```matlab
% ME564 - HW9
% Q1

% Define system matrices
A = [2 1 1; 5 3 6; -5 -1 -4];
B = [1; 0; 0];
C = [1 1 2];

% (a) Identify the three modes of the system

% Calculate eigenvalues and eigenvectors
[eigVectors, eigValues] = eig(A);

% Extract the eigenvalues into a vector
eigValues = diag(eigValues);

% Display the eigenvalues and corresponding eigenvectors
disp('Eigenvalues:');
disp(eigValues);
disp('Eigenvectors:');
disp(eigVectors);

% Check for defective eigenvalues and display modes
for i = 1:length(eigValues)
    algebraicMultiplicity = sum(eigValues(i) == eigValues);
    geometricMultiplicity = rank(eigVectors(:, eigValues == eigValues(i)));

    if algebraicMultiplicity == geometricMultiplicity
        % Non-defective eigenvalue
        mode = eigVectors(:, i) * exp(eigValues(i) * 't');
        disp(['Mode for eigenvalue ', num2str(eigValues(i)), ':']);
        disp(mode);
    else
        % Defective eigenvalue, attempt to find generalized eigenvector
        genEigVector = null((A - eigValues(i) * eye(size(A)))^2, 'r');
        if ~isempty(genEigVector)
            % Generalized eigenvector found
            mode = genEigVector * exp(eigValues(i) * 't');
            disp(['Defective mode for eigenvalue ',
 num2str(eigValues(i)), ':']);
            disp(mode);
        else
            % Unable to find a generalized eigenvector
            disp(['Eigenvalue ', num2str(eigValues(i)), ' is defective, but
 unable to find a generalized eigenvector.']);
        end
    end
end

% (b) Identify whether each mode is controllable or uncontrollable

% Calculate the controllability matrix
```

```
n = size(A, 1);
controllabilityMatrix = B;
for i = 1:n-1
    controllabilityMatrix = [controllabilityMatrix, A^i * B];
end

% Check controllability for each mode
for i = 1:n
    eigVec = eigVectors(:, i);
    if rank([controllabilityMatrix, eigVec]) == rank(controllabilityMatrix)
        disp(['Mode corresponding to eigenvector ', num2str(i), ' is
 controllable.']);
    else
        disp(['Mode corresponding to eigenvector ', num2str(i), ' is
 uncontrollable.']);
    end
end

% (c) Identify whether each mode is observable or unobservable

% Calculate the observability matrix
observabilityMatrix = C;
for i = 1:n-1
    observabilityMatrix = [observabilityMatrix; C * A^i];
end

% Check observability for each mode
for i = 1:n
    eigVec = eigVectors(:, i);
    if rank([observabilityMatrix; eigVec']) == rank(observabilityMatrix)
        disp(['Mode corresponding to eigenvector ', num2str(i), ' is
 observable.']);
    else
        disp(['Mode corresponding to eigenvector ', num2str(i), ' is
 unobservable.']);
    end
end

% (d) Transfer Function Y(s)/U(s) Confirmation

% Define the symbolic variable s
syms s

% Compute the transfer function
transferFunction = C * inv(s * eye(size(A)) - A) * B;

% Simplify and display the transfer function
transferFunction = simplify(transferFunction);
disp('Transfer Function Y(s)/U(s):');
disp(transferFunction);

% (e) Transform the system into modal form

% Find the Jordan normal form J of matrix A and the change of basis matrix V
```

```
[V, J] = jordan(A);

% Compute the transformed B and C matrices
B_bar = inv(V) * B;
C_bar = C * V;

% Display the results
disp('Jordan normal form J:');
disp(J);
disp('Change of basis matrix V:');
disp(V);
disp('Transformed B matrix (B_bar):');
disp(B_bar);
disp('Transformed C matrix (C_bar):');
disp(C_bar);
```

```
Eigenvalues:
   -3.0000
    2.0000
    2.0000

Eigenvectors:
         0   -0.5774   -0.5774
   -0.7071   -0.5774   -0.5774
    0.7071    0.5774    0.5774

Mode for eigenvalue -3:
  1.0e-151 *

         0
   -0.5188
    0.5188

Defective mode for eigenvalue 2:
  1.0e+100 *

         0   -5.7058
    5.7058         0
         0    5.7058

Defective mode for eigenvalue 2:
  1.0e+100 *

         0   -5.7058
    5.7058         0
         0    5.7058

Mode corresponding to eigenvector 1 is controllable.
Mode corresponding to eigenvector 2 is controllable.
Mode corresponding to eigenvector 3 is controllable.
Mode corresponding to eigenvector 1 is unobservable.
Mode corresponding to eigenvector 2 is unobservable.
Mode corresponding to eigenvector 3 is unobservable.
Transfer Function Y(s)/U(s):
```

```
1/(s + 3)

Jordan normal form J:
    -3     0     0
     0     2     1
     0     0     2

Change of basis matrix V:
         0    1.0000   -1.2000
    0.2000    1.0000   -0.2000
   -0.2000   -1.0000    1.2000

Transformed B matrix (B_bar):
    -5
     1
     0

Transformed C matrix (C_bar):
   -0.2000         0    1.0000
```

