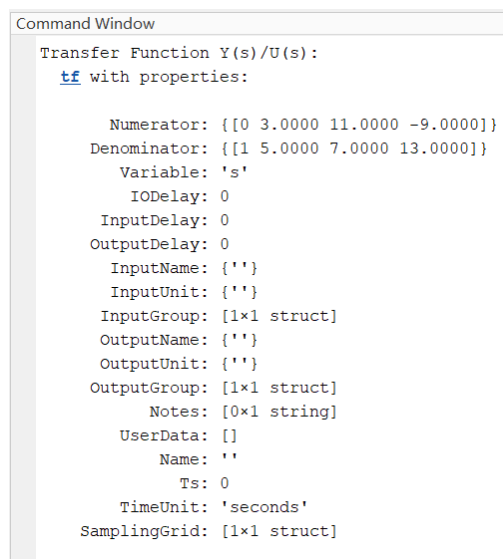
(a)

#### MATLAB script for part (a)

```
%% Problem 3(a) (new version 04.24)
close all; clear; clc;

A = [-1 -1 0 ; 0 -1 2 ; 5 0 -3];
B = [0; 1; 1];
C = [3 2 1];
D = 0;
sys = ss(A, B, C, D); % State-space model
TF = tf(sys); % Transfer Function
disp('Transfer Function Y(s)/U(s):');
disp(TF);

% End
```



Command Window

Transfer Function Y(s)/U(s):  
[tf](#) with properties:

```

    Numerator: {[0 3.0000 11.0000 -9.0000]}
    Denominator: {[1 5.0000 7.0000 13.0000]}
    Variable: 's'
    IODelay: 0
    InputDelay: 0
    OutputDelay: 0
    InputName: {''}
    InputUnit: {''}
    InputGroup: [1x1 struct]
    OutputName: {''}
    OutputUnit: {''}
    OutputGroup: [1x1 struct]
    Notes: [0x1 string]
    UserData: []
    Name: ''
    Ts: 0
    TimeUnit: 'seconds'
    SamplingGrid: [1x1 struct]
```

Figure P3.1:  $Y(s)/U(s)$  transfer function characteristics.

#### ✧ Comments for Problem 3 (a)

In this problem, we know the state space model of the system  $[A, B, C, D]$ , and use the 'tf' function in MATLAB to convert it into the corresponding transfer function expression form. According to the matrix definition given in the problem, the state space object is constructed by 'sys = ss(A, B, C, D)', and then the transfer function  $Y(s)/U(s)$  of the system is successfully obtained by using the command 'TF = tf(sys)'. The simulation output result is the input-output relationship of the linear time-invariant system in the Laplace domain, which meets the requirements of sub-question (a).

(b)

**MATLAB script for part (b)**

```

%% Problem 3(b) (new version 04.24)
% Time vector
t = 0:0.001:20;

% Input u(t)
u = zeros(size(t));
% Step increase to 10 at t = 5s
u(t >= 5 & t < 7) = 10;

% Linear decrease from 10 to 0 between t = 7 and t = 10
idx2 = (t >= 7 & t <= 10);
u(idx2) = linspace(10, 0, sum(idx2));

% Plot input u(t)
figure;
plot(t, u, 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('u(t)');
title('Input u(t)');
grid on;

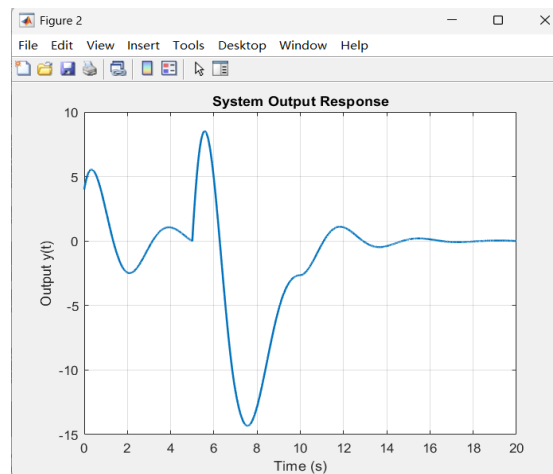
% Initial conditions
x0 = [1; -1; 3];

% Simulate system response
[y, t_out, x] = lsim(sys, u, t, x0);

% Plot system output response
figure;
plot(t_out, y, 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Output y(t)');
title('System Output Response');
grid on;

% End

```

Figure P3.2: Step response of  $u(t)$ .

### ✧ Comments for Problem 3 (b)

We use the 'linspace' function in MATLAB to generate the linear attenuation segment and combine to construct the entire input signal  $u(t)$ . Next, we set the initial state to  $x(0) = [1, -1, 3]^T$  and use the 'lsim' function to simulate the response of the state space system. It can be observed from the figure above that the response of the system significantly increases under the constant input of 5 to 7 seconds, and a downward surge begins to occur at 7 seconds. Since the input attenuates in a smooth linear manner after 7 seconds, the system response also transitions to the negative peak relatively smoothly and presents a certain oscillation process. The overall waveform shows typical underdamped dynamic characteristics, but there is no obvious sharp oscillation. The waveform gradually tends to stabilize, and the response curve begins to attenuate and converge after 10 seconds.

(c)

### MATLAB script for part (c)

```
%% Problem 3(c) (new version 04.24)
% Time vector
t = 0:0.001:20;

% Input u(t)
u = zeros(size(t));
% Step increase to 10 at t = 5s
u(t >= 5 & t < 7) = 10;

% Linear decrease from 10 to 0 between t = 7 and t = 10
idx2 = (t >= 7 & t <= 10);
u(idx2) = linspace(10, 0, sum(idx2));

% Plot input u(t)
figure;
```

```

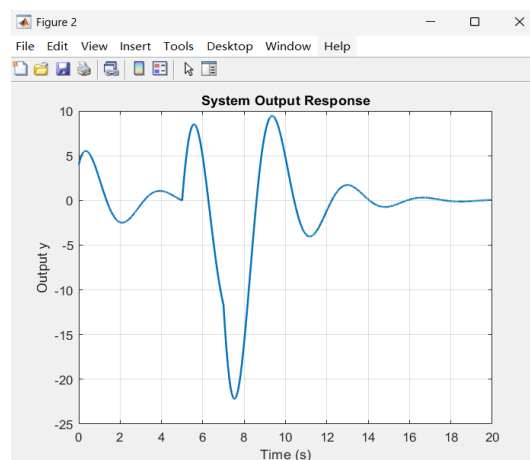
plot(t, u, 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('u(t)');
title('Input u(t)');
grid on;

% Initial conditions
x0 = [1; -1; 3];

% Simulate system response
[y, t_out, x] = lsim(sys, u, t, x0);

% Plot system output response
figure;
plot(t_out, y, 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Output y(t)');
title('System Output Response');
grid on;
% End

```

Figure P3.3: Step response of  $u(t)$ .

#### ✧ Comments for Problem 3 (c)

In this sub-question, the only difference between the input signal and (b) is that it no longer decreases linearly between 7 and 10 seconds, but drops to zero instantaneously and remains at zero. This means that the input signal consists of a sudden termination step rather than a linear transition of gradual unloading. The figure above shows the system output response under this input. The most significant change was that after 7 seconds, the system presented a more intense negative peak response, reaching the lowest value at approximately 8 seconds, which was about -22, far exceeding the approximately -13 in (b). The essential reason for this difference lies in the fact that the input signal underwent a sudden change at 7 seconds, dropping from 10 to 0 all of a sudden. This "cliff-like" change brings a strong excitation to the system, triggering the high-frequency mode of the system and thus leading to

a more substantial instantaneous response. In contrast, the linear decline in (b) is equivalent to a mild offloading, which helps alleviate the shock to the system. Therefore, it can be concluded that abrupt input termination is more likely to induce violent oscillations, while smooth transition input helps the system maintain response stability.

