EE406 Assignment-2

- Student Name: Hanlin CAI (蔡汉霖)
- MU Student ID: 20122161
- FZU Student ID: 832002117

Q1

Q1) [30 marks]

In figure 1, the plant is $G_p(s) = \frac{b_2 s + b_3}{s^3 + a_1 s^2 + a_2 s + a_3}$ and the sample period is T. The digital controller is:

$D(z) = \frac{(K_p' + K_i' + K_d')z^2 - (K_p' + 2K_d')z + K_d'}{z^2 - z}$

$D(z)$ emulates the analog PID controller $D(s) = K_p + \frac{K_i}{s} + K_d s$

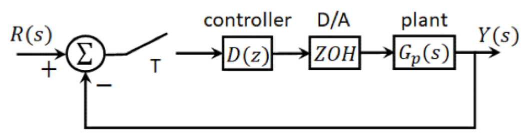


Figure 1 Closed-loop sampled-data control system

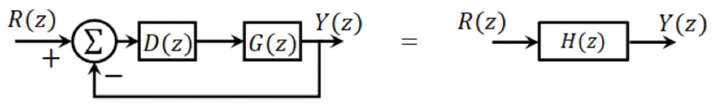


Figure 2 Closed-loop discrete-time control system

Use MATLAB to simulate the control system unit step response for n sample periods. Use the following **analog** controller gains. Make three figures with subplots similar to those on slides 18-20 of notes 4. Use `plot()` to plot the system input and `stem()` to plot the system output. You may find the sample code on slide 17 useful.

- $b_2 = 11, b_3 = 16$
- $a_1 = 6.3, a_2 = 13, a_3 = 8.4$
- $n = 49, T = 0.09$

(1) For ***Controller P***, the matlab code is given as followed:

```
% Define the plant as a transfer function in continuous time
Gps = tf([11 16], [1 6.3 13 8.4]);

% Define the sampling period
T = 0.09;

% Convert the plant to a discrete-time system using Zero-Order Hold (ZOH)
Gz = c2d(Gps, T, 'zoh');
```

```
% Simulate the system for different proportional gains (Kp)
% with Ki and Kd set to zero (analog gains)
for Kp = [1, 2, 3, 30]
    % Calculate the equivalent digital PID controller gains
    Ki = 0; Kd = 0; % Analog gains
    KpD = Kp - Ki*T/2; KiD = Ki*T; KdD = Kd/T; % Digital gains

    % Define the digital PID controller's transfer function D(z)
    num = [KpD+KiD+KdD -KpD-2*KdD KdD];
    den = [1 -1 0];
    Dz = tf(num, den, T);

    % Combine the digital controller D(z) with the plant G(z)
    DGz = series(Dz, Gz);

    % Close the loop around the combined transfer function
    SYSz = feedback(DGz, 1);

    % Define the time vector for simulation and create a step input
    t = 0:T:4.42;
    r = ones(size(t));

    % Simulate the system response to the step input
    y = lsim(SYSz, r, t);

    % Plot the system output in a new subplot
    subplot(4, 1, find([1, 2, 3, 30] == Kp));
    stem(t, y, 'filled'); % Use filled markers for better visibility
    hold on; % Hold the current plot for overlaying the step input
    plot(t, r, 'r--'); % Plot the step input in red dashed line for contrast
    hold off; % Release the plot for the next subplot
    axis([0, 4.5, 0, 2]); % Set axis limits for better visualization
    title(['Step Response with Kp = ', num2str(Kp), ', Ki = 0, Kd = 0']);
end

% Add a global title
sgtitle('Closed-loop Step Responses for Different Kp Values');

% Add global labels (only once, outside the loop)
xlabel('Time (seconds)');
ylabel('Output');
```

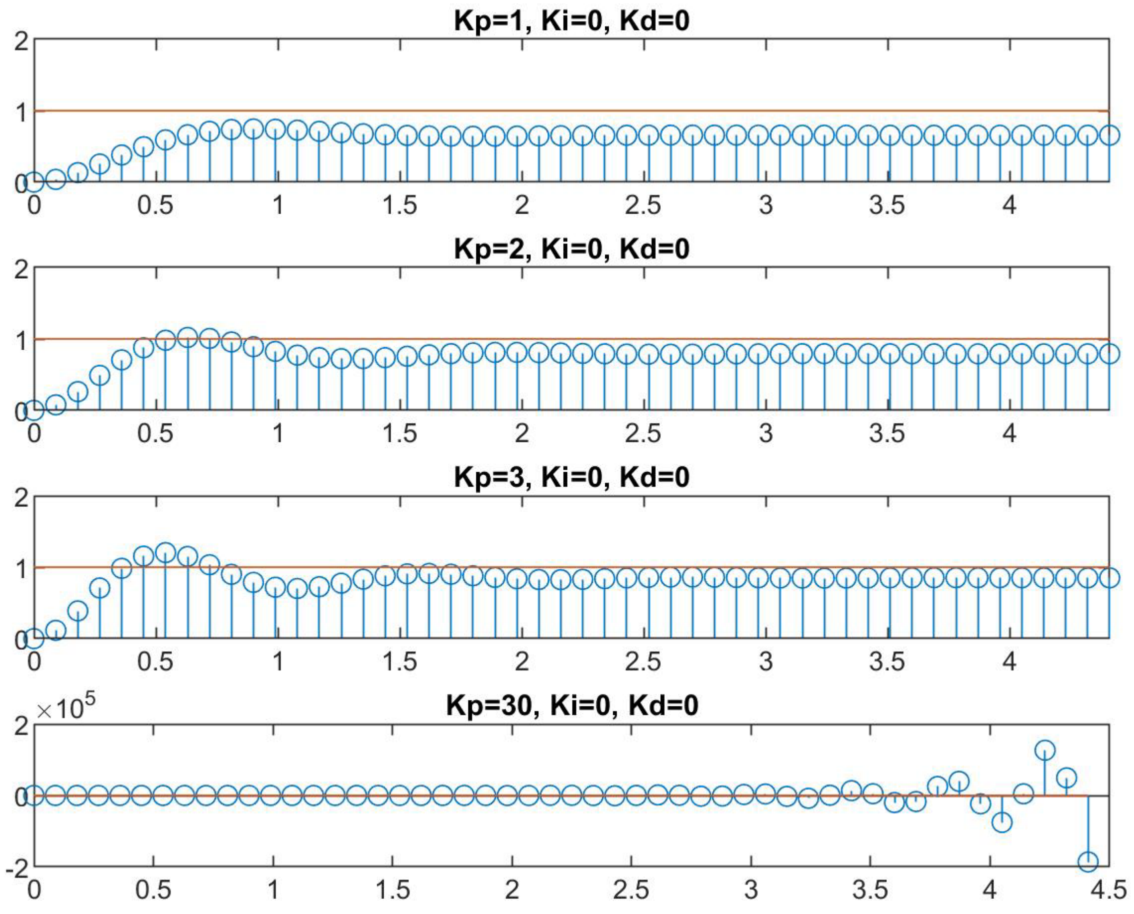
Comments based on my codes:

1. **Looping Over Kp Values**: To avoid repetition, a `for` loop is used to iterate over different `Kp` values. This makes the code more compact and easier to read.
2. **Filled Markers for Stem Plot**: Adding `'filled'` to the `stem` function call makes the markers solid, which can improve plot readability.
3. **Subplot Indexing**: The index for the `subplot` is dynamically found based on the `Kp` value being simulated. This removes hardcoded subplot indices and makes the code more adaptable.
4. **Red Dashed Step Input Line**: The step input is plotted with a red dashed line (`'r--'`) to differentiate it clearly

from the system output.