Published with MATLAB® R2022b

```
% ME564 HW9
% q2

% Transfer function G(s)
numerator = [1 3];
denominator = conv([1 2], [1 1]);
G = tf(numerator, denominator);

% Function to display matrices
displayMatrices = @(A, B, C, D, label) fprintf('%s:\nMatrix A:\n%s\n\nMatrix
 B:\n%s\n\nMatrix C:\n%s\n\nMatrix D:\n%s\n\n', label, mat2str(A), mat2str(B),
 mat2str(C), mat2str(D));

% State-space representation
[A, B, C, D] = ssdata(ss(G));

% a - controllable canonical realization
displayMatrices(A, B, C, D, 'Controllable Canonical Realization');

% b - observable canonical realization
n = size(A, 1); % Number of states
Ao = [A; eye(n-1) zeros(n-1, 1)]; % Extend A matrix
Bo = [zeros(1, n-1) 1]'; % Extend B matrix
Co = C; % C matrix remains the same
displayMatrices(Ao, Bo, Co, D, 'Observable Canonical Realization');

% c - modal realization
Am = diag(eig(A)); % Diagonal matrix with eigenvalues of A
Bm = B; % B matrix remains the same
Cm = C; % C matrix remains the same
displayMatrices(Am, Bm, Cm, D, 'Modal Realization');

% d - non-minimal realization
Anm = [A, zeros(size(A, 1), 1); zeros(1, size(A, 2)), -1]; % Extend A to 4x4
Bnm = [B; 0]; % Extend B to 4x1
Cnm = [C, zeros(1, size(C, 2))]; % Extend C to 1x4
displayMatrices(Anm, Bnm, Cnm, D, 'Non-minimal Realization');
```

```
Controllable Canonical Realization:
Matrix A:
[-3 -2;1 0]

Matrix B:
[2;0]

Matrix C:
[0.5 1.5]

Matrix D:
0

Observable Canonical Realization:
```

```
Matrix A:
[-3 -2;1 0;1 0]

Matrix B:
[0;1]

Matrix C:
[0.5 1.5]

Matrix D:
0

Modal Realization:
Matrix A:
[-2 0;0 -1]

Matrix B:
[2;0]

Matrix C:
[0.5 1.5]

Matrix D:
0

Non-minimal Realization:
Matrix A:
[-3 -2 0;1 0 0;0 0 -1]

Matrix B:
[2;0;0]

Matrix C:
[0.5 1.5 0 0]

Matrix D:
0
```

