Table of Contents

```
% Ruipu Ji
% SE 265
% Homework #7

clc; clear; close all;

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

set(0, 'DefaultAxesFontSize', 15);
set(0, 'DefaultTextFontSize', 15);
```

Task 1: Load data and create arrays.

```
load('4-Story Structure Data/data3SS2009.mat'); % Load the data file.
Sensor4 = squeeze(dataset(:,5,:)); % Sensor4 = Data from channel 5
(acceleration response for Level-4).
% squeeze() is to remove the dimension with length of 1.
NumOfPoints = size(Sensor4,1); % NumOfPoints = Number of data points in each
set of signal.
NumOfTests = size(Sensor4,2); % NumOfTests = Number of tests for each floor.
State = [1 3 10 12 14]; % Create a vector for all the states used for
analysis.
```

Task 1-A: Develop 30th-order AR model based on the data from state 1 (column 1).

```
Order = 30; % Order for the auto-regressive time series model.

% Create [X] matrix.
X = zeros(NumOfPoints-Order, Order);
```

```
for i = 1:Order
    X(:,i) = Sensor4(i:i+NumOfPoints-Order-1, 1);
end

B = Sensor4(Order+1:NumOfPoints, 1); % Create {B} vector.
a = pinv(X)*B; % Solve the coefficients vector {a}.
```

Task 1-B: Generate an estimate of the measured signals using the AR model.

Initialization for a matrix to store the reconstructed signals of each state. Each column represents a set of reconstructed signal. Note that each reconstructed signal has (NumOfPoints - Order) data points, which is different from the original signal.

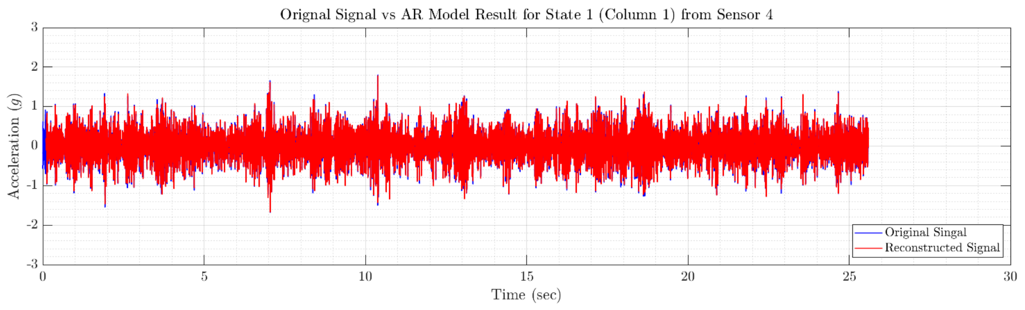
```
Reconstruction = zeros(NumOfPoints-Order, size(State,2));

% Calculate the reconstructed signals for each state.
for i = 1:size(State,2) % Loop over each state.
    % Create [X] matrix for each state.
    X = zeros(NumOfPoints-Order, Order);
    for j = 1:Order
        X(:,j) = Sensor4(j:j+NumOfPoints-Order-1, 50*(State(i)-1)+1);
    end

    % Reconstructed signal = [X] * coefficients vector {a}.
    Reconstruction(:,i) = X*a;
end
```

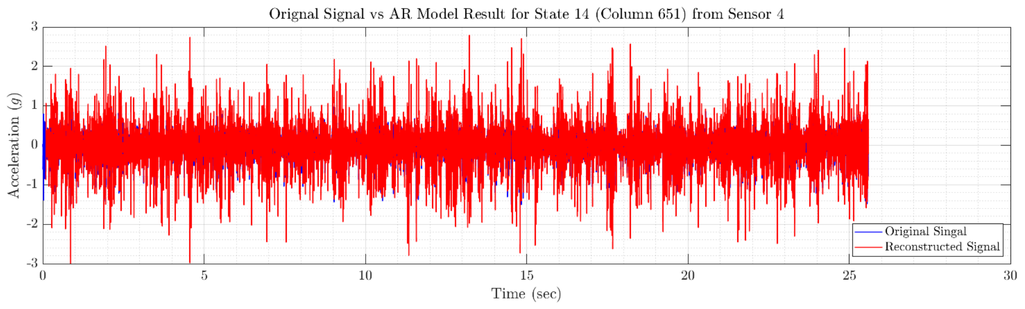
Task 1-C: Plot the time series.