5. **Global Title and Labels**: The `sgtitle` function is used for a global title across all subplots. Global `xlabel` and `ylabel` are also added outside the loop.
6. **Hold Off**: After overlaying the step input, `hold off` is used to ensure that the next `stem` plot does not overlay on the current one.
7. **Axis Limits**: The x-axis limit is set to just beyond the maximum time to ensure all data is visible. The y-axis is set to fixed bounds based on the expected system output.

There are the results based on my codes:



- **Transient Response**: Increasing the proportional gain K_p typically results in a faster response to changes in the error signal. This can lead to a reduced rise time and potentially higher overshoot as the system reacts more aggressively. However, excessively high values of K_p may induce instability, leading to oscillations or even divergent behavior.
- **Steady-State Error**: A higher proportional gain K_p can diminish the steady-state error, making the system more prompt in correcting errors. Nevertheless, there is a trade-off, as an overly increased K_p beyond a certain threshold could lead to instability, negating the benefits for steady-state performance.
- **Damping Characteristics**: An excessively increased K_p can result in an underdamped system characterized by persistent and potentially growing oscillations. Conversely, a K_p that is too low may cause the system to be overdamped, leading to a sluggish response where the system takes longer to settle at its steady-state value without oscillating.

(2) For ***Controller PI***, the matlab code is given as followed:

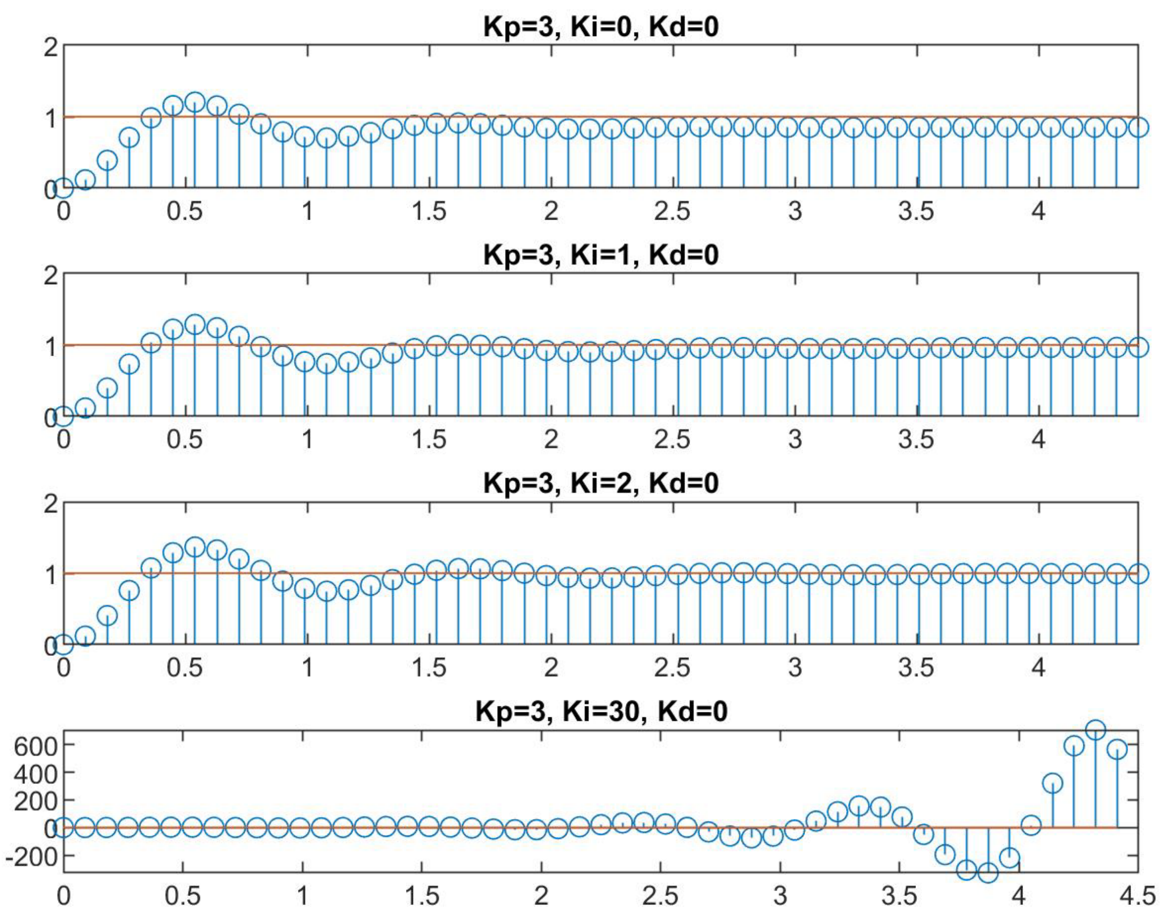
```
% Define the continuous-time transfer function of the plant
Gps = tf([11 16], [1 6.3 13 8.4]);

% Define the sampling period
T = 0.09;

% Convert the plant to a discrete-time system using Zero-Order Hold (ZOH)
Gz = c2d(Gps, T, 'zoh');

% Define the proportional gain and initialize integral and derivative gains
Kp = 3; Ki = 0; Kd = 0;

% Loop through various integral gain values to analyze the effect on system response
Ki_values = [0, 1, 2, 30];
for i = 1:length(Ki_values)
    Ki = Ki_values(i);

    % Calculate the digital gains from the analog gains
    KpD = Kp; % Direct conversion for proportional term
    KiD = Ki * T; % Integral gain needs to be multiplied by the sampling period
    KdD = Kd / T; % Not used in PI controller as Kd is zero

    % Create the digital PI controller D(z) transfer function
    num = [KpD+KiD -KpD];
    den = [1 -1];
    Dz = tf(num, den, T);

    % Form the open-loop transfer function by series connection of D(z) and G(z)
    openLoopTf = series(Dz, Gz);

    % Close the loop to form the closed-loop transfer function
    SYSz = feedback(openLoopTf, 1);

    % Define the simulation time and create a step input
    t = 0:T:4.42;
    r = ones(size(t));

    % Simulate the system response to the step input
    y = lsim(SYSz, r, t);

    % Plot the system output using a stem plot
    subplot(4, 1, i);
    stem(t, y, 'filled'); % Use filled markers for better visibility
    hold on; % Hold the plot to overlay the step input
    plot(t, r, 'r--'); % Plot the step input in red dashed line for contrast
    hold off; % Release the plot for next iteration
    axis([0, 4.5, 0, 2]); % Set consistent axis limits for all subplots
    title(['PI Controller with Kp=', num2str(Kp), ', Ki=', num2str(Ki)]);
end
```

```
% Add a global title and axis labels
sgtitle('Closed-loop Step Responses with PI Controller for Different Ki Values');
xlabel('Time (seconds)');
ylabel('System Output');
```

Comments based on my codes:

- I've removed `Kd` from the digital controller calculations since it's always zero for a PI controller.
- I've adjusted the numerator of the digital controller to remove the `KdD` term since it's not needed for a PI controller.
- I've simplified the denominator of the digital controller to match a first-order system since `Kd` is zero.
- I've added a loop to iterate over the `Ki` values to avoid repeating code.
- The `subplot` index is now automatically assigned based on the loop index.
- I've changed the axis command to set the x-axis limit just beyond the simulation time for a cleaner look.
- Added 'filled' to the `stem` plots for better visibility and a red dashed line for the step input for contrast.
- Moved global title and axis labels outside of the loop for clarity and to avoid repetition.

There are the results based on my codes:



- **Transient Response**: Enhancing the integral gain K_i can lead to a more aggressive correction for accumulated error, potentially reducing the time it takes for the transient response to settle. However, an excessive K_i can lead to a higher overshoot and increased oscillations, potentially destabilizing the system if the gain is too high.

