## Table of Contents



# Clean workspace

```
clear all; clf; close all; clc;
```

# Data Preparation

```
% Section 1 - Load Data:
% Load the data provided for each input channel.

% Sampling parameters
ts = 1/50; % Sample period in seconds
fs = 1/ts; % Sampling frequency in Hz

% Load data for u1
load('random_u1.mat'); % Contains u1, y1, y2
u11 = u1;
y11 = y1; % Output y1 due to input u1
y21 = y2; % Output y2 due to input u1

% Load data for u2
load('random_u2.mat'); % Contains u2, y1, y2
u22 = u2;
y12 = y1; % Output y1 due to input u2
y22 = y2; % Output y2 due to input u2

% Load data for u3
load('random_u3.mat'); % Contains u3, y1, y2
u33 = u3;
y13 = y1; % Output y1 due to input u3
y23 = y2; % Output y2 due to input u3

% Time vector
Ndat = length(u1); % Assuming all datasets have the same length
t = (0:Ndat-1) * ts;
ax = [0 10 -6 6];

% Section 2 -  Code for Plotting:
% Plot the input and output signals to get an initial understanding of the
```

```
system's behavior.

% % Plot for u1
% figure;
% subplot(3,1,1);
% plot(t, u1);
% title('Input u_1');
% xlabel('Time (s)');
% ylabel('Amplitude');
% grid on;
%
% subplot(3,1,2);
% plot(t, y11);
% title('Output y_1 due to u_1');
% xlabel('Time (s)');
% ylabel('Amplitude');
% grid on;
%
% subplot(3,1,3);
% plot(t, y21);
% title('Output y_2 due to u_1');
% xlabel('Time (s)');
% ylabel('Amplitude');
% grid on;

figure(1)
subplot(311)
plot(t,u11)
title('u_1 input')
grid on; axis(ax); legend('u_1')
subplot(312)
plot(t,y11)
grid on; axis(ax); legend('y_1')
subplot(313)
plot(t,y21)

grid on; axis(ax); legend('y_2')
xlabel('Time (s)')
figure(2)
subplot(311)
plot(t,u22)
title('u_2 input')
grid on; axis(ax); legend('u_2')
subplot(312)
plot(t,y12)
grid on; axis(ax); legend('y_1')
subplot(313)
plot(t,y22)
grid on; axis(ax); legend('y_2')
xlabel('Time (s)')
figure(3)
subplot(311)
plot(t,u33)
title('u_3 input')
```

```
grid on; axis(ax); legend('u_3')
subplot(312)
plot(t,y13)
grid on; axis(ax); legend('y_1')
subplot(313)
plot(t,y23)
grid on; axis(ax); legend('y_2')
xlabel('Time (s)')
```

# Task 1 - Empirical Frequency Response Estimates

```
% Section 1 - Compute Spectra Using cpsd:

% Parameters for cpsd
nfft = 250; % Number of FFT points
window = hamming(nfft);
noverlap = []; % Default overlap

% Compute auto-spectra
[Suu1, f] = cpsd(u1, u1, window, noverlap, nfft, fs, 'twosided');
[Suu2, ~] = cpsd(u2, u2, window, noverlap, nfft, fs, 'twosided');
[Suu3, ~] = cpsd(u3, u3, window, noverlap, nfft, fs, 'twosided');

Suu1_average = mean(abs(Suu1));
Suu1_variance = var(abs(Suu1));
Suu2_average = mean(abs(Suu2));
Suu2_variance = var(abs(Suu2));
Suu3_average = mean(abs(Suu3));
Suu3_variance = var(abs(Suu3));

% Compute cross-spectra
[Su1u2, ~] = cpsd(u1, u2, window, noverlap, nfft, fs, 'twosided');
[Su1u3, ~] = cpsd(u1, u3, window, noverlap, nfft, fs, 'twosided');
[Su2u3, ~] = cpsd(u2, u3, window, noverlap, nfft, fs, 'twosided');

Su1u2_average = mean(abs(Su1u2));
Su1u3_average = mean(abs(Su1u3));
Su2u3_average = mean(abs(Su2u3));
Su1u2_variance = var(abs(Su1u2));
Su1u3_variance = var(abs(Su1u3));
Su2u3_variance = var(abs(Su2u3));
% Section 2 - Plotting Auto-Spectra:

% Plot auto-spectra
% figure("Position",[100,200,1500,500]);
figure;
% tiledlayout(1,2)
% nexttile;
loglog(f, abs(Suu1), 'r', f, abs(Suu2), 'g', f, abs(Suu3), 'b');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

