Table of Contents

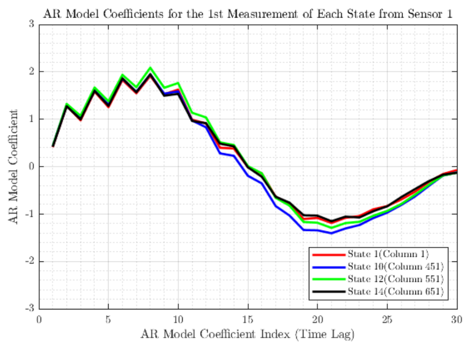
```
% Ruipu Ji
% SE 265
% Homework #8

clc; clear; close all;

set(0, 'DefaultTextInterpreter', 'latex');
set(0, 'DefaultLegendInterpreter', 'latex');
set(0, 'DefaultAxesTickLabelInterpreter', 'latex');

set(0, 'DefaultAxesFontSize', 10);
set(0, 'DefaultTextFontSize', 10);
```

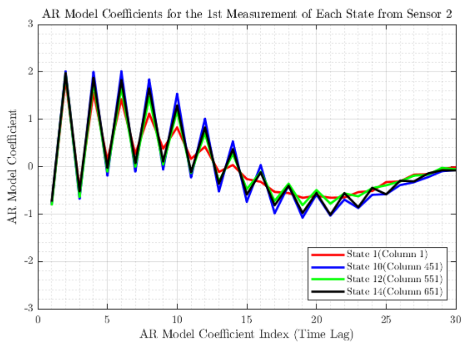
Task 0: Load data and create arrays.

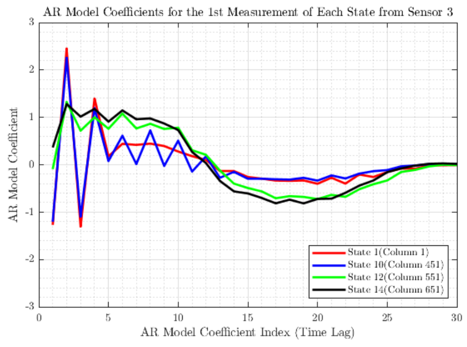
```
load('4-Story Structure Data/data3SS2009.mat'); % Load the data file.
Input = squeeze(dataset(:,1,:)); % Input = Data from channel 1 (input time
history from the load cell).
Response = dataset(:,2:5,:); % Response = Data from channel 2-5
(acceleration response for each level).
% squeeze() is to remove the dimension with length of 1.
NumOfPoints = size(Input,1); % NumOfPoints = Number of data points in each
set of signal.
NumOfLevels = size(Response,2); % NumOfLevels = Number of levels (number of
input-sensor pairs) in the structure.
NumOfTests = 50; % NumOfTests = Number of tests for each state at each floor.
State = [1 10 12 14]; % Create a vector for all the states used for analysis.
```

