CH5019 Mathematical Foundations for Data Science

Course Project

Gowthamaan

ED23S037

Q1 - Linear Regression

Implementation

The question asked is to perform Linear Regression using Least Squares, Least Median of Squares and Least Trimmed Squares. All the above algorithms are implemented in MATLAB. The `matlab\Q1` folder has all the MATLAB files related to this question.

Least Squares

```
%% Ordinary Least Squares using Gradient Descent
function [beta, cost_history] = ordinaryLeastSquares(X, y, alpha, epochs)
    %% Input Arguments
    %   X: Input features (n x m matrix, where n is the number of data points
    %      and m is the number of features)
    %   y: Target variable (n x 1 vector)
    %   alpha: Learning rate (scalar)
    %   epochs: Number of epochs (scalar)

    %% Output Arguments
    %   beta: Learned or Estimated parameters of the model(m+1 x 1 vector)
    %   cost_history: Cost history for each iteration (epochs x 1 vector)

    %% Initialize parameters
    n = length(y);                  % Number of data points
    m = size(X, 2);                 % Number of features
    beta = zeros(m + 1, 1);         % Initialize parameters
    cost_history = zeros(epochs, 1); % Store cost for each iteration
    X = [ones(n, 1) X];             % Add bias term to X

    %% Gradient Descent
    for epoch = 1:epochs
        % Compute y_hat and cost
        y_hat = X * beta;             % Hypothesis: y_hat = X * theta
        cost = sum((y_hat - y).^2) / (2 * n); % Ordinary Least Squares
        cost_history(epoch) = cost; % Store cost for plotting

        % Compute gradients
        grad = (X' * (y_hat - y)) / n;

        % Update parameters
        beta = beta - alpha * grad;
    end
end
```

Least Median of Squares

```
%% Least Median Squares using Gradient Descent
function [beta, cost_history] = leastMedianSquares(X, y, alpha, epochs)
    %% Input Arguments
    %   X: Input features (n x m matrix, where n is the number of data points
    %      and m is the number of features)
    %   y: Target variable (n x 1 vector)
    %   alpha: Learning rate (scalar)
    %   epochs: Number of epochs (scalar)

    %% Output Arguments
    %   beta: Learned or Estimated parameters of the model(m+1 x 1 vector)
    %   cost_history: Cost history for each iteration (epochs x 1 vector)

    %% Initialize parameters
    n = length(y);                  % Number of data points
    m = size(X, 2);                 % Number of features
    beta = zeros(m + 1, 1);         % Initialize parameters
    cost_history = zeros(epochs, 1); % Store cost for each iteration
    X = [ones(n, 1) X];             % Add bias term to X

    %% Gradient Descent
    for epoch = 1:epochs
        % Compute y_hat and cost
        y_hat = X * beta;             % Hypothesis: y_hat = X * theta
        e = y_hat - y;                % Residual
        squared_e = e.^2;             % Squared Residual
        cost = median(squared_e)/ (2 * n);      % Least Median Squares
        cost_history(epoch) = cost; % Store cost for plotting

        % Compute gradients (Technically Subgradient)
        grad = (2/n) * (X' * (e.* xsign(squared_e, cost)));

        % Update parameters
        beta = beta - alpha * grad;
    end
end
```

