## Table of Contents



```
% Ruipu Ji
% SE 265
% Homework #5

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 1: 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.
```

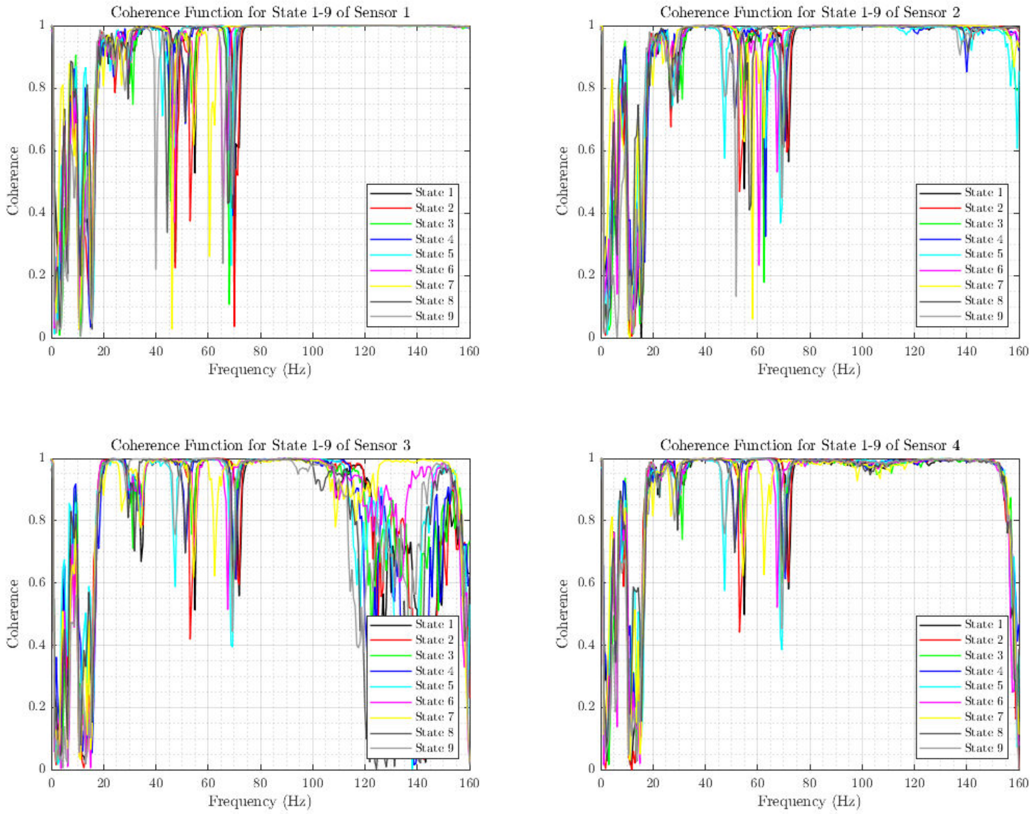
# Task 2: Calcualte and plot the coherence functions.

```
SamplingFrequency = 320; % SamplingFrequency = Sampling frequency in Hz.

% Initialize a 3-D matrix for coherence function values.
% Dimension-1 = 4, which represents the total number of levels in the
structure.
% Dimension-2 = 850, which represents the total number of tests.
% Dimension-3 = 257, which represents the total number of frequency
components.
coh = zeros(NumOfLevels, size(Response,3), NumOfPoints/16/2+1);

% Calculate coherence function values.
for Level = 1:NumOfLevels % Loop over all the 4 levels.
```