```
title('Auto-Spectral Densities of Inputs');
legend('S_{u_1u_1}', 'S_{u_2u_2}', 'S_{u_3u_3}',Location='southwest');
axis([0.1 fs/2 1e-5 1e-2]);
grid on;


% Section 3 - Plotting Cross-Spectra:
% Plot cross-spectra
% nexttile
figure;
loglog(f, abs(Su1u2), 'r', f, abs(Su1u3), 'g', f, abs(Su2u3), 'b');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Cross-Spectral Densities between Inputs');
legend('S_{u_1u_2}', 'S_{u_1u_3}', 'S_{u_2u_3}',Location='southwest');
axis([0.1 fs/2 1e-5 1e-2]);
grid on;
```

# Section 4 - Calculating Mean Square Values:

```
% Compute the variance (mean square value) of each input signal in the time
domain:
var_u1 = mean(u1.^2);
var_u2 = mean(u2.^2);
var_u3 = mean(u3.^2);

% Compute the mean of the auto-spectral densities and multiply by the
sampling frequency:
mean_Suu1 = mean(abs(Suu1)) * fs;
mean_Suu2 = mean(abs(Suu2)) * fs;
mean_Suu3 = mean(abs(Suu3)) * fs;

var_table = array2table([var_u1',var_u2',var_u3']);
mean_table = array2table([mean_Suu1',mean_Suu2',mean_Suu3']);
```

# Section 5 - Estimating Frequency Responses

```
% For each input-output pair, compute the frequency response:

% For input u1
[Sy1u1, ~] = cpsd(y11, u1, window, noverlap, nfft, fs, 'twosided');
[Sy2u1, ~] = cpsd(y21, u1, window, noverlap, nfft, fs, 'twosided');

H11 = Sy1u1 ./ Suu1;
H21 = Sy2u1 ./ Suu1;

% For input u2
[Sy1u2, ~] = cpsd(y12, u2, window, noverlap, nfft, fs, 'twosided');
[Sy2u2, ~] = cpsd(y22, u2, window, noverlap, nfft, fs, 'twosided');

H12 = Sy1u2 ./ Suu2;
H22 = Sy2u2 ./ Suu2;
```

```matlab
% For input u3
[Sy1u3, ~] = cpsd(y13, u3, window, noverlap, nfft, fs, 'twosided');
[Sy2u3, ~] = cpsd(y23, u3, window, noverlap, nfft, fs, 'twosided');

H13 = Sy1u3 ./ Suu3;
H23 = Sy2u3 ./ Suu3;


% Section 6 - Plotting Frequency Responses

% Create the figures directory if it doesn't exist
if ~exist('figures', 'dir')
    mkdir('figures');
end

% Magnitude for H11 and H21
figure;
loglog(f, abs(H11), 'r', f, abs(H21), 'b');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Response Magnitudes for Input u_1');
legend('H_{11}', 'H_{21}',Location='southwest');
axis([0.1 fs/2 1e-3 1e2]);
grid on;
saveas(gcf, fullfile('figures', 'T1S5_Freq_u_1.png'));

% Magnitude for H12 and H22
figure;
loglog(f, abs(H12), 'r', f, abs(H22), 'b');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Response Magnitudes for Input u_2');
legend('H_{12}', 'H_{22}',Location='southwest');
axis([0.1 fs/2 1e-3 1e2]);
grid on;
saveas(gcf, fullfile('figures', 'T1S5_Freq_u_2.png'));