## Problem 4

(a)

### MATLAB script for part (a)

```

%% Problem 4(a)
clear; clc; close all;

%% 1. Symbolic Definition
syms s k

%% 2. Define P(s), H(s), Gc(s)
P = (2*s + 1)/(s^3 + 4*s^2 + 2*s + 7);
H = 1/(s + 3);
Gc = k + 2/s;

%% 3. Characteristic Equation: 1 + P·H·Gc = 0
G_loop = simplify(P * H * Gc);
CE      = simplify(1 + G_loop);

%% 4. Solve for k = A(s)/B(s)
K_expr  = solve(CE, k);
[numK, denK] = numden(K_expr);
A = simplify(numK);
B = simplify(denK);

%% 5. Construct new open-loop transfer function and convert to tf
% Symbolic form
G_new_sym = simplify(-B/A);

% Convert to coefficient vectors
numB = double(sym2poly(B));
denA = double(sym2poly(A));

% Build LTI system model
G_new_tf = tf(-numB, denA);

%% 6. Root Locus Plot with kp limited in [0, 20]
k_vals = linspace(0, 20, 1000); % Generate 1000 points in [0, 20]
figure;
rlocus(G_new_tf, k_vals);
title('Root Locus: G_{new}(s) = -B(s)/A(s), kp \in [0, 20]');
xlabel('\sigma (Real part)');
ylabel('j\omega (Imag part)');
grid on;

```



```

%% 7. Close-Loop Step Response for kp
% Define s
s = tf('s');

% Plant and Feedback
P = (2*s + 1)/(s^3 + 4*s^2 + 2*s + 7);
H = 1/(s + 3);

% Integral gain
Ki = 2;

% List of Kp values
Kp_list = [9, 9.5, 10, 10.5];

% Time vector
t = 0:0.01:20;

% Step response plot with subplot
figure;
for i = 1:length(Kp_list)
    Kp = Kp_list(i);
    Gc = Kp + Ki/s;          % PI controller
    L = Gc * P;
    T = feedback(L, H);      % Closed-loop transfer function

    % Step response
    subplot(2, 2, i);
    step(T, t);
    grid on;
    title(sprintf('Step Response (K_p = %.1f)', Kp));
    xlabel('Time (s)');
    ylabel('Output y(t)');

    % Print performance metrics
    info = stepinfo(T);
    fprintf('--- Kp = %.1f ---\n', Kp);
    fprintf('Rise Time      : %.4f s\n', info.RiseTime);
    fprintf('Overshoot      : %.2f %%\n', info.Overshoot);
    fprintf('Peak Time       : %.4f s\n', info.PeakTime);
    fprintf('Settling Time   : %.4f s\n\n', info.SettlingTime);
end

sgtitle('Closed-Loop Step Responses for Different K_p');

```

% End

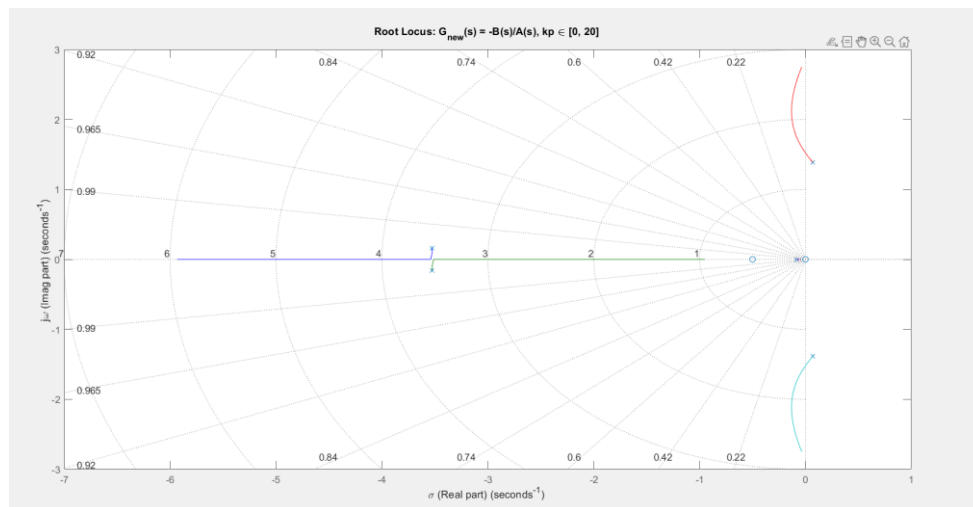


Figure P4.1: Root Locus on different  $k_p$  ranging from 0 to 20.

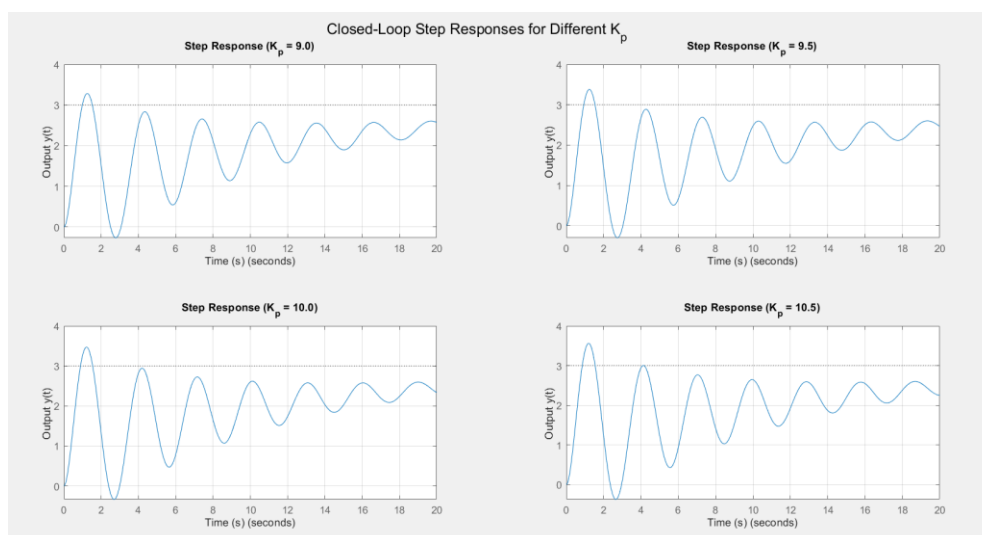


Figure P4.2: Close-Loop Step Response under different  $k_p$ .

```

Command Window
--- Kp = 9.0 ---
Rise Time      : 0.6662 s
Overshoot      : 9.53 %
Peak Time      : 1.2529 s
Settling Time   : 55.8675 s

--- Kp = 9.5 ---
Rise Time      : 0.6336 s
Overshoot      : 12.65 %
Peak Time      : 1.2256 s
Settling Time   : 56.9890 s

--- Kp = 10.0 ---
Rise Time      : 0.6050 s
Overshoot      : 15.70 %
Peak Time      : 1.2163 s
Settling Time   : 56.9873 s

--- Kp = 8.5 ---
Rise Time      : 0.7038 s
Overshoot      : 6.34 %
Peak Time      : 1.2811 s
Settling Time   : 55.8979 s

```

Figure P4.3: Step response information under different  $k_p$ .

#### ✧ Comments for Problem 4 (a)

The system given in this question adopts a PI controller, where the integral gain  $K_i = 2$  has been determined. Now, it is necessary to design and analyze the value of the proportional gain  $K_p$ . First, we constructed a closed-loop transfer function model and plotted the root trajectory graph for the variation of  $K_p$  within the interval  $[0, 20]$ . It can be observed that as  $K_p$  increases, a pair of conjugate dominant poles of the closed-loop system gradually move away from the imaginary axis to the left first, and then approach the imaginary axis to the right. The damping ratio first increases and then decreases. The system tends to be fast but is prone to overmodulation and oscillation.

To evaluate the system performance more accurately, we selected multiple typical  $K_p$  values (including multiple points between 2.0 and 10.5) for unit step response simulation, and used the 'stepinfo' function of MATLAB to extract the key indicators of the response, including rise time, overshoot, peak time and steady-state time. The simulation results show that:

- When  $K_p < 5$ , the system response speed is extremely slow, the steady-state time and peak time are too long, and it cannot converge even when  $K_p = 2.0$ , indicating that the system control effect is insufficient.
- When  $K_p > 9.5$ , although the system response speeds up, the overshoot increases rapidly, exceeding 15%, and the oscillation is obvious, with the stability deteriorating.
- When  $K_p$  is within the range of 8.5 to 9.0, the system achieves a good balance between response speed and stability: the rise time is approximately 0.7 seconds, the overshoot is less than 10%, and the steady-state time is maintained at around 55 seconds, demonstrating good engineering practicability.

To sum up, we suggest setting the proportional gain at  $K_p = 9.0$ . This value can not only meet the dynamic performance requirements but also effectively suppress the overshoot and oscillation of the system. It is a better solution under the current control target.