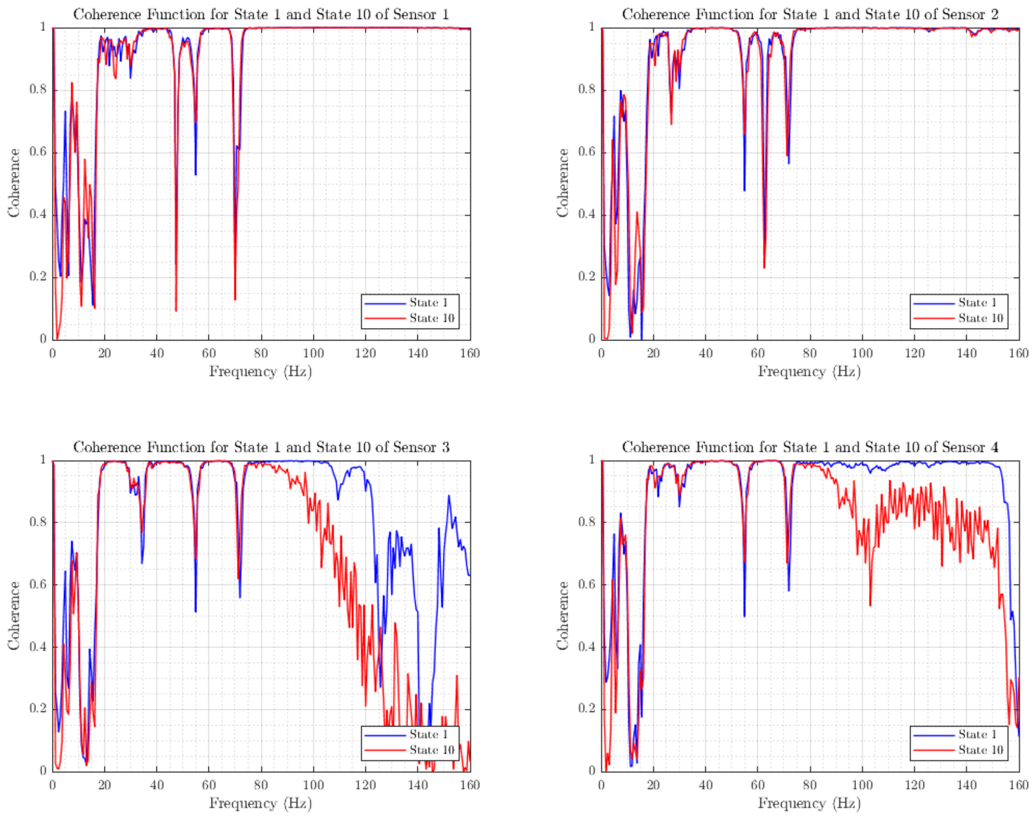
```
    for NumOfTest = 1:size(Response,3) % Loop over all the tests.
        [coh(Level, NumOfTest, :), f] = mscohere(Input(:,NumOfTest),
Response(:,Level,NumOfTest), hann(NumOfPoints/16), 0, [], SamplingFrequency);
    end
end

% Plot 1: Coherence Functions for the 1st Measurement of Each State.
% --------------------------------------------------------------------------
% Create the color map and the plot legend cell array for the plot
corresponding to different damage states.
color = [0 0 0; 1 0 0; 0 1 0;
         0 0 1; 0 1 1; 1 0 1;
         1 1 0; 0.3 0.3 0.3; 0.6 0.6 0.6];
LegendState = cell(9,1);

figure('Renderer', 'painters', 'Position', [10 10 1200 900]);

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

    for NumOfState = 1:9
        plot(f, squeeze(coh(Level, (NumOfState-1)*50+1, :)), 'Color',
color(NumOfState,:), 'LineWidth', 1); % Plot the coherence function vs
frequency.
        LegendState{NumOfState} = sprintf(['State ', num2str(NumOfState)]);

    end

    grid on;
    grid minor;
    box on;
    xlim([0 SamplingFrequency/2]);
    ylim([0 1]);
    xticks(0:20:SamplingFrequency/2);
    yticks(0:0.2:1);
    xlabel('Frequency (Hz)');
    ylabel('Coherence');
    legend(LegendState, 'Location', 'southeast');
    title(sprintf(['Coherence Function for State 1-9 of Sensor ',
num2str(Level)]));
end

sgtitle('Plot 1: Coherence Functions for the 1st Measurement of Each State');

% Plot 2: Coherence Functions for the 1st Measurement of State 1 and State
10.
% --------------------------------------------------------------------------
figure('Renderer', 'painters', 'Position', [10 10 1200 900]);

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

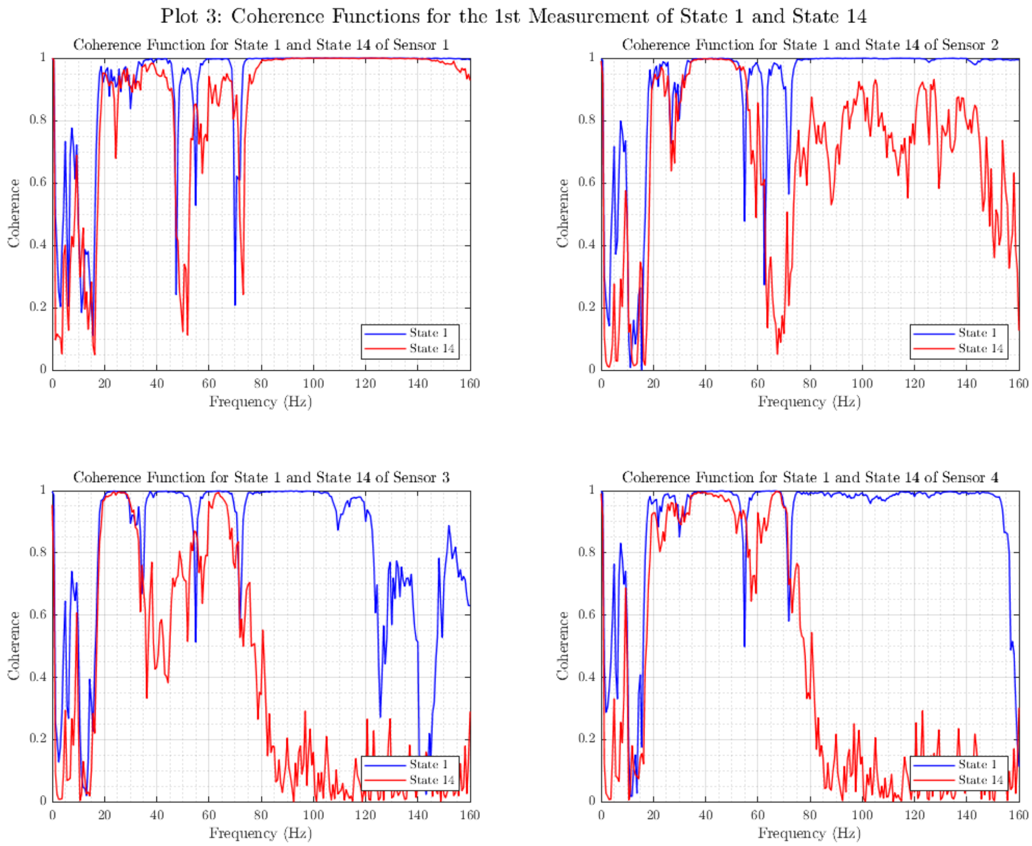
```
    plot(f, squeeze(coh(Level, 1, :)), 'Color', 'b', 'LineWidth', 1); % Plot
the coherence function vs frequency.
    plot(f, squeeze(coh(Level, 451, :)), 'Color', 'r', 'LineWidth', 1); %
Plot the coherence function vs frequency.
    grid on;
    grid minor;
    box on;
    xlim([0 SamplingFrequency/2]);
    ylim([0 1]);
    xticks(0:20:SamplingFrequency/2);
    yticks(0:0.2:1);
    xlabel('Frequency (Hz)');
    ylabel('Coherence');
    legend('State 1', 'State 10', 'Location', 'southeast');
    title(sprintf(['Coherence Function for State 1 and State 10 of Sensor ',
num2str(Level)]));
end

