Experiment No. 01: Time Domain Analysis (Using Formulas)

What is Time Domain Analysis?

Time domain analysis in control systems involves examining the system's output response as a function of time when subjected to various standard test input signals, such as step, ramp, parabolic, or impulse inputs. This type of analysis directly reveals how the system behaves over time, including its transient response (how it moves from an initial state to a final state) and its steady-state response (its behavior as time approaches infinity). It provides crucial insights into the system's performance characteristics without requiring frequency transformations.

Significance of Time Domain Analysis

The significance of time domain analysis lies in its direct relevance to how a system will perform in a real-world scenario. Key performance characteristics observed in the time domain include:

- **Transient Response:** This part of the response is concerned with the initial behavior of the system as it transitions from its initial state to its final steady-state. It often involves oscillations or decaying exponentials.
- **Steady-State Response:** This is the behavior of the system output as time approaches infinity. For stable systems, the output typically settles to a constant value or follows the input with a constant error.

From the unit step response (response to a sudden, constant input), several important performance metrics are typically determined:

- **Rise Time (t_r):** The time required for the response to rise from 10% to 90% of its final value for overdamped systems, or from 0% to 100% for underdamped systems. It indicates how quickly the system responds.
- **Peak Time (t_p):** The time required for the response to reach the first peak of the overshoot. It is primarily relevant for underdamped systems.
- **Maximum Overshoot (M_p):** The maximum peak value of the response curve measured from the final steady-state value, usually expressed as a percentage of the final value. It indicates the relative stability of the system. A large overshoot means less stable or more oscillatory.
- **Settling Time (t_s):** The time required for the response to reach and stay within a specified percentage (typically $\pm 2\%$ or $\pm 5\%$) of its final value. It indicates how long it takes for the oscillations to die out and for the system to settle.

These metrics are vital for evaluating whether a system meets desired performance criteria, such as speed of response, accuracy, and stability.

Necessary Formulas for Second-Order System Step Response

For a standard second-order underdamped system of the form $G(s) = \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$, the unit step response $C(s) = \frac{1}{s} \cdot \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$. The inverse Laplace transform gives the time-domain response

$c(t)$: $c(t) = 1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1-\zeta^2}} \sin(\omega_d t + \phi)$ where $\omega_d = \omega_n\sqrt{1-\zeta^2}$ (damped natural frequency) and $\phi = \arctan\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right) = \arccos(\zeta)$.

The key performance metrics are defined by the following formulas for an underdamped second-order system ($0 < \zeta < 1$):

- **Rise Time (t_r):** $t_r = \frac{\pi-\phi}{\omega_d} = \frac{\pi-\arccos(\zeta)}{\omega_n\sqrt{1-\zeta^2}}$
- **Peak Time (t_p):** $t_p = \frac{\pi}{\omega_d} = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}}$
- **Maximum Overshoot (M_p):** (as a percentage) $M_p = e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}} \times 100\%$
- **Settling Time (t_s):**
  - For 2% criterion: $t_s \approx \frac{4}{\zeta\omega_n}$
  - For 5% criterion: $t_s \approx \frac{3}{\zeta\omega_n}$

For critically damped ($\zeta = 1$) and overdamped ($\zeta > 1$) systems, there is no overshoot and the rise/settling times are calculated differently or approximated. For these cases, we will primarily rely on plotting the analytical response and observing the times.

MATLAB Code for Problems (Using Formulas, without built-in 'step'/'impulse'/'lsim functions)

1. Determine the Rise Time (t_r), Peak Time (t_p), Maximum Overshoot (M_p), and Settling Time (t_s) in the Unit Step Response.

(a) $C(s)/R(s) = \frac{9}{s^2+5s+9}$ **Explanation:** This is a second-order system. We first extract the natural frequency (ω_n) and damping ratio (ζ) by comparing the denominator to the standard form $s^2+2\zeta\omega_n s+\omega_n^2$. From s^2+5s+9: The coefficient of s^0 is 9, so $\omega_n^2 = 9 \implies \omega_n = 3$ rad/s. The coefficient of s^1 is 5, so $2\zeta\omega_n = 5 \implies 2\zeta(3) = 5 \implies 6\zeta = 5 \implies \zeta = 5/6 \approx 0.8333$. Since $0 < \zeta < 1$, the system is underdamped, and we can use the provided formulas for t_r, t_p, M_p, t_s. The analytical step response $c(t)$ will be plotted.

How to Solve (Analyze) in MATLAB

MATLAB's Control System Toolbox provides powerful functions for time domain analysis.

```
% Define the system (e.g., as a transfer function)
num = [numerator_coefficients];
den = [denominator_coefficients];
sys = tf(num, den);

% Unit Step Response
figure;
step(sys);
grid on;
title('Unit Step Response');

% Get step response characteristics
info = stepinfo(sys);
fprintf('Rise Time: %.4f s\n', info.RiseTime);
```

```
    fprintf('Peak Time: %.4f s\n', info.PeakTime);
    fprintf('Maximum Overshoot: %.2f %%\n', info.Overshoot);
    fprintf('Settling Time (2%%): %.4f s\n', info.SettlingTime);

    % Unit Impulse Response
    figure;
    impulse(sys);
    grid on;
    title('Unit Impulse Response');

    % Response to other inputs (e.g., ramp, parabolic) using lsim
    t = 0:0.01:10; % Define time vector
    u_ramp = t; % Unit ramp input
    figure;
    lsim(sys, u_ramp, t);
    grid on;
    title('Unit Ramp Response');

    u_parabolic = 0.5 * t.^2; % Unit parabolic input (t^2/2)
    figure;
    lsim(sys, u_parabolic, t);
    grid on;
    title('Unit Parabolic Response');

```

Listing 1: General MATLAB usage for Time Domain Analysis

MATLAB Code for Problems

We will now provide the MATLAB code for each problem, along with detailed explanations.

1. Determine the Rise Time (t_r), Peak Time (t_p), Maximum Overshoot (M_p), and Settling Time (t_s) in the Unit Step Response.

(a) $C(s)/R(s) = \frac{9}{s^2+5s+9}$ **Explanation:** This is a second-order system. We define its numerator and denominator polynomials and create a transfer function object. The 'stepinfo' function is then used to directly compute the desired time domain specifications from the unit step response. We also plot the step response for visualization.

