Table of Contents


```
% Ruipu Ji
% SE 265
% Homework #9

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 the data. ---

```
load('4-Story Structure Data/data3SS2009.mat'); % Load the data file.
dataset = double(dataset); % Convert the data into double precision.
TestingData = squeeze(dataset(:,5,:)); % TestingData = Data from channel 5
 (acceleration response at level-4).
% squeeze() is to remove the dimension with length of 1.

% Generate the time vector. -----------------------------------------------
SamplingFrequency = 320; % SamplingFrequency = Sampling frequency in Hz.
timestamps = 0: 1/SamplingFrequency : (size(TestingData,1)-1)/
SamplingFrequency; % Create the time vector.
```

Task 2.a.

Calculate the coefficients for a 5-th order linear AR model. ------------

```
AR5_Coefficients = zeros(5, size(TestingData,2)); % Initialization.

for TestIndex = 1:size(TestingData,2) % Loop over all the tests.
    % First create a temperory matrix to store the result including the 1 in
```

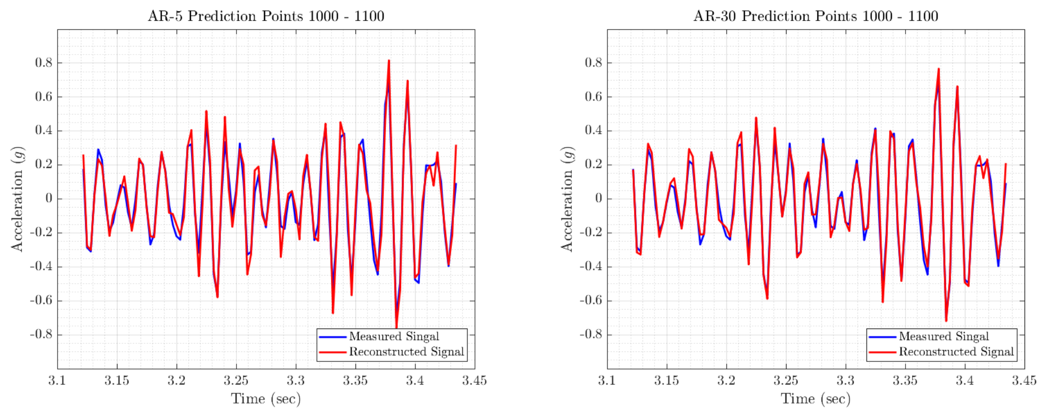
```
the first column.
    % lpc(x, p) finds the coefficients of a p-th order linear predictor and
returns to a 1-D row vector.
    % If x is a 2-D matrix, then the function will treat each column as a
separate channel.
    % The input x must be in double precision.
    Coefficients_temp = lpc(TestingData(:,TestIndex),5);

    % Remove the 1 in the first column and store the coefficient vector in
the final output matrix.
    % Note that the AR coefficients should be in reverse order and of
opposite sign.
    Coefficients_temp(:,1) = [];
    AR5_Coefficients(:,TestIndex) = -flipud(Coefficients_temp');
end

% Calculate the coefficients for a 30-th order linear AR model. -----------
AR30_Coefficients = zeros(30, size(TestingData,2)); % Initialization.

for TestIndex = 1:size(TestingData,2) % Loop over all the tests.
    % 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(TestingData(:,TestIndex),30);

    % Remove the 1 in the first column and store the coefficient vector in
the final output matrix.
    % Note that the AR coefficients should be in reverse order and of
opposite sign.
    Coefficients_temp(:,1) = [];
    AR30_Coefficients(:,TestIndex) = -flipud(Coefficients_temp');
end
```

Task 2.b.

Calculate the reconstructed signal of the 1st time history from AR-5 model. -- Create [X] matrix.