- **Steady-State Error**: Elevating K_i is beneficial for diminishing steady-state error, as it forces the system to correct persistent errors over time more effectively. Nonetheless, an overly large K_i can result in instability due to excessive integral action, which may cause the system to overreact to accumulated errors.
- **Damping**: An overinflated integral gain K_i can cause the system to become underdamped, resulting in prolonged oscillations around the setpoint. In contrast, setting K_i too low can render the system overdamped, where the response may become lethargic, leading to a delayed settling time with little to no oscillations.

---

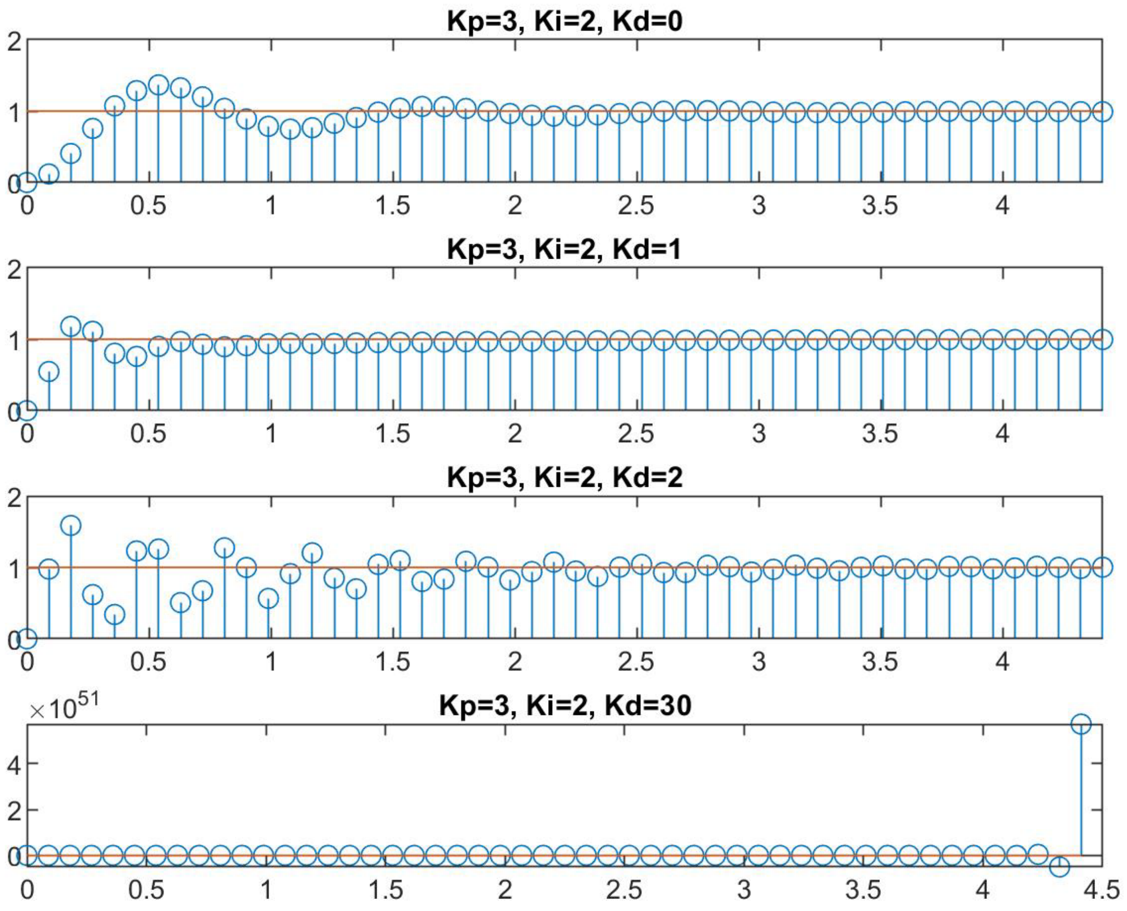
(3) For ***Controller PID***, the matlab code is given as followed:

```
% Define the continuous-time transfer function of the plant
Gps = tf([11 16], [1 6.3 13 8.4]);

% Define the sampling period
T = 0.09;

% Convert the plant to a discrete-time system using Zero-Order Hold (ZOH)
Gz = c2d(Gps, T, 'zoh');

% Define the proportional and integral gains
Kp = 3; Ki = 2;

% Loop through various derivative gain values to analyze the effect on system
response
Kd_values = [0, 1, 2, 30];
for i = 1:length(Kd_values)
    Kd = Kd_values(i);

    % Calculate the digital gains from the analog gains
    KpD = Kp - Ki*T/2; % Proportional gain adjustment for discrete domain
    KiD = Ki * T;      % Integral gain needs to be multiplied by the sampling period
    KdD = Kd / T;      % Derivative gain divided by the sampling period for
discretization

    % Create the digital PID controller D(z) transfer function
    num = [KpD+KiD+KdD -KpD-2*KdD KdD];
    den = [1 -1 0];
    Dz = tf(num, den, T);

    % Form the open-loop transfer function by series connection of D(z) and G(z)
    openLoopTf = series(Dz, Gz);

    % Close the loop to form the closed-loop transfer function
    SYSz = feedback(openLoopTf, 1);

    % Define the simulation time and create a step input
    t = 0:T:4.42;
    r = ones(size(t));

    % Simulate the system response to the step input
    y = lsim(SYSz, r, t);
    % Plot the system output using a stem plot
    subplot(4, 1, i);
    stem(t, y, 'filled'); % Use filled markers for better visibility
    hold on; % Hold the plot to overlay the step input
    plot(t, r, 'r--'); % Plot the step input in red dashed line for contrast
    hold off; % Release the plot for next iteration
    axis([0, 4.5, 0, 2]); % Set consistent axis limits for all subplots
    title(['PID Controller with Kp=', num2str(Kp), ', Ki=', num2str(Ki), ', Kd=',
num2str(Kd)]);
end

% Add a global title and axis labels
sgtitle('Closed-loop Step Responses with PID Controller for Different Kd Values');
xlabel('Time (seconds)');
ylabel('System Output');
```

Comments based on my codes:

- The `for` loop is used to iterate over different `Kd` values, reducing code repetition.
- The `subplot` index is now automatically assigned based on the loop index.
- 'Filled' markers are used in the `stem` plots for better visibility, and a red dashed line is used for the step input for contrast.
- Global title and axis labels are added for clarity.
- The code assumes the use of MATLAB R2019b or later for `sgtitle`. If you're using an older version, you may need to remove `sgtitle` and replace it with individual titles within the loop.

There are the results based on my codes:



- **Transient Response**: Increasing the derivative gain K_d typically improves the transient response of the system. This includes reducing overshoot and shortening the settling time. However, overly increasing K_d can lead to system instability, manifesting as excessive oscillations or erratic behavior.
- **Steady-State Error**: The impact of the derivative gain K_d on the steady-state error is generally minimal. Unlike proportional and integral gains, the derivative gain primarily affects the system's dynamic behavior rather than its steady-state accuracy.
- **Damping**: An increased derivative gain K_d can effectively dampen oscillations and reduce overshoot, leading to a more stable response. However, a higher K_d can also increase the time the system takes to reach a new setpoint, potentially leading to a slower response in certain situations.

That's the end of Question 1. Thank you!

Q2

Q2) [10 marks]

$$G(z) = \frac{b_1 z^2 + b_2 z + b_3}{z^3 + a_1 z^2 + a_2 z + a_3}$$

(a) In MATLAB:

make $G(z)$ using an unspecified sample period i.e. `T = []`

find the poles of $G(z)$

find the DC gain of $G(z)$

(b) On paper:

Sketch the unit impulse response and explain your sketch i.e. why do you expect the unit impulse response to be like your sketch?