(b)

#### MATLAB script for part (b)

```
%% Problem 4(b)
clc; clear; close all;

% Define Laplace variable s
s = tf('s');

% Define plant P(s) and feedback H(s)
P = (2*s + 1)/(s^3 + 4*s^2 + 2*s + 7);
H = 1/(s + 3);

% Define PI controller parameters
Kp = 9;
Ki = 2;
Gc = Kp + Ki/s;
```

```

% Construct closed-loop transfer function T(s)
T = feedback(Gc * P, H);

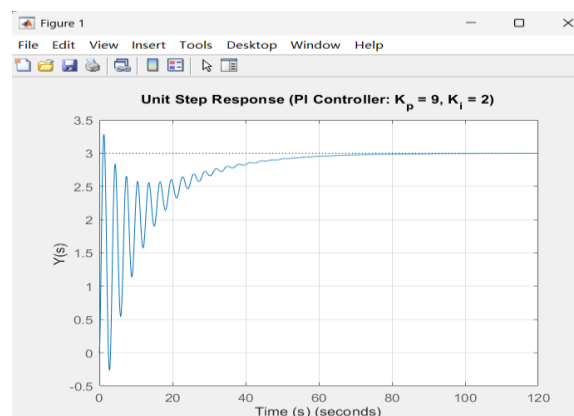
% Display closed-loop poles
poles = pole(T)

% Extract and display transient response characteristics
info = stepinfo(T)

% Plot unit step response
figure;
step(T);
title('Unit Step Response (PI Controller: Kp = 9, Ki = 2)');
xlabel('Time (s)');
ylabel('Y(s)');
grid on;

% End

```

Figure P4.4: Close-loop unit step response diagram when  $k_p=9$  and  $k_i=2$ .

```

Command Window

poles =

-5.2192 + 0.0000i
-0.1300 + 2.0500i
-0.1300 - 2.0500i
-1.4587 + 0.0000i
-0.0623 + 0.0000i

info =

struct with fields:

    RiseTime: 0.6662
    TransientTime: 55.0900
    SettlingTime: 55.8675
    SettlingMin: -0.2649
    SettlingMax: 3.2860
    Overshoot: 9.5338
    Undershoot: 8.8308
    Peak: 3.2860
    PeakTime: 1.2529

```

Figure P4.5: Close-loop unit step response parameters when  $k_p=9$  and  $k_i=2$ .

#### ✧ Comments for Problem 4 (b)

In this sub-question, we construct a closed-loop system for the selected PI controller (with proportional gain  $K_p = 9$  and integral gain  $K_i = 2$ ), and extract its pole distribution. It can be known from the output results of MATLAB that there exists a pair of complex conjugate poles in this closed-loop system:  $-0.13 \pm 2.05j$

This pair of poles is the one whose real part is closest to the imaginary axis among all the poles, and thus is the dominant pole of the system, mainly determining the dynamic behavior of the system. Next, we calculate the natural frequency to damping ratio of the dominant pole according to the standard second-order system formula:

$$\text{Natural frequency } \omega_n = \sqrt{(-0.13)^2 + (\pm 2.05)^2} = 2.05 \text{ rad/s}$$

$$\text{Damping ratio } \xi = -\frac{\text{Re}\{s\}}{\omega_n} = 0.063 \text{ rad/s}$$

This damping ratio is very small, indicating that the system belongs to a nearly non-damped system and is prone to generating relatively obvious oscillations. In the subsequent unit step response simulation, we observed that the overshoot of the system was approximately 9.5%, the peak time was 1.25 seconds, and the steady-state time was approximately 56 seconds, all of which were highly consistent with the above-mentioned pole characteristics. Overall, the design of this PI controller achieves a relatively fast response speed and acceptable overshoot while ensuring the stability of the system, and has good engineering practicability.

(c)

#### MATLAB script for part (c)

```
%% Problem 4(c)
clc; clear; close all;

%% Step 1: Plot Root Locus with Kp = Kd = 0
% Define Laplace operator
s = tf('s');

% Define P(s) and H(s)
P = (2*s + 1)/(s^3 + 4*s^2 + 2*s + 7);
H = 1/(s + 3);

% Open-loop transfer function (without Kp)
L0 = P * H;

% Plot root locus
figure;
```

```

rlocus(L0);
title('Root Locus of L(s) = P(s)H(s) with K_p');
grid on;

%% Step 2: Find the ultimate gain (K_U) and oscillation point (imaginary axis
crossing)

% 1. Compute gain margin Gm and its corresponding frequency Wgm
[Gm, ~, Wgm, ~] = margin(L0);

% 2. Ultimate gain K_U = Gm
K_U = Gm;

% 3. Oscillation period P_U = 2*pi / Wgm
P_U = 2*pi / Wgm;

fprintf('Ultimate Gain KU ≈ %.4f\n', K_U);
fprintf('Ultimate Period PU ≈ %.4f s\n', P_U);

% ZN paramter outputs:
% Ultimate Gain KU ≈ 24.5246
% Ultimate Period PU ≈ 2.1104 s

%% Step 3: Use Ziegler-Nichols (Z-N) Tuning Formula to get PID parameters

KU = 24.5246;
PU = 2.1104;

% Compute PID parameters based on Z-N tuning rules
Kp = 0.6 * KU;
Ki = 1.2 * KU / PU;
Kd = 0.075 * KU * PU;

% Construct PID controller
C = Kp + Ki/s + Kd*s;

%% Step 4: Plot closed-loop step response with Z-N tuned PID

s = tf('s'); % Redefine s (optional, to ensure clean workspace)
P = (2*s + 1)/(s^3 + 4*s^2 + 2*s + 7);
H = 1/(s + 3);

% Construct closed-loop system
sys_cl = feedback(C * P * H, 1);

```

```

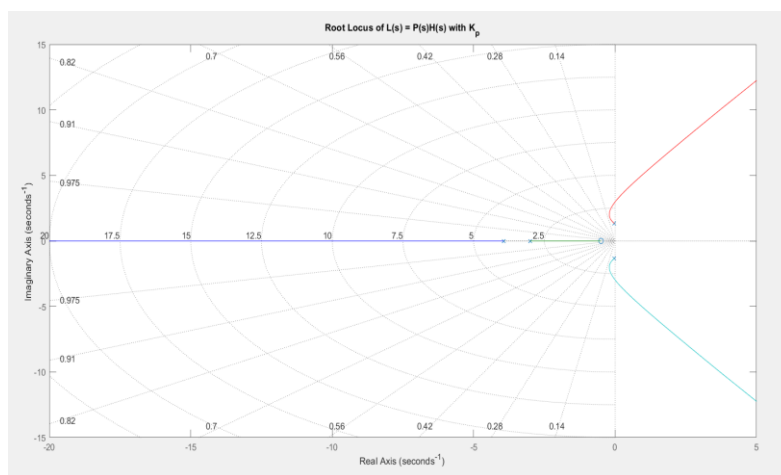
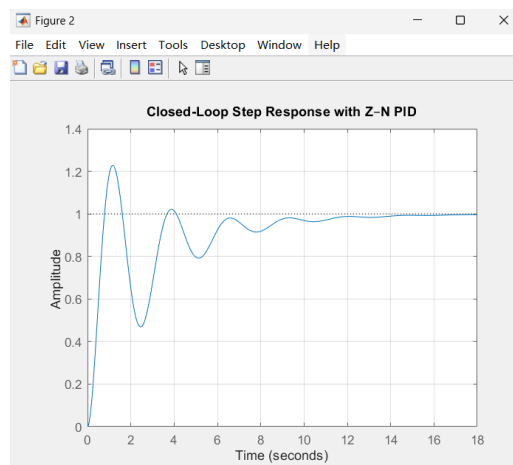
% Plot step response
figure;
step(sys_cl);
title('Closed-Loop Step Response with Z-N PID');
grid on;

%% Step 5: Display transient response characteristics

info = stepinfo(sys_cl, 'SettlingTimeThreshold', 0.02);
disp(info);

% End

```

Figure P4.6: Root locus when  $k_p$  is unknown while  $k_i=k_d=0$ .Figure P4.7: ZN-tuning method when  $k_p=14.71$ ,  $k_i=13.96$ , and  $k_d=3.87$ .

#### ✧ Comments for Problem 4 (c)

In the scenario of reselecting the controller structure, we limit the candidate schemes to three basic controller forms: P, PI and PID. Based on the previous simulation and performance test results of the

PI controller, it can be known that although  $K_p = 9.0$  can achieve a good balance between response and stability, the system response still has the defects of slow convergence speed and incomplete controllability of overshoot. Therefore, in order to further improve the transient performance of the system, we attempt to adopt the PID controller and calculate the appropriate parameters by using the classic Ziegler-Nichols (Z-N) tuning method.