```
AR5_X = zeros(size(TestingData,1)-5, 5);
for i = 1:5
    AR5_X(:,i) = TestingData(i:i+size(TestingData,1)-5-1, 1);
end

% Calcualte the reconstructed signal of the 1st time history.
AR5_Reconstruction = AR5_X * AR5_Coefficients(:,1);

% Calculate the reconstructed signal of the 1st time history from AR-30
model.
% -------------------------------------------------------------------------
```

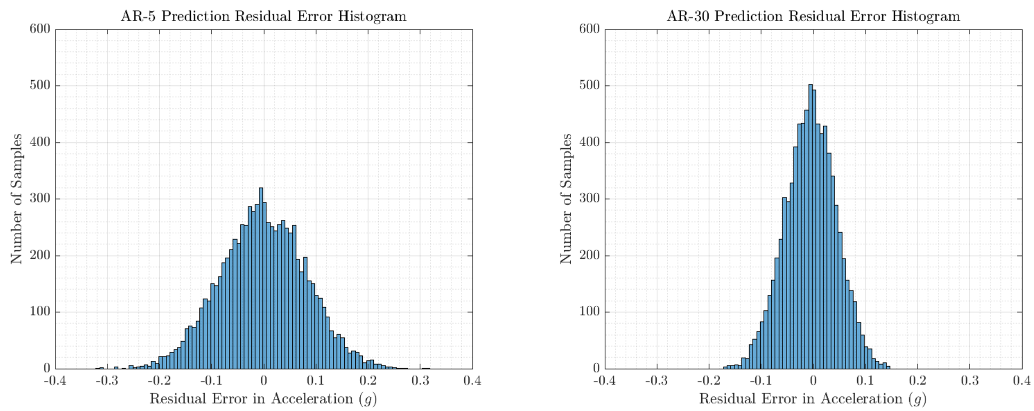
```
% Create [X] matrix.
AR30_X = zeros(size(TestingData,1)-30, 30);
for i = 1:30
    AR30_X(:,i) = TestingData(i:i+size(TestingData,1)-30-1, 1);
end

% Calcualte the reconstructed signal of the 1st time history.
AR30_Reconstruction = AR30_X * AR30_Coefficients(:,1);

% Plot the measured and reconstructed time history. -----------------------
figure('Renderer', 'painters', 'Position', [10 10 1800 600]);

subplot(1,2,1); % Plot for AR-5 model.
hold on;
plot(timestamps(1000:1100), TestingData(1000:1100,1), 'b', 'LineWidth', 2);
% Plot the measured signal.
plot(timestamps(1000:1100), AR5_Reconstruction(995:1095,1), 'r',
'LineWidth', 2); % Plot the reconstructed signal.
grid on;
grid minor;
box on;
xlim([3.1 3.45]);
ylim([-1 1]);
xticks(3.1:0.05:3.45);
yticks(-0.8:0.2:0.8);
xlabel('Time (sec)');
ylabel('Acceleration ($g$)');
legend('Measured Singal', 'Reconstructed Signal', 'Location', 'southeast');
title('AR-5 Prediction Points 1000 - 1100');
hold off;

subplot(1,2,2); % Plot for AR-30 model.
hold on;
plot(timestamps(1000:1100), TestingData(1000:1100,1), 'b', 'LineWidth', 2);
% Plot the measured signal.
plot(timestamps(1000:1100), AR30_Reconstruction(970:1070,1), 'r',
'LineWidth', 2); % Plot the reconstructed signal.
grid on;
grid minor;
box on;
xlim([3.1 3.45]);
ylim([-1 1]);
xticks(3.1:0.05:3.45);
yticks(-0.8:0.2:0.8);
xlabel('Time (sec)');
ylabel('Acceleration ($g$)');
legend('Measured Singal', 'Reconstructed Signal', 'Location', 'southeast');
title('AR-30 Prediction Points 1000 - 1100');
hold off;
```



Task 3.a.

Calculate residual errors for AR-5 and AR-30 model.