Published with MATLAB® R2022b

```
% ME 564 HW 9
% q3

% a
% 1: Find Eigenvalues and Eigenvectors
A = [-2 -1; 0 -3];
[V, D] = eig(A);  % V: eigenvectors, D: eigenvalues as a diagonal matrix

disp('Eigenvalues:');
disp(diag(D));

disp('Eigenvectors:');
disp(V);

% 2: Form the Transformation Matrix V
% V is already obtained from eig function

% 3: Transform the System to Modal Form
A_modal = inv(V) * A * V;
B_modal = inv(V) * [2; 1];

disp('Modal Form Matrices:');
disp('A_modal:');
disp(A_modal);

disp('B_modal:');
disp(B_modal);


% b
% Given matrices
A_modal = [-2 0; 0 -3];
B_modal = [2; 1];

% Time vector
% assume:
t = linspace(0, 5, 100);

% Define the input function u(t)
% assume input function is:
u = @(t) sin(t);

% Compute Lu(t) - Zero-state response
Lu = zeros(size(B_modal, 1), length(t));
for i = 1:length(t)
    Lu(:, i) = integral(@(tau) expm(A_modal*(t(i)-tau))*B_modal*u(tau), 0,
 t(i), 'ArrayValued', true);
end

% Compute Lo(t) - Zero-input response with x(0) = 0
Lo = zeros(size(A_modal, 1), length(t));
for i = 1:length(t)
```

```
    Lo(:, i) = expm(A_modal*t(i)) * zeros(size(A_modal, 1), 1);
end

% Display the results
figure;

subplot(2, 1, 1);
plot(t, Lu(1, :), 'r', t, Lu(2, :), 'b', 'LineWidth', 2);
title('Zero-State Response (Lu)');
xlabel('Time');
ylabel('State Variables');
legend('Lu (z1)', 'Lu (z2)');

subplot(2, 1, 2);
plot(t, Lo(1, :), 'r', t, Lo(2, :), 'b', 'LineWidth', 2);
title('Zero-Input Response (Lo)');
xlabel('Time');
ylabel('State Variables');
legend('Lo (z1)', 'Lo (z2)');


% c

% Given matrices
A = [-2 -1; 0 -3];
B = [2 1];
C = [1; -1]; % Make C a column vector for proper transpose

% Step 1: Find Adjoint Matrix A*
A_star = conj(A.');

u = @(t) sin(t);


% Step 2: Compute Pu(t) - Zero-state response for adjoint system
Pu = zeros(size(A_star, 1), length(t));
for i = 1:length(t)
    % Define the matrix exponential function
    expAt = @(t) expm(A_star * t);

    % Define the system dynamics for the integral
    % dynamics = @(tau, z) expAt(tau) * C.' * u(tau);

    % expAt_t = expAt(t(1));
    % C_t = C.' * u(t(1)).';
    % disp(size(expAt_t));
    % disp(size(C_t));

    dynamics = @(tau, z) expAt(tau) * (C.' * u(tau)).';

    % Initialize state variable for Pu
    z0 = zeros(size(A_star, 2), 1);

    % Integrate the system using a loop
```