% Magnitude for H13 and H23
figure;
loglog(f, abs(H13), 'r', f, abs(H23), 'b');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Response Magnitudes for Input u_3');
legend('H_{13}', 'H_{23}',Location='southwest');
axis([0.1 fs/2 1e-3 1e2]);
grid on;
saveas(gcf, fullfile('figures', 'T1S5_Freq_u_3.png'));

% Phase of H11 and H21
figure;
semilogx(f, angle(H11)*(180/pi), 'r', f, angle(H21)*(180/pi), 'b');
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
title('Frequency Response Phases for Input u_1');
```

```
legend('H_{11}', 'H_{21}',Location='southwest');
axis([0.1 fs/2 -200 200]);
grid on;
saveas(gcf, fullfile('figures', 'T1S5_Phase_u_1.png'));

% Phase for H12 and H22
figure;
semilogx(f, angle(H12)*(180/pi), 'r', f, angle(H22)*(180/pi), 'b');
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
title('Frequency Response Phases for Input u_2');
legend('H_{12}', 'H_{22}',Location='southwest');
axis([0.1 fs/2 -200 200]);
grid on;
saveas(gcf, fullfile('figures', 'T1S5_Phase_u_2.png'));

% Phase for H13 and H23
figure;
semilogx(f, angle(H13)*(180/pi), 'r', f, angle(H23)*(180/pi), 'b');
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
title('Frequency Response Phases for Input u_3');
legend('H_{13}', 'H_{23}',Location='southwest');
axis([0.1 fs/2 -200 200]);
grid on;
saveas(gcf, fullfile('figures', 'T1S5_Phase_u_3.png'));
```

# Task 2 - Pulse Response Estimates

```
% Section 1 - Compute Pulse Responses using IFFT

% Compute the IFFT of the frequency responses to obtain pulse responses
h11 = ifft(H11);
h21 = ifft(H21);

h12 = ifft(H12);
h22 = ifft(H22);

h13 = ifft(H13);
h23 = ifft(H23);

% Time vector for pulse responses
n_pulse = length(h11); % Should be equal to nfft
t_pulse = (0:n_pulse-1) * ts; % Starts at t=0


% Section 2 - Plotting Pulse Responses

% Plot h11 and h21
figure;
subplot(2,1,1);
plot(t_pulse, real(h11), 'r');
xlabel('Time (s)');
```

```
ylabel('Amplitude');
title('Pulse Response h_{11}');
axis([0 3 -2 3]);
grid on;

subplot(2,1,2);
plot(t_pulse, real(h21), 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title('Pulse Response h_{21}');
axis([0 3 -2 3]);
grid on;
saveas(gcf, fullfile('figures/Task2', 'T2S1_pr_h11_h21.png'));

% Plot h12 and h22
figure;
subplot(2,1,1);
plot(t_pulse, real(h12), 'r');
xlabel('Time (s)');
ylabel('Amplitude');
title('Pulse Response h_{12}');
axis([0 3 -2 3]);
grid on;

subplot(2,1,2);
plot(t_pulse, real(h22), 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title('Pulse Response h_{22}');
axis([0 3 -2 3]);
grid on;
saveas(gcf, fullfile('figures/Task2', 'T2S1_pr_h12_h22.png'));

% Plot h13 and h23
figure;
subplot(2,1,1);
plot(t_pulse, real(h13), 'r');
xlabel('Time (s)');
ylabel('Amplitude');
title('Pulse Response h_{13}');
axis([0 3 -2 3]);
grid on;

subplot(2,1,2);
plot(t_pulse, real(h23), 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title('Pulse Response h_{23}');
axis([0 3 -2 3]);
grid on;
saveas(gcf, fullfile('figures/Task2', 'T2S1_pr_h13_h23.png'));