MATLAB Code:

```
% G(s) = 9 / (s^2 + 5s + 9)

d = [1 5 9];        % denominator coefficients
n = 9;              % numerator (wn^2)
wn = sqrt(d(3));    % natural frequency
z = d(2)/(2*wn);    % damping ratio

fprintf('-- System Specs --\n');
fprintf('wn = %.4f rad/s\n', wn);
fprintf('zeta = %.4f\n', z);

if z >= 0 && z < 1
wd = wn * sqrt(1 - z^2);      % damped frequency
phi = acos(z);                % phase angle

tr = (pi - phi) / wd;         % rise time
tp = pi / wd;                 % peak time
Mp = exp(-z * pi / sqrt(1 - z^2)) * 100;  % overshoot
ts2 = 4 / (z * wn);           % 2% settling time
ts5 = 3 / (z * wn);           % 5% settling time

fprintf('tr = %.4f s\n', tr);
fprintf('tp = %.4f s\n', tp);
fprintf('Mp = %.2f %%\n', Mp);
fprintf('ts (2%%) = %.4f s\n', ts2);
fprintf('ts (5%%) = %.4f s\n', ts5);
else
fprintf('System is not underdamped (z = %.4f)\n', z);
ts2 = 4 / (z * wn);
fprintf('Approx ts (2%%) = %.4f s\n', ts2);
end

%output:
-- System Specs --
wn = 3.0000 rad/s
zeta = 0.8333
tr = 1.5413 s
tp = 1.8945 s
Mp = 0.88 %
ts (2%) = 1.6000 s
ts (5%) = 1.2000 s

```

Listing 2: MATLAB Code for Problem 1(a)

(b) $S = -3 - j5$ **Explanation:** A single pole at $S = -3 - j5$ implies a complex conjugate pair of poles for a real system. The other pole must be $S^* = -3 + j5$. These poles are of a second-order system characteristic equation $s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$. From the pole form $s = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2}$: Real part: $-\zeta\omega_n = -3 \implies \zeta\omega_n = 3$ Imaginary part: $\omega_d = \omega_n\sqrt{1-\zeta^2} = 5$ Squaring and adding the real and imaginary parts: $\omega_n^2 = (-3)^2 + (5)^2 = 9 + 25 = 34 \implies \omega_n = \sqrt{34} \approx 5.831$. Then, $\zeta = 3/\omega_n = 3/\sqrt{34} \approx 0.5145$. To analyze the unit step response, we need a complete transfer function. A standard second-order system with unity DC gain is $G(s) = \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$. The denominator polynomial will be $s^2 + 2(0.5145)(\sqrt{34})s + (\sqrt{34})^2 = s^2 + 6s + 34$. The numerator will be $\omega_n^2 = 34$.

MATLAB Code:

```
% Given pole: S = -3 - j5
% Corresponds to second-order system: G(s) = wn^2 / (s^2 + 2*zeta*wn*s + wn^2)

% Step 1: Calculate natural frequency (wn) and damping ratio (zeta)
sig = 3;      % Real part = zeta * wn
wd  = 5;      % Imaginary part = wn * sqrt(1 - zeta^2)

wn = sqrt(sig^2 + wd^2);      % Natural frequency
z  = sig / wn;                % Damping ratio

% Step 2: Calculate time-domain specs using standard formulas
ph = atan(sqrt(1 - z^2)/z);                % Phase angle in radians

tr = (pi - ph) / wd;                       % Rise time
tp = pi / wd;                              % Peak time
Mp = exp(-z * pi / sqrt(1 - z^2)) * 100;   % Maximum overshoot in %
ts_2 = 4 / (z * wn);                         % Settling time (2%)
ts_5 = 3 / (z*wn);
% Step 3: Display results
fprintf('Rise Time (tr) = %.4f s\n', tr);
fprintf('Peak Time (tp) = %.4f s\n', tp);
fprintf('Max Overshoot (Mp) = %.2f %%\n', Mp);
fprintf('Settling Time 2%% (ts) = %.4f s\n', ts_2);
fprintf('Settling Time 5%% (ts) for = %.4f s\n', ts_5);

%output:
--- System 1(b) Specifications (from poles) ---
Rise Time (tr) = 0.4222 s
Peak Time (tp) = 0.6283 s
Max Overshoot (Mp) = 15.18 %
Settling Time 2% (ts) = 1.3333 s
Settling Time 5% (ts) for = 1.0000 s

```

Listing 3: MATLAB Code for Problem 1(b)

0.0.1 Using built-in function

```
  % Given pole: S = -3 - j5. This implies a conjugate pair: -3 +/- j5.
  % This corresponds to a second-order system characteristic equation:
  % s^2 + 2*zeta*wn*s + wn^2 = 0
  % From s = -zeta*wn +/- j*wn*sqrt(1-zeta^2)
  % -zeta*wn = -3  => zeta*wn = 3
  % wn*sqrt(1-zeta^2) = 5
  % wn^2 = (-3)^2 + (5)^2 = 9 + 25 = 34
  wn_1b = sqrt(34);
  zeta_1b = 3 / wn_1b;

  % Construct the transfer function assuming a standard second-order system
  % with unity DC gain (numerator = wn^2) for step response analysis
  num_1b = wn_1b^2; % 34
  den_1b = [1, 2*zeta_1b*wn_1b, wn_1b^2]; % s^2 + 6s + 34
  sys_1b = tf(num_1b, den_1b);

  % Plot the unit step response
```

```
    figure;
    step(sys_1b);
    grid on;
    title('Unit Step Response for System with Poles at -3 +/- j5');

    % Determine and display time domain specifications
    info_1b = stepinfo(sys_1b);
    fprintf('--- System 1(b) Specifications (from poles) ---\n');
    fprintf('Natural Frequency (wn): %.4f rad/s\n', wn_1b);
    fprintf('Damping Ratio (zeta): %.4f\n', zeta_1b);
    fprintf('Rise Time (tr): %.4f s\n', info_1b.RiseTime);
    fprintf('Peak Time (tp): %.4f s\n', info_1b.PeakTime);
    fprintf('Maximum Overshoot (Mp): %.2f %%\n', info_1b.Overshoot);
    fprintf('Settling Time (ts, 2%%): %.4f s\n', info_1b.SettlingTime);

```

Listing 4: MATLAB Code for Problem 1(b)

2. Step Response Comparison for Various Damping Ratios (ζ) when $\omega_n = 1.5$

$C(s)/R(s) = \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$ (with $\omega_n = 1.5$)

Explanation: We are given the standard form of a second-order system and a constant natural frequency $\omega_n = 1.5$. We need to vary the damping ratio ζ and observe its effect on the step response. The problem states to vary ζ from $-\infty$ to $+\infty$. This is a theoretical range. In practice, we choose representative values to illustrate the different damping conditions:

- $\zeta < 0$: Unstable (poles in RHP)
- $\zeta = 0$: Undamped (oscillatory, no decay)
- $0 < \zeta < 1$: Underdamped (oscillatory with decay)
- $\zeta = 1$: Critically damped (fastest response without overshoot)
- $\zeta > 1$: Overdamped (slower response, no overshoot)

We will create a loop, define the transfer function for each ζ, and plot the step response on the same figure using 'hold on'.

MATLAB Code:

```
% Natural frequency (assumed constant)
wn = 5;

% Time vector
t = 0:0.005:5;

% Prepare figure
figure;
sgtitle('Step Responses for Different \zeta Values');

% Loop for 6 different zeta inputs
for i = 1:6
% Take zeta input from user
zeta = input(['Enter value of zeta for case ' num2str(i) ': ']);

% Define transfer function: H(s) = wn^2 / (s^2 + 2*zeta*wn*s + wn^2)
num = [0 0 wn^2];
den = [1 2*zeta*wn wn^2];
```

```
% Calculate step response
[y, ~] = step(tf(num, den), t);

% Plot subplot
subplot(3, 2, i);
plot(t, y, 'b', 'LineWidth', 1.5);
grid on;
title(['\zeta = ' num2str(zeta)]);
xlabel('Time (s)');
ylabel('Response');
end
```

Listing 5: Step response vs damping ratio (manual)

3. Show the Unit-Step Response, Unit-Ramp Response, Unit-Parabolic Response and Unit-Impulse Response.

(a) $C(s)/R(s) = \frac{3}{s^2+3s+3}$ **Explanation:** This is a standard second-order transfer function. We will use 'step' for the unit step response, 'impulse' for the unit impulse response, and 'lsim' for the unit ramp and unit parabolic responses. For 'lsim', we need to define the time vector 't' and the corresponding input signal 'u'. A unit ramp input is $u(t) = t$ and a unit parabolic input is $u(t) = t^2/2$.