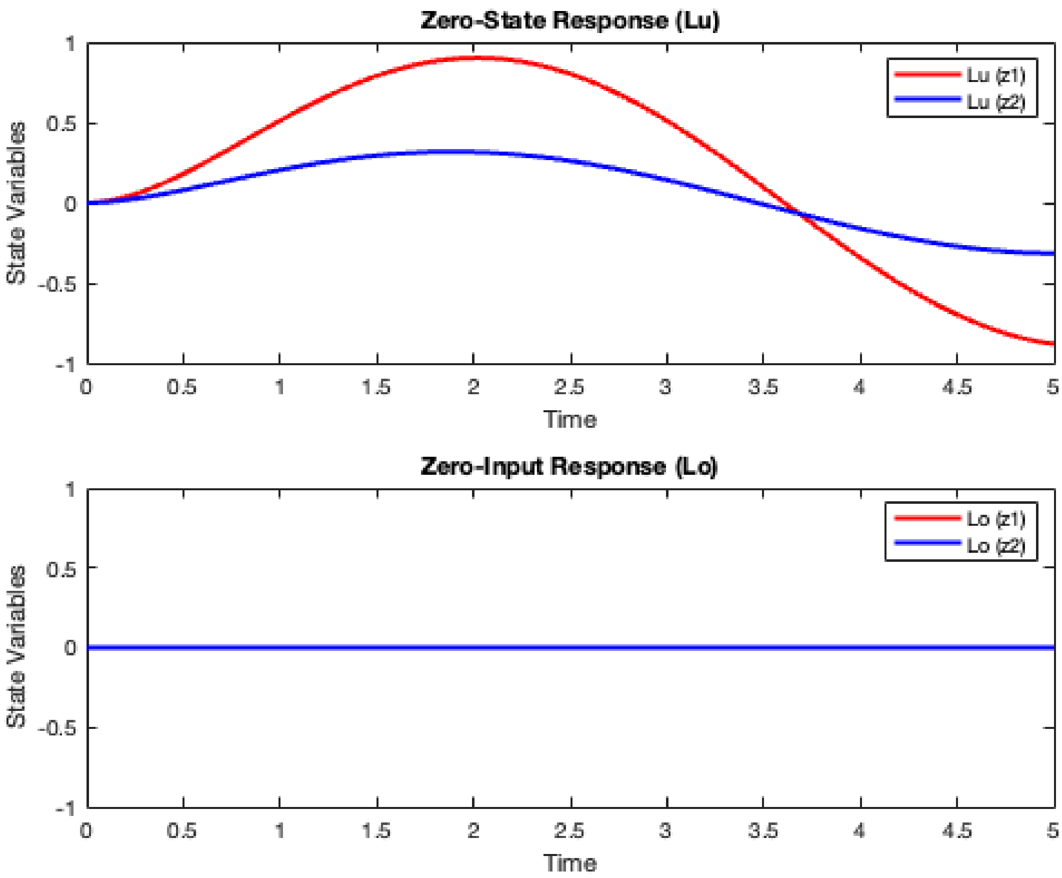
```
    for j = 1:length(t)
        [~, z] = ode45(@(tau, z) dynamics(tau, z), [eps t(j)], z0); % Start
 from a small positive value
        z0 = z(end, :).';  % Update initial condition for the next step
    end

    % Store the final state for Pu
    Pu(:, i) = z0;
end

% Step 3: Compute Po(t) - Zero-input response for adjoint system
Po = zeros(size(A_star, 1), length(t));
x0 = [0; 0]; % Initial conditions for the adjoint system
for i = 1:length(t)
    % Initialize state variable for Po
    z0 = x0;

    % Integrate the system using a loop
    for j = 1:length(t)
        [~, z] = ode45(@(tau, z) A_star * z, [eps t(j)], z0); % Start from a
 small positive value
        z0 = z(end, :).';  % Update initial condition for the next step
    end

    % Store the final state for Po
    Po(:, i) = z0;
end

% Display the results
figure;

subplot(2, 1, 1);
plot(t, Pu, 'k', 'LineWidth', 2);
title('Zero-State Response (Pu) - Adjoint System');
xlabel('Time');
ylabel('State Variable');
legend('Pu');

subplot(2, 1, 2);
plot(t, Po, 'k', 'LineWidth', 2);
title('Zero-Input Response (Po) - Adjoint System');
xlabel('Time');
ylabel('State Variable');
legend('Po');

% d

% Calculate the adjoint matrices for Lo_star
A_star_lo = conj(A_modal.');  % Adjoint of A_modal
B_star_lo = V * B_modal;       % Adjoint of B_modal
C_star_lo = C.' * inv(V);      % Adjoint of C

% Time vector for response comparison
t_compare = linspace(0, 5, 100);
```