sgtitle('Plot 2: Coherence Functions for the 1st Measurement of State 1 and
State 10');

% Plot 3: Coherence Functions for the 1st Measurement of State 1 and State
14.
% ------------------------------------------------------------------------
figure('Renderer', 'painters', 'Position', [10 10 1200 900]);

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

    plot(f, squeeze(coh(Level, 1, :)), 'Color', 'b', 'LineWidth', 1); % Plot
the coherence function vs frequency.
    plot(f, squeeze(coh(Level, 651, :)), 'Color', 'r', 'LineWidth', 1); %
Plot the coherence function vs frequency.
    grid on;
    grid minor;
    box on;
    xlim([0 SamplingFrequency/2]);
    ylim([0 1]);
    xticks(0:20:SamplingFrequency/2);
    yticks(0:0.2:1);
    xlabel('Frequency (Hz)');
    ylabel('Coherence');
    legend('State 1', 'State 14', 'Location', 'southeast');
    title(sprintf(['Coherence Function for State 1 and State 14 of Sensor ',
num2str(Level)]));
end

sgtitle('Plot 3: Coherence Functions for the 1st Measurement of State 1 and
State 14');
```

Plot 1: Coherence Functions for the 1st Measurement of Each State



Plot 2: Coherence Functions for the 1st Measurement of State 1 and State 10






# Task 3: Calculate a coherence comparison metric.

Initialize a 2-D matrix for unity deviation metric (UDM) values. Dimension-1 = 4, which represents the total number of levels in the structure. Dimension-2 = 850, which represents the total number of tests.

```
UDM = zeros(NumOfLevels, size(Response,3));
min_Damaged_UDM = zeros(NumOfLevels, 1);

% Calculate the absolute unity deviation.
abs_coh_deviation = abs(1-coh);