MATLAB Code:

```
% Define the transfer function
num_3a = 3;
den_3a = [1 3 3]; % s^2 + 3s + 3
sys_3a = tf(num_3a, den_3a);

% Define a common time vector for all responses
t_3a = 0:0.01:10; % From 0 to 10 seconds with 0.01s step

% --- Unit Step Response ---
figure;
step(sys_3a, t_3a);
grid on;
title('Unit Step Response for G(s) = 3/(s^2+3s+3)');

% --- Unit Impulse Response ---
figure;
impulse(sys_3a, t_3a);
grid on;
title('Unit Impulse Response for G(s) = 3/(s^2+3s+3)');

% --- Unit Ramp Response ---
u_ramp_3a = t_3a; % Unit ramp input u(t) = t
figure;
lsim(sys_3a, u_ramp_3a, t_3a);
grid on;
title('Unit Ramp Response for G(s) = 3/(s^2+3s+3)');
xlabel('Time (seconds)');
ylabel('Output');

% --- Unit Parabolic Response ---
u_parabolic_3a = 0.5 * t_3a.^2; % Unit parabolic input u(t) = t^2/2
figure;
```

```
    lsim(sys_3a, u_parabolic_3a, t_3a);
    grid on;
    title('Unit Parabolic Response for G(s) = 3/(s^2+3s+3)');
    xlabel('Time (seconds)');
    ylabel('Output');

```

Listing 6: MATLAB Code for Problem 3(a)

(b) State-Space Representation:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -1 & -0.5 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} [U]$$

$$y = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [0][U]$$

Explanation: This system is given in state-space form. We first define the A, B, C, and D matrices and create an 'ss' object. MATLAB's 'step', 'impulse', and 'lsim' functions can directly accept state-space system objects, making the process straightforward.

MATLAB Code:

```
    % Define state-space matrices
    A_3b = [-1 -0.5; 1 0];
    B_3b = [0.5; 0];
    C_3b = [1 -1];
    D_3b = 0;

    % Create the state-space system
    sys_3b_ss = ss(A_3b, B_3b, C_3b, D_3b);

    % Define a common time vector for all responses
    t_3b = 0:0.01:10; % From 0 to 10 seconds with 0.01s step

    % --- Unit Step Response ---
    figure;
    step(sys_3b_ss, t_3b);
    grid on;
    title('Unit Step Response for State-Space System');

    % --- Unit Impulse Response ---
    figure;
    impulse(sys_3b_ss, t_3b);
    grid on;
    title('Unit Impulse Response for State-Space System');

    % --- Unit Ramp Response ---
    u_ramp_3b = t_3b; % Unit ramp input u(t) = t
    figure;
    lsim(sys_3b_ss, u_ramp_3b, t_3b);
    grid on;
    title('Unit Ramp Response for State-Space System');
    xlabel('Time (seconds)');
    ylabel('Output');

    % --- Unit Parabolic Response ---
    u_parabolic_3b = 0.5 * t_3b.^2; % Unit parabolic input u(t) = t^2/2
    figure;
    lsim(sys_3b_ss, u_parabolic_3b, t_3b);
    grid on;
    title('Unit Parabolic Response for State-Space System');
    xlabel('Time (seconds)');
```

```
41     ylabel('Output');
42
```

Listing 7: MATLAB Code for Problem 3(b)

MATLAB Code for Root Locus Plots

Experiment No. 02: Root Locus

What is the Root Locus?

The Root Locus is a graphical method used in control systems engineering to analyze the behavior of the closed-loop poles of a system as a single system parameter (typically the proportional gain K) is varied from 0 to ∞. For a feedback control system with an open-loop transfer function $L(s) = K \cdot G(s)H(s)$, the characteristic equation of the closed-loop system is $1 + K \cdot G(s)H(s) = 0$. The roots of this characteristic equation are the closed-loop poles. The Root Locus plot shows the trajectories of these roots in the complex s-plane as K changes.

The plot consists of:

- **Open-loop poles ($\times$):** The roots of the denominator of $G(s)H(s)$. These are the starting points of the locus ($K = 0$).
- **Open-loop zeros ($\circ$):** The roots of the numerator of $G(s)H(s)$. These are the ending points of the locus ($K \to \infty$).
- **Branches:** Paths along which the closed-loop poles move. The number of branches is equal to the number of open-loop poles.

Significance of the Root Locus

The Root Locus plot is immensely significant in control system analysis and design for several reasons:

- **Stability Analysis:** The primary use of the Root Locus is to determine the stability of the closed-loop system. If any branch of the root locus crosses into the right-half of the s-plane (where real part of poles is positive), the system becomes unstable for the corresponding gain K. The value of K at which the locus crosses the imaginary axis gives the marginal gain for stability.
- **Performance Prediction:** The location of closed-loop poles in the s-plane directly correlates with the system's time-domain performance characteristics:
  - Poles further to the left (more negative real part) generally lead to faster response and quicker decay of transients.
  - Poles closer to the imaginary axis result in slower responses or oscillations.
  - Complex conjugate poles near the real axis but further from the origin (low damping ratio ζ) result in oscillatory responses with large overshoots.
  - Poles on the real axis typically lead to non-oscillatory (overdamped or critically damped) responses.

  - The damping ratio ζ and natural frequency ω_n can be graphically determined from the pole locations, which directly relate to overshoot, settling time, and peak time.

- **Gain Selection:** It helps engineers select an appropriate value of gain K to meet desired stability and performance specifications. By visually inspecting the plot, one can choose a K that places the closed-loop poles in desirable regions of the s-plane.
- **Compensator Design:** The Root Locus forms the basis for designing compensators (e.g., lead, lag, PID controllers). By adding poles and zeros through a compensator, the Root Locus is reshaped to achieve desired pole locations, thereby improving system performance.
- **Understanding System Behavior:** It provides an intuitive understanding of how changes in gain affect the system's dynamic behavior, including oscillations, damping, and speed of response.

How to Solve (Plot) in MATLAB

MATLAB's Control System Toolbox provides a dedicated function 'rlocus' for drawing root locus plots.

```
% Define the open-loop transfer function G(s)H(s)
% For a system G_ol(s) = N_ol(s)/D_ol(s), where N_ol(s) and D_ol(s) are polynomials in s
num_ol = [numerator_coefficients];   % e.g., [1 2] for (s+2)
den_ol = [denominator_coefficients]; % e.g., [1 3 2] for s^2+3s+2
sys_ol = tf(num_ol, den_ol);

% Plot the Root Locus
figure; % Create a new figure window
rlocus(sys_ol);
grid on; % Add a grid for better readability
title('Root Locus of G(s)H(s)'); % Add a descriptive title

% Optional: To find specific gain values or pole locations, you can click on the plot.
% Or use [r, k] = rlocus(sys_ol) to get all root loci and corresponding gains.
```

Listing 8: General MATLAB usage for Root Locus Plot

MATLAB Code for Problems

We will now provide the MATLAB code for each problem from Experiment No. 02, along with detailed explanations.