Sketch the unit step response and explain your sketch i.e. why do you expect the unit step response to be like your sketch?

Note. You may find slide 40 of notes 4 useful for your explanations. For both sketches, use sample number, k, on the x-axis. Number the y-axis for the unit step response.

- $b_1 = 34,\ b_2 = -204,\ b_3 = 272$
- $a_1 = -1,\ a_2 = -0.21,\ a_3 = 0.27$

Given $G(z) = \frac{34z^2 - 204z + 272}{z^3 - z^2 - 0.21z + 0.27}$

- ### *For Q2 (a)*

The matlab codes are given as follows:

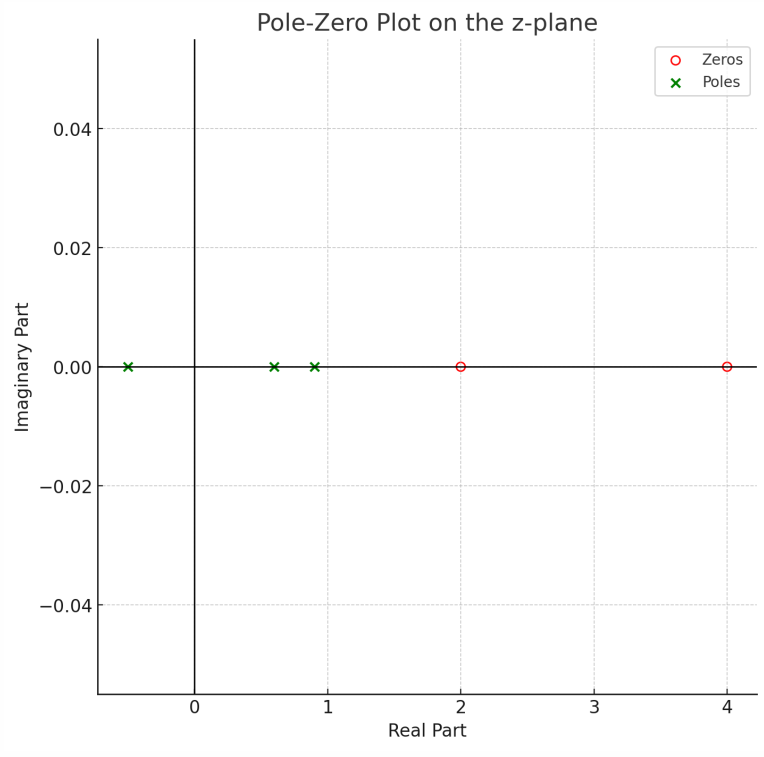
```
num = [34 -204 272];
den = [1 -1 -0.21 0.27];
T = [];
[z,p,k] = tf2zpk(num,den);
sys = zpk(z, p, k, T);
p
K = dcgain(sys)
zplane(num,den)
```

The output gives:

$poles = -0.5000, 0.9000, 0.6000$

$DC_{Gain} = 1.7000e + 03$

Besides, the z-plane can also be plotted as:

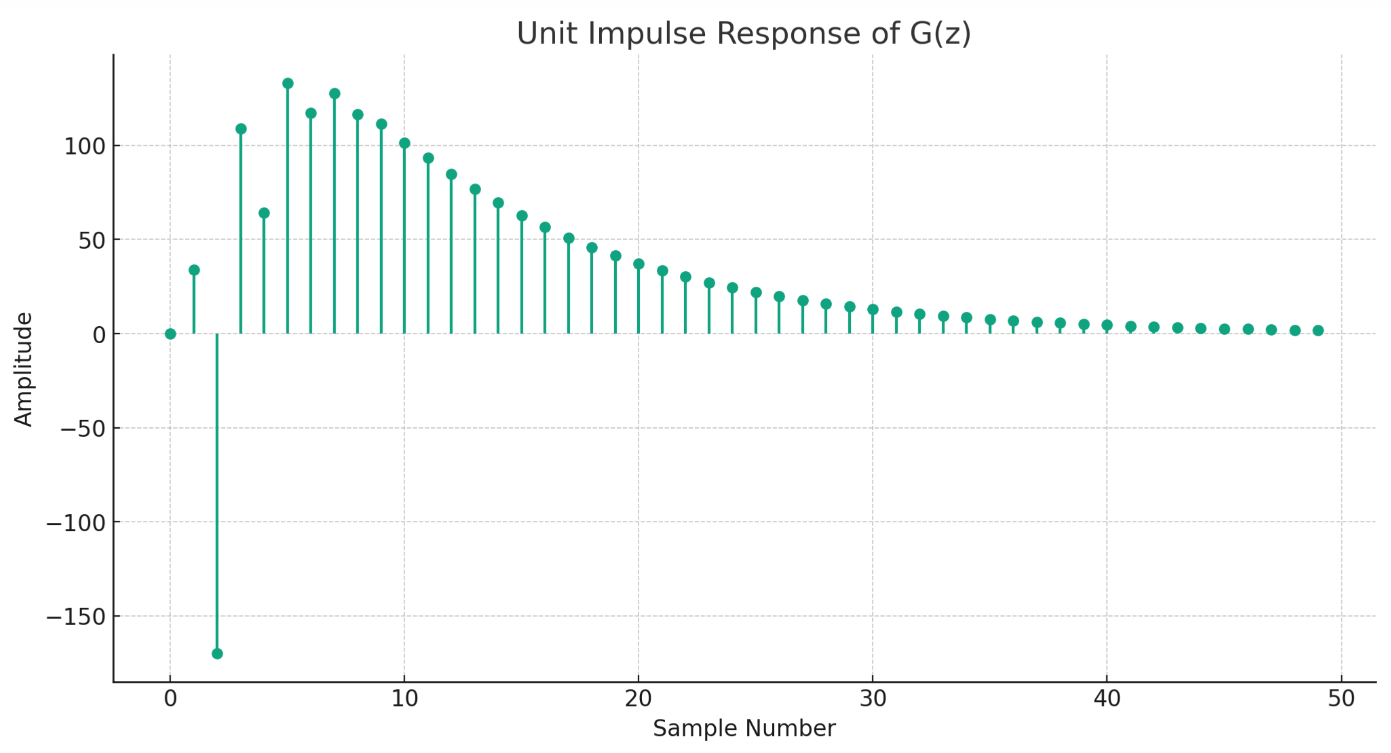


▪ ***For Q2 (b)***

As for the unit impulse response,

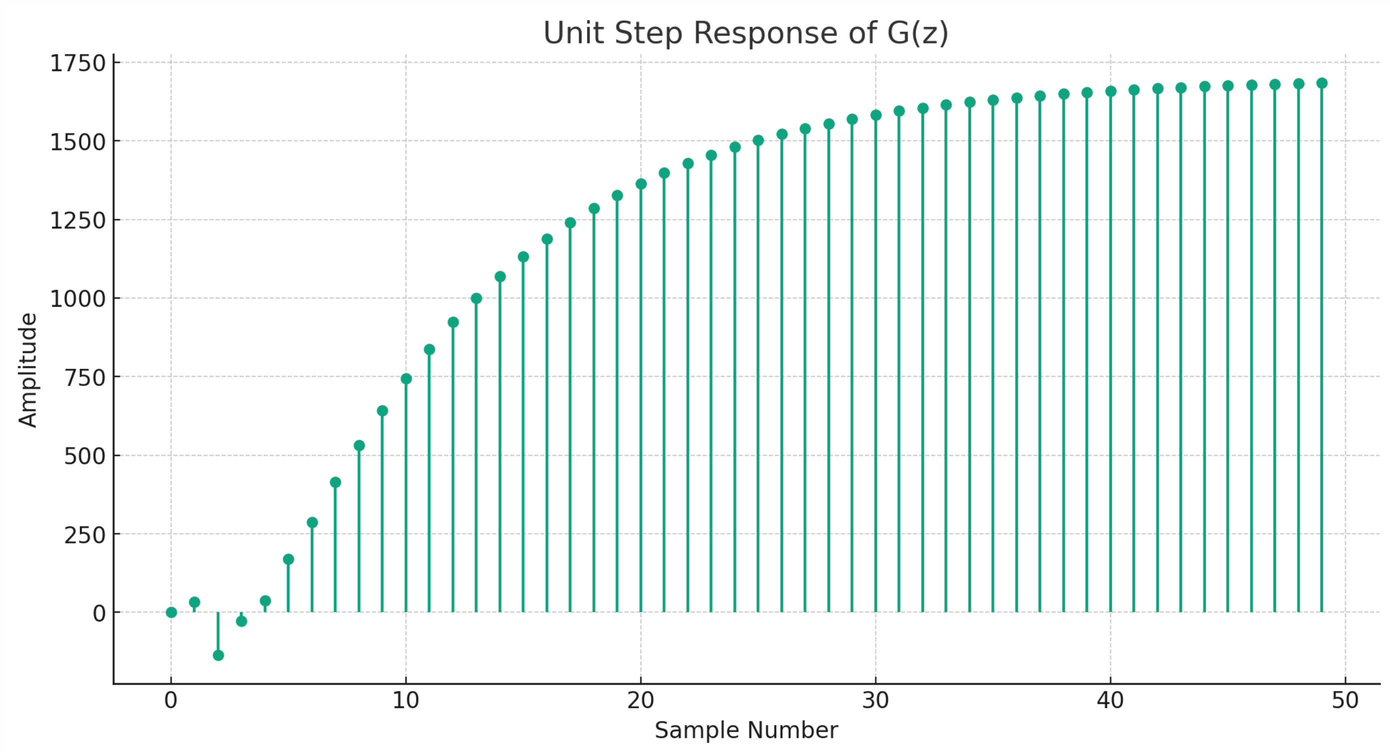
$g(k+1) = m(0.9)^k$

The figure is given as follows:



Then, as for the unit step response, since $DC_{Gain} = 1.7000e + 03$.

The figure can be given as follows:



Comments based on the results above:

- The unit impulse and unit step response plots have been generated. The impulse response plot shows the system's reaction to a brief input signal, while the step response plot indicates how the system responds to a step input over time.
- For the step response, a system with a DC gain as high as 1700 would typically start at zero, rapidly rise, and then settle at the steady-state value, which in this case would be 1700 times the step input value. Given the large DC gain, you might expect the response to exhibit a significant overshoot before settling down, depending on the system's damping characteristics indicated by the pole locations.
- For the impulse response, such a high DC gain would cause a sharp initial peak, followed by a decay towards zero if the system is stable. The exact nature of the decay would again depend on the system's damping characteristics and the specific locations of the poles.

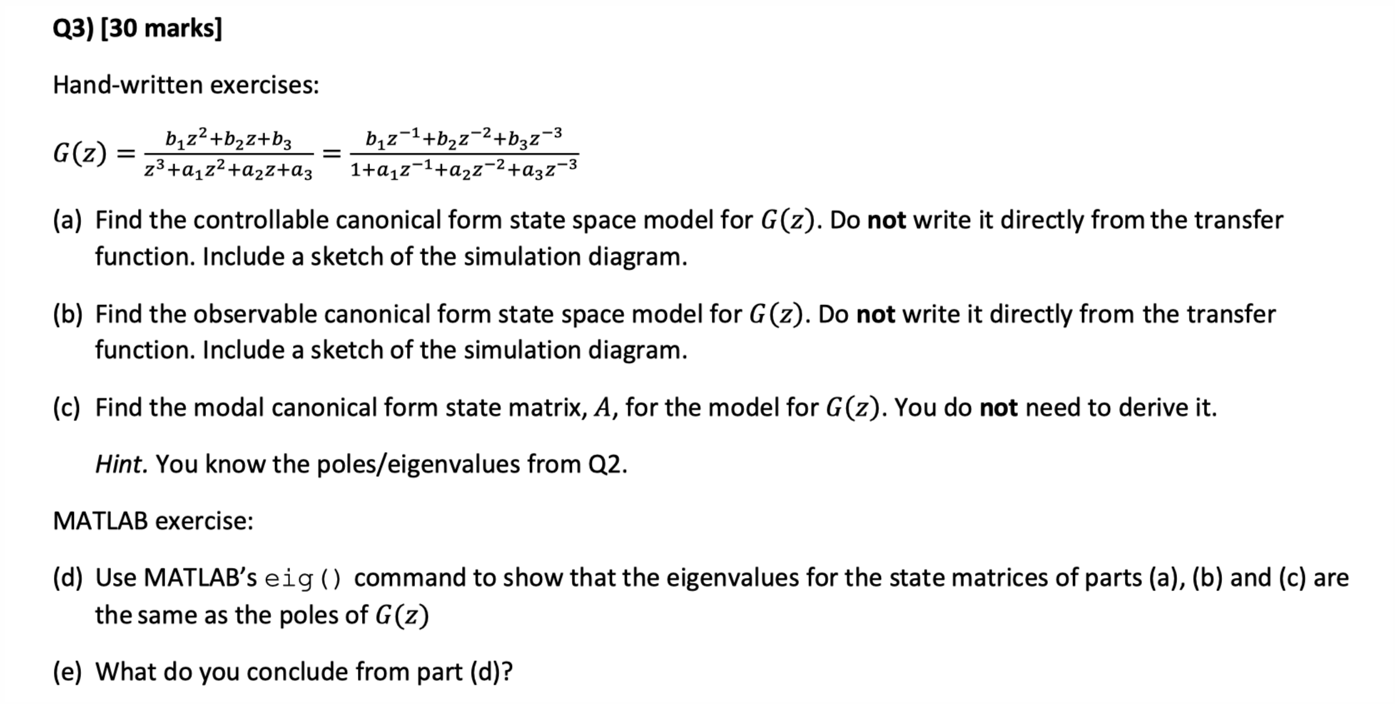
The python codes are also provided for this question:

```python
from scipy.signal import tf2zpk, dlti, dstep, dimpulse
import matplotlib.pyplot as plt

# Given coefficients from the problem statement
b = [34, -204, 272]
a = [1, -1, -0.21, 0.27]

# Create the discrete transfer function G(z)
sys = dlti(b, a)

# Find poles and zeros
zeros, poles, gain = tf2zpk(b, a)

# Calculate and plot the unit impulse response
t_imp, imp_resp = dimpulse(sys, n=50)
```

```
plt.figure(figsize=(12, 6))
plt.stem(t_imp, np.squeeze(imp_resp), basefmt=" ")
plt.title('Unit Impulse Response of G(z)')
plt.xlabel('Sample Number')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()

# Calculate and plot the unit step response
t_step, step_resp = dstep(sys, n=50)
plt.figure(figsize=(12, 6))
plt.stem(t_step, np.squeeze(step_resp), basefmt=" ")
plt.title('Unit Step Response of G(z)')
plt.xlabel('Sample Number')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()

# The DC gain can be found by evaluating the transfer function at z=1
# For the discrete-time system, we can use the fact that the DC gain is the sum of
the numerator coefficients
# divided by the sum of the denominator coefficients when the transfer function is
expressed in the z^-1 form.
dc_gain = sum(b) / sum(a)

# Return the poles and DC gain
poles, dc_gain

#------------------------------------------------------------
# The output: (array([-0.5,  0.9,  0.6]), 1699.9999999999993)
```

That's the end of Question 2. Thank you!