Least Trimmed Squares

```
%% Least Trimmed Squares using Gradient Descent
function [beta, cost_history] = leastTrimmedSquares(X, y, alpha, epochs)
    %% Input Arguments
    %   X: Input features (n x m matrix, where n is the number of data points
    %      and m is the number of features)
    %   y: Target variable (n x 1 vector)
```

```
    %   alpha: Learning rate (scalar)
    %   epochs: Number of epochs (scalar)

    %% Output Arguments
    %   beta: Learned or Estimated parameters of the model(m+1 x 1 vector)
    %   cost_history: Cost history for each iteration (epochs x 1 vector)

    %% Initialize parameters
    n = length(y);                   % Number of data points
    q = ceil((n/2)+1);              % Number of samples to consider for computing loss
    m = size(X, 2);                  % Number of features
    beta = zeros(m + 1, 1);         % Initialize parameters
    cost_history = zeros(epochs, 1); % Store cost for each iteration
    X = [ones(n, 1) X];              % Add bias term to X
    index_column = (1:n)';           % Indexing Column

    %% Gradient Descent
    for epoch = 1:epochs
        % Compute y_hat and cost
        y_hat = X * beta;              % Hypothesis: y_hat = X * theta
        e = y_hat - y;                 % Residual

        e_indexed = [e.^2, index_column]; % Add index column (Useful
                                  while calculating the gradients)
        e_sorted = sortrows(e_indexed, 1); % Ordered squared residuals
        e_sampled = e_sorted(1:q, :); % Consider the first q samples
        cost = sum(e_sampled(:, 1))/ (2 * q); % Least Trimmed Squares
        cost_history(epoch) = cost; % Store cost for plotting

        % Compute gradients
        Xs = X(e_sampled(:, 2),:); % Sampled Input
        es = e(e_sampled(:, 2)); % Sampled Error
        grad = (Xs' * (es)) / q;

        % Update parameters
        beta = beta - alpha * grad;
    end
end
```

A few helper functions are also defined and used in the above functions which are in the `matlab\Q1` folder.

Synthetic Data

As per the question, synthetic data is generated and the models are evaluated on this data.

The following is the result of that analysis,

Parameters	OLS	LMS	LTS
MSE	0.034016	0.01881	0.26578
RB	6.7453	8.1251	3.9865
MAD	18.367	22.303	10.385

From the above, we can observe the following:

- LMS excels in terms of the **lowest MSE**, which implies better accuracy in terms of squared error minimization, but it has the highest RB and MAD, indicating **higher bias** and **less consistency**.
- LTS shows the **least bias** (lowest RB) and **highest consistency** (lowest MAD), making it robust in terms of bias and deviation, but it suffers from a high MSE.
- OLS stands in the middle ground with moderate values for MSE, RB, and MAD, indicating a **balanced performance** but **not excelling in any specific metric** of comparison.

These inferences suggest that while **LMS minimizes squared errors effectively, LTS is better for minimizing bias and maintaining consistency**. The choice between these methods depends on the specific requirements of the problem, whether minimizing errors, reducing bias, or ensuring consistent performance is the priority.

Real Dataset - Medical Insurance

The medical insurances dataset in the `dataset` folder is used in this question. Linear regression is carried out to predict the medical charges for a person based on the regressors.

All the above three methods are used on the data with MSE as the comparison metric. The given data also has categorical variables like gender, religion, smoker or not which are encoded into numerical values. The following is the result of the analysis,

Parameters	OLS	LMS	LTS
MSE	3.9839e+07	5.0851e+07	2.664e+08

Based on the given table, we can observe the following:

- OLS demonstrates the **best performance** with the lowest MSE, suggesting that it is the most accurate method in terms of minimizing squared errors.
- LMS has a higher MSE than OLS, indicating that it is less accurate than OLS but still performs better than LTS.

- LTS has the highest MSE, suggesting that it is the least accurate method among the three in terms of minimizing squared errors.

Given these results, OLS is the preferred method in this task when the goal is to achieve the lowest mean square error. LMS, while not as effective as OLS, still provides better performance than LTS. LTS, with the highest MSE, is the least desirable method in this context for minimizing squared errors.

Question - 1 (PART- II)

```
% Clear workspace and command window
clear; clc; close all;
% For reproducibility
rng(13);
```

Given Data

```
N = 20;  % Number of Observations
R = 200; % Number of Realizations
beta = [0;3;5]; % True Beta
```

Initialize variables

```
beta_estimates_ols = zeros(R, 3); % Store parameter estimates
beta_estimates_lms = zeros(R, 3);
beta_estimates_lts = zeros(R, 3);
loss_ols = zeros(R, 1); % Store final cost
loss_lms = zeros(R, 1);
loss_lts = zeros(R, 1);
alpha = 0.01; % Learning rate
epochs = [200, 150, 800]; % Epochs or Number of Iterations
```