Generate the timestamps vector.

```
SamplingFrequency = 320; % samplingFrequency = Sampling frequency in Hz.
timestamps = 0: 1/SamplingFrequency : (NumOfPoints-1)/SamplingFrequency; %
Create the timestamps vector.

% Plot orginal signal vs AR model result for state 1 and state 14.
figure('Renderer', 'painters', 'Position', [10 10 1800 1000]);

subplot(2,1,1); % Plot for state 1 (column 1).
hold on;
plot(timestamps, Sensor4(:,1), 'b', 'LineWidth', 1); % Plot the original
signal.
plot(timestamps(Order+1:NumOfPoints), Reconstruction(:,1), 'r', 'LineWidth',
1); % Plot the reconstructed signal.
grid on;
grid minor;
box on;
xlim([0 ceil(max(timestamps)/5)*5]);
ylim([-ceil(max(abs(Sensor4),[],"all")) ceil(max(abs(Sensor4),[],"all"))]);
xticks(0:5:ceil(max(timestamps)/5)*5);
yticks(-ceil(max(abs(Sensor4),[],"all")):1:ceil(max(abs(Sensor4),[],"all")));
```

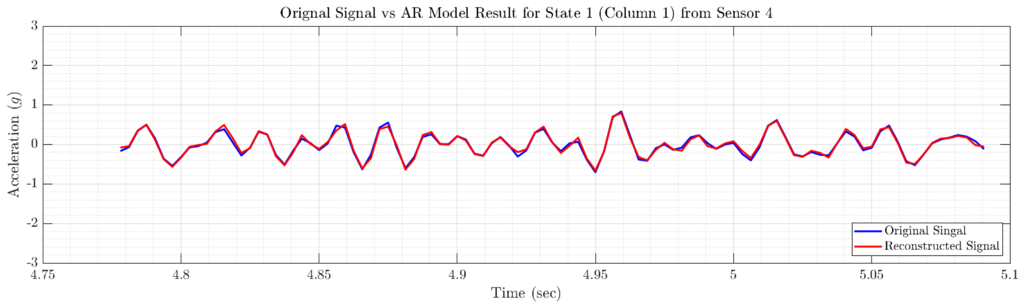
```
xlabel('Time (sec)');
ylabel('Acceleration ($g$)');
legend('Original Singal', 'Reconstructed Signal', 'Location', 'southeast');
title('Orignal Signal vs AR Model Result for State 1 (Column 1) from Sensor
4');

subplot(2,1,2); % Plot for state 14 (column 651).
hold on;
plot(timestamps, Sensor4(:,651), 'b', 'LineWidth', 1); % Plot the original
signal.
plot(timestamps(Order+1:NumOfPoints), Reconstruction(:,5), 'r', 'LineWidth',
1); % Plot the reconstructed signal.
grid on;
grid minor;
box on;
xlim([0 ceil(max(timestamps)/5)*5]);
ylim([-ceil(max(abs(Sensor4),[],"all")) ceil(max(abs(Sensor4),[],"all"))]);
xticks(0:5:ceil(max(timestamps)/5)*5);
yticks(-ceil(max(abs(Sensor4),[],"all")):1:ceil(max(abs(Sensor4),[],"all")));
xlabel('Time (sec)');
ylabel('Acceleration ($g$)');
legend('Original Singal', 'Reconstructed Signal', 'Location', 'southeast');
title('Orignal Signal vs AR Model Result for State 14 (Column 651) from
Sensor 4');
```