```
Re5 = TestingData(6:size(TestingData,1),1) - AR5_Reconstruction;
Re30 = TestingData(31:size(TestingData,1),1) - AR30_Reconstruction;

% Plot the histogram of residual errors.
figure('Renderer', 'painters', 'Position', [10 10 1800 600]);

subplot(1,2,1); % Plot for AR-5 model.
h = histogram(Re5, 90); % Generate a 90-bin histogram for AR-5 model.
grid on;
grid minor;
box on;
xlim([-0.4 0.4]);
ylim([0 600]);
xticks(-0.4:0.1:0.4);
yticks(0:100:600);
xlabel('Residual Error in Acceleration ($g$)');
ylabel('Number of Samples');
title('AR-5 Prediction Residual Error Histogram');

subplot(1,2,2); % Plot for AR-30 model.
histogram(Re30, h.BinEdges); % Generate a histogram with the same bins as
the AR-5 model.
grid on;
grid minor;
box on;
xlim([-0.4 0.4]);
ylim([0 600]);
xticks(-0.4:0.1:0.4);
yticks(0:100:600);
xlabel('Residual Error in Acceleration ($g$)');
ylabel('Number of Samples');
title('AR-30 Prediction Residual Error Histogram');
```

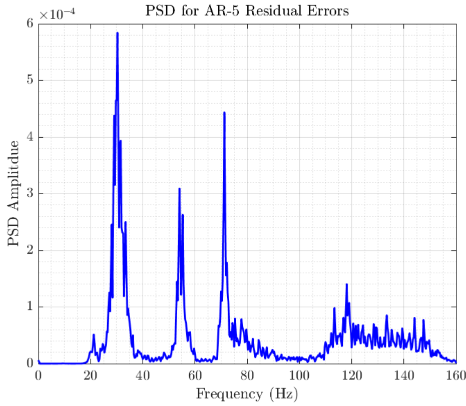


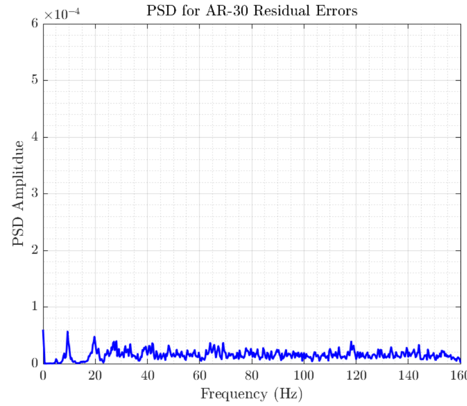
Task 3.b.

Calculate the power spectral density (PSD) of residual errors for AR-5 and AR-30 model. Hanning window with 8 averages and zero overlap is applied here.

```
[AR5_psd, AR5_f] = pwelch(Re5, hann(size(Re5,1)/8), 0, [],
SamplingFrequency);
[AR30_psd, AR30_f] = pwelch(Re30, hann(size(Re30,1)/8), 0, [],
SamplingFrequency);

% Plot the power spectral density.
figure('Renderer', 'painters', 'Position', [10 10 1800 600]);

subplot(1,2,1); % Plot for AR-5 model.
plot(AR5_f, AR5_psd, 'Color', 'b', 'LineWidth', 2);
grid on;
grid minor;
box on;
xlim([0 160]);
ylim([0 6e-4]);
xticks(0:20:160);
yticks(0:1e-4:6e-4);
xlabel('Frequency (Hz)');
ylabel('PSD Amplitdue');
title('PSD for AR-5 Residual Errors');

subplot(1,2,2); % Plot for AR-30 model.
plot(AR30_f, AR30_psd, 'Color', 'b', 'LineWidth', 2);
grid on;
grid minor;
box on;
xlim([0 160]);
ylim([0 6e-4]);
xticks(0:20:160);
yticks(0:1e-4:6e-4);
xlabel('Frequency (Hz)');
ylabel('PSD Amplitdue');
title('PSD for AR-30 Residual Errors');
```

Warning: Rounding order to nearest integer.
Warning: Rounding order to nearest integer.





Task 4.a.