Perform Regression for R realizations

```
for r = 1:R
    % Generate Data
    X1 = randn(N, 1);
    X2 = randn(N, 1);
    E = randn(N, 1);
    y = beta(2)*X1 + beta(3)*X2 + E;
    X = [X1 X2];

    % Perform Regression
    [beta_hat_ols, cost_history_ols] = ordinaryLeastSquares(X, y, alpha, epochs(1));
    [beta_hat_lms, cost_history_lms] = leastMedianSquares(X, y, alpha, epochs(2));
    [beta_hat_lts, cost_history_lts] = leastTrimmedSquares(X, y, alpha, epochs(3));

    % Record final variables value
    beta_estimates_ols(r, :) = beta_hat_ols';
    loss_ols(r) = cost_history_ols(end);

    beta_estimates_lms(r, :) = beta_hat_lms';
    loss_lms(r) = cost_history_lms(end);

    beta_estimates_lts(r, :) = beta_hat_lts';
    loss_lts(r) = cost_history_lts(end);
end
```

Best beta across realizations

```
% Find the realization with the minimum sum of squared residuals
[min_ssr_ols, best_parameter_ols] = min(loss_ols);
[min_ssr_lms, best_parameter_lms] = min(loss_lms);
[min_ssr_lts, best_parameter_lts] = min(loss_lts);
fprintf('Best realization Index - OLS: %d\n\n', best_parameter_ols);
```

```
Best realization Index - OLS: 116
```

```
fprintf('Best realization Index - LMS: %d\n\n', best_parameter_lms);
```

```
Best realization Index - LMS: 179
```

```
fprintf('Best realization Index - LTS: %d\n\n', best_parameter_lts);
```

```
Best realization Index - LTS: 88
```

```
best_estimates = [beta_estimates_ols(best_parameter_ols, :)' beta_estimates_lms(best_parameter_
% Display best parameters
% Model parameter names
parameters = {'beta0', 'beta1', 'beta2'};
methods = {'OLS', 'LMS', 'LTS'};
TP = table(best_estimates(:, 1), best_estimates(:, 2), best_estimates(:, 3), 'VariableNames', m
TPD = table(parameters', TP, 'VariableNames', {'Parameters', 'Optimum'});
disp(TPD);
```

```
    Parameters                Optimum
                     OLS          LMS          LTS
    _________    ____________________________________

     'beta0'      0.0076753    -0.03846    -0.32516
     'beta1'         2.7235      2.8845      2.3579
     'beta2'         4.5065      4.8435      4.1158
```

```
TL = table(methods', [min_ssr_ols, min_ssr_lms, min_ssr_lts]', 'VariableNames', {'Methods', 'Lo
TLD = table(TL, 'VariableNames', {'Minimum_Loss_across_R'});
disp(TLD);
```

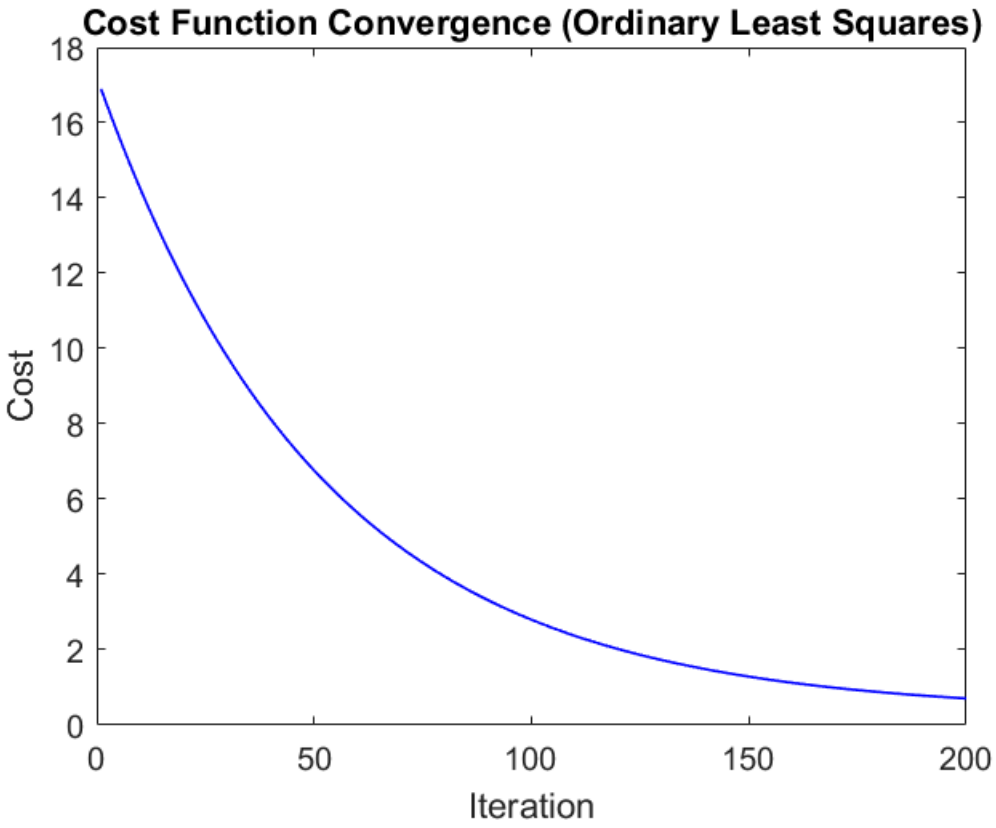
```
    Minimum_Loss_across_R
    Methods      Loss
    ____________________

     'OLS'          0.53
     'LMS'     0.0013857
     'LTS'       0.15288
```