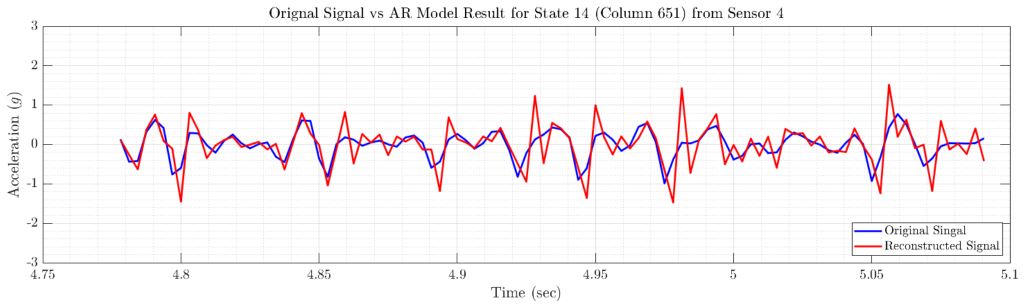




Task 1-D: Plot the time series for only the points with index 1500-1600.

```
IndexRange = (1500:1:1600) + Order; % Define the index range for the zoom-in
plot.

% Plot orginal signal vs AR model result for state 1 and state 14.
figure('Renderer', 'painters', 'Position', [10 10 1800 1000]);

subplot(2,1,1); % Plot for state 1 (column 1).
hold on;
plot(timestamps(IndexRange), Sensor4(IndexRange,1), 'b', 'LineWidth', 2); %
Plot the original signal.
plot(timestamps(IndexRange), Reconstruction(IndexRange-Order,1), 'r',
'LineWidth', 2); % Plot the reconstructed signal.
grid on;
grid minor;
box on;
xlim([floor(min(timestamps(IndexRange))/0.05)*0.05
ceil(max(timestamps(IndexRange))/0.05)*0.05]);
ylim([-ceil(max(abs(Sensor4),[],"all")) ceil(max(abs(Sensor4),[],"all"))]);
xticks(floor(min(timestamps(IndexRange))/0.05)*0.05: 0.05:
ceil(max(timestamps(IndexRange))/0.05)*0.05);
yticks(-ceil(max(abs(Sensor4),[],"all")):1:ceil(max(abs(Sensor4),[],"all")));
xlabel('Time (sec)');
ylabel('Acceleration ($g$)');
legend('Original Singal', 'Reconstructed Signal', 'Location', 'southeast');
title('Orignal Signal vs AR Model Result for State 1 (Column 1) from Sensor
4');

subplot(2,1,2); % Plot for state 14 (column 651).
hold on;
plot(timestamps(IndexRange), Sensor4(IndexRange,651), 'b', 'LineWidth', 2);
% Plot the original signal.
plot(timestamps(IndexRange), Reconstruction(IndexRange-Order,5), 'r',
'LineWidth', 2); % Plot the reconstructed signal.
grid on;
grid minor;
box on;
xlim([floor(min(timestamps(IndexRange))/0.05)*0.05
ceil(max(timestamps(IndexRange))/0.05)*0.05]);
ylim([-ceil(max(abs(Sensor4),[],"all")) ceil(max(abs(Sensor4),[],"all"))]);
xticks(floor(min(timestamps(IndexRange))/0.05)*0.05: 0.05:
ceil(max(timestamps(IndexRange))/0.05)*0.05);
yticks(-ceil(max(abs(Sensor4),[],"all")):1:ceil(max(abs(Sensor4),[],"all")));
xlabel('Time (sec)');
ylabel('Acceleration ($g$)');
legend('Original Singal', 'Reconstructed Signal', 'Location', 'southeast');
title('Orignal Signal vs AR Model Result for State 14 (Column 651) from
Sensor 4');
```





Task 1-E: Calculate the residual error time series.

Initialization for a matrix to store the residual errors of each state. Each column represents a set of residual error time series. Note that each residual error time series has (NumOfPoints - Order) data points, which is different from the original signal.