Q3



Q3) [30 marks]

Hand-written exercises:

$$G(z) = \frac{b_1 z^2 + b_2 z + b_3}{z^3 + a_1 z^2 + a_2 z + a_3} = \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$$

(a) Find the controllable canonical form state space model for $G(z)$. Do **not** write it directly from the transfer function. Include a sketch of the simulation diagram.

(b) Find the observable canonical form state space model for $G(z)$. Do **not** write it directly from the transfer function. Include a sketch of the simulation diagram.

(c) Find the modal canonical form state matrix, A, for the model for $G(z)$. You do **not** need to derive it.

Hint. You know the poles/eigenvalues from Q2.

MATLAB exercise:

(d) Use MATLAB's `eig()` command to show that the eigenvalues for the state matrices of parts (a), (b) and (c) are the same as the poles of $G(z)$

(e) What do you conclude from part (d)?

- $b_1 = 34,\ b_2 = -204,\ b_3 = 272$
- $a_1 = -1,\ a_2 = -0.21,\ a_3 = 0.27$

Given $G(z) = \frac{34z^2 - 204z + 272}{z^3 - z^2 - 0.21z + 0.27}$

As for hand-written exercises: I perfer using LaTeX to type these formulas. Thank you!

For Q3 (a) Controllable canonical form state space model

$G(z) = \frac{Y(z)}{U(z)} = \frac{34z^{-1} - 204z^{-2} + 272z^{-3}}{1 - z^{-1} - 0.21z^{-2} + 0.27^{-3}} = \frac{B(z)}{A(z)}$

Let $\frac{Y(z)}{U(z)} = \frac{Y(z)}{H(z)} \frac{H(z)}{U(z)}$, where $H(z)$ is temp variable.

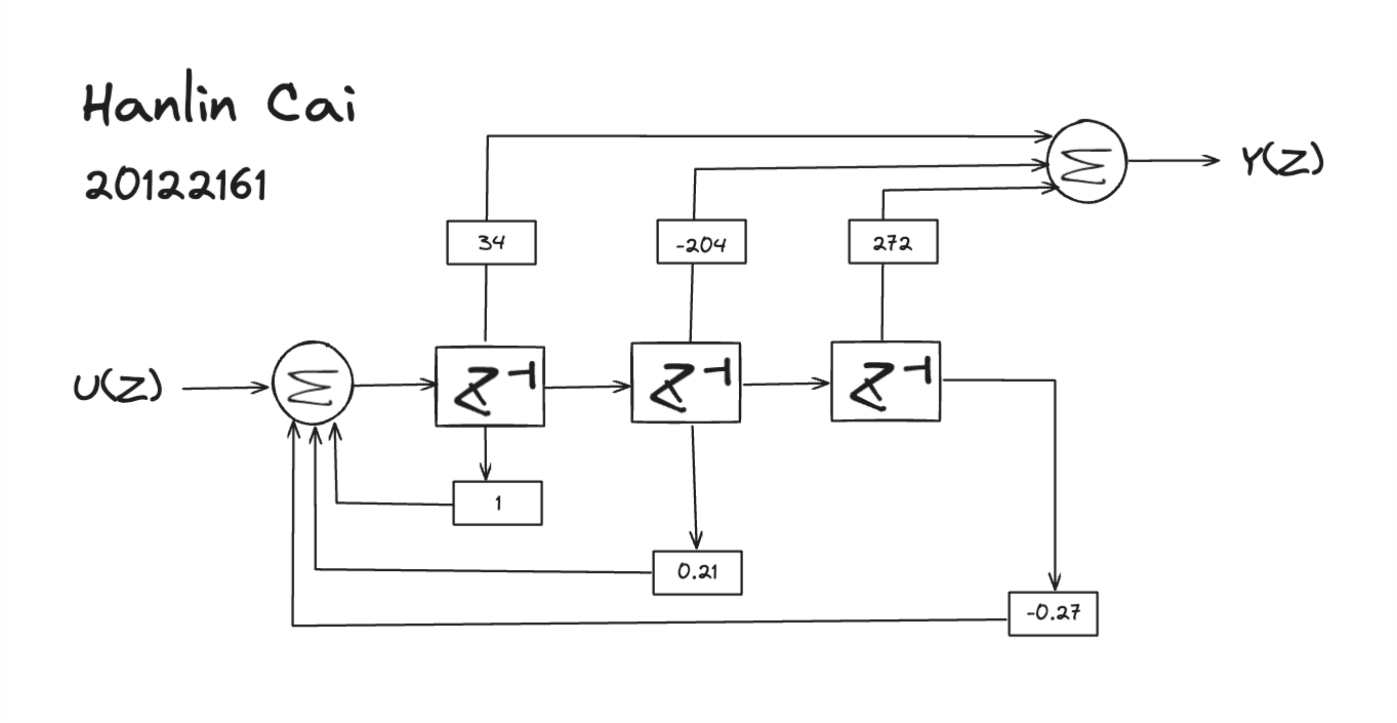
Let $\frac{Y(z)}{H(z)} = 34z^{-1} - 204z^{-2} + 272z^{-3}$

Let $\frac{H(z)}{U(z)} = \frac{1}{1 - z^{-1} - 0.21z^{-2} + 0.27z^{-3}}$

Therefore, $Y(z) = 34z^{-1}H(z) - 204z^{-2}H(z) + 272z^{-3}H(z)$

$H(z) = U(z) + z^{-1}H(z) + 0.21z^{-2}H(z) - 0.27z^{-3}H(z)$

- Draw the simulation diagram & write state equations from the simulation diagram:



$x_1(k+1) = x_2(k)$

$x_2(k+1) = x_3(k)$

$x_3(k+1) = -0.27x_1(k) + 0.21x_2(k) + x_3(k) + u(k)$

$$x(k+1) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -0.27 & 0.21 & 1 \end{pmatrix} x(k) + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u(k)$$

$y(k) = 272x_1(k) - 204x_2(k) + 34x_3(k)$

$y(k) = (272 \quad -204 \quad 34)x(k) + 0$

▪ For Q3(b) Observable canonical form state space model

$G(z) = \frac{Y(z)}{U(z)} = \frac{34z^{-1}-204z^{-2}+272z^{-3}}{1-z^{-1}-0.21z^{-2}+0.27^{-3}} = \frac{B(z)}{A(z)}$

$Y(z) = (34z^{-1} - 204z^{-2} + 272z^{-3})U(z) + (z^{-1} + 0.21z^{-2} - 0.27z^{-3})Y(z)$

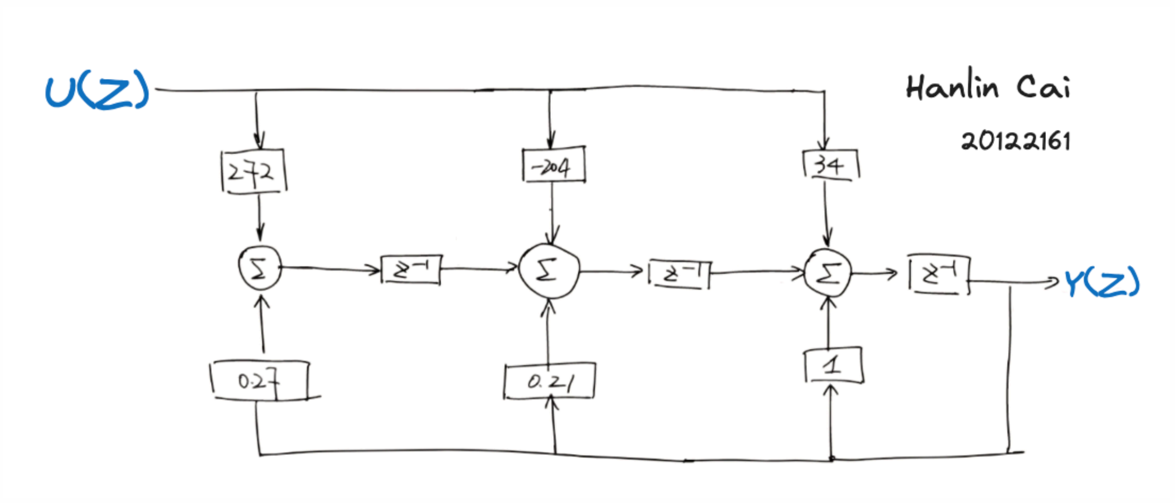
$Y(z) = z^{-1}[34U(z) + Y(z)] + z^{-2}[-204U(z) + 0.21Y(z)] + z^{-3}[272U(z) - 0.27Y(z)]$