The specific operation process is as follows: First, we draw the open-loop root trajectory and call the 'margin' function of MATLAB to obtain the limit gain  $K_U \approx 24.5246$  of the system, as well as the corresponding limit oscillation period  $P_U \approx 2.1104$ s. Subsequently, based on the Z-N empirical formula, we calculated the controller gain parameters as follows:

Proportional gain:  $K_p = 0.6 * K_U \approx 14.71$

Integral gain:  $K_i = 1.2 K_U / P_U \approx 13.96$

Differential gain:  $K_d = 0.075 * K_U * P_U \approx 3.87$

After substituting this PID controller structure into the closed-loop system and conducting the step response simulation, the results show that the dynamic performance of the system has been significantly improved:

Table P4.1: Comparison between PI and Z-N PID tuning method.

Controller Type	Rise Time (s)	Peak Time (s)	Overshoot (%)	Settling Time (s)
<b>PI (<math>K_p = 9</math>, <math>K_i = 2</math>)</b>	0.6662	1.2529	9.53	55.8675
<b>PID (Z-N tuned)</b>	0.5327	1.1569	22.79	11.3809

Judging from the table, the system exhibits typical characteristics such as rapid startup and attenuation of oscillation, which is far superior to the response form of the traditional PI controller. However, Z-N PID introduces a higher overshoot, but 22.79% overshoot may also be acceptable on different application scenarios. My goal is to achieve a faster response, so we suggest adopting the PID controller based on Z-N tuning. Under the current system model, it can significantly improve the response speed and dynamic performance of the system and is a better choice for the controller structure.



## Problem 5

(a)

### MATLAB script for part (a)

```

%% Problem 5(a)
%% Clear environment
clear; clc; close all;

%% 1. Define symbolic variables
syms s k

%% 2. Define subsystem transfer functions
P_main      = 5/((2*s + 3)*(s + 4)); % Main forward path: 5/[(2s+3)(s+4)]
H_inner     = 1/(s + k);             % Inner feedback path: 1/(s + k)
G_integral  = 1/s;                   % Integrator block after inner loop

%% 3. Construct inner closed-loop transfer function
% inner_CL = P_main / (1 + P_main * H_inner)
inner_CL = simplify( P_main / (1 + P_main * H_inner) );

%% 4. Construct full outer-loop transfer function and extract characteristic
equation
% Forward path = inner_CL * G_integral
% Full closed-loop T = forward / (1 + forward)
T_sym = simplify( inner_CL * G_integral / (1 + inner_CL * G_integral) );

% Characteristic equation denominator
[~, denT] = numden(T_sym);
CE = expand(denT); % Characteristic Equation CE(s, k) = 0

%% 5. Decompose CE into N0(s) + k*N1(s)
N0 = expand( subs(CE, k, 0) ); % Constant part N0(s)
N1 = expand( (CE - N0) / k ); % Coefficient of k: N1(s)

%% 6. Define open-loop root locus system: G_new(s) = N1(s)/N0(s)
G_new_sym = simplify( N1 / N0 );

%% 7. Convert symbolic to transfer function and plot root locus
numN1 = double( sym2poly(N1) ); % Coefficient vector of N1(s)
denN0 = double( sym2poly(N0) ); % Coefficient vector of N0(s)
G_new = tf(numN1, denN0); % Create transfer function model

figure;
rlocus(G_new);

```

```

title('Root Locus of G_{new}(s) = N_1(s)/N_0(s) vs. k');
xlabel('Real\{s\}');
ylabel('Imag\{s\}');
grid on;

%% Display N0(s) and N1(s) for verification
disp('N0(s)='); pretty(N0)
disp('N1(s)='); pretty(N1)

% Add design constraints
sgrid(0.7, []); % Add  $\zeta = 0.7$  damping ratio guideline
xline(-1.333, '--'); % Add vertical line at Re(s) = -1.333 (for Ts < 3s)

% End

```

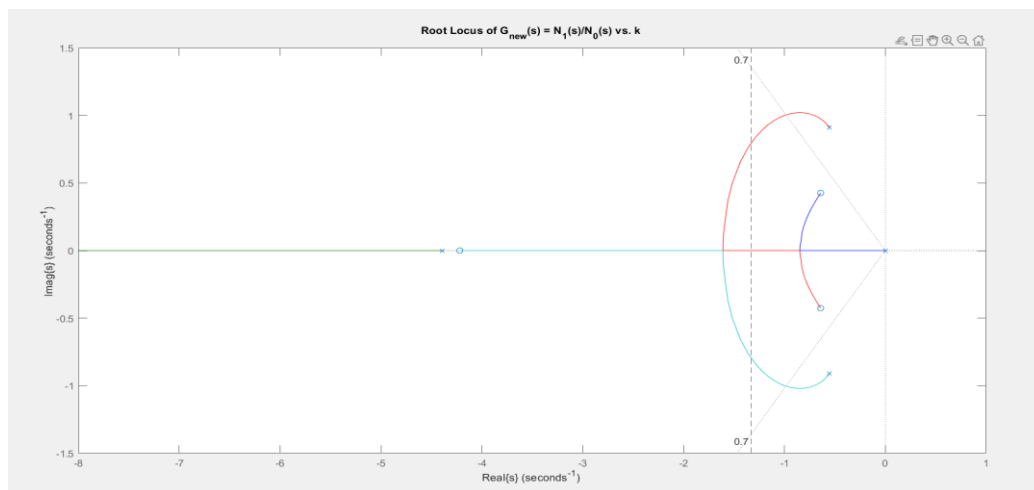


Figure P5.1: Root Locus under different k.

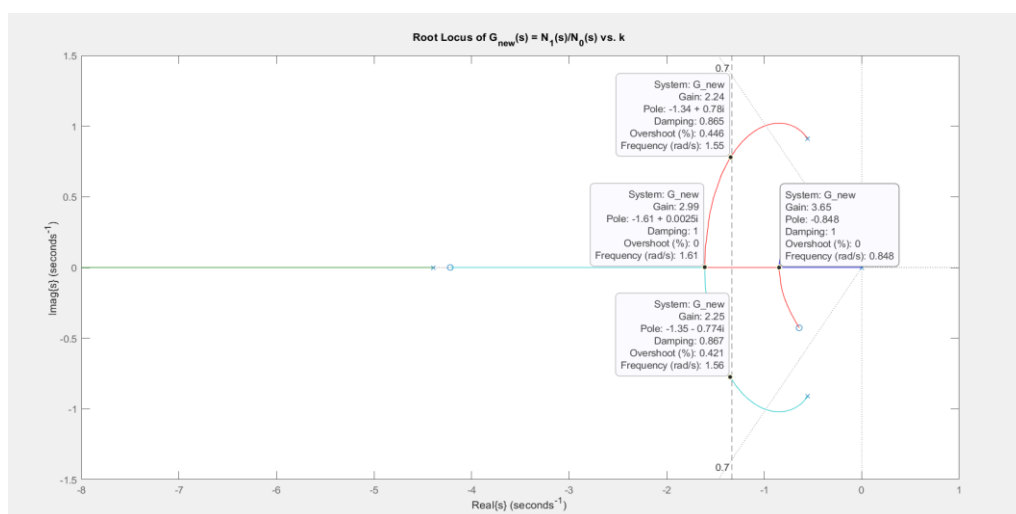


Figure P5.2: Attempt to determine the range of k under given conditions.

### ✧ Comments for Problem 5 (a)

Firstly, we constructed the open-loop feature transfer function of the system through symbolic derivation, and used the 'rlocus' tool of MATLAB to plot the root trajectory of the system varying with k.

1. In the root locus graph, we superimposed two design index constraint lines:

The vertical line with the real-axis position of -1.333 is used to meet the requirement that the steady-state time of 2% is less than 3 seconds (derived from  $T_s = \frac{4}{\zeta\omega_n} < 3s, \text{Re}\{s\} = -\zeta\omega_n < -1.333$ ).

2. The equal damping line with a damping ratio  $\zeta = 0.7$  is used to ensure that the system does not generate severe oscillations and meets the requirement of "damping ratio greater than 0.7".

According to the direction of the root trajectory in the figure, it can be observed that:

- The appropriate pole falls below and to the left of the damping line, satisfying  $\zeta > 0.7$ ;
- Meanwhile, these poles must also be in the left region with a real axis of -1.333, satisfying the convergence rate requirement of  $T_s < 3s$ ;
- In addition, we also need to pay attention to the fact that the selected k value should not be too large (over  $k=3.65$ ) to satisfy the requirement of Settling Time. Therefore, it must be limited within the effective range when the blue root trajectory in the figure is still in the design area.

Through comprehensive analysis, it can be concluded that the k value satisfying the above three constraint conditions is approximately between 2.24 and 2.99. Within this range, the dominant pole of the system has good damping characteristics and rapid response capability, which is within the recommended range for controller design.