(a) $G(s)H(s) = \frac{3(s+2)(s+3)}{s(s+6)(s+8)(s+5)}$

Explanation: This is a standard open-loop transfer function. To plot the root locus in MATLAB, we first define the numerator and denominator polynomials of the transfer function. The 'tf' function is then used to create the transfer function object. Finally, the 'rlocus' function is called with this transfer function object to generate the root locus plot. The 'grid on' command adds a grid to the plot for better readability, and 'title' adds a descriptive title.

MATLAB Code:

```
% Define the numerator and denominator polynomials
num_a = 3 * conv([1 2], [1 3]); % 3*(s+2)*(s+3)
den_a = conv([1 0], conv([1 6], conv([1 8], [1 5]))); % s*(s+6)*(s+8)*(s+5)

% Create the transfer function
sys_a = tf(num_a, den_a);

% Plot the root locus
figure; % Create a new figure window
rlocus(sys_a);
grid on;
title('Root Locus for G(s)H(s) = 3(s+2)(s+3) / (s(s+6)(s+8)(s+5))');
```

Listing 9: MATLAB Code for System (a)

(b) $G(s)H(s) = \frac{4(s+3)}{s^3(s+1)(s+2)}$

Explanation: Similar to system (a), we define the numerator and denominator polynomials. Note that s^3 in the denominator means three poles at the origin, which is represented by the polynomial '[1 0 0 0]'. The 'conv' function is used to multiply the polynomial terms.

MATLAB Code:

```
% Define the numerator and denominator polynomials
num_b = 4 * [1 3]; % 4*(s+3)
den_b = conv([1 0 0 0], conv([1 1], [1 2])); % s^3*(s+1)*(s+2)

% Create the transfer function
sys_b = tf(num_b, den_b);

% Plot the root locus
figure;
rlocus(sys_b);
grid on;
title('Root Locus for G(s)H(s) = 4(s+3) / (s^3(s+1)(s+2))');
```

Listing 10: MATLAB Code for System (b)

(c) $G(s)H(s) = \frac{4s}{(s+1)(s+2)^2}$

Explanation: Here, the numerator is simply $4s$, represented by '[4 0]'. The term $(s+2)^2$ is calculated as 'conv([1 2], [1 2])'. The process remains the same: define numerator and denominator, create transfer function, and plot.

MATLAB Code:

```
% Define the numerator and denominator polynomials
num_c = [4 0]; % 4s
den_c = conv([1 1], conv([1 2], [1 2])); % (s+1)*(s+2)^2

% Create the transfer function
sys_c = tf(num_c, den_c);

% Plot the root locus
figure;
```

```
    rlocus(sys_c);
    grid on;
    title('Root Locus for G(s)H(s) = 4s / ((s+1)(s+2)^2)');

```

Listing 11: MATLAB Code for System (c)

(d) $G(s)H(s) = \frac{2(s+1)^2}{(s^2-9)(s+5)^2(s+2)}$

Explanation: The term $(s^2 - 9)$ can be represented directly as a polynomial '[1 0 -9]'. Alternatively, it can be factored as $(s - 3)(s + 3)$ and represented using 'conv([1 -3], [1 3])'. The term $(s + 1)^2$ and $(s + 5)^2$ are handled using 'conv'.

MATLAB Code:

```
    % Define the numerator and denominator polynomials
    num_d = 2 * conv([1 1], [1 1]); % 2*(s+1)^2
    den_d_s2_minus_9 = [1 0 -9]; % s^2 - 9
    den_d = conv(den_d_s2_minus_9, conv(conv([1 5], [1 5]), [1 2])); % (s
    ^2-9)*(s+5)^2*(s+2)

    % Create the transfer function
    sys_d = tf(num_d, den_d);

    % Plot the root locus
    figure;
    rlocus(sys_d);
    grid on;
    title('Root Locus for G(s)H(s) = 2(s+1)^2 / ((s^2-9)(s+5)^2(s+2))');

```

Listing 12: MATLAB Code for System (d)

(e) $\frac{C(s)}{R(s)} = \frac{s(s+2)}{1+(s^2+2s)(s+3)}$

Explanation: The 'rlocus' function plots the root locus of an open-loop transfer function $G(s)H(s)$ in the unity feedback configuration, where the closed-loop transfer function is $\frac{G(s)H(s)}{1+G(s)H(s)}$. Comparing the given closed-loop transfer function $\frac{s(s+2)}{1+(s^2+2s)(s+3)}$ to the standard form, it implies that the open-loop transfer function for which the root locus is to be plotted is $G(s)H(s) = (s^2 + 2s)(s + 3)$. We define this as our system and then plot its root locus.

MATLAB Code:

```
num = conv([1 2 0], [1 3]);
dem = [1];
sys = tf(num,dem);
figure;
rlocus(sys);
grid on;

```

Listing 13: MATLAB Code for System (e)

(f) State-Space Representation:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -1 & -0.5 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} [U]$$

$$y = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [0][U]$$

Explanation: To plot the root locus of a system given in state-space form, we first define the state-space matrices A, B, C, and D. Then, we can create a state-space system object using 'ss'. The 'rlocus' function can directly accept a state-space system object. It plots the locus of poles of $sI - (A - BKC)$, assuming output feedback or input gain K. More commonly, for open-loop root locus, we convert the state-space model to its equivalent transfer function $G(s) = C(sI - A)^{-1}B + D$ using 'ss2tf' or 'tf' function directly on the 'ss' object, and then use 'rlocus'. Here, we'll convert to transfer function first to explicitly show the open-loop system.

MATLAB Code:

```
% Define state-space matrices
A_f = [-1 -0.5; 1 0];
B_f = [0.5; 0];
C_f = [1 -1];
D_f = 0;

% Create the state-space system
sys_f_ss = ss(A_f, B_f, C_f, D_f);

% Convert the state-space system to a transfer function (open-loop G(s))
% This G(s) is typically what is used for root locus analysis when a
gain K is
% applied in feedback (e.g., in a unity feedback system).
[num_f, den_f] = ss2tf(A_f, B_f, C_f, D_f);
sys_f_tf = tf(num_f, den_f);

% Plot the root locus
figure;
rlocus(sys_f_tf);
grid on;
title('Root Locus for State-Space System (converted to TF)');

```

Listing 14: MATLAB Code for System (f)

(g) Block Diagram System

Explanation: For a block diagram, we first need to determine the overall open-loop transfer function $G(s)H(s)$ that corresponds to the characteristic equation $1 + KG(s)H(s) = 0$.

1. **Inner Loop Calculation:** The inner loop has a forward path $G_1(s) = \frac{3}{(s+2)(s+3)}$ and a feedback path $H_1(s) = 0.3s$. The closed-loop transfer function of this inner loop is $T_{inner}(s) = \frac{G_1(s)}{1+G_1(s)H_1(s)}$.

2. **Series Connection:** This $T_{inner}(s)$ is in series with another block $\frac{1}{2s}$. So, the effective forward path $G_{forward}(s) = T_{inner}(s) \cdot \frac{1}{2s}$.