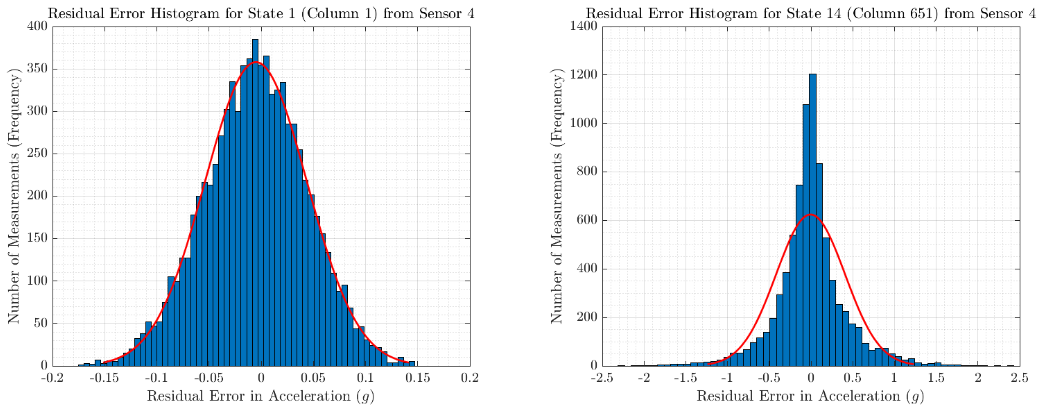
```
Error = zeros(NumOfPoints-Order, size(State,2));

% Loop over each state, calculate the residual error time series.
for i = 1:size(State,2)
    Error(:,i) = Sensor4(Order+1:NumOfPoints, (State(i)-1)*50+1) -
Reconstruction(:,i);
end
```

Task 1-F: Plot residual error histograms.

Plot residual error histograms for state 1 and state 14.

```
figure('Renderer', 'painters', 'Position', [10 10 1800 600]);

subplot(1,2,1); % Plot for state 1.
histfit(Error(:,1), 60);
grid on;
grid minor;
box on;
xlim([-0.2 0.2]);
ylim([0 400]);
xticks(-0.2:0.05:0.2);
```

```
yticks(0:50:400);
xlabel('Residual Error in Acceleration ($g$)');
ylabel('Number of Measurements (Frequency)');
title('Residual Error Histogram for State 1 (Column 1) from Sensor 4');

subplot(1,2,2); % Plot for state 14.
histfit(Error(:,5), 60);
grid on;
grid minor;
box on;
xlim([-2.5 2.5]);
ylim([0 1400]);
xticks(-2.5:0.5:2.5);
yticks(0:200:1400);
xlabel('Residual Error in Acceleration ($g$)');
ylabel('Number of Measurements (Frequency)');
title('Residual Error Histogram for State 14 (Column 651) from Sensor 4');
```



Task 1-G: Residual error normaliztion.

Calcualate the normalized residual errors for each state.

```
ErrorNormalized = (Error - mean(Error(:,1))) / std(Error(:,1));
```