```matlab
Train_u5 = zeros(5,225); % A training set of the AR(5) model parameters that correspond to the odd-numbered undamaged cases (cases 1, 3, 5...449).
Test_u5 = zeros(5,225); % A testing set of AR(5) model parameters corresponding to even undamaged cases (cases 2, 4, 6...450).
Test_d5 = zeros(5,400); % A testing set of the AR(5) model parameters that correspond to the all damaged cases (cases 451, 452, 453...850).
Train_d5 = zeros(5,200); % A training set of the AR(5) model parameters that correspond to the odd-numbered damaged cases (cases 451, 453...849).
Test_svm_d5 = zeros(5,200); % A testing set of the AR(5) model parameters that correspond to the even-numbered damaged cases (cases 452, 454, 456...850).

for i = 1:225
    Train_u5(:,i) = AR5_Coefficients(:,2*i-1);
    Test_u5(:,i) = AR5_Coefficients(:,2*i);
end

Test_d5 = AR5_Coefficients(:,451:850);

for i = 1:200
    Train_d5(:,i) = AR5_Coefficients(:,450+2*i-1);
    Test_svm_d5(:,i) = AR5_Coefficients(:,450+2*i);
end
```

Task 4.b.

```matlab
Train_u30 = zeros(30,225); % A training set of the AR(30) model parameters that correspond to the odd-numbered undamaged cases (cases 1, 3, 5...449).
Test_u30 = zeros(30,225); % A testing set of AR(30) model parameters corresponding to even undamaged cases (cases 2, 4, 6...450).
Test_d30 = zeros(30,400); % A testing set of the AR(30) model parameters that correspond to the all damaged cases (cases 451, 452, 453...850).
Train_d30 = zeros(30,200); % A training set of the AR(30) model parameters
```

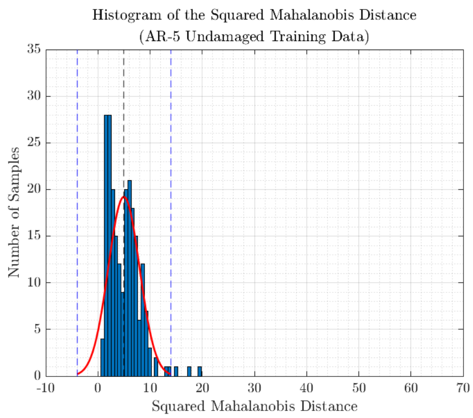
```
that correspond to the odd-numbered damaged cases (cases 451, 453...849).
Test_svm_d30 = zeros(30,200); % A testing set of the AR(30) model parameters
that correspond to the even-numbered damaged cases (cases 452, 454,
456...850).

for i = 1:size(Train_u30,2)
    Train_u30(:,i) = AR30_Coefficients(:,2*i-1);
    Test_u30(:,i) = AR30_Coefficients(:,2*i);
end

Test_d30 = AR30_Coefficients(:,451:850);

for i = 1:size(Train_d30,2)
    Train_d30(:,i) = AR30_Coefficients(:,450+2*i-1);
    Test_svm_d30(:,i) = AR30_Coefficients(:,450+2*i);
end
```

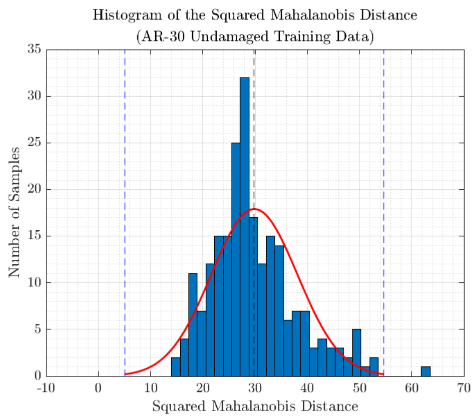
Task 5.a.

Calculate the squared Mahalanobis distances for the undamaged training data.