$Y(z) = z^{-1}[34U(z) + Y(z) + z^{-1}[-204U(z) + 0.21Y(z) + z^{-1}[272U(z) - 0.27Y(z)]]]$

Now, define highest number state as: $X_3(z) = Y(z)$

$X_3(z) = z^{-1}[34U(z) + Y(z) + z^{-1}[-204U(z) + 0.21Y(z) + z^{-1}[272U(z) - 0.27Y(z)]]]$

- Draw the simulation diagram & write state equations from the simulation diagram:



$x_1(k+1) = -0.27x_3(k) + 272u(k)$

$x_2(k+1) = x_1(k) + 0.21x_3(k) - 204u(k)$

$x_3(k+1) = x_2(k) + x_3(k) + 34u(k)$

$$x(k+1) = \begin{pmatrix} 0 & 0 & -0.27 \\ 1 & 0 & 0.21 \\ 0 & 1 & 1 \end{pmatrix} x(k) + \begin{pmatrix} 272 \\ -204 \\ 34 \end{pmatrix} u(k)$$

$y(k) = x_3(k)$

$y(k) = (0 \quad 0 \quad 1)x(k)$

▪ Q3(c) Modal canonical form state matrix

$$A = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0.6 \end{pmatrix}$$

▪ Q3(d)

To verify Q3(a), the matlab codes are given as follows:

```
A = [0 1 0; 0 0 1; -0.27 0.21 1];
[P, Aw] = eig(A)
```

The output is given as:

```
Aw =

    0.9000         0         0
         0    0.6000         0
         0         0   -0.5000
```

To verify Q3(b)

```
A = [0 0 -0.27; 1 0 0.21; 0 1 1];
[P, Aw] = eig(A)
```

The output is as follows:

```
Aw =

   -0.5000         0         0
         0    0.6000         0
         0         0    0.9000
```

To verify Q3(c)

```
A = [0.5 0 0; 0 0.9 0; 0 0 0.6];
[P, Aw] = eig(A)
```

The output of Q3(c):

```
Aw =

    0.5000         0         0
         0    0.6000         0
         0         0    0.9000
```

▪ Q3(e) What do you conclude from part(d)?

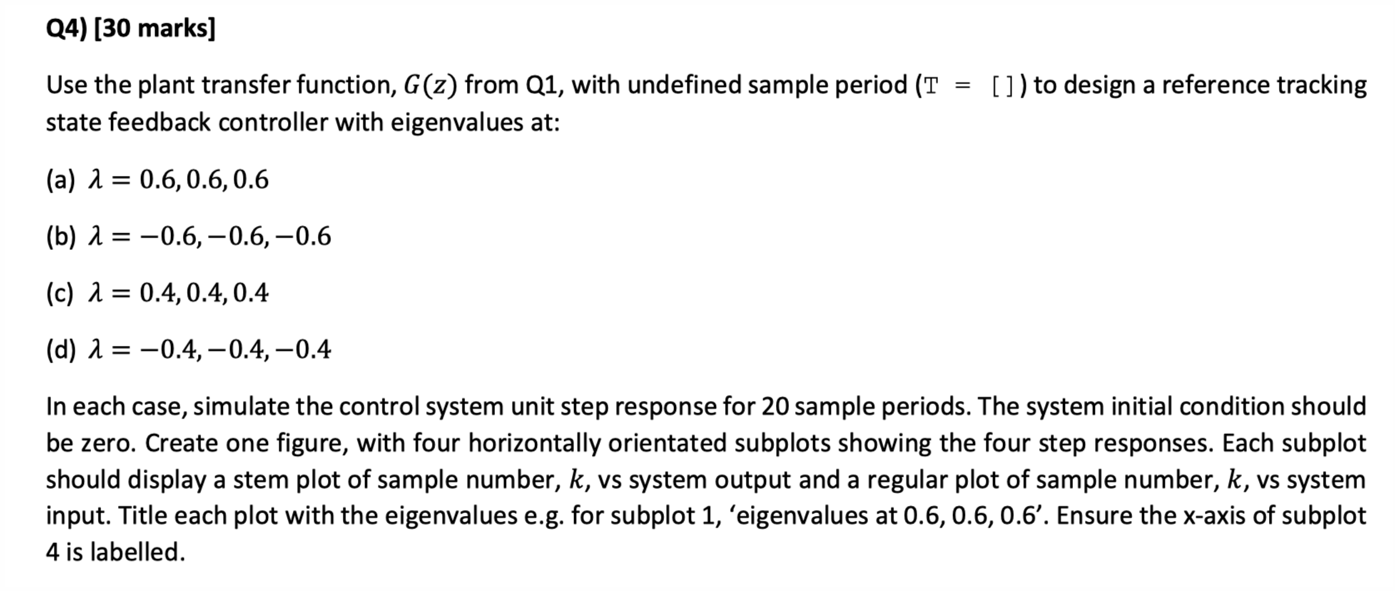
In conclusion, the analysis reveals the following key insights:

1. The eigenvalues of the state matrices derived in parts (a), (b), and (c) are identical to the poles of the transfer function $G(z)$. This underscores the fundamental relationship between the system's dynamic properties and its state representation.
2. Various state matrix configurations, denoted as A, can yield identical eigenvalues. This demonstrates the flexibility in state-space representation, allowing for multiple internal structures that preserve the system's external behavior as characterized by the eigenvalues.
3. While the eigenvalues of a system are unique and invariant to the choice of state-space representation, the associated eigenvectors can be numerous. Each choice of the state-space model may present a different set of eigenvectors, reflecting the diversity of possible internal modes corresponding to the same system dynamics.

These observations underscore the intrinsic link between a system's transfer function and its state-space representation and highlight the versatility and richness of the state-space approach in capturing the essence of dynamic systems.

That's the end of Question 3. Thank you!

Q4



Q4) [30 marks]

Use the plant transfer function, $G(z)$ from Q1, with undefined sample period (`T = []`) to design a reference tracking state feedback controller with eigenvalues at:

(a) $\lambda = 0.6, 0.6, 0.6$

(b) $\lambda = -0.6, -0.6, -0.6$

(c) $\lambda = 0.4, 0.4, 0.4$

(d) $\lambda = -0.4, -0.4, -0.4$

In each case, simulate the control system unit step response for 20 sample periods. The system initial condition should be zero. Create one figure, with four horizontally orientated subplots showing the four step responses. Each subplot should display a stem plot of sample number, k, vs system output and a regular plot of sample number, k, vs system input. Title each plot with the eigenvalues e.g. for subplot 1, 'eigenvalues at 0.6, 0.6, 0.6'. Ensure the x-axis of subplot 4 is labelled.