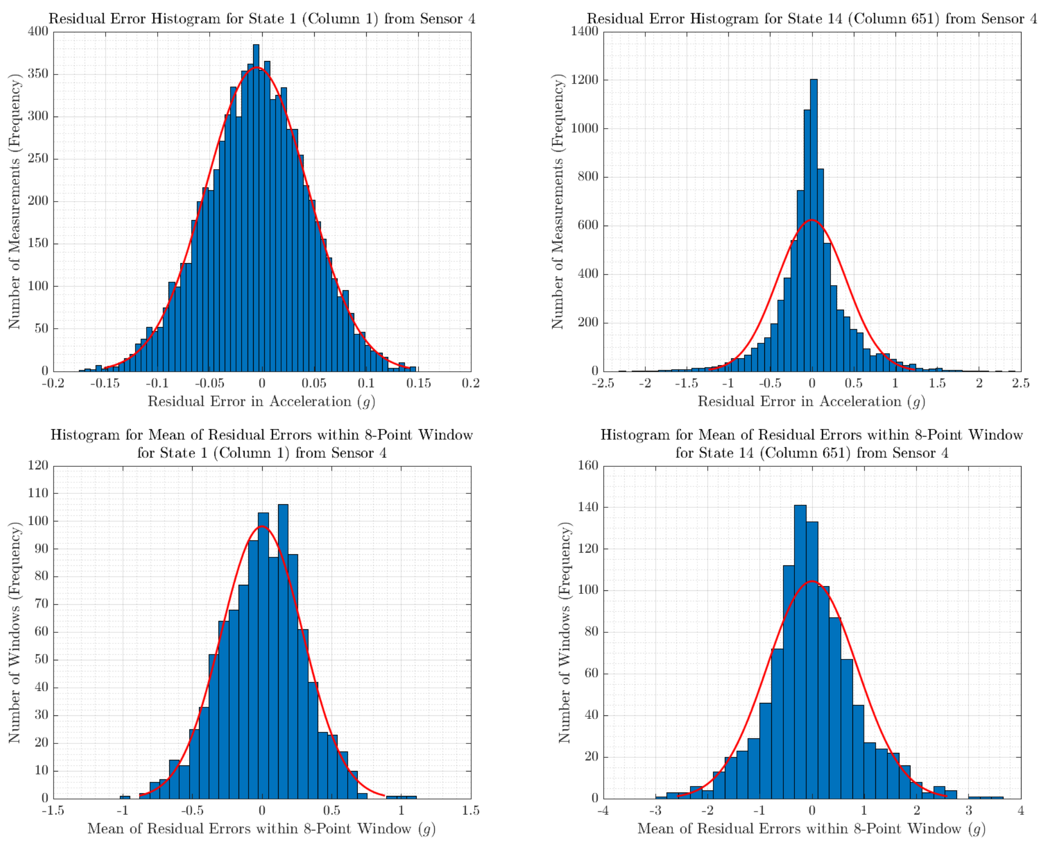
Task 1-H: X-bar control charts.

```
m = 8; % m = window size.

% Initialization for matricies to store the mean and standard deviation of
residual errors within each m-points window.
Mean = zeros(floor(size(ErrorNormalized,1)/m), size(ErrorNormalized,2));
Deviation = zeros(floor(size(ErrorNormalized,1)/m), size(ErrorNormalized,2));

% Calculate the mean and standard deviation of residual errors within each
m-points window.
for i = 1:size(ErrorNormalized,2) % Loop over each state.
    for j = 1:floor(size(ErrorNormalized,1)/m) % Loop over each window.
        Mean(j,i) = mean(ErrorNormalized((j-1)*m+1:j*m, i));
        Deviation(j,i) = std(ErrorNormalized((j-1)*m+1:j*m, i));
    end
end

% Calculate the control limits using the result for state 1 only.
s = mean(Deviation(:,1));
CL = mean(Mean(:,1));
UCL = CL + 3*s/sqrt(m); % UCL = Upper control limit.
LCL = CL - 3*s/sqrt(m); % LCL = Lower control limit.
```

Task 1-I: Plot histograms for the mean values of residual errors within each m-points window.

Plot histograms for the mean values of residual errors within each m-points window for state 1 and state 14.