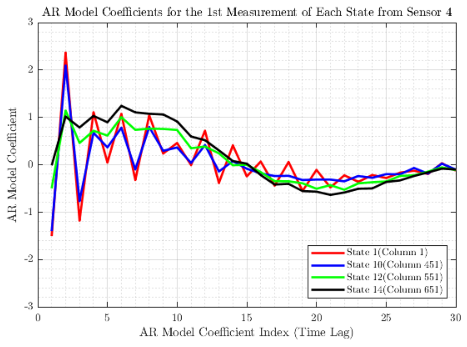
Task 1: Fit an AR model to the data from states 1, 10, 12 and 14.

```
Order = 30; % Define the order of the AR model.

% Initialize a 3-D matrix for the coefficients of the AR model.
% Dimension-1 = 4, which represents the total number of levels in the
structure.
% Dimension-2 = 4, which represents the 4 states of interest.
% Dimension-3 = 50, which represents the number of measurements for each
state.
% Dimension-4 = 30, which represents the order of the AR model.
Coefficients = zeros(NumOfLevels, size(State,2), NumOfTests, Order);

% Calculate the coefficients for a 30-th order linear AR model.
for LevelIndex = 1:NumOfLevels % Loop over all the levels.
    for StateIndex = 1:size(State,2) % Loop over all the states of interest
(State 1, 10, 12 and 14).
        % First create a temperory matrix to store the result including the
1 in the first column.
        % lpc(x, p) finds the coefficients of a p-th order linear predictor
and returns to a 1-D row vector.
        % If x is a 2-D matrix, then the function will treat each column as
a separate channel.
        % The input x must be in double precision.
        Coefficients_temp = lpc(double(squeeze(Response(:, LevelIndex,
(State(StateIndex)-1)*NumOfTests+1:State(StateIndex)*NumOfTests))), Order);

        % Remove the 1 in the first column and store the coefficient matrix
in the final output matrix.
        Coefficients_temp(:,1) = [];
        Coefficients(LevelIndex,StateIndex,:,:) = Coefficients_temp;
    end
end

% Plot AR model coefficients for the first measurement from each state.
% Create the color map and the plot legend cell array for the plot
corresponding to different damage states.
color = ['r', 'b', 'g', 'k'];
LegendState = cell(size(State,2),1);

figure('Renderer', 'painters', 'Position', [10 10 1500 1000]);

for LevelIndex = 1:NumOfLevels
    subplot(2,2,LevelIndex);
    hold on;

    for StateIndex = 1:size(State,2)
        plot(squeeze(Coefficients(LevelIndex, StateIndex, 1, :)), 'Color',
color(StateIndex), 'LineWidth', 2); % Plot the AR model coefficients.
        LegendState{StateIndex} = sprintf(['State
', num2str(State(StateIndex)), ' (Column ',
num2str((State(StateIndex)-1)*NumOfTests+1), ')']);
    end

    grid on;
    grid minor;
    box on;
    xlim([0 Order]);
    ylim([-3 3]);
    xticks(0:5:Order);
    yticks(-3:1:3);
    xlabel('AR Model Coefficient Index (Time Lag)');
    ylabel('AR Model Coefficient');
    legend(LegendState, 'Location', 'southeast');
    title(sprintf(['AR Model Coefficients for the 1st Measurement of Each
State from Sensor ', num2str(LevelIndex)]));
end

hold off;
```









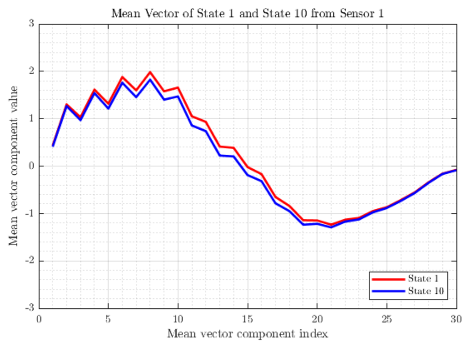
Task 2: Calculate the mean and covariance matrices for the Fisher Discriminant.

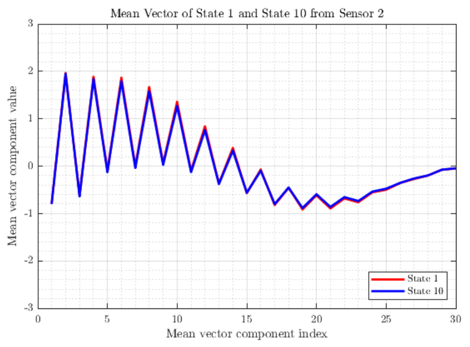
Initialize the mean and covariance matrices.