% Check maximum imaginary part
```

```
max_imag_h11 = max(abs(imag(h11)));
disp(['Maximum imaginary part of h11: ', num2str(max_imag_h11)]);
```

# Task 3 - Hankel Matrix Analysis and Parametric Model

```
function h = compute_h(h11, h12, h13, h21, h22, h23, n)
    h = [h11(n) h12(n) h13(n); h21(n) h22(n) h23(n)];
end

% Section 1 - Construct the Hankel Matrix M_n

% Number of samples to use for constructing M_n
n = 25;
K = 2*n; % Number of required pulse response samples

% Ensure that K does not exceed the length of h11
if K > length(h11)
    error('Not enough data points in h11 to construct M_n with n = %d', n);
end

% Initialize variables
m = 2; % Number of outputs
q = 3; % Number of inputs

% Initialize h{1} as a zero matrix representing h[0]
h{1} = zeros(m, q);

% Assign the actual impulse responses starting from h{2}
for k = 1:K
    h_k = [h11(k), h12(k), h13(k);  % First row for output y1
           h21(k), h22(k), h23(k)]; % Second row for output y2
    h{k + 1} = h_k; % Shifted by +1
end

M_n = zeros(m*n, q*n);

for i = 1:n
    for j = 1:n
        h_ij = h{i+j+1};
        % hij = compute_h(h11, h12, h13, h21, h22, h23, i+j)
        M_n(2*i-1:2*i, 3*j-2:3*j) = compute_h(h11, h12, h13, h21, h22, h23,
i+j);
    end
end

M_new = zeros(m*n, q*n);
for i = 1:n
    for j = 1:n
        M_new(2*i-1:2*i, 3*j-2:3*j) = compute_h(h11, h12, h13, h21, h22,
h23, i+j+1); %[h11(i+j+1), h12(i+j+1), h13(i+j+1); h21(i+j+1), h22(i+j+1),
h23(i+j+1)];
```

```
    end
end
Mn_tilde = M_new;


% Section 2 - Perform SVD on M_n

[U, S, V] = svd(M_n);

% Plot singular values
singular_values = diag(S);
figure;
semilogy(singular_values, 'o-');
xlabel('Index');
ylabel('Singular Value (log scale)');
title('Singular Values of M_n');
grid on;
saveas(gcf, fullfile('figures/Task3', 'T3S1_M_singular.png'));
% Section 3 - Estimate models for different model orders

model_orders = [7, 8, 10, 16];
num_models = length(model_orders);

% Preallocate cell arrays to store models
A_models = cell(num_models, 1);
B_models = cell(num_models, 1);
C_models = cell(num_models, 1);
D_models = cell(num_models, 1); % D is assumed to be zero


% Section 3 - Estimate models for different model orders

model_orders = [7, 8, 10, 16];
num_models = length(model_orders);

% Preallocate cell arrays to store models
A_models = cell(num_models, 1);
B_models = cell(num_models, 1);
C_models = cell(num_models, 1);
D_models = cell(num_models, 1); % D is assumed to be zero

for idx = 1:num_models
    n_s = model_orders(idx);

    % Truncate SVD matrices to model order n_s
    U1 = U(:, 1:n_s);          % m*n x n_s (50x8 for n_s=8)
    S1 = S(1:n_s, 1:n_s);      % n_s x n_s
    V1 = V(:, 1:n_s);          % q*n x n_s (75x8 for n_s=8)

    % Compute Projection Matrices
    L = U1; %
    % L = sqrt(S1);          % m*n x n_s (50x8)
    R = S1* V1'; %
    % R = sqrt(S1) * V1';         % n_s x q*n (8x75)
```

```
    % Compute State Matrix A using standard ERA formula
    % A = inv(L'*L)*L' * Mn_tilde * R'*inv(R*R');
    A = pinv(L) * Mn_tilde * pinv(R); % n_s x n_s matrix

    % Extract Output Matrix C (first m*n rows of L)
    C = L(1:m, :);        % (m*n) x n_s (50x8)

    % Extract Input Matrix B (first q columns of R)
    B = R(:, 1:q);              % n_s x q (8x3)

    % Assume D is a zero matrix
    D = zeros(m, q);            % 2x3 matrix

    % Store the state-space matrices
    A_models{idx} = A;
    B_models{idx} = B;
    C_models{idx} = C;
    D_models{idx} = D;

    % Check stability
    eig_A = eig(A);
    max_abs_eig = max(abs(eig_A));
    fprintf('Model Order %d: Max |eig(A)| = %.4f\n', n_s, max_abs_eig);

    if all(abs(eig_A) < 1)
        fprintf('Model Order %d is asymptotically stable.\n\n', n_s);
    else
        fprintf('Warning: The model with order %d is unstable.\n\n', n_s);
    end
end

close all

% Section 4 - Simulate the Impulse Response of Each Model and Plot

% Number of time steps to simulate
num_steps = length(t_pulse);

% Define input-output pairs for plotting
input_output_pairs = {'h11', 'h21'; 'h12', 'h22'; 'h13', 'h23'};

graph_positions = [200,100,1000,700];
for idx = 1:num_models
    if idx ~= 4
        n_s = model_orders(idx);
        A = A_models{idx};
        B = B_models{idx};
        C = C_models{idx};
        D = D_models{idx};

        % Initialize storage for model impulse responses
        h_model = struct();
```

```
% Simulate impulse responses for each input
for input_idx = 1:q
    % Reset state vector
    x = zeros(n_s, num_steps + 1); % +1 for initial state
    % Initialize output storage
    y_model = zeros(m, num_steps);

    for k = 1:num_steps
        u = zeros(q, 1);
        if k == 1
            u(input_idx) = 1; % Impulse at k=1 for input_idx
        end

        x(:, k+1) = A * x(:, k) + B * u;
        y = C * x(:, k) + D * u;

        y_model(:, k) = y;
    end

    % Store the outputs
    switch input_idx
        case 1
            h_model.h11 = y_model(1, :)';
            h_model.h21 = y_model(2, :)';
        case 2
            h_model.h12 = y_model(1, :)';
            h_model.h22 = y_model(2, :)';
        case 3
            h_model.h13 = y_model(1, :)';
            h_model.h23 = y_model(2, :)';
    end
end

% Define frequency vector (same as in Task 1)
omega = 2 * pi * f; % Convert frequency to radians per second

% Number of frequency points
num_freq = length(omega);

% Preallocate frequency response matrices
H_model = struct();

for k = 1:num_freq
    s = exp(1j * omega(k) * ts);
    G = C * ((s * eye(n_s) - A) \ B) + D; % Solve (sI - A)^{-1} * B
    % G is m x q

    % Store frequency responses
    H_model.H11(k) = G(1,1);
    H_model.H21(k) = G(2,1);
    H_model.H12(k) = G(1,2);
    H_model.H22(k) = G(2,2);
    H_model.H13(k) = G(1,3);
    H_model.H23(k) = G(2,3);
```

```
        end

        % Plot Impulse Responses with Subplots
        for pair = 1:size(input_output_pairs,1)
            % Extract the pair of responses
            resp1 = input_output_pairs{pair, 1}; % e.g., 'h11'
            resp2 = input_output_pairs{pair, 2}; % e.g., 'h21'

            % Parse output and input channels from the response names
            out1 = resp1(2); in1 = resp1(3);  % for 'h11', out1='1', in1='1'
            out2 = resp2(2); in2 = resp2(3);  % for 'h21', out2='2', in2='1'

            % Create a new figure for each pair
            figure(Position=graph_positions);

            % Plot both responses (estimated and model) on the same axes
            hold on;
            plot(t_pulse, real(eval(resp1)), 'r', 'DisplayName', ['Estimated
h_{',out1,in1,'}']);
            plot(t_pulse, h_model.(resp1), 'r--', 'DisplayName', ['Model
h_{',out1,in1,'}']);
            plot(t_pulse, real(eval(resp2)), 'b', 'DisplayName', ['Estimated
h_{',out2,in2,'}']);
            plot(t_pulse, h_model.(resp2), 'b--', 'DisplayName', ['Model
h_{',out2,in2,'}']);

            % Set labels and title
            xlabel('Time (s)');
            ylabel('Amplitude');
            title(['Impulse Response Comparisons (', resp1, ' & ', resp2, ')
- Model Order ', num2str(n_s)]);

            legend('Location', 'northeast');
            axis([0 3 -2 3]);
            grid on;

            saveas(gcf, fullfile('figures/Task3',
sprintf('T3S2_impulse_model_order_%d_%s_%s.png', n_s, resp1, resp2)));




            % plot frequency magnitude response
            figure(Position=graph_positions);

            % Plot estimated vs model for resp1
            loglog(f, abs(eval(upper(resp1))), 'r', 'DisplayName',
['Estimated H_{',out1,in1,'}']);

            hold on; % hold on must come after log log graph otherwise the
formatting will be incorrect

            loglog(f, abs(H_model.(upper(resp1))), 'r--', 'DisplayName',
['Model H_{',out1,in1,'}']);
```

```
            % Plot estimated vs model for resp2
            loglog(f, abs(eval(upper(resp2))), 'b', 'DisplayName',
['Estimated H_{',out2,in2,'}']);
            loglog(f, abs(H_model.(upper(resp2))), 'b--', 'DisplayName',
['Model H_{',out2,in2,'}']);


            xlabel('Frequency (Hz)');
            ylabel('Magnitude');
            title(['Frequency Response Magnitude Comparisons (',
upper(resp1), ' & ', upper(resp2), ') - Model Order ', num2str(n_s)]);
            legend('Location','southwest');
            axis([0.1 fs/2 1e-3 1e2]);
            grid on;
            hold off;

            saveas(gcf, fullfile('figures/Task3',
sprintf('T3S3_freq_magnitude_model_order_%d_%s_%s.png', n_s, resp1, resp2)));



            % plot frequency phase response
            figure(Position=graph_positions);


            % Plot estimated vs model for resp1
            semilogx(f, angle(eval(upper(resp1)))*(180/pi), 'r',
'DisplayName',['Estimated H_{',out1,in1,'}']);
            hold on;
            semilogx(f, angle(H_model.(upper(resp1)))*(180/pi), 'r--',
'DisplayName',['Model H_{',out1,in1,'}']);

            % Plot estimated vs model for resp2
            semilogx(f, angle(eval(upper(resp2)))*(180/pi), 'b',
'DisplayName',['Estimated H_{',out2,in2,'}']);
            semilogx(f, angle(H_model.(upper(resp2)))*(180/pi), 'b--',
'DisplayName',['Model H_{',out2,in2,'}']);

            xlabel('Frequency (Hz)');
            ylabel('Phase (degrees)');
            title(['Frequency Response Phase Comparisons (', upper(resp1), '
& ', upper(resp2), ') - Model Order ', num2str(n_s)]);
            legend('Location','southwest');
            axis([0.1 fs/2 -200 200]);
            grid on;

            saveas(gcf, fullfile('figures/Task3',
sprintf('T3S3_phase_model_order_%d_%s_%s.png', n_s, resp1, resp2)));


        end

    end
```

```
end
```

# Task 4 - Transmission Zeros and Eigenvalue-Zero Cancellations

```
function tz = compute_transmission_zeros(A, B, C, D)
% Dimensions
n = size(A, 1);  % Number of states

% Construct the generalized eigenvalue matrices
E = [A, B; -C, -D];          % Combined matrix for the system
F = [eye(n), zeros(n, 1);    % Block diagonal identity-zeros matrix
    zeros(1, n), 0];

% Solve the generalized eigenvalue problem
[~, Z] = eig(E, F);

% Extract eigenvalues
z_vals = diag(Z);

% Filter out inf and nan
tz = z_vals(~isinf(z_vals) & ~isnan(z_vals));
end

model_orders = [7,8,10,16];
% Number of models
num_models = length(model_orders);
zeros_models = cell(num_models, 1);
poles_models = cell(num_models, 1);



for idx = 1:num_models
    if idx == 2 || idx == 3
        n_s = model_orders(idx);
        A = A_models{idx};
        B = B_models{idx};
        C = C_models{idx};
        D = D_models{idx};

        % Create a state-space model
        sys = ss(A, B, C, D, ts);

        % Compute poles (eigenvalues of A)
        tp = eig(A);

        % Store poles
        poles_models{idx} = tp;

        % Calculate magnitudes of poles
        pole_magnitudes = abs(tp);
```

```
        % Display poles with additional information
        disp(['Model Order ', num2str(n_s), ':']);
        disp(['Number of Poles: ', num2str(length(tp))]);
        disp(['Poles (Eigenvalues of A): ', num2str(tp.')]);
        disp(['Pole Magnitudes: ', num2str(pole_magnitudes.')]);
        disp(['Maximum Pole Magnitude: ', num2str(max(pole_magnitudes))]);
        disp(['Minimum Pole Magnitude: ', num2str(min(pole_magnitudes))]);
        disp('----------------------------------------');

        % Plot poles for each model
        figure(Position=[200,0,700,700])
        hold on;

        % Plot poles
        plot(real(tp), imag(tp), 'bx', 'MarkerSize', 10, 'LineWidth', 2); %
Poles as blue 'x'

        % Plot unit circle for reference
        theta = linspace(0, 2*pi, 300);
        plot(cos(theta), sin(theta), 'k--');

        xlabel('Real Part');
        ylabel('Imaginary Part');
        title(['Poles in z-plane - Model Order ', num2str(n_s)]);
        grid on;
        axis square;
        hold off;

        saveas(gcf, fullfile('figures/Task4',
sprintf('T4_pole_zero_model_order_%d.png',n_s)));
    end
end
```

Tolerance for zero-pole cancellation

```
tol = 0.2;


% Initialize cancellation tracking
cancellations = cell(num_models, 1);

for idx = 1:num_models
    if idx == 2 || idx == 3
        n_s = model_orders(idx);
        A = A_models{idx};
        B = B_models{idx};
        C = C_models{idx};
        D = D_models{idx};
        tp = poles_models{idx};  % Poles for the current model


        cancellations{idx} = cell(size(C, 1), size(B, 2));  % One cell per
```

```
output-input pair
        for out_idx = 1:size(C,1)  % Loop over outputs
            for in_idx = 1:size(B,2)  % Loop over inputs
                C_ch = C(out_idx, :);
                D_ch = D(out_idx, in_idx);
                B_ch = B(:, in_idx);

                % Create SISO state-space model
                sys_ch = ss(A, B_ch, C_ch, D_ch, ts);

                % Compute transmission zeros
                % tz = tzero(sys_ch);
                fprintf("model %d ic %d oc
%d\n",model_orders(idx),in_idx,out_idx)
                tz = compute_transmission_zeros(A,B_ch,C_ch,D_ch);


                tz = tz(abs(tz) <= 1e3);

                % Initialize tracking for this channel
                canceled_pairs = [];


                % Compare zeros and poles
                for i = 1:length(tz)
                    for j = 1:length(tp)
                        if abs(tz(i) - tp(j)) < tol
                            canceled_pairs = [canceled_pairs; tz(i), tp(j)];
%#ok<AGROW>
                            break;  % Each zero cancels with one pole at most
                        end
                    end
                end

                % Store cancellations for this channel
                cancellations{idx}{out_idx, in_idx} = canceled_pairs;

                % Display cancellations
                disp(['Model Order ', num2str(n_s), ...
                    ', Output ', num2str(out_idx), ...
                    ', Input ', num2str(in_idx)]);
                disp('Canceled Zero-Pole Pairs:');
                disp(canceled_pairs);

            end
        end
    end

end
disp("------------------------------")
```

Filter out large zeros (inf zeros)

```
threshold = 1;
% Plot zeros and poles for each channel
for idx = 1:num_models
    if idx == 2 || idx == 3
        n_s = model_orders(idx);
        A = A_models{idx};
        B = B_models{idx};
        C = C_models{idx};
        D = D_models{idx};

        tp = poles_models{idx};

        for out_idx = 1:size(C,1)      % Loop over outputs
            for in_idx = 1:size(B,2)   % Loop over inputs
                C_ch = C(out_idx, :);
                D_ch = D(out_idx, in_idx);
                B_ch = B(:, in_idx);  % Single input column

                % Create SISO state-space model for this input-output pair
                sys_ch = ss(A, B_ch, C_ch, D_ch, ts);

                % Compute transmission zeros for this channel
                % tz = tzero(sys_ch);
                tz = compute_transmission_zeros(A,B_ch,C_ch,D_ch);


                %
                % % Identify inf zeros
                inf_zeros = tz(abs(tz) > threshold);

                % Identify finite zeros
                finite_zeros = tz(abs(tz) <= threshold);

                tz = finite_zeros;

                % Plot poles and zeros for this channel
                figure(Position=[200,0,700,700]);
                hold on;

                % Plot poles (eigenvalues)
                plot(real(tp), imag(tp), 'bx', 'MarkerSize', 10,
'LineWidth', 2);

                % Plot zeros if they exist
                if ~isempty(tz)
                    plot(real(tz), imag(tz), 'ro', 'MarkerSize', 8,
'LineWidth', 2);
                    legend_entries = {'Poles', 'Zeros'};
                else
                    legend_entries = {'Poles'};
                end

                % Plot unit circle
                theta = linspace(0, 2*pi, 300);
```

```
                plot(cos(theta), sin(theta), 'k--');

                axis square;
                grid on;
                xlabel('Real Part');
                ylabel('Imaginary Part');
                title(['Poles and Zeros within Norm = ',num2str(threshold),'
| Model Order ', num2str(n_s), ', Input ', num2str(in_idx), ', Output ',
num2str(out_idx)]);        legend(legend_entries);
                hold off;
                saveas(gcf, fullfile('figures/Task4',
sprintf('T4_pole_zero_model_order_%d_output_%d_input_%d.png',n_s,out_idx,in_i
dx)));

            end
        end
    end
end
```

# Task 5 - Controller Design

```
% Number of models
num_models = length(model_orders);

% Preallocate cell arrays for K
K_models = cell(num_models, 1);

idx = 2; % we only care about model 8 with index 2

n_s = model_orders(idx);
A = A_models{idx};
B = B_models{idx};
C = C_models{idx};

% Extract discrete-time eigenvalues
eig_d = eig(A)
% log_eigd = log(eig_d)
% log_eigd_add = log(eig_d) + 2*pi*2j

% Discrete to continuous
eig_c = log(eig_d) / ts
% eig_c_add = log_eigd_add / ts

eig_d_c = [eig_d, eig_c]

% Get one of the eigenvalues cont of resonators
resonators_eigs_c = [eig_c(5);eig_c(7)]

fn = abs(resonators_eigs_c)/2/pi
```

```
% Check controllability
Co = ctrb(A, B);
rank_Co = rank(Co)

disp(['Model Order ', num2str(n_s), ':']);
disp(['Rank of Controllability Matrix: ', num2str(rank_Co)]);

if rank_Co == n_s
    disp('The system is controllable.');
else
    disp('The system is NOT controllable.');
    disp('----------------------------------------');

end
```

*Published with MATLAB® R2024b*