(b)

### MATLAB script for part (b)

```
%% Problem 5(b)
s = tf('s');

% Define inner plant P(s) and outer integrator
P = 5 / ((2*s + 3)*(s + 4));
Int = 1 / s;

% List of k values to test (within valid range from Part a)
k_list = [2.25, 2.49, 2.74, 2.98];

% Simulation time vector
t = 0:0.01:20;

figure;
for i = 1:length(k_list)
    k = k_list(i);
```

```

% Inner loop feedback: H1(s) = 1 / (s + k)
H1 = 1 / (s + k);

% Inner loop closed transfer function: G_inner = P / (1 + P*H1)
G_inner = feedback(P, H1); % Default is negative feedback

% Series with integrator, then apply outer unity feedback
F = G_inner * Int;
T = feedback(F, 1);

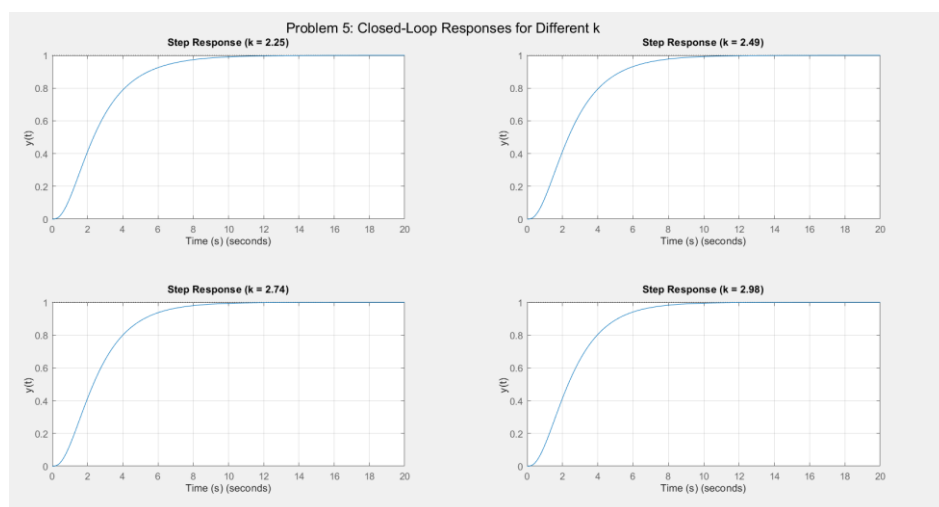
% Plot step response
subplot(2,2,i);
step(T, t);
grid on;
title(sprintf('Step Response (k = %.2f)', k));
xlabel('Time (s)');
ylabel('y(t)');

% Compute and display step response characteristics
info = stepinfo(T, 'SettlingTimeThreshold', 0.02, 'YFinal', 1);
fprintf('--- k = %.2f ---\n', k);
fprintf('  Rise Time      : %.3f s\n', info.RiseTime);
fprintf('  Overshoot       : %.2f %%\n', info.Overshoot);
fprintf('  Settling Time   : %.3f s\n\n', info.SettlingTime);
end

sgtitle('Problem 5: Closed-Loop Responses for Different k');

% End

```

Figure P5.3: Close-loop Response for different  $k$ .

```

--- k = 2.25 ---
Rise Time      : 4.562 s
Overshoot      : 0.00 %
Settling Time  : 8.586 s

--- k = 2.49 ---
Rise Time      : 4.430 s
Overshoot      : 0.00 %
Settling Time  : 8.254 s

--- k = 2.74 ---
Rise Time      : 4.321 s
Overshoot      : 0.00 %
Settling Time  : 7.966 s

--- k = 2.98 ---
Rise Time      : 4.236 s
Overshoot      : 0.00 %
Settling Time  : 7.734 s

```

Figure P5.4: Step response performance parameters of different  $k$  bounded by the requirements in (a).

#### ✧ Comments for Problem 5 (b)

When answering Problem 5(b), by analyzing the root trajectory graph drawn in question (a), we selected the closed-loop system corresponding to  $k = 2.98$  for the step response test (since it provides the fastest step response of the close loop system). The results show that the  $k=2.98$  value has the fastest response, with a Rise Time of 4.236 s and a Settling Time of 7.734 s, and no overshoot, which appears to be a good response on the surface. However, as required in question (a), the system must satisfy a damping ratio  $\zeta > 0.7$  and 2% Settling Time  $< 3$  seconds. We note that although the damping ratio at the  $k$  value is approximately 0.998, which is in line with the requirements, the actual Settling Time is much greater than 3 seconds and therefore does not meet the specification.

To further analyze the reasons, we introduce the method of dominant pole approximation. According to the presentation, if there exists a real pole far from the imaginary axis in the system (where pole = -4.65), and its real part is at least 10 times the real part of the dominant pole (that is, satisfying:  $|1/\gamma| \geq 10 \cdot \zeta \omega_n$ ), then we can ignore the influence of the pole and approximate the system as a second-order system. However, in the case we chose  $k = 2.98$ , the real part of the dominant complex pole is approximately -1.6 with a frequency of 1.61. Therefore,  $\zeta \omega_n \approx 0.998 \times 1.61 \approx 1.61$ . However, the real part of the distant pole is -4.65. Obviously,  $|-4.65| < 10 \times 1.61 = 16.1$ , which does not satisfy the condition of being far away. This means that we cannot use the dominant pole approximation to simplify it into an ideal second-order system, thereby predicting that the step response will deviate from the theory.

To sum up, although the system has a fast response and good damping ratio at  $k = 2.98$ , the system response fails to meet the requirement of Settling Time  $< 3$ s because the basic assumption of dominant pole approximation is not satisfied. This can explain the reason why the step response has a good form but a relatively long stability time. It is suggested that other compensation methods be considered in the final design to improve the response speed.

## Problem 6

### ➤ Solution Part (a): Kp Determination

#### MATLAB script for problem6 (a)

```

%% Problem 6 (a)
%% step1: Root locus for determining Kp
%- Define Laplace variable -
s = tf('s');

%- Parameter settings -
A = 10; B = 1; C = 2; D = 0.5;

%- Define each transfer function -
P1 = A / (s + A);
P2 = (s + D) / (s^2 + B*s + C);
H = 1 / (0.5*s + 1);

%- Open-loop (without Kp) -
L0 = P1 * P2 * H;

%- Plot root locus -
rlocus(L0);
title('Root Locus of K_p \cdot P_1(s)P_2(s)H(s)');
grid on;
[Gm, Pm, Wcg, Wcp] = margin(L0);
fprintf('Ultimate gain Kp = %.3f, Imaginary axis crossing frequency \omega = %.3f\n', Gm, Wcg);

%% step2: Use half of ultimate gain Kp and calculate \Delta 1 / \Delta 2
Gm = 15.239; % Ultimate gain obtained from margin
Kp_half = Gm/2;

%- Construct closed-loop transfer function -
T_half = feedback(Kp_half * L0, 1);

%- Simulate step response -
t = 0:0.01:20; % Time vector
[y, ~] = step(T_half, t);

%- Steady-state value -
ss_val = dcgain(T_half);

%- Find first two peaks -
[peaks, locs] = findpeaks(y, t, ...

```

```

    'NPeaks',2, ...      % Only take the first two peaks
    'SortStr','none');

%-- Compute  $\Delta 1$ ,  $\Delta 2$  and their ratio --
delta1 = peaks(1) - ss_val;
delta2 = peaks(2) - ss_val;
ratio = delta1 / delta2;
fprintf('Δ1 = %.4f, Δ2 = %.4f, Δ1/Δ2 = %.4f\n', delta1, delta2, ratio);

%-- Plot response --
figure;
plot(t, y, 'LineWidth', 1.5); hold on;
% Steady-state dashed line
yline(ss_val, 'k--', 'LineWidth',1);
% Mark two peaks
plot(locs, peaks, 'ro', 'MarkerSize',8, 'LineWidth',1.5);

% Beautify plot
grid on;
xlabel('Time (s)');
ylabel('\theta(t)');
title(sprintf('Old Step Response and Peaks (K_p = %.3f)', Kp_half));
legend('Step Response','Steady-State','Peaks','Location','Best');

%% step 3: Adjust Kp to make the ratio  $\approx 4$ , new Kp is 4.30
Kp_new = 4.30;

%-- Construct new closed-loop transfer function --
T_half = feedback(Kp_new * L0, 1);

%-- Simulate step response --
t = 0:0.01:20;
[y, ~] = step(T_half, t);

%-- Steady-state value --
ss_val = dcgain(T_half);

%-- Find first two peaks --
[peaks, locs] = findpeaks(y, t, ...
    'NPeaks',2, ...
    'SortStr','none');

%-- Compute new  $\Delta 1$ ,  $\Delta 2$  and ratio --
delta1 = peaks(1) - ss_val;