3. **Overall Open-Loop TF:** The overall system has unity feedback (indicated by the summing junction at the input $R(s)$ with a minus sign from $C(s)$). So, the open-loop transfer function for the root locus plot is $G_{overall}(s) = G_{forward}(s)$.

We will perform these calculations in MATLAB using ‘tf‘ and ‘series‘ functions, and explicit arithmetic for the feedback loop.

Mathematical Derivation for (g): Inner loop: $G_1(s) = \frac{3}{(s+2)(s+3)}$, $H_1(s) = 0.3s$. $T_{inner}(s) = \frac{G_1(s)}{1+G_1(s)H_1(s)} = \frac{\frac{3}{(s+2)(s+3)}}{1+\frac{3}{(s+2)(s+3)}\cdot 0.3s} = \frac{3}{(s+2)(s+3)+0.9s}$ $T_{inner}(s) = \frac{3}{s^2+5s+6+0.9s} = \frac{3}{s^2+5.9s+6}$.

Forward path for outer loop: $G_{forward}(s) = T_{inner}(s) \cdot \frac{1}{2s}$ $G_{forward}(s) = \frac{3}{s^2+5.9s+6} \cdot \frac{1}{2s} = \frac{3}{2s(s^2+5.9s+6)}$.

This $G_{forward}(s)$ is the open-loop transfer function for which we plot the root locus, assuming an overall unity feedback with a variable gain K.

MATLAB Code:

```
num_1 = [3];
dem_1 =conv([1 0],conv([1 2] , [1 3]));
sys_1 =tf(num_1,dem_1);

num_2 = [0.2 0];
dem_2 = [ 1];
sys_2 =tf(num_2,dem_2);

sys_3 =feedback(sys_1,sys_2,-1);

num_4 =[1];
dem_4 =[2 0];
sys_4 = tf(num_4 , dem_4);
sys_5 = series(sys_3, sys_4);
sys_6 =parallel(sys_3,sys_4);

figure;
rlocus(sys_5);
grid on;

```

Listing 15: MATLAB Code for System (g)

Experiment No. 03: Nyquist Plot and Nyquist Stability Criterion

What is a Nyquist Plot?

A Nyquist plot is a parametric plot of a frequency response function $G(j\omega)$ or $L(j\omega) = G(j\omega)H(j\omega)$ for all frequencies from $\omega = -\infty$ to $\omega = +\infty$. It is plotted in the complex plane, where the horizontal axis represents the real part of $G(j\omega)$ and the vertical axis represents the imaginary part. Each point on the Nyquist plot corresponds to a specific frequency ω, where its distance from the origin represents the magnitude $|G(j\omega)|$ and its angle with the positive real axis represents the phase angle $\angle G(j\omega)$.

The Nyquist plot is constructed by mapping a special contour in the s-plane, called the Nyquist contour, to the $G(s)$ or $L(s)$ plane. The Nyquist contour typically encloses the entire right-half of the s-plane, including the imaginary axis from $-j\infty$ to $+j\infty$ and a large semicircle of infinite radius in the right-half plane. If there are poles of $L(s)$ on the imaginary axis (e.g., at $s = 0$), the contour must be indented around them to avoid passing through them.

Significance of the Nyquist Plot

The primary significance of the Nyquist plot lies in its ability to determine the stability of a closed-loop control system directly from its open-loop frequency response, without explicitly calculating the closed-loop poles. This is achieved through the Nyquist Stability Criterion.

Nyquist Stability Criterion

For a stable closed-loop system with unity feedback (or where the characteristic equation is $1 + L(s) = 0$), the Nyquist Stability Criterion states: $Z = N + P$ where:

- Z: Number of zeros of $1 + L(s)$ in the right-half s-plane (RHP). These are the unstable closed-loop poles. For stability, Z must be zero.
- N: Number of clockwise encirclements of the critical point $(-1, 0)$ by the Nyquist plot of $L(j\omega)$.
- P: Number of open-loop poles of $L(s)$ in the RHP.

For a stable closed-loop system, we require $Z = 0$. Therefore, the criterion becomes $N = -P$. If $P = 0$ (no open-loop poles in RHP), then N must be 0 (no encirclements of $(-1, 0)$).

Relative Stability

Beyond absolute stability, the Nyquist plot provides insights into relative stability through:

- **Gain Margin (GM):** The amount of gain that can be added to the system before it becomes unstable. On the Nyquist plot, it is the reciprocal of the magnitude of $L(j\omega)$ when its phase is -180°. If the Nyquist plot crosses the negative real axis at $-a$, then $GM = 1/|a|$.
- **Phase Margin (PM):** The additional phase lag required to make the system unstable at unity gain. It is the angle by which the Nyquist plot differs from -180° when its magnitude is unity (i.e., when it crosses the unit circle). $PM = 180^\circ + \phi_{gc}$, where ϕ_{gc} is the phase angle at the gain crossover frequency.

A larger GM and PM generally indicate a more robust and stable system.

How to Solve (Plot) in MATLAB

MATLAB's Control System Toolbox provides the 'nyquist' function, which is specifically designed for generating Nyquist plots.

```
% Define the transfer function
num = [numerator_coefficients]; % e.g., [10] for 10
den = [denominator_coefficients]; % e.g., [1 6 11 6] for s^3+6s^2+11s+6
sys = tf(num, den);

% Plot the Nyquist diagram
figure; % Create a new figure window
nyquist(sys);
grid on; % Add a grid for better readability
title('Nyquist Plot of G(s)'); % Add a title

```

Listing 16: General MATLAB usage for Nyquist Plot

MATLAB Code for Problems

We will now provide the MATLAB code for each problem, along with detailed explanations.

1. Unity Feedback Control System with Given Open-Loop Transfer Functions

(a) $G(s) = \frac{10}{(s+1)(s+2)(s+3)}$ **Explanation:** This is a straightforward transfer function. We define the numerator as '[10]' and the denominator polynomial is obtained by multiplying $(s+1)$, $(s+2)$, and $(s+3)$ using the 'conv' (convolution) function. 'conv([1 1], conv([1 2], [1 3]))' calculates $(s+1)(s+2)(s+3)$. Finally, 'tf' creates the transfer function object, and 'nyquist' plots it.

MATLAB Code:

```
% Define the numerator and denominator polynomials
num_1a = 10;
den_1a = conv([1 1], conv([1 2], [1 3])); % (s+1)(s+2)(s+3) = s^3 + 6s^2
+ 11s + 6

% Create the transfer function
G_1a = tf(num_1a, den_1a);

% Plot the Nyquist diagram
figure;
nyquist(G_1a);
grid on;
title('Nyquist Plot for G(s) = 10 / ((s+1)(s+2)(s+3))');

```

Listing 17: MATLAB Code for Problem 1(a)

(b) $G(s) = \frac{6(s+2)}{s^2(s+3)(s+4)}$ **Explanation:** Here, the denominator has s^2, which means two poles at the origin. This is represented by '[1 0 0]'. The 'conv' function is used again to multiply the polynomial terms for the denominator. Since there are poles at the origin, the Nyquist contour must be indented around the origin, which 'nyquist' function handles automatically.