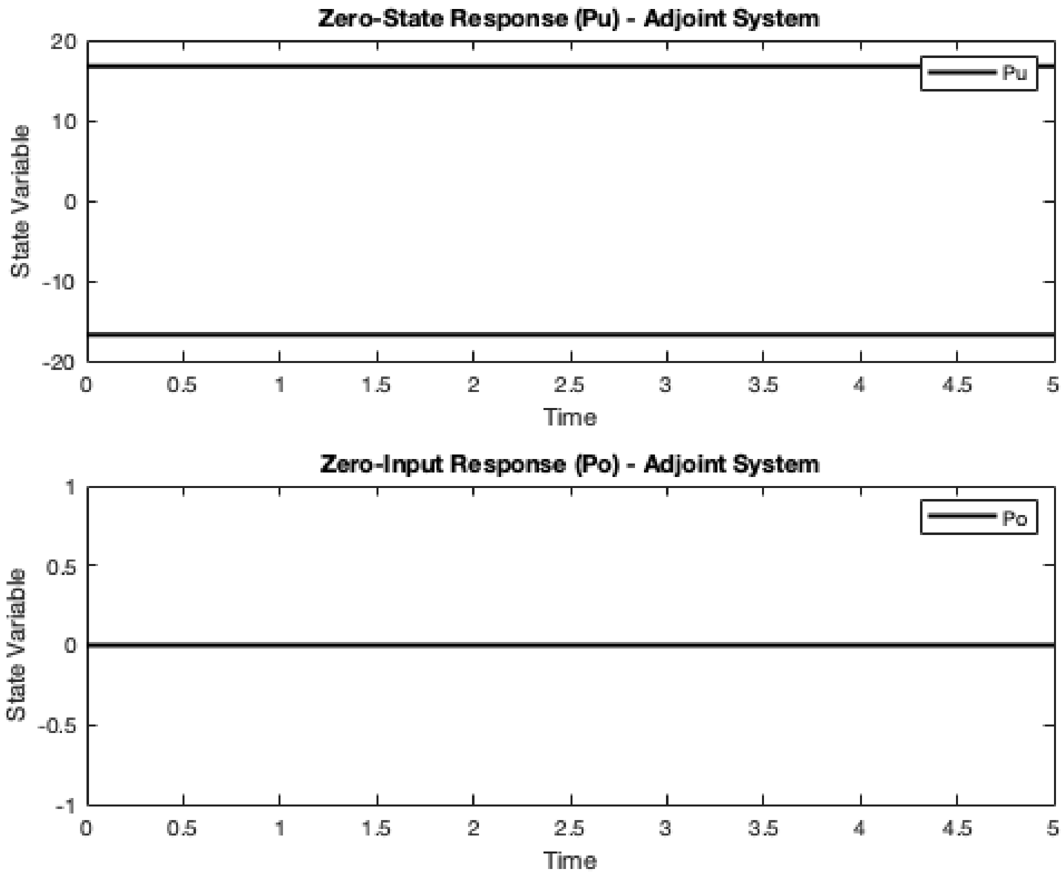
```
% Compute Lo_star(t) - Zero-input response for adjoint system
Lo_star = zeros(size(A_star_lo, 1), length(t_compare));
x0_lo_star = [0; 0]; % Initial conditions for Lo_star
for i = 1:length(t_compare)
    % Initialize state variable for Lo_star
    z0_lo_star = x0_lo_star;

    % Integrate the system using a loop
    for j = 1:length(t_compare)
        [~, z_lo_star] = ode45(@(tau, z) A_star_lo * z, [eps t_compare(j)],
 z0_lo_star); % Start from a small positive value
        z0_lo_star = z_lo_star(end, :).';  % Update initial condition for the
 next step
    end

    % Store the final state for Lo_star
    Lo_star(:, i) = z0_lo_star;
end

% Compare Lo_star with Pu
figure;

subplot(2, 1, 1);
plot(t_compare, Pu, 'k', 'LineWidth', 2);
title('Zero-State Response (Pu)');
xlabel('Time');
ylabel('State Variable');
legend('Pu');

subplot(2, 1, 2);
plot(t_compare, Lo_star, 'k--', 'LineWidth', 2);
title('Zero-Input Response (Lo_star)');
xlabel('Time');
ylabel('State Variable');
legend('Lo\_star');

% Display legend for comparison
legend({'Pu', 'Lo\_star'});


% % e
% % Given matrices
% A = [-2 -1; 0 -3];
% B = [2 1];
% C = [1; -1]; % Make C a column vector for proper transpose
%
% % Time vector for response comparison
% t_compare = linspace(0, 5, 100);
%
% % Define the input function u(t)
% u = @(t) sin(t);
%
% % Step 1: Find Adjoint Matrix A*
```