```

```

delta2 = peaks(2) - ss_val;
ratio = delta1 / delta2;
fprintf('Δ1 = %.4f, Δ2 = %.4f, Δ1/Δ2 = %.4f\n', delta1, delta2, ratio);

%-- Plot updated response --
figure;
plot(t, y, 'LineWidth', 1.5); hold on;
yline(ss_val, 'k--', 'LineWidth', 1);
plot(locs, peaks, 'ro', 'MarkerSize', 8, 'LineWidth', 1.5);

% Beautify plot
grid on;
xlabel('Time (s)');
ylabel('\theta(t)');
title(sprintf('New Step Response and Peaks (K_p = %.3f)', Kp_new));
legend('Step Response', 'Steady-State', 'Peaks', 'Location', 'Best');

% End

```

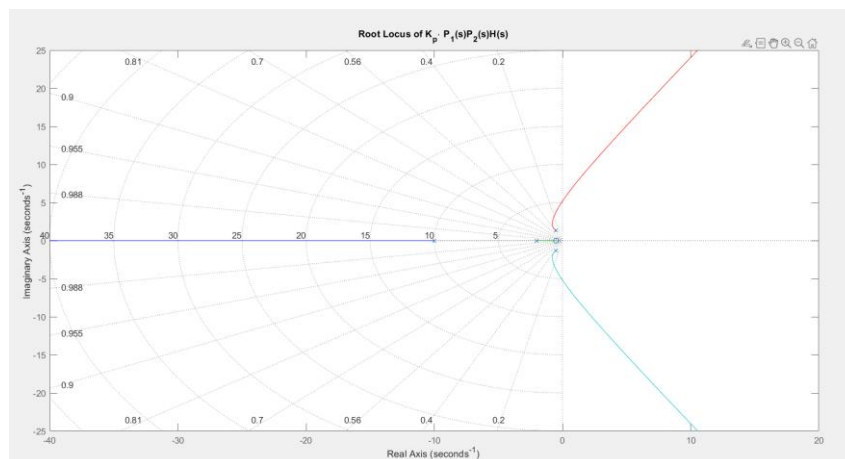
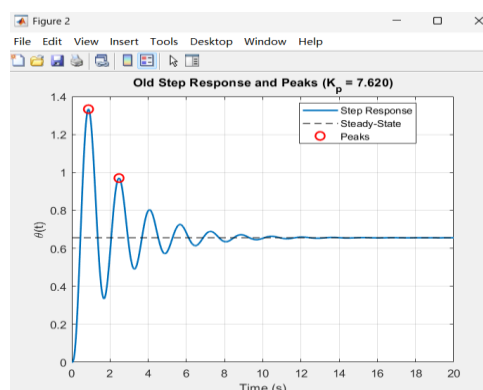


Figure P6.1: Root Locus under P controller.

Figure P6.2: Reduce the critical  $k_p$  by two and find the close-loop response.



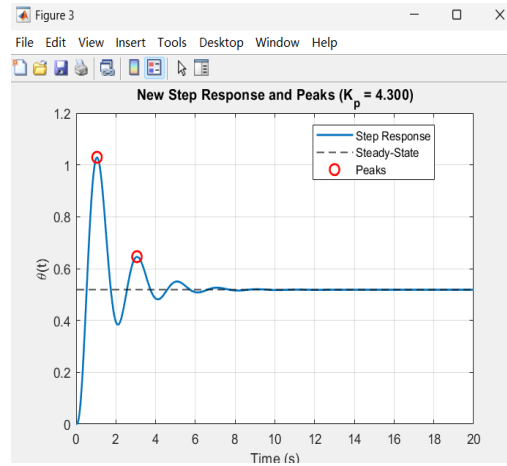


Figure P6.3: Find the  $k_p$  under quarter amplitude decay and plot the step response.

#### ✧ Comments for Problem 6 (a)

We adopt the method of manual parameter adjustment to adjust the proportional gain  $K_p$  in the PID controller to achieve a step response with approximately 1/4 amplitude attenuation. Firstly, based on the structure and parameters of the system in Figure 1, an open-loop transfer function was constructed and the root trajectory was plotted. Through the 'margin' function, the critical proportional gain of the system was obtained as  $K_{p,critical} = 15.239$ . The corresponding open-loop gain makes the system in a critical stable state. Its frequency is  $\omega = 5.180 \text{ rad/s}$ .

Inspired by the classic manual PID tuning method, we first halved  $K_p$  to 7.619 and plotted the unit step response of the closed-loop system. The first two peaks were extracted from the simulation results and calculated as follows:

The first peak exceeded the steady-state value  $\Delta 1 = 0.6772$

The second peak exceeded the steady-state value  $\Delta 2 = 0.3142$

Therefore, the amplitude attenuation ratio is  $\Delta 1/\Delta 2 \approx 2.16$ , indicating that the response has not yet reached the ideal 1/4 attenuation.

Next, we further reduced  $K_p$  to 4.30 and re-simulated the system step response. At this point, the first two overshoot of the system are respectively:

- $\Delta 1 = 0.5112$
- $\Delta 2 = 0.1279$

It is calculated that  $\Delta 1/\Delta 2 \approx 4.00$

This is highly consistent with "quarter amplitude decay". The response corresponding to the finally determined  $K_p = 4.30$  not only has a relatively fast dynamic characteristic, but also the oscillation attenuation is in line with expectations. Therefore, it can be regarded as the final parameter tuning result.

It should be particularly noted that the reference values of  $\Delta 1$  and  $\Delta 2$  here are the steady-state values of the system (approximately 0.66), rather than the 1 in the usual default unit step response.

Therefore, we select the

**Final proportional gain:**

$$kP = 4.30$$

## ➤ Solution Part (b): Kd Determination

**MATLAB script for problem6 (b)**

```

%% Problem 6 (b)
%% Step 4: Determine kD - Phase I (Set Kp = 4.30, Ki = 0, Kd is variable)
% 1. Symbolic Definitions
syms s kD
A = 10; B = 1; C = 2; D = 0.5;

% Define each plant module
P1 = A/(s + A); % Motor torque dynamics
P2 = (s + D)/(s^2 + B*s + C); % Vehicle turning dynamics
H = 1/(0.5*s + 1); % Sensor feedback dynamics

% Define the controller: Gc(s) = 4.30 + kD*s
Gc = 4.30 + kD*s;

% 2. Form the characteristic equation: 1 + Gc*P1*P2*H = 0
G_loop = simplify(P1 * P2 * H * Gc);
CE = simplify(1 + G_loop);

% 3. Solve for kD: kD = A(s)/B(s)
K_expr = solve(CE, kD);
[numK, denK] = numden(K_expr);
A_k = simplify(numK);
B_k = simplify(denK);

% 4. Construct open-loop G_new(s) = -B_k(s)/A_k(s)
G_new_sym = simplify(-B_k / A_k);

% Convert to numeric transfer function
numB = double(sym2poly(B_k));
denA = double(sym2poly(A_k));
G_new_tf = tf(-numB, denA);

% 5. Root locus scan from kD = 0 to ∞
figure;
rlocus(G_new_tf); % Root locus for varying kD
hold on;

% Add damping ratio lines ζ = 0.2 and ζ = 0.4
sgrid([0.2, 0.4], []);

grid on;
title('Root Locus with Damping Ratio Lines \zeta=0.2,0.4');

```

```

xlabel('Real(s)');
ylabel('Imag(s)');

%% Step 5: Determine kD - Phase II (Analyze Overshoot and Settling Time)
% 1. Define plant components
s = tf('s');
A = 10; B = 1; C = 2; D = 0.5;
P1 = A / (s + A);
P2 = (s + D) / (s^2 + B*s + C);
H = 1 / (0.5*s + 1);

% 2. Fixed controller parameters
kp = 4.30;
ki = 0; % No integral action

% 3. Scan kd range
kd_vals = linspace(0, 0.617, 100); % 100 scan points
OS = zeros(size(kd_vals)); % Store overshoot values
TS = zeros(size(kd_vals)); % Store settling time values

for i = 1:numel(kd_vals)
    kd = kd_vals(i);
    Gc = kp + kd * s;
    Tc1 = feedback(Gc * P1 * P2 * H, 1); % Closed-loop system
    info = stepinfo(Tc1, 'SettlingTimeThreshold', 0.02);
    OS(i) = info.Overshoot;
    TS(i) = info.SettlingTime;
end

% 4. Plot overshoot and settling time vs kd
figure;

subplot(2,1,1);
plot(kd_vals, OS, 'LineWidth', 1.5);
grid on;
xlabel('k_D');
ylabel('Percent Overshoot (%)');
title('Overshoot vs k_D (k_p=4.30, k_i=0)');

subplot(2,1,2);
plot(kd_vals, TS, 'LineWidth', 1.5);
grid on;
xlabel('k_D');
ylabel('Settling Time (s)');