MATLAB Code:

```
% Define the numerator and denominator polynomials
num_1b = 6 * [1 2]; % 6*(s+2)
den_1b = conv([1 0 0], conv([1 3], [1 4])); % s^2*(s+3)*(s+4) = s^4 + 7s^3 + 12s^2

% Create the transfer function
G_1b = tf(num_1b, den_1b);

% Plot the Nyquist diagram
figure;
nyquist(G_1b);
grid on;
title('Nyquist Plot for G(s) = 6(s+2) / (s^2(s+3)(s+4))');

```

Listing 18: MATLAB Code for Problem 1(b)

(c) $G(s) = \frac{10(s+1)}{s(s+6)(s+2)(s+5)}$ **Explanation:** Similar to the previous cases, we define the numerator and denominator polynomials. The 's' term in the denominator is '[1 0]'. The 'conv' function combines all the denominator terms.

MATLAB Code:

```
% Define the numerator and denominator polynomials
num_1c = 10 * [1 1]; % 10*(s+1)
den_1c = conv([1 0], conv([1 6], conv([1 2], [1 5]))); % s*(s+6)*(s+2)*(s+5)

% Create the transfer function
G_1c = tf(num_1c, den_1c);

% Plot the Nyquist diagram
figure;
nyquist(G_1c);
grid on;
title('Nyquist Plot for G(s) = 10(s+1) / (s(s+6)(s+2)(s+5))');

```

Listing 19: MATLAB Code for Problem 1(c)

2. Multivariable State-Space System

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 6.5 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Explanation: This is a Multi-Input Multi-Output (MIMO) system in state-space form. We need to plot four individual Nyquist plots, corresponding to each input-output transfer function: $G_{11}(s) = Y_1(s)/U_1(s)$, $G_{12}(s) = Y_1(s)/U_2(s)$, $G_{21}(s) = Y_2(s)/U_1(s)$, and $G_{22}(s) = Y_2(s)/U_2(s)$.

First, define the state-space matrices A, B, C, and D. Then, create an 'ss' object. To extract individual transfer functions, we can specify the input and output indices when using 'tf' or '$ss2tf$'. For example, '$tf(sys_ss, 1, 1)$' gives Y_1/U_1.

MATLAB Code:

```
% Define state-space matrices
A_2 = [-1 -1; 6.5 0];
B_2 = [1 1; 1 0];
C_2 = [1 0; 0 1];
D_2 = [0 0; 0 1];

% Create the state-space system
sys_2_ss = ss(A_2, B_2, C_2, D_2);


% Extract all transfer functions
G11 = tf(sys_2_ss(1,1));
G12 = tf(sys_2_ss(1,2));
G21 = tf(sys_2_ss(2,1));
G22 = tf(sys_2_ss(2,2));

% Plot all Nyquist diagrams in one figure
figure;
subplot(2,2,1); nyquist(G11); title('G11(s)'); axis equal; grid on;
subplot(2,2,2); nyquist(G12); title('G12(s)'); axis equal; grid on;
subplot(2,2,3); nyquist(G21); title('G21(s)'); axis equal; grid on;
subplot(2,2,4); nyquist(G22); title('G22(s)'); axis equal; grid on;

```

Listing 20: MATLAB Code for Problem 2

3. Nyquist Plot for Only Positive Frequency

(a) $G(s) = \frac{s^2+3s+1}{s^3+1.2s^2+2s+1}$ **Explanation:** The standard '$nyquist(sys)$' command plots the Nyquist curve for both positive and negative frequencies. To plot only for positive frequencies, we need to manually specify a frequency range 'omega' that spans from 0 to a sufficiently high value. We then use '$[re, im, omega_out] = nyquist(sys, omega)$' to get the real and imaginary parts of $G(j\omega)$ at these frequencies. Finally, we plot 'im' versus 're' using the 'plot' command.

MATLAB Code:

```
  num_3a = [1 3 1]; % s^2+3s+1
den_3a = [1 1.2 2 1]; % s^3+1.2s^2+2s+1
G_3a = tf(num_3a, den_3a);


omega_3a = logspace(-2, 2, 500); % Logarithmically spaced frequencies from
  10^-2 to 10^2

[re_3a, im_3a] = nyquist(G_3a, omega_3a);

re_3a = squeeze(re_3a);
im_3a = squeeze(im_3a);

% Plot only the positive frequency part
figure;
plot(re_3a, im_3a, 'b'); % 'b' for blue line

xlabel('Real Axis');
ylabel('Imaginary Axis');
grid on;

```

Listing 21: MATLAB Code for Problem 3(a)

(b) $G(s) = \frac{5(s+8)}{(s+4)(s+3)}$ **Explanation:** The approach is identical to 3(a). We define the transfer function and then generate the Nyquist data for a positive frequency range, plotting 'im' versus 're'.

MATLAB Code:

```
num = [5 40];
dem = conv([1 4] , [1 3]);
sys =tf(num,dem);
w = logspace(-2,2,1000);
[r,i] = nyquist(sys,w);
rs= squeeze(r);
is=squeeze(i);
figure;
plot(rs,-is,'b'); %negative freq
hold on
plot(rs,is,'r'); %positive freq
xlabel('real axis');
ylabel('imaginary aaxis');
axis equal;
grid on;

```

Listing 22: MATLAB Code for Problem 3(b)

4. Different Feedback Configurations

$G(s) = \frac{s^2+5s+4}{s(s^2+5s+2s+2)}$ **Clarification on Denominator:** Assuming $s^2 + 5s + 2s + 2$ simplifies to $s^2 + 7s + 2$. So, $G(s) = \frac{s^2+5s+4}{s(s^2+7s+2)}$.

Explanation: The Nyquist plot is always of the open-loop transfer function $L(s) = G(s)H(s)$. The difference in feedback configuration changes how the Nyquist Stability Criterion is applied (specifically, the critical point for encirclements).

(i) Unity Negative Feedback **Explanation:** This is the standard case. The characteristic equation is $1 + G(s) = 0$. We plot $G(s)$ and analyze encirclements of $(-1, 0)$.

MATLAB Code:

```
% Define the transfer function
num_4 = [1 5 4]; % s^2+5s+4
% Assuming s^2+5s+2s+2 simplifies to s^2+7s+2
den_4_poly = [1 7 2]; % s^2+7s+2
den_4 = conv([1 0], den_4_poly); % s*(s^2+7s+2) = s^3+7s^2+2s
G_4 = tf(num_4, den_4);

% Plot the Nyquist diagram for unity negative feedback
figure;
nyquist(G_4);
grid on;
title('Nyquist Plot for G(s) with Unity Negative Feedback');

```

Listing 23: MATLAB Code for Problem 4(i) (Unity Negative Feedback)

(ii) Unity Positive Feedback **Explanation:** For unity positive feedback, the closed-loop characteristic equation is $1 - G(s) = 0$. In this case, the critical point for stability analysis shifts from $(-1, 0)$ to $(+1, 0)$. We still plot the Nyquist diagram of $G(s)$, but we count encirclements of $(+1, 0)$ instead of $(-1, 0)$.