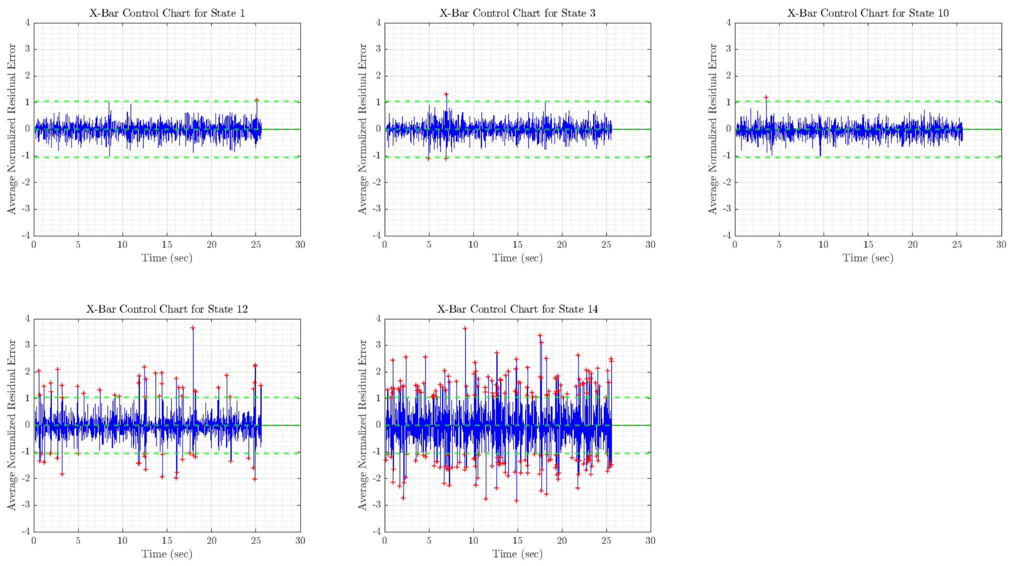
```
figure('Renderer', 'painters', 'Position', [10 10 1800 600]);

subplot(1,2,1); % Plot for state 1.
histfit(Mean(:,1), 30);
grid on;
grid minor;
box on;
xlim([-1.5 1.5]);
ylim([0 120]);
xticks(-1.5:0.5:1.5);
yticks(0:10:120);
xlabel('Mean of Residual Errors within 8-Point Window ($g$)');
ylabel('Number of Windows (Frequency)');
title({'Histogram for Mean of Residual Errors within 8-Point Window', 'for
State 1 (Column 1) from Sensor 4'});

subplot(1,2,2); % Plot for state 14.
histfit(Mean(:,5), 30);
grid on;
grid minor;
box on;
xlim([-4 4]);
ylim([0 160]);
xticks(-4:1:4);
yticks(0:20:160);
xlabel('Mean of Residual Errors within 8-Point Window ($g$)');
ylabel('Number of Windows (Frequency)');
title({'Histogram for Mean of Residual Errors within 8-Point Window', 'for
State 14 (Column 651) from Sensor 4'});
```



Task 1-J: Plot the control charts.

Create another timestamps vector for the average of each window. The timestamps are selected as the end points for each window. Also it accounts for the mean time series start at the 31st point (Order+1) of the original timestamps vector.

```
timestamps_window = timestamps(Order+m:m:Order+m*size(Mean,1));

% Find the indicies for the outliers in the [Mean] matrix.
% The indicies matrix has the same dimension as the [Mean] matrix with 1
represents outlier and 0 represents inlier.
OutlierIndicies = Mean<LCL | Mean>UCL;

% Find the values for the outliers in the [Mean] matrix.
% The non-zero values represent the outliers. The inliers are shown as 0.
OutlierValues = Mean.*OutlierIndicies;

% Plot the control charts for all the states.
figure('Renderer', 'painters', 'Position', [10 10 2000 1000]);
set(0, 'DefaultAxesFontSize', 12);
set(0, 'DefaultTextFontSize', 12);

for i = 1:size(State,2) % Loop over each state.
    subplot(2,3,i);
    hold on;
    bar(timestamps_window, Mean(:,i), 3, 'blue'); % Plot the control plot.
    yline(CL, 'g--', 'LineWidth', 2); % Plot the mean value for the average
normalized residual error.
    yline(UCL, 'g--', 'LineWidth', 2); % Plot the upper control limit.
    yline(LCL, 'g--', 'LineWidth', 2); % Plot the lower control limit.

    for j = 1:size(Mean,1) % Loop over each value in the average normalized
residual error vector.
        if OutlierValues(j,i) ~= 0 % Check if the value is an outlier (non-
zero).
            plot(timestamps_window(j), OutlierValues(j,i), '+',
'MarkerSize', 5, 'MarkerEdgeColor', 'r', 'LineWidth', 1); % Plot the outlier.
        end
    end

    grid on;
    grid minor;
    box on;
    xlim([0 ceil(max(timestamps)/5)*5]);
    ylim([-4 4]);
    xticks(0:5:ceil(max(timestamps)/5)*5);
    yticks(-4:1:4);
    xlabel('Time (sec)');
    ylabel('Average Normalized Residual Error');
    title(sprintf(['X-Bar Control Chart for State ', num2str(State(i))]));
end
```