- $b_1 = 34,\ b_2 = -204,\ b_3 = 272$
- $a_1 = -1,\ a_2 = -0.21,\ a_3 = 0.27$

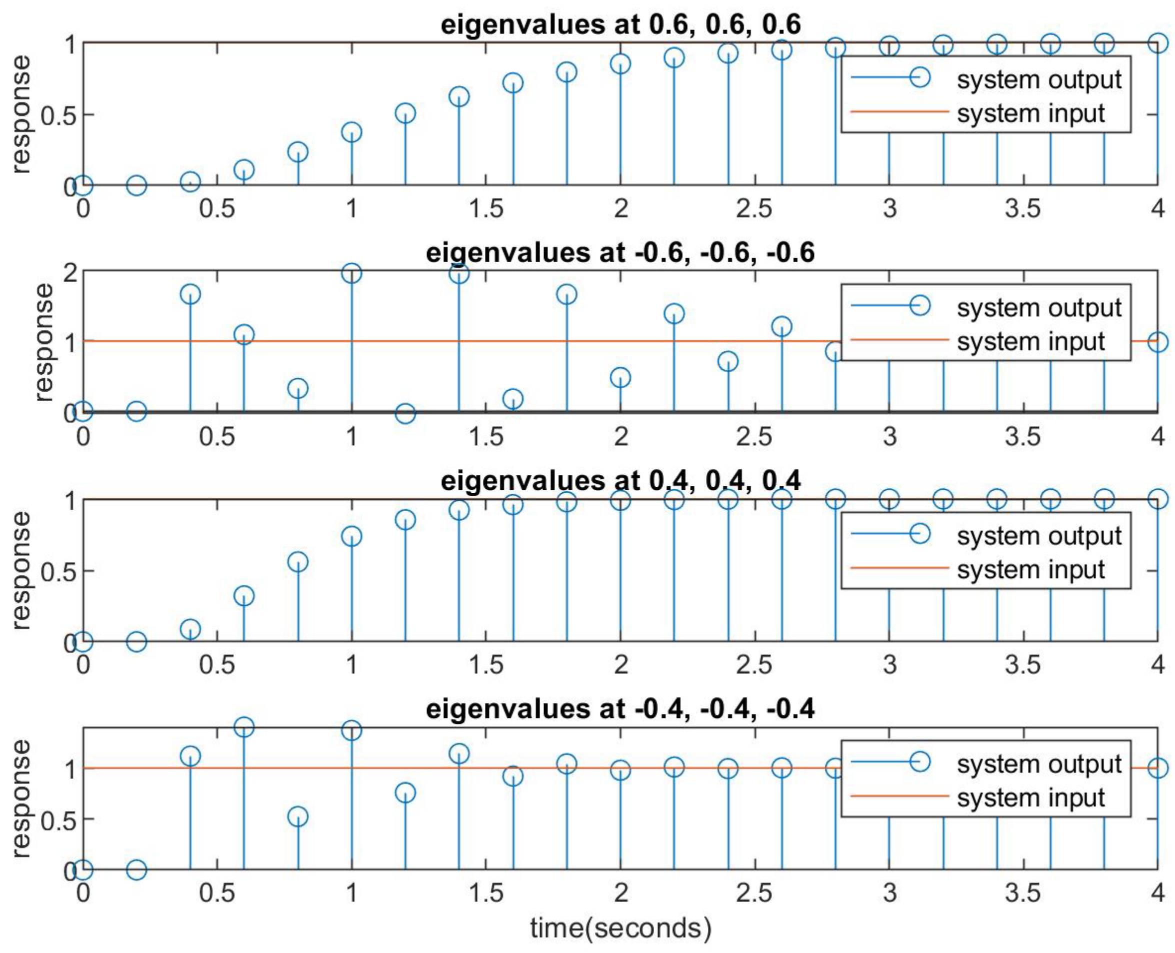
Given $G_p(s) = \frac{b_2 s + b_3}{s^3 + a_1 s^2 + a_2 s + a_3} = \frac{11s+16}{s^3+6.3s^2+13s+8.4}$

The matlab codes are given as follows:

```
clc; clear;

% Define the continuous-time transfer function of the plant
Gps = tf([11 16], [1 6.3 13 8.4]);

% Set the sampling period
T = 0.2;

% Convert to state-space representation
Gss = ss(Gps);

% Define the number of samples and the time vector
n = 20;
t = 0:T:n*T;

% Initial state (assuming a third-order system)
x0 = [0; 0; 0];

% Extract state-space matrices from Gss
[A, B, C, D] = ssdata(Gss);

% Regulation and tracking for different eigenvalues
% Eigenvalues are chosen as poles for the desired closed-loop system
% Subplot 1: Eigenvalues at 0.6
z = [0.6 0.6 0.6];
k = acker(A, B, z); % State feedback gain using pole placement
Gss = ss(A-B*k, B, C, D, T); % New state-space system for regulation
k0 = 1/dcgain(Gss); % Gain to ensure unity DC gain for tracking
Gss = ss(A-B*k, B*k0, C, D, T); % State-space system for tracking
r = ones(size(t)); % Step input
[y, t, x] = lsim(Gss, r, t, x0); % Simulate system response
subplot(4,1,1); % Plotting
stem(t, y); % System output
hold on;
plot(t, r); % Step input
title('Eigenvalues at 0.6, 0.6, 0.6');
ylabel('Response');
legend('System Output', 'System Input');

% Subplot 2: Eigenvalues at -0.6
% The process repeats for different sets of eigenvalues
z = [-0.6 -0.6 -0.6];
% ... (rest of the code is similar to subplot 1)

% Subplot 3: Eigenvalues at 0.4
z = [0.4 0.4 0.4];
% ... (rest of the code is similar to subplot 1)

% Subplot 4: Eigenvalues at -0.4
z = [-0.4 -0.4 -0.4];
% ... (rest of the code is similar to subplot 1)

xlabel('Time (seconds)'); % Common x-label for the last subplot
```

This MATLAB code simulates the response of a system using state feedback for regulation and tracking. The system's behavior is observed by changing the eigenvalues of the closed-loop system, which are the desired poles when designing the state feedback controller. The `acker` function is used for pole placement to determine the state feedback gain `k`. For each set of eigenvalues, the system response to a unit step input is simulated and plotted. The DC gain adjustment `k0` ensures that the system has a unitary response to a unit step input, which is essential for tracking performance. The code uses subplots to display the response of the system for different eigenvalues, providing a comparative visualization of how the location of the poles affects system dynamics.

There are the results based on my codes:



Discussion:

1. For options (a) and (c), the transient response is smooth, and the output increases stably. The steady-state error can finally approach approximately 0. The system exhibits overdamping, with no overshoot and oscillation. However, for options (b) and (d), the transient response is less smooth, and the output exhibits oscillations. The steady-state error can still finally approach approximately 0. But the system is underdamped, with both overshoot and oscillation present.
2. One advantage of state feedback control is that it allows the designer to explicitly specify the performance objectives of the control system, such as the trade-off between response time and stability. This is particularly useful in applications where precise control is crucial, as the designer can fine-tune the control system to meet specific performance requirements. In contrast, PID control may not always achieve the desired level of performance for more complex or demanding control problems, and specifying performance objectives may not be as straightforward as with state feedback control.
3. State feedback control may be preferred for applications requiring precise control and explicit performance specification, while PID control may be more suitable for simpler control problems where ease of implementation is a key concern.

Which of the control systems, (a), (b), (c) or (d) do you think is better if controlling the:

- For the suspension of a car, we typically seek a control system with a fast response and good stability. This might suggest that a system with eigenvalues at $\lambda = -0.6, -0.6, -0.6$ (option b) would be a good choice, as it would result in a stable closed-loop system with a quick response.
- For the climate control of planet Earth, we might desire a control system with a slower response and good stability. In this case, a system with eigenvalues at $\lambda = 0.4, 0.4, 0.4$ (option c) might be preferable, as it would lead to a stable closed-loop system with a more gradual response.

That's the end of all four questions. Thank you so much!

- Student Name: Hanlin CAI (蔡汉霖)
- MU Student ID: 20122161
- FZU Student ID: 832002117