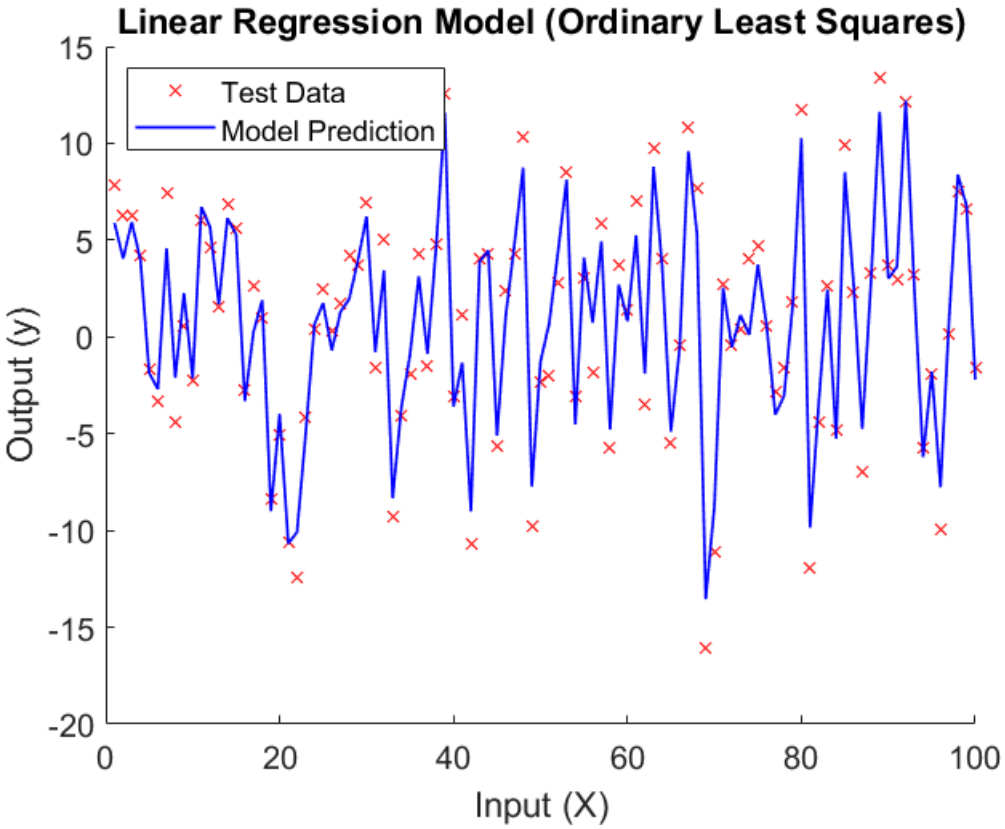
Model Testing

```
% Generate Test Data
rng(27); % Change reproducibility to generate new data
X1 = randn(N, 1);
X2 = randn(N, 1);
E = randn(N, 1);
y = beta(2)*X1 + beta(3)*X2 + E;
X = [X1 X2];
```

Ordinary Least Squares