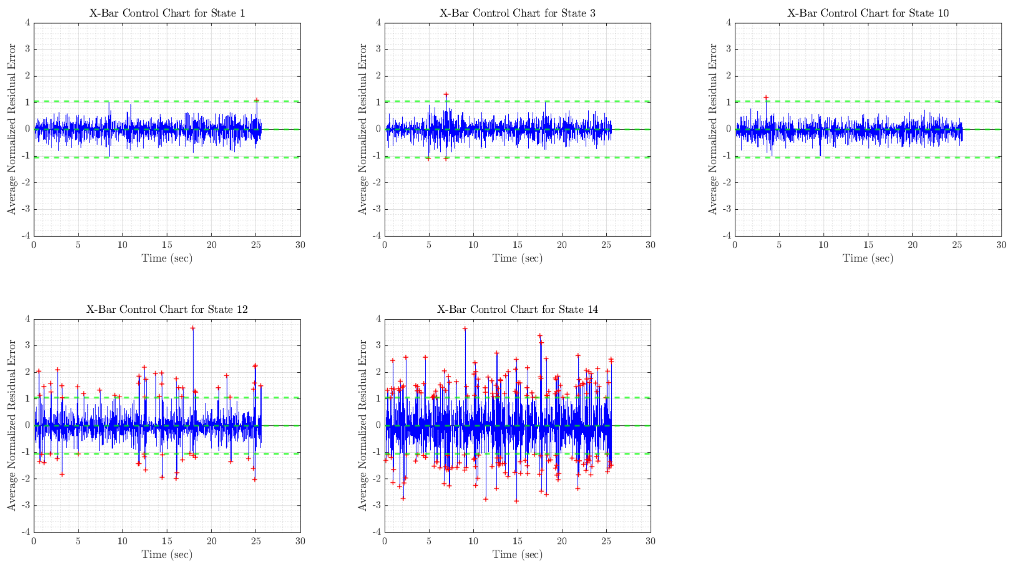


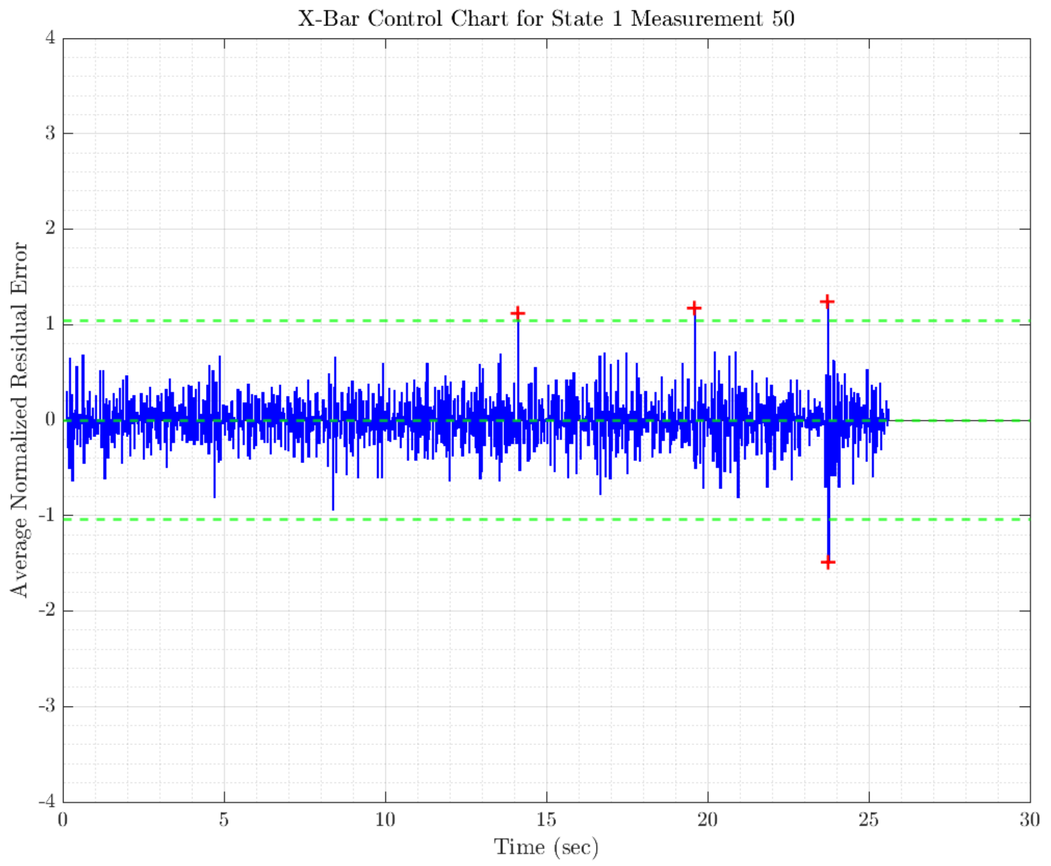
Task 2: False positive study for state 1 measurement 50.