% Calculate unity deviation metric (UDM) values.
for Level = 1:NumOfLevels % Loop over all the 4 levels.
    for NumOfTest = 1:size(Response,3) % Loop over all the tests.
        % Sum the absolute unity deviation values corresponding to
frequencies between 20 Hz and 150 Hz (index 33-241).
        UDM(Level, NumOfTest) = sum(abs_coh_deviation(Level, NumOfTest,
33:241));
    end
end
```

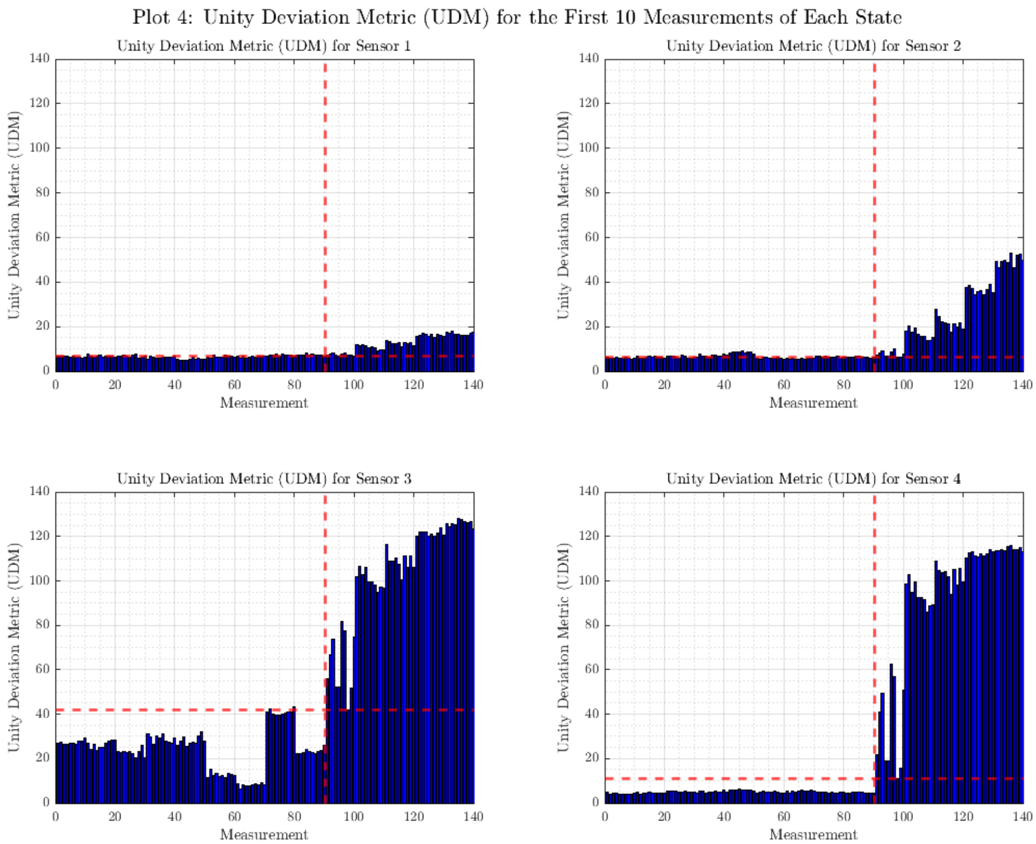
```
% Plot 4: Bar plots for the unity deviation metric (UDM) values.
% -----------------------------------------------------------------------
figure('Renderer', 'painters', 'Position', [10 10 1200 900]);

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

    for NumOfState = 1:14
        for i = 1:10
            % Plot the bar plot of the UDM values for the first 10
measurements of each state.
            bar((NumOfState-1)*10+i, UDM(Level,(NumOfState-1)*50+i), 0.75,
'blue');
        end
    end

    grid on;
    grid minor;
    box on;
    xlim([0 140]);
    ylim([0 ceil(max(UDM(:))/20)*20]);
    xticks(0:20:140);
    yticks(0:20:ceil(max(UDM(:))/20)*20);
    xlabel('Measurement');
    ylabel('Unity Deviation Metric (UDM)');
    title(sprintf(['Unity Deviation Metric (UDM) for Sensor ',
num2str(Level)]));

    % Plot a vertical red line that separates the undamaged cases (1-90) and
damaged cases (91-140).
    xline(90.5, 'r--', 'LineWidth', 2);

    % Calculate the minimum Unity Deviation Metric (UDM) value for the
damaged cases of each sensor.
    min_Damaged_UDM(Level) = min(UDM(Level, [451:460, 501:510, 551:560,
601:610, 651:660]));

    % Show the minimum Unity Deviation Metric (UDM) value in the command
window.
    disp(sprintf(['Minimum Unity Deviation Metric (UDM) value for the
damaged cases of sensor ', num2str(Level), ':']));
    disp(min_Damaged_UDM(Level));

    % Plot a horizontal red line corresponding to the lowest value of all
the damaged cases.
    yline(min_Damaged_UDM(Level), 'r--', 'LineWidth', 2);

end

sgtitle('Plot 4: Unity Deviation Metric (UDM) for the First 10 Measurements
of Each State');
```

*Minimum Unity Deviation Metric (UDM) value for the damaged cases of sensor 1: 6.6443*

*Minimum Unity Deviation Metric (UDM) value for the damaged cases of sensor 2: 6.5046*

*Minimum Unity Deviation Metric (UDM) value for the damaged cases of sensor 3: 41.9372*

*Minimum Unity Deviation Metric (UDM) value for the damaged cases of sensor 4: 10.9578*




*Published with MATLAB® R2023b*