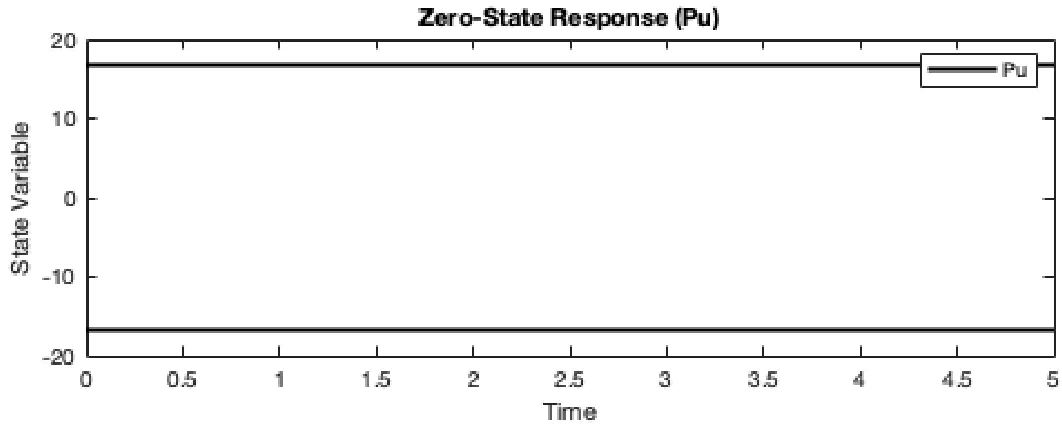
```
% A_star = conj(A.');
%
% % Step 2: Compute Lu_star(t) - Adjoint response for the adjoint system
% Lu_star = zeros(size(A_star, 1), length(t_compare));
% for i = 1:length(t_compare)
%     % Define the matrix exponential function
%     expAt_star = @(t) expm(A_star.' * t);
%
%     % Define the system dynamics for the integral
%     dynamics_star = @(tau, z) expAt_star(t_compare(i) - tau) * (C.' *
 u(tau));
%
%     % Initialize state variable for Lu_star
%     z0_star = zeros(size(A_star, 2), 1);
%
%     % Integrate the system using a loop
%     for j = 1:length(t_compare)
%         [~, z_star] = ode45(@(tau, z) dynamics_star(tau, z), [eps
 t_compare(j)], z0_star); % Start from a small positive value
%         z0_star = z_star(end, :).';  % Update initial condition for the next
 step
%     end
%
%     % Store the final state for Lu_star
%     Lu_star(:, i) = z0_star;
% end
%
% % Compare Lu_star with Po
% figure;
%
% subplot(2, 1, 1);
% plot(t_compare, Po, 'k', 'LineWidth', 2);
% title('Zero-Input Response (Po)');
% xlabel('Time');
% ylabel('State Variable');
% legend('Po');
%
% subplot(2, 1, 2);
% plot(t_compare, Lu_star, 'k--', 'LineWidth', 2);
% title('Adjoint Response (Lu_star)');
% xlabel('Time');
% ylabel('State Variable');
% legend('Lu\_star');
%
% % Display legend for comparison
% legend({'Po', 'Lu\_star'});
```