Create [X] matrix.

```
X_State50 = zeros(NumOfPoints-Order, Order);
for j = 1:Order
    X_State50(:,j) = Sensor4(j:j+NumOfPoints-Order-1, 50);
end

Reconstruction_State50 = X_State50*a; % Calculate reconstructed signal.
Error_State50 = Sensor4(Order+1:NumOfPoints, 50) - Reconstruction_State50; %
Calculate residual error time series.
ErrorNormalized_State50 = (Error_State50 - mean(Error(:,1))) /
std(Error(:,1)); % Calcualate the normalized residual errors using the mean
and standard deviation from state 1.

% Initialization for vectors to store the mean of residual errors within
each m-points window.
Mean_State50 = zeros(floor(size(ErrorNormalized_State50,1)/m), 1);

% Calculate the mean of residual errors within each m-points window.
for j = 1:floor(size(ErrorNormalized_State50)/m) % Loop over each window.
    Mean_State50(j) = mean(ErrorNormalized_State50((j-1)*m+1:j*m));
end

% Find the indicies for the outliers in the [Mean_State50] matrix.
% The indicies matrix has the same dimension as the [Mean_State50] matrix
with 1 represents outlier and 0 represents inlier.
OutlierIndicies_State50 = Mean_State50<LCL | Mean_State50>UCL;

% Find the values for the outliers in the [Mean_State50] matrix.
% The non-zero values represent the outliers. The inliers are shown as 0.
OutlierValues_State50 = Mean_State50.*OutlierIndicies_State50;

% Plot the control chart.
figure('Renderer', 'painters', 'Position', [10 10 1200 900]);
set(0, 'DefaultAxesFontSize', 15);
set(0, 'DefaultTextFontSize', 15);
hold on;

bar(timestamps_window, Mean_State50, 3, 'blue'); % Plot the control plot.
yline(CL, 'g--', 'LineWidth', 2); % Plot the mean value for the averagce
normalized residual error.
yline(UCL, 'g--', 'LineWidth', 2); % Plot the upper control limit.
yline(LCL, 'g--', 'LineWidth', 2); % Plot the lower control limit.

for j = 1:size(Mean_State50,1) % Loop over each value in the average
normalized residual error vector.
    if OutlierValues_State50(j) ~= 0 % Check if the value is an outlier (non-
zero).
        plot(timestamps_window(j), OutlierValues_State50(j), '+',
'MarkerSize', 10, 'MarkerEdgeColor', 'r', 'LineWidth', 2); % Plot the
outlier.
    end
end

grid on;
grid minor;
```

```
box on;
xlim([0 ceil(max(timestamps)/5)*5]);
ylim([-4 4]);
xticks(0:5:ceil(max(timestamps)/5)*5);
yticks(-4:1:4);
xlabel('Time (sec)');
ylabel('Average Normalized Residual Error');
title('X-Bar Control Chart for State 1 Measurement 50');
```





Published with MATLAB® R2023b