MATLAB Code:

```
% The open-loop transfer function remains G_4
% Plot the Nyquist diagram. The interpretation for stability changes
% to encirclements of (+1, 0)
figure;
nyquist(G_4);
grid on;
title('Nyquist Plot for G(s) with Unity Positive Feedback (Critical
Point is +1,0)');
% Optionally, you can mark the critical point for positive feedback
hold on;
plot(1, 0, 'rx', 'MarkerSize', 10, 'LineWidth', 2); % Mark (+1,0)
text(1.1, 0.1, 'Critical Point (+1,0)', 'Color', 'red');

```

Listing 24: MATLAB Code for Problem 4(ii) (Unity Positive Feedback)

(iii) Unity Positive-Negative System **Explanation:** The term "unity positive-negative system" is not a standard control system configuration. It might imply a system with a feedback path that introduces both positive and negative components, or a scenario where the gain K in $1 + KL(s) = 0$ could be negative. However, the standard Nyquist plot is of $L(s) = G(s)H(s)$. If $H(s)$ itself has terms that can be positive or negative, then $L(s)$ incorporates that. If this implies a general non-unity feedback system $L(s) = G(s)H(s)$ where $H(s)$ is a complex function, the procedure is still to plot $L(s)$. Without a specific definition for "unity positive-negative system," the most reasonable interpretation for plotting purposes remains plotting $G(s)$, as this is the $L(s)$ in a unity feedback context. The stability analysis would then depend on the specific interpretation of the "positive-negative" aspect, usually involving shifts of the critical point or modified encirclement rules based on the characteristic equation. Since the problem asks to consider the system "with the help of the open loop transfer function" $G(s)$, we will simply plot $G(s)$ again and note that the interpretation depends on the exact characteristic equation.

MATLAB Code:

```
% The term "unity positive-negative system" is ambiguous in standard
control theory.
% It typically refers to a characteristic equation different from 1+G(s)
=0 or 1-G(s)=0.
% However, the Nyquist plot itself is always of the open-loop transfer
function L(s).
% Assuming L(s) is still G(s) for the plot, but the stability
interpretation changes.

% Using the same G_4 as defined before
figure;
nyquist(G_4);
grid on;
title('Nyquist Plot for G(s) (Interpretation for "Positive-Negative
System" Varies)');
% Note: The critical point for stability analysis will depend on the
exact
% characteristic equation derived from the "positive-negative" feedback.
% Without a specific characteristic equation, we plot G(s) as the base.

```

Listing 25: MATLAB Code for Problem 4(iii) (Unity Positive-Negative System)

Experiment No. 04: Bode Diagram

What is a Bode Plot?

A Bode plot is a graphical representation of the frequency response of a linear time-invariant (LTI) system. It consists of two separate plots:

1. **Magnitude Plot:** A plot of the magnitude of the system's frequency response, $|G(j\omega)|$, typically in decibels (dB), versus the logarithm of the angular frequency ω (in rad/s). The magnitude in dB is calculated as $20 \log_{10} |G(j\omega)|$.

2. **Phase Plot:** A plot of the phase angle of the system's frequency response, $\angle G(j\omega)$, in degrees, versus the logarithm of the angular frequency ω.

Bode plots are fundamental tools in control systems engineering for analyzing and designing control systems.

Significance of the Bode Plot

The Bode plot offers several significant advantages and insights into system behavior:

- **Frequency Response Analysis:** It provides a clear picture of how a system responds to different input frequencies. This includes understanding gain at various frequencies (e.g., resonant peaks, attenuation at high frequencies) and phase shifts.

- **Stability Analysis:** The Bode plot is directly used to determine the stability of a closed-loop system from its open-loop frequency response. Key metrics for stability are:

  - **Gain Margin (GM):** The amount of gain (in dB) that can be added to the system before it becomes unstable. It is found at the phase crossover frequency (ω_{pc}), where the phase plot crosses -180°.
  - **Phase Margin (PM):** The additional phase lag (in degrees) required to make the system unstable at unity gain. It is found at the gain crossover frequency (ω_{gc}), where the magnitude plot crosses 0 dB.

  Positive gain and phase margins indicate a stable system. Larger margins generally imply more robust stability.

- **Controller Design:** Bode plots are extensively used in classical control design techniques (e.g., lead-lag compensators) to shape the frequency response of a system to meet desired performance specifications (e.g., bandwidth, settling time, overshoot).

- **System Identification:** By observing the slopes and breakpoints in a Bode plot, one can infer the order of the system, the presence of poles and zeros, and approximate their locations.

- **Ease of Plotting (Asymptotic Approximation):** For simple systems, approximate Bode plots can be sketched quickly using straight-line asymptotes, which aids in understanding the general behavior without complex calculations.

How to Solve (Plot) in MATLAB

MATLAB's Control System Toolbox provides the 'bode' function for generating Bode plots.

```
% Define the transfer function
% For a system G(s) = N(s)/D(s), where N(s) and D(s) are polynomials in
s
num = [numerator_coefficients];   % e.g., [1 2] for (s+2)
den = [denominator_coefficients]; % e.g., [1 3 2] for s^2+3s+2
sys = tf(num, den);

% Plot the Bode diagram
figure; % Create a new figure window
bode(sys);
grid on; % Add a grid for better readability
title('Bode Diagram of G(s)'); % Add a descriptive title

```

Listing 26: General MATLAB usage for Bode Plot

The 'bode' function automatically selects an appropriate frequency range. You can also specify a custom frequency range using 'bode(sys, omega)'.

MATLAB Code for Problems

We will now provide the MATLAB code for each problem, along with detailed explanations. For functions given as $G(j\omega)$, we will convert them to $G(s)$ by replacing $j\omega$ with s.

1. Draw the Bode Diagram of the Following System: ($G(j\omega)$)

(a) $G(j\omega) = \frac{1}{j\omega 4}$ Equivalent $G(s) = \frac{1}{4s}$ **Explanation:** This is a simple integrator with a gain. We define the numerator as 1 and the denominator as '[4 0]' (for $4s$). Then, create the transfer function and plot its Bode diagram.

MATLAB Code:

```
% Define the transfer function G(s) = 1/(4s)
num_1a = 1;
den_1a = [4 0]; % Represents 4s
G_1a = tf(num_1a, den_1a);

% Plot the Bode diagram
figure;
bode(G_1a);
grid on;
title('Bode Diagram for G(j\omega) = 1/(j\omega4)');

```

Listing 27: MATLAB Code for Problem 1(a)

(b) $G(j\omega) = j\omega 5$ Equivalent $G(s) = 5s$ **Explanation:** This represents a differentiator with a gain. The numerator is '[5 0]' (for $5s$) and the denominator is '[1]' (for 1).

MATLAB Code:

```
% Define the transfer function G(s) = 5s
num_1b = [5 0]; % Represents 5s
den_1b = 1;
G_1b = tf(num_1b, den_1b);

```

```
    % Plot the Bode diagram
    figure;
    bode(G_1b);
    grid on;
    title('Bode Diagram for G(j\omega) = j\omega5');

```

Listing 28: MATLAB Code for Problem 1(b)

(c) $G(j\omega) = (1 + j\omega 3)$ Equivalent $G(s) = (1 + 3s)$ **Explanation:** This is a lead term. The numerator is '[3 1]' (for $3s + 1$) and the denominator is '[1]'.