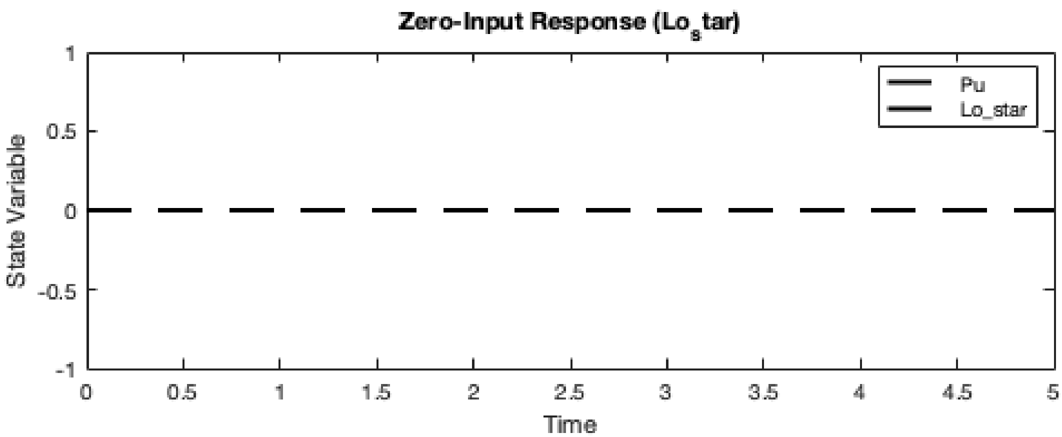
```
Eigenvalues:
    -2
    -3

Eigenvectors:
    1.0000    0.7071
         0    0.7071
```

```
Modal Form Matrices:
A_modal:
    -2     0
     0    -3

B_modal:
    1.0000
    1.4142
```




Zero-State Response (Pu) - Adjoint System
State Variable
Time
Pu
20
10
0
-10
-20
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Zero-Input Response (Po) - Adjoint System
State Variable
Time
Po
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5





Published with MATLAB® R2022b

```
% ME564 HW9
% q4

% Given system matrices
A = [3 3 0 2; 0 87 0 60; 6 3 -3 2; 0 126 0 -87];
B = [0; 3; -1; 4];

% Controllability matrix
Co = ctrb(A, B);

% Check if the system is controllable
if rank(Co) == size(A, 1)
    % Perform Kalman decomposition
    Ac = A;
    Bc = B;
    Cc = eye(size(A));
    Tu = eye(size(A));
    Au = zeros(size(A));

    % Display the controllable and uncontrollable parts
    disp('Controllable Part:');
    disp(Ac);
    disp('Uncontrollable Part:');
    disp(Au);
else
    disp('System is not controllable');
end
```

```
Controllable Part:
     3     3     0     2
     0    87     0    60
     6     3    -3     2
     0   126     0   -87

Uncontrollable Part:
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Published with MATLAB® R2022b

```
% ME 564 HW9
% q5

% a
% Given system matrices
A = [0 1; -1 -2];
B = [1 0; 0 1];

% Desired eigenvalues
desired_eigenvalues = [0 0];

% Find state feedback matrix K using the place function
K = place(A, B, desired_eigenvalues);

% Display the state feedback matrix K
disp('State Feedback Matrix K:');
disp(K);

% Explanation:
% - desired_eigenvalues specifies the desired closed-loop evalues.
%   Here, we want both evalues to be at 0.
% - The place function calculates the state feedback matrix K s.t.
%   the closed-loop eigenvalues of A + BK match the desired_eigenvalues.
% - K is the state feedback matrix that satisfies the desired closed-loop
%   eigenvalues and is used in the state-feedback control law u(k) = Kx(k).