```
MD_train_u5 = mahal(Train_u5', Train_u5');
MD_train_u30 = mahal(Train_u30', Train_u30');

% Plot the 30-bin histogram of the squared Mahalanobis distances
corresponding to the undamaged training data.
figure('Renderer', 'painters', 'Position', [10 10 1800 600]);

subplot(1,2,1); % Plot for AR-5 model.
hold on;
histfit(MD_train_u5, 30);
xline(mean(MD_train_u5), 'k--', 'LineWidth', 1);
xline(mean(MD_train_u5) + 3*std(MD_train_u5), 'b--', 'LineWidth', 1);
xline(mean(MD_train_u5) - 3*std(MD_train_u5), 'b--', 'LineWidth', 1);
grid on;
grid minor;
box on;
xlim([-10 70]);
ylim([0 35]);
xticks(-10:10:70);
yticks(0:5:35);
xlabel('Squared Mahalanobis Distance');
ylabel('Number of Samples');
title('Histogram of the Squared Mahalanobis Distance', '(AR-5 Undamaged
Training Data)');
hold off;

subplot(1,2,2); % Plot for AR-30 model.
hold on;
histfit(MD_train_u30, 30);
xline(mean(MD_train_u30), 'k--', 'LineWidth', 1);
xline(mean(MD_train_u30) + 3*std(MD_train_u30), 'b--', 'LineWidth', 1);
xline(mean(MD_train_u30) - 3*std(MD_train_u30), 'b--', 'LineWidth', 1);
```

```
grid on;
grid minor;
box on;
xlim([-10 70]);
ylim([0 35]);
xticks(-10:10:70);
yticks(0:5:35);
xlabel('Squared Mahalanobis Distance');
ylabel('Number of Samples');
title('Histogram of the Squared Mahalanobis Distance', '(AR-30 Undamaged
Training Data)');
hold off;
```





Task 5.b.

1. Calculate the squared Mahalanobis distances for the undamaged testing data.

```
MD_test_u5 = mahal(Test_u5', Train_u5');
MD_test_u30 = mahal(Test_u30', Train_u30');

% 2. Calculate the number of false-positives for AR-5 and AR-30 model.
FP_u5 = 0;
FP_u30 = 0;

for i = 1:size(MD_test_u5,1)
    if MD_test_u5(i) < mean(MD_train_u5)-3*std(MD_train_u5) || MD_test_u5(i)
> mean(MD_train_u5)+3*std(MD_train_u5)
        FP_u5 = FP_u5 + 1;
    end

    if MD_test_u30(i) < mean(MD_train_u30)-3*std(MD_train_u30) ||
MD_test_u30(i) > mean(MD_train_u30)+3*std(MD_train_u30)
        FP_u30 = FP_u30 + 1;
    end
end

% 3. Calculate the squared Mahalanobis distances for all damaged data.
MD_test_d5 = mahal(Test_d5', Train_u5');
MD_test_d30 = mahal(Test_d30', Train_u30');
```

```
% 4. Calculate the number of false-negatives for AR-5 and AR-30 model.
FN_d5 = 0;
FN_d30 = 0;

for i = 1:size(MD_test_d5,1)
    if MD_test_d5(i) > mean(MD_train_u5)-3*std(MD_train_u5) && MD_test_d5(i)
< mean(MD_train_u5)+3*std(MD_train_u5)
        FN_d5 = FN_d5 + 1;
    end

    if MD_test_d30(i) > mean(MD_train_u30)-3*std(MD_train_u30) &&
MD_test_d30(i) < mean(MD_train_u30)+3*std(MD_train_u30)
        FN_d30 = FN_d30 + 1;
    end
end

% 5. Calculate the number of true-positives and true-negatives for AR-5 and
AR-30 model.
TP_5 = size(MD_test_d5,1) - FN_d5;
TP_30 = size(MD_test_d30,1) - FN_d30;

TN_5 = size(MD_test_u5,1) - FP_u5;
TN_30 = size(MD_test_u30,1) - FP_u30;