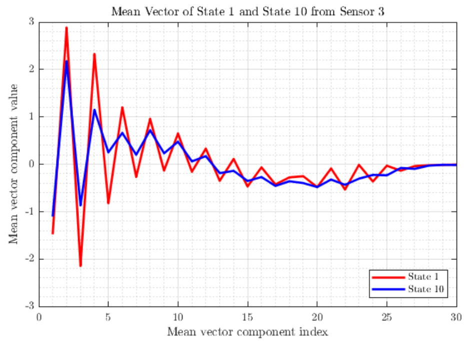
```
MeanVectors = zeros(NumOfLevels, 2, Order);
CovarianceMatrices = zeros(NumOfLevels, 2, Order, Order);

% Calculate the mean and covariance matrices for all the measurements of
state 1 and state 10.
for LevelIndex = 1:NumOfLevels % Loop over all the levels.
    for StateIndex = 1:2 % Loop over all the states of interest (State 1 and
10).
        MeanVectors(LevelIndex, StateIndex, :) =
mean(Coefficients(LevelIndex, StateIndex, :, :)); % Calcualte mean vectors.
        CovarianceMatrices(LevelIndex, StateIndex, :, :) =
cov(squeeze(Coefficients(LevelIndex, StateIndex, :, :))); % Calculate
```

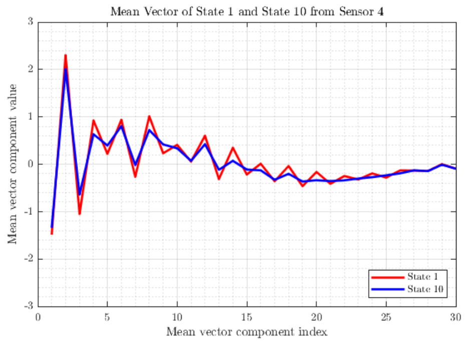
```
covariance matrices.
    end
end

% Plot mean vectors of the 2 states for all the 4 sensors.
figure('Renderer', 'painters', 'Position', [10 10 1500 1000]);

for LevelIndex = 1:NumOfLevels
    subplot(2,2,LevelIndex);
    hold on;

    for StateIndex = 1:2
        plot(squeeze(MeanVectors(LevelIndex, StateIndex, :)), 'Color',
color(StateIndex), 'LineWidth', 2); % Plot the mean vector components.
        LegendState{StateIndex} = sprintf(['State ',
num2str(State(StateIndex))]);
    end

    grid on;
    grid minor;
    box on;
    xlim([0 Order]);
    ylim([-3 3]);
    xticks(0:5:Order);
    yticks(-3:1:3);
    xlabel('Mean vector component index');
    ylabel('Mean vector component value');
    legend(LegendState, 'Location', 'southeast');
    title(sprintf(['Mean Vector of State 1 and State 10 from Sensor ',
num2str(LevelIndex)]));
end

hold off;
```

```
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
```









Task 3: Calculate the within-class and between-class scatter matrices.

Initialize the with-in class (Sw) and between-class (Sb) scatter matrices.