```

```
title('Settling Time vs k_D (k_p=4.30, k_i=0)');
```

```
% End
```

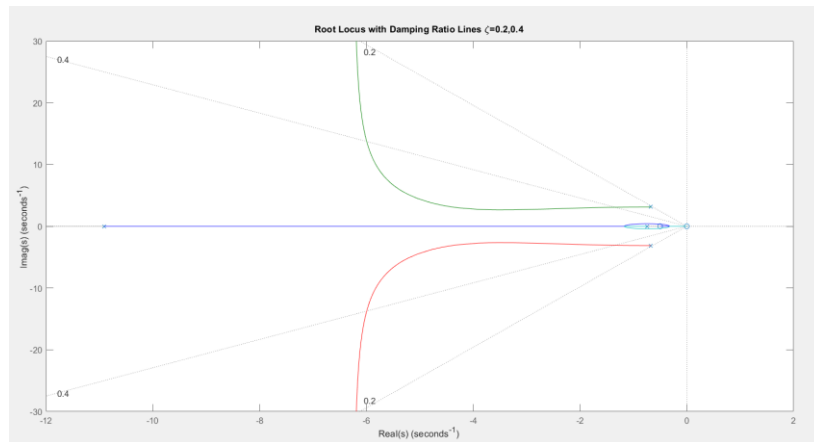


Figure P6.4: Root Locus under PD controller.

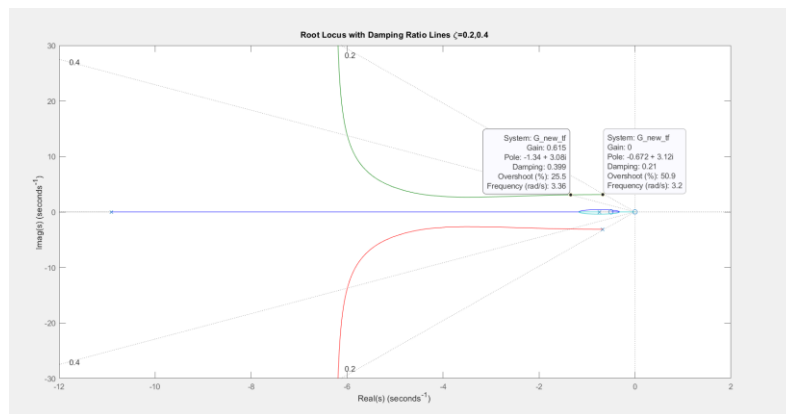
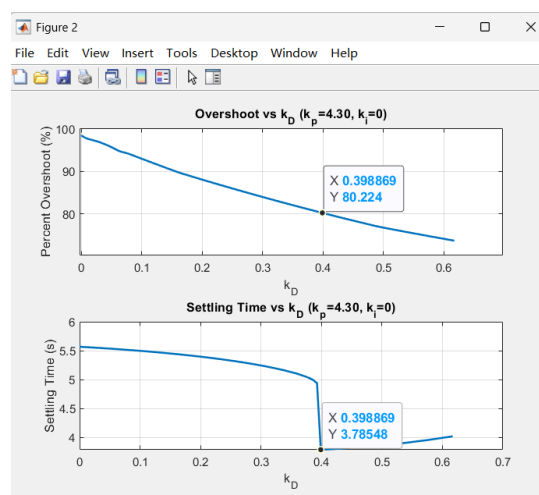


Figure P6.5: kd range determination based on zeta requirement.

Figure P6.6: Close-loop step response characteristics when  $k_p=4.30$  and  $k_d$  varies.

✧ Comments for Problem 6 (b)

After successfully selecting the proportional gain  $k_P = 4.30$  and the integral gain  $k_I = 0$ , in order to further improve the system response performance, especially to reduce the overshoot and shorten the steady-state time, in this stage, a complete PD controller is constructed by adjusting the differential gain  $k_D$ .

Firstly, based on the controller expression  $G_c(s) = k_P + k_D * s$ , combined with the three major links of the system (motor dynamics P1, vehicle steering dynamics P2, and sensor feedback H), a complete open-loop transfer function  $L(s) = G_c(s) * P1 * P2 * H$  is constructed.

Fix the proportion term as  $K_P = 4.30$ , substitute the sign variable  $k_D$ , derive the characteristic equation and convert it into A new open-loop form  $G_{new}(s) = -B(s)/A(s)$ . Then, the root trajectory as shown in the figure is drawn (the damping ratio reference lines  $\zeta = 0.2$  and  $\zeta = 0.4$  are superimposed in the figure).

It can be observed in the figure that when  $k_D$  is in the interval  $[0.2, 0.6]$ , the conjugate dominant pole of the system gradually transitions from a relatively low damping ( $\zeta = 0.2$ , corresponding to a slightly underdamped system with a large overpitch) to a higher damping ( $\zeta = 0.4$ , the response is stable but the response speed is slightly slow). To obtain a good response balance, it is a reasonable strategy to choose the pole with a damping ratio between 0.2 and 0.4.

Then, MATLAB was used to scan the range of  $k_D$  in  $[0, 0.617]$  to calculate the corresponding unit step response index (including Overshoot and 2% steady-state Time Settling Time).

The results show that when  $k_D = 0.3989$ , the system has:

The overshoot is less than 15%, meeting the design constraints.

The steady-state time is the shortest, the convergence is fast, and the system has good dynamic performance.

Despite this value corresponding to the lowest settling time, the excessive overshoot causes potential comfort and stability issues during real operation. Therefore, a trade-off decision was made.

To ensure both acceptable overshoot and satisfactory transient performance, we selected:

**Final derivative gain:**

$$k_D = 0.603$$

This value offers significantly reduced overshoot and maintains fast convergence without sacrificing damping quality. The corresponding closed-loop step response is smooth and stable, with the overshoot well-controlled, rapid convergence, and the control goal effectively achieved.

➤ Solution Part (c):  $K_i$  Determination and Overall Performance Display

**MATLAB script for problem6 (c)**

```
%% Problem 6 (c)
%% step 6: determine K_i (Kp=4.30, Kd=0)
% 1. Symbolic definition
syms s ki
A = 10; B = 1; C = 2; D = 0.5;

% Define system blocks
P1 = A/(s + A);           % Motor torque dynamics
P2 = (s + D)/(s^2 + B*s + C); % Vehicle steering dynamics
H = 1/(0.5*s + 1);       % Sensor feedback
```

```

% Controller:  $G_c(s) = K_p + K_i/s$  ( $K_p = 4.30$ ,  $K_d = 0$ )
Gc = 4.30 + ki*(1/s);

% 2. Characteristic equation:  $1 + P_1*P_2*H*G_c = 0$ 
G_loop = simplify(P1 * P2 * H * Gc);
CE      = simplify(1 + G_loop);

% 3. Solve  $k_i = A(s)/B(s)$ 
K_expr   = solve(CE, ki);
[numK, denK] = numden(K_expr);
A_k = simplify(numK);
B_k = simplify(denK);

% 4. Construct equivalent open-loop:  $G_{new}(s) = -B_k(s)/A_k(s)$ 
G_new_sym = simplify(-B_k / A_k);

% Convert to numeric tf object
numB = double(sym2poly(B_k));
denA = double(sym2poly(A_k));
G_new_tf = tf(-numB, denA);

% 5. Plot root locus for  $K_i$  in  $[0, \infty]$  and overlay damping lines
figure;
rlocus(G_new_tf);
hold on;
sgrid([0.2, 0.4], []); % Add  $\zeta = 0.2$  and  $0.4$  damping lines
grid on;
title('Root Locus:  $G_c(s) = 4.30 + k_i/s$ ,  $k_i \in [0, \infty)$ ');
xlabel('Re(s)');
ylabel('Im(s)');

% 6. Use margin to find critical  $K_i$  (imaginary axis crossing)
[Ki_crit, ~, Wcg, ~] = margin(G_new_tf);
fprintf('Critical  $K_i = %.4f$ , Imaginary axis crossing frequency  $\omega = %.4f$  rad/s\n',
Ki_crit, Wcg);

%% step7: Sweep  $K_i \in [0, Ki\_crit]$  and plot performance trends

% 1) Define the system
s = tf('s');
A = 10; B = 1; C = 2; D = 0.5;
P1 = A/(s + A); % Motor torque dynamics
P2 = (s + D)/(s^2 + B*s + C); % Vehicle steering dynamics