% 6. Display the confusion matrix in the command window.
helperCommandWindowDisplay(0, 5, TN_5, FN_d5, TP_5, FP_u5);
helperCommandWindowDisplay(0, 30, TN_30, FN_d30, TP_30, FP_u30);
```

Unsupervised Learning Confusion Matrix AR(5) Model

	Actual Undamaged	*Actual Damaged*	*Total*
Predicted undamaged	*222*	*145*	*367*
Predicted damaged	*3*	*255*	*258*
Total	*225*	*400*	*NaN*

True Positive (damage) Rate: 63.8%
True Negative (undamaged) Rate: 98.7%
False Positive Rate: 1.3%
False Negative Rate: 36.2%
Overall Classification Accuracy: 76.3%

_

Unsupervised Learning Confusion Matrix AR(30)

```
Model
_____________________________________________________________________________
_
                         Actual Undamaged    Actual Damaged    Total
                         ________________    ______________    _____

    Predicted undamaged         206                 70           276
    Predicted damaged            19                330           349
    Total                       225                400           NaN

True Positive (damage) Rate: 82.5%
True Negative (undamaged) Rate: 91.6%
False Positive Rate: 8.4%
False Negative Rate: 17.5%
Overall Classification Accuracy: 85.8%
```

Task 6.

1. Combine undamaged training data and damaged training data into one matrix.

```
svm_train5 = [Train_u5, Train_d5];
svm_train30 = [Train_u30, Train_d30];

% 2. Combine undamaged testing data and damaged testing data into one matrix.
svm_test5 = [Test_u5, Test_svm_d5];
svm_test30 = [Test_u30, Test_svm_d30];

% 3. Create a binary classification label row vector (0 = undamaged; 1 =
damaged).
Class1 = ones(1, size(svm_train5,2));
Class1(1, 1:size(Train_u5,2)) = 0;

% 4. Train the support vector machine classifier.
Model_AR5 = fitclinear(svm_train5', Class1);
Model_AR30 = fitclinear(svm_train30', Class1);

% 5. Classify the testing data.
Prediction_AR5 = predict(Model_AR5, svm_test5');
Prediction_AR30 = predict(Model_AR30, svm_test30');

% 6. Calculate the true positives, false positives, true negatives and false
negatives.
TP_svm5 = sum(Prediction_AR5' .* Class1);
FP_svm5 = sum(Prediction_AR5) - TP_svm5;
TN_svm5 = size(Test_u5,2) - FP_svm5;
FN_svm5 = size(Test_svm_d5,2) - TP_svm5;

TP_svm30 = sum(Prediction_AR30' .* Class1);
FP_svm30 = sum(Prediction_AR30) - TP_svm30;
```

```
TN_svm30 = size(Test_u30,2) - FP_svm30;
FN_svm30 = size(Test_svm_d30,2) - TP_svm30;

% 7. Display the confusion matrix in the command window.
helperCommandWindowDisplay(1, 5, TN_svm5, FN_svm5, TP_svm5, FP_svm5);
helperCommandWindowDisplay(1, 30, TN_svm30, FN_svm30, TP_svm30, FP_svm30);
```

Supervised Learning Confusion Matrix AR(5) Model

	Actual Undamaged	Actual Damaged	Total
Predicted undamaged	225	55	280
Predicted damaged	0	145	145
Total	225	200	NaN

True Positive (damage) Rate: 72.5%
True Negative (undamaged) Rate: 100.0%
False Positive Rate: 0.0%
False Negative Rate: 27.5%
Overall Classification Accuracy: 87.1%

Supervised Learning Confusion Matrix AR(30) Model

	Actual Undamaged	Actual Damaged	Total
Predicted undamaged	221	22	243
Predicted damaged	4	178	182
Total	225	200	NaN

True Positive (damage) Rate: 89.0%
True Negative (undamaged) Rate: 98.2%
False Positive Rate: 1.8%
False Negative Rate: 11.0%
Overall Classification Accuracy: 93.9%

Published with MATLAB® R2023b