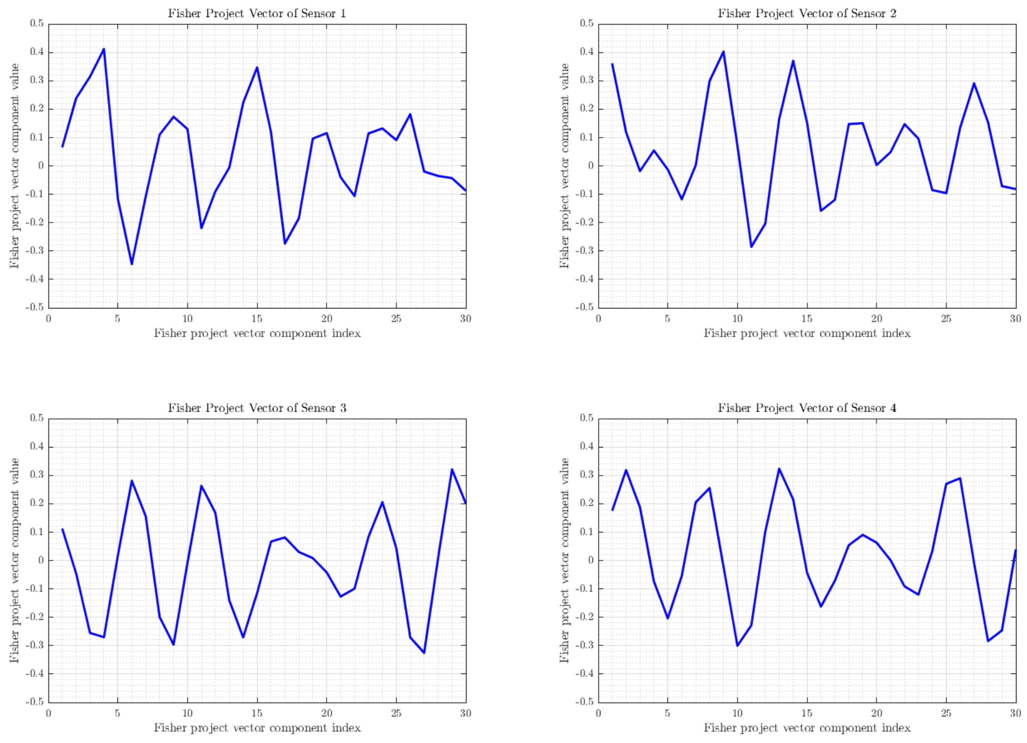
```
Sw = zeros(NumOfLevels, Order, Order);
Sb = zeros(NumOfLevels, Order, Order);

% Calculate the with-in class and between-class scatter matrices.
for LevelIndex = 1:NumOfLevels % Loop over all the levels.
    Sw(LevelIndex,:,:) = CovarianceMatrices(LevelIndex, 1, :, :) +
CovarianceMatrices(LevelIndex, 2, :, :);
    Sb(LevelIndex,:,:) = squeeze(MeanVectors(LevelIndex, 1, :) -
MeanVectors(LevelIndex, 2, :)) * squeeze(MeanVectors(LevelIndex, 1, :) -
MeanVectors(LevelIndex, 2, :))';
end
```

Task 4: Solve the eigenvalue problem to find the Fisher project vector.

Initialize the Fisher project vector.

```
w = zeros(Order, NumOfLevels);
```

```
% Calculate the Fisher project vector for each sensor.
for LevelIndex = 1:NumOfLevels % Loop over all the levels.
    [V, D] =
eig(inv(squeeze(Sw(LevelIndex,:,:)))*squeeze(Sb(LevelIndex,:,:))); % Solve
the eigenvalue problem: V = eigenvectors, D = eigenvalues (diagonal matrix).
    [MaxEigenvalues, MaxIndex] = max(diag(D)); % Find the maximum eigenvalue
and its corresponding index.
    w(:,LevelIndex) = V(:, MaxIndex); % The Fisher project vector is the
eigenvector corresponding to the largest eigenvalue.
end

% Plot Fisher project vectors of for all the 4 sensors.
figure('Renderer', 'painters', 'Position', [10 10 1500 1000]);

for LevelIndex = 1:NumOfLevels
    subplot(2,2,LevelIndex);
    hold on;

    plot(w(:, LevelIndex), 'Color', 'b', 'LineWidth', 2); % Plot the Fisher
project vector.

    grid on;
    grid minor;
    box on;
    xlim([0 Order]);
    ylim([-0.5 0.5]);
    xticks(0:5:Order);
    yticks(-0.5:0.1:0.5);
    xlabel('Fisher project vector component index');
    ylabel('Fisher project vector component value');
    title(sprintf(['Fisher Project Vector of Sensor ',
num2str(LevelIndex)]));
end

hold off;
```



Task 5: Project the AR features for all the classes onto the Fisher coordinate.