% b
% Extract the first row of B
B1 = B(1, :);

% Solve for the state feedback matrix K directly
K = acker(A, B1', desired_eigenvalues);

% Display the state feedback matrix K
disp('State Feedback Matrix K (for u1 only):');
disp(K);

% Explanation:
% - B1 is the first row of B, corresponding to the first input.
% - The acker function calculates the state feedback matrix K s.t.
%   the closed-loop eigenvalues of A + B1*K match the desired_eigenvalues.
% - K is the state feedback matrix for u1(k) = Kx(k) using only the first
 input.

% c
% Extract the second row of B
B2 = B(2, :);

% Solve for the state feedback matrix K directly for u2
K = acker(A, B2', desired_eigenvalues);

% Display the state feedback matrix K (for u2 only)
```

```
disp('State Feedback Matrix K (for u2 only):');
disp(K);
```

```
State Feedback Matrix K:
     0     1
    -1    -2

State Feedback Matrix K (for u1 only):
    -2    -3

State Feedback Matrix K (for u2 only):
    -1    -2
```

Published with MATLAB® R2022b

Q6. $\dot{x} = Ax + Bu$ —— (1)

$y = Cx$ —— (2)

$u = -Kx + v$ —— (3)

a) CLS; $\bar{A}, \bar{B}$ s.t. $\dot{x} = \bar{A}x + \bar{B}v$ (by plug. (3) into (1))

(3) into (1) : $\dot{x} = Ax + B(-Kx + v)$

$= (A - BK)x + Bv$

$\Updownarrow$ same form w/

$= \bar{A}x + \bar{B}v$

where $\left.\begin{array}{l} \bar{A} \triangleq A - BK \\ \bar{B} \triangleq B \end{array}\right\}$ (4)

b) show if the original sys (A, B) is controllable, then $(\bar{A}, \bar{B})$ is also controllable for any K.

(A, B) controllable $\rightarrow$ $(\bar{A}, \bar{B})$ controllable (?)

cont. of CLS : $\bar{\mathcal{C}}_o = [\bar{B} \;\; \bar{A}\bar{B} \ldots \bar{A}^{n-1}\bar{B}]$

if (A, B) controllable $\rightarrow$ $\mathcal{C}_o$ must be full rank

using (4): $\bar{\mathcal{C}}_o = [B \;\; (A - BK)B \;\ldots\; (A - BK)^{n-1}B]$

$= [B, AB - BKB, A^2B - ABK - BKB^2, \ldots\ldots]$

$= \underbrace{[B \;\; AB \;\; A^2B \ldots]}_{\text{original full-rank sys.}} - \underbrace{[BK, BKA \ldots BK(A^{n-2}B)]}_{\text{subtracting a lin. combo of its cols. won't change the rank.}}$

$\Rightarrow$ $\bar{\mathcal{C}}_o$ has full rank. (?) ✓

c) $A \in \mathbb{R}^{2\times 2}$. Find ex. for (A, C) observable, $(\bar{A}, C)$ is not.

Obs (A, C) $\rightarrow$ $\mathcal{O} = \begin{bmatrix} C \\ CA \end{bmatrix}$

let's say $\left.\begin{array}{l} A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \\ C = [1 \;\; 0] \end{array}\right\}$ $\mathcal{O} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ → full rank

$B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$\left.\begin{array}{l} \bar{A} = A - BK = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \bar{C} = C \end{array}\right\}$ $\bar{\mathcal{O}} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ → not full rank

so (A, C) observable, $(\bar{A}, C)$ not.

Office of the Registrar - Evaluations

Inbox - Google 27 November 2023

Thank you for providing feedback for Fall 2023 Teaching Evaluations

To: Rabia Konuk,

Reply-To: ro.evaluations@umich.edu

Dear Rabia,

Your feedback for MECHENG 564-001: Lin Systems Theory (Combined Section) was submitted.

Thank you,
Office of the Registrar Evaluations team