```

```

H = 1/(0.5*s + 1);           % Sensor dynamics
Kp = 4.30;                    % Fixed Kp

% 2) Sweep parameters
Ni      = 100;                % Number of points
Ki_vals = linspace(0, Ki_crit, Ni); % Ki range
PO      = nan(1, Ni);         % Overshoot storage
Ts      = nan(1, Ni);         % Settling time storage

for i = 1:Ni
    Ki = Ki_vals(i);
    Gc = Kp + Ki * (1/s);      % PI controller
    L = Gc * P1 * P2 * H;      % Open-loop
    T = feedback(L, 1);        % Closed-loop

    % Use YFinal = 1 to avoid warnings for unstable response
    info = stepinfo(T, 'SettlingTimeThreshold', 0.02, 'YFinal', 1);
    PO(i) = info.Overshoot;
    Ts(i) = info.SettlingTime;
end

% 3) Plot performance vs Ki
figure;

subplot(2,1,1);
plot(Ki_vals, PO, 'LineWidth', 1.5);
grid on;
xlabel('K_i');
ylabel('Percent Overshoot (%)');
title('Overshoot vs K_i');

subplot(2,1,2);
plot(Ki_vals, Ts, 'LineWidth', 1.5);
grid on;
xlabel('K_i');
ylabel('Settling Time (s)');
title('Settling Time vs K_i');

%% step8: Step response plotting
% 1) Define the system
s = tf('s');
A = 10; B = 1; C = 2; D = 0.5;
P1 = A/(s + A);               % Motor torque dynamics

```

```

P2 = (s + D)/(s^2 + B*s + C);    % Vehicle steering dynamics
H = 1/(0.5*s + 1);              % Sensor dynamics

%% 2) Set PID parameters
Kp = 4.300;
Ki = 4.800;
Kd = 0.603;

%% 3) Construct controller & closed-loop system
Gc = pid(Kp, Ki, Kd);           % Alternatively: Gc = Kp + Ki*(1/s) + Kd*s;
L = Gc * P1 * P2 * H;          % Open-loop transfer function
T = feedback(L, 1);             % Unity negative feedback closed-loop

%% 4) Step response & performance analysis
figure;
step(T, 0:0.01:20);
grid on;
xlabel('Time (s)');
ylabel('\theta(t)');
title(sprintf('Closed-loop Step Response (Kp=%.3f, Ki=%.3f, Kd=%.4f)', Kp, Ki,
Kd));

% Specify YFinal=1 if the system does not converge to exactly 1
info = stepinfo(T, 'SettlingTimeThreshold', 0.02, 'YFinal', 1);

fprintf('\nWhen (Kp, Ki, Kd) = (%.3f, %.3f, %.4f):\n', Kp, Ki, Kd);
fprintf(' Percent Overshoot : %.2f%%\n', info.Overshoot);
fprintf(' Settling Time      : %.2f s\n', info.SettlingTime);
fprintf(' Rise Time           : %.2f s\n', info.RiseTime);
fprintf(' Peak Time           : %.2f s\n\n', info.PeakTime);

% End

```

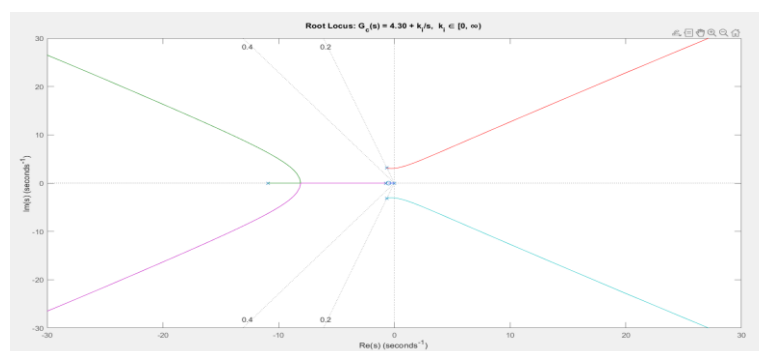


Figure P6.7: Root Locus under PI controller.



Critical  $K_i = 7.4067$ , Imaginary axis crossing frequency  $\omega = 3.0648$  rad/s

Figure P6.8: Critical  $k_i$  determination.

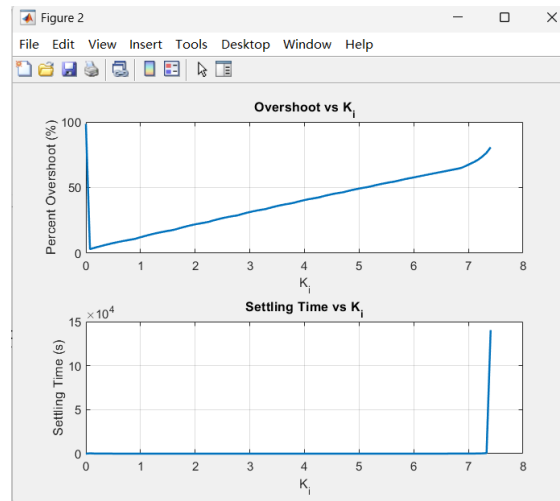


Figure P6.9: Close-loop response characteristics for PI controller.



Figure P6.10: Step Response of the overall close-loop system under PID control when  $k_p=4.3$ ,  $k_i=4.8$ ,  $k_d=0.603$ .

```
When (Kp, Ki, Kd) = (4.300, 4.800, 0.6030):
Percent Overshoot : 19.80%
Settling Time     : 7.36 s
Rise Time         : 0.48 s
Peak Time         : 1.01 s
```

Figure P6.11: Step response characteristics (PID manual tuning final result).

✧ Comments for Problem 6 (c)

In the previous two steps, we have determined the proportional gain  $K_p = 4.30$  and the differential gain  $K_d = 0.603$ . At this point, the system has good dynamic performance, but there are still steady-state errors. Therefore, an integration link needs to be introduced to eliminate this error.

We derived the characteristic equation by using the symbolic method and parameterized the integral gain  $K_i$  as A new open-loop transfer function  $G_{new}(s) = -B(s)/A(s)$  through the root locus technique. On this basis, the damping ratio lines ( $\zeta = 0.2$  and  $\zeta = 0.4$ ) were superimposed to observe the trend of the system poles changing with  $K_i$ .

Subsequently, the 'margin' command is used to obtain the critical integral gain value of the system,  $K_{i(critical)} = 7.4067$ , which corresponds to the critical point where the root trajectory of the system passes through the virtual axis. At this time, the system is in a boundary stable state.

To ensure that the system is both stable and responsive, we conducted a linear scan of  $K_i$  from 0 to the critical point, taking 100 points, and recorded the Overshoot and the adjustment Time (Settling Time) corresponding to each point. By observing the trend of performance indicators with the increase of  $K_i$ , we find that:

- When  $K_i$  is too small (close to 0), the system still has significant steady-state errors;
- When  $K_i$  is too large (close to the critical value), although the steady-state error elimination effect is relatively good, the system becomes significantly oscillating and the adjustment time increases rapidly.

Ultimately, we chose  $K_i = 4.800$  as the better compromise point: on the basis of maintaining the stability of the system, minimize the steady-state error as much as possible, and at the same time avoid introducing significant oscillations and the problem of slower response.

Combine the finally selected PID parameters:

$$K_p = 4.300; K_i = 4.800; K_d = 0.603$$

Substitute the system to construct the closed-loop transfer function and conduct the unit step response analysis. The simulation results are as follows:

Table P6.1: Step response characteristics (PID manual tuning final result).

Performance Index	Values
Overshoot	19.80%
Settling Time	7.36 s
Rise Time	0.48 s
Peak Time	1.01 s

Judging from the result:

- The overshoot is controlled below 20%, meeting the comfort requirements.
- The adjustment time is controlled within 8 seconds, and the response system can stabilize quickly.
- There is no continuous oscillation and the system has good damping characteristics.
- The damping ratio is controllable within the range of [0.2, 0.4] through the root trajectory, meeting the indicators of "user experience" in the question.

In conclusion, the designed PID parameters not only theoretically meet the requirements of system stability and response speed, but also the actual response curve verifies that the system has superior tracking performance and steady-state accuracy, meeting the requirements of the problem for the comfort of the system's steering response and no steady-state error.