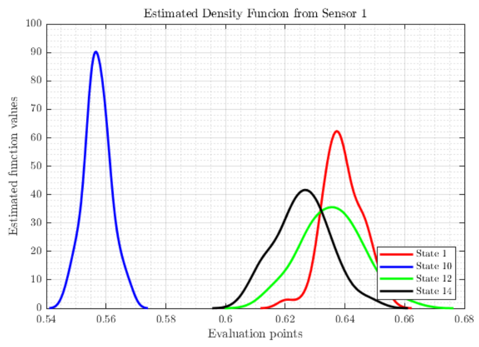
Initialize the projection matrix [y].

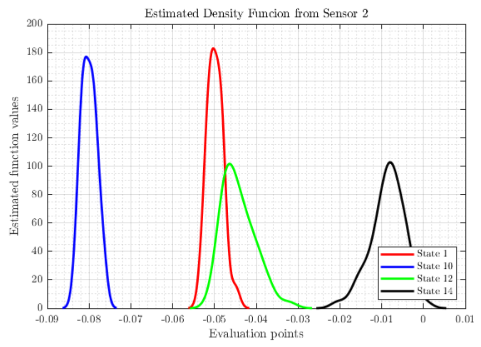
```
y = zeros(NumOfLevels, size(State,2), NumOfTests);

% Calculate the projection in Fisher coordinate for each sensor.
for LevelIndex = 1:NumOfLevels % Loop over all the levels.
    for StateIndex = 1:size(State,2) % Loop over different states of
interest.
        y(LevelIndex, StateIndex, :) = w(:,LevelIndex)' *
squeeze(Coefficients(LevelIndex, StateIndex, :, :))';
    end
end
```

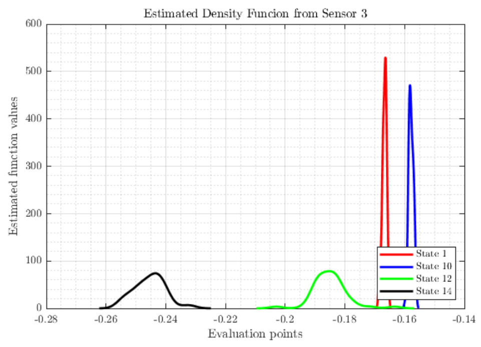
Task 6: Estimate the density function of the projected data sets using a kernel density estimator.

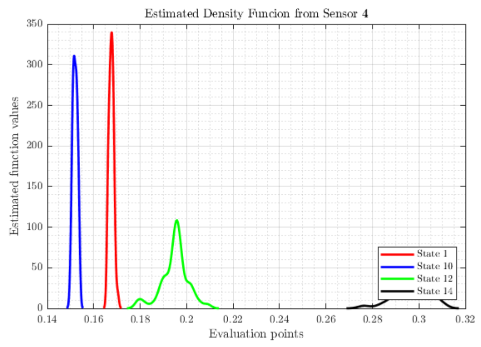
Plot the density functions for the projected data from each state.

```
figure('Renderer', 'painters', 'Position', [10 10 1500 1000]);

for LevelIndex = 1:NumOfLevels % Loop over all the levels.
    subplot(2,2,LevelIndex);
    hold on;

    for StateIndex = 1:size(State,2) % Loop over different states of
interest.
        % Estimate the density function of the projected data.
        % [f,xi] = ksdensity(x) returns to a probability density estimate.
        % f = Estimated function values. xi = Evaluation points.
        [f, xi] = ksdensity(squeeze(y(LevelIndex, StateIndex, :)));
        plot(xi, f, 'Color', color(StateIndex), 'LineWidth', 2); % Plot the
density functions.
        LegendState{StateIndex} = sprintf(['State ',
num2str(State(StateIndex))]);
    end

    grid on;
    grid minor;
    box on;
    xlabel('Evaluation points');
    ylabel('Estimated function values');
    legend(LegendState, 'Location', 'southeast');
    title(sprintf(['Estimated Density Funcion from Sensor ',
num2str(LevelIndex)]));
end
```









Published with MATLAB® R2023b