MATLAB Code:

```
    % Define the transfer function G(s) = (1+3s)
    num_1c = [3 1]; % Represents 3s + 1
    den_1c = 1;
    G_1c = tf(num_1c, den_1c);

    % Plot the Bode diagram
    figure;
    bode(G_1c);
    grid on;
    title('Bode Diagram for G(j\omega) = (1+j\omega3)');

```

Listing 29: MATLAB Code for Problem 1(c)

(d) $G(j\omega) = \frac{1}{3+j\omega 5}$ Equivalent $G(s) = \frac{1}{3+5s}$ **Explanation:** This is a lag term. The numerator is '[1]' and the denominator is '[5 3]' (for $5s + 3$).

MATLAB Code:

```
    % Define the transfer function G(s) = 1/(3+5s)
    num_1d = 1;
    den_1d = [5 3]; % Represents 5s + 3
    G_1d = tf(num_1d, den_1d);

    % Plot the Bode diagram
    figure;
    bode(G_1d);
    grid on;
    title('Bode Diagram for G(j\omega) = 1/(3+j\omega5)');

```

Listing 30: MATLAB Code for Problem 1(d)

(e) $G(j\omega) = \frac{1}{j\omega(1+j\omega 7)}$ Equivalent $G(s) = \frac{1}{s(1+7s)}$ **Explanation:** This system has an integrator and a lag term. The denominator is '$s * (7s + 1)$', which expands to '$7s^2 + s$'. We use 'conv' to multiply the polynomial terms.

MATLAB Code:

```
    % Define the transfer function G(s) = 1 / (s(1+7s))
    num_1e = 1;
    den_1e = conv([1 0], [7 1]); % s * (7s + 1) = 7s^2 + s
    G_1e = tf(num_1e, den_1e);

    % Plot the Bode diagram
    figure;
```

```
    bode(G_1e);
    grid on;
    title('Bode Diagram for G(j\omega) = 1/(j\omega(1+j\omega7))');

```

Listing 31: MATLAB Code for Problem 1(e)

(f) $G(j\omega) = \frac{1+j\omega 5}{j\omega(2+j\omega)(4+j2\omega)}$ Equivalent $G(s) = \frac{1+5s}{s(2+s)(4+2s)}$ **Explanation:** This is a more complex transfer function involving a zero, an integrator, and two poles. The numerator is '[5 1]' (for $5s+1$). The denominator is 's*(s+2)*(2s+4)'. We use 'conv' to multiply all denominator terms.

MATLAB Code:

```
    % Define the transfer function G(s) = (1+5s) / (s(2+s)(4+2s))
    num_1f = [5 1]; % Represents 5s + 1
    den_term_s = [1 0]; % s
    den_term_s_plus_2 = [1 2]; % s + 2
    den_term_2s_plus_4 = [2 4]; % 2s + 4
    den_1f = conv(den_term_s, conv(den_term_s_plus_2, den_term_2s_plus_4));
    % s * (s+2) * (2s+4)
    G_1f = tf(num_1f, den_1f);

    % Plot the Bode diagram
    figure;
    bode(G_1f);
    grid on;
    title('Bode Diagram for G(j\omega) = (1+j\omega5)/(j\omega(2+j\omega)(4+
    j2\omega))');

```

Listing 32: MATLAB Code for Problem 1(f)

2. Draw the Bode Diagram of the Following System: (Controllers)

(a) $G(s) = 1.1 + \frac{3}{s} + 2.5s$ **(For PID Controller)** **Explanation:** This is the standard form of a PID controller. To get a single transfer function, we combine the terms over a common denominator s: $G(s) = \frac{1.1s+3+2.5s^2}{s} = \frac{2.5s^2+1.1s+3}{s}$. The numerator polynomial is '[2.5 1.1 3]' and the denominator is '[1 0]'.

MATLAB Code:

```
    % Define the PID controller transfer function
    % G(s) = (2.5s^2 + 1.1s + 3) / s
    num_2a = [2.5 1.1 3];
    den_2a = [1 0]; % s
    G_2a = tf(num_2a, den_2a);

    % Plot the Bode diagram
    figure;
    bode(G_2a);
    grid on;
    title('Bode Diagram for PID Controller G(s) = 1.1 + 3/s + 2.5s');

```

Listing 33: MATLAB Code for Problem 2(a)

(b) $G(s) = 12(1 + \frac{6}{s})$ **(For PI Controller)** **Explanation:** This is a PI controller. Expand and simplify: $G(s) = 12 + \frac{72}{s} = \frac{12s+72}{s}$. The numerator polynomial is '[12 72]' and the denominator is '[1 0]'.

MATLAB Code:

```
% Define the PI controller transfer function
% G(s) = (12s + 72) / s
num_2b = [12 72];
den_2b = [1 0]; % s
G_2b = tf(num_2b, den_2b);

% Plot the Bode diagram
figure;
bode(G_2b);
grid on;
title('Bode Diagram for PI Controller G(s) = 12(1+6/s)');

```

Listing 34: MATLAB Code for Problem 2(b)

(c) $G(s) = 0.5(2 + 1.5s)$ **(For PD Controller)** **Explanation:** This is a PD controller. Expand: $G(s) = 1 + 0.75s = 0.75s + 1$. The numerator polynomial is '[0.75 1]' and the denominator is '[1]'.

MATLAB Code:

```
% Define the PD controller transfer function
% G(s) = 0.75s + 1
num_2c = [0.75 1];
den_2c = 1;
G_2c = tf(num_2c, den_2c);

% Plot the Bode diagram
figure;
bode(G_2c);
grid on;
title('Bode Diagram for PD Controller G(s) = 0.5(2+1.5s)');

```

Listing 35: MATLAB Code for Problem 2(c)

3. Draw the Bode Diagram of the Following System: (State-Space)

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Explanation: This is a Multi-Input Multi-Output (MIMO) system in state-space form. We need to plot four individual Bode diagrams, corresponding to each input-output transfer function: $G_{11}(s) = Y_1(s)/U_1(s)$, $G_{12}(s) = Y_1(s)/U_2(s)$, $G_{21}(s) = Y_2(s)/U_1(s)$, and $G_{22}(s) = Y_2(s)/U_2(s)$.

First, define the state-space matrices A, B, C, and D. Then, create an ''ss' object. To extract individual transfer functions, we use the 'tf' function on the 'ss' object, specifying the output index and input index. For example,'$tf(sys_ss, 1, 1)$' gives Y_1/U_1.

MATLAB Code:

```
  % Define state-space matrices
A_3 = [0 -1; -3 4];
B_3 = [1 1; 0 1];
C_3 = [1 0; 0 1];
D_3 = [0 0; 0 0.1];

sys_2_ss = ss(A_3 ,B_3 ,C_3 ,D_3);
figure;
bode(A_3 ,B_3 ,C_3 ,D_3);
grid on;

%this is for details
% Extract all transfer functions
G11 = tf(sys_2_ss(1,1));
G12 = tf(sys_2_ss(1,2));
G21 = tf(sys_2_ss(2,1));
G22 = tf(sys_2_ss(2,2));

% Plot all Nyquist diagrams in one figure
figure;
subplot(2,2,1); bode(G11); title('G11(s)');  grid on;
subplot(2,2,2); bode(G12); title('G12(s)');  grid on;
subplot(2,2,3); bode(G21); title('G21(s)');  grid on;
subplot(2,2,4); bode(G22); title('G22(s)');  grid on;
```

Listing 36: MATLAB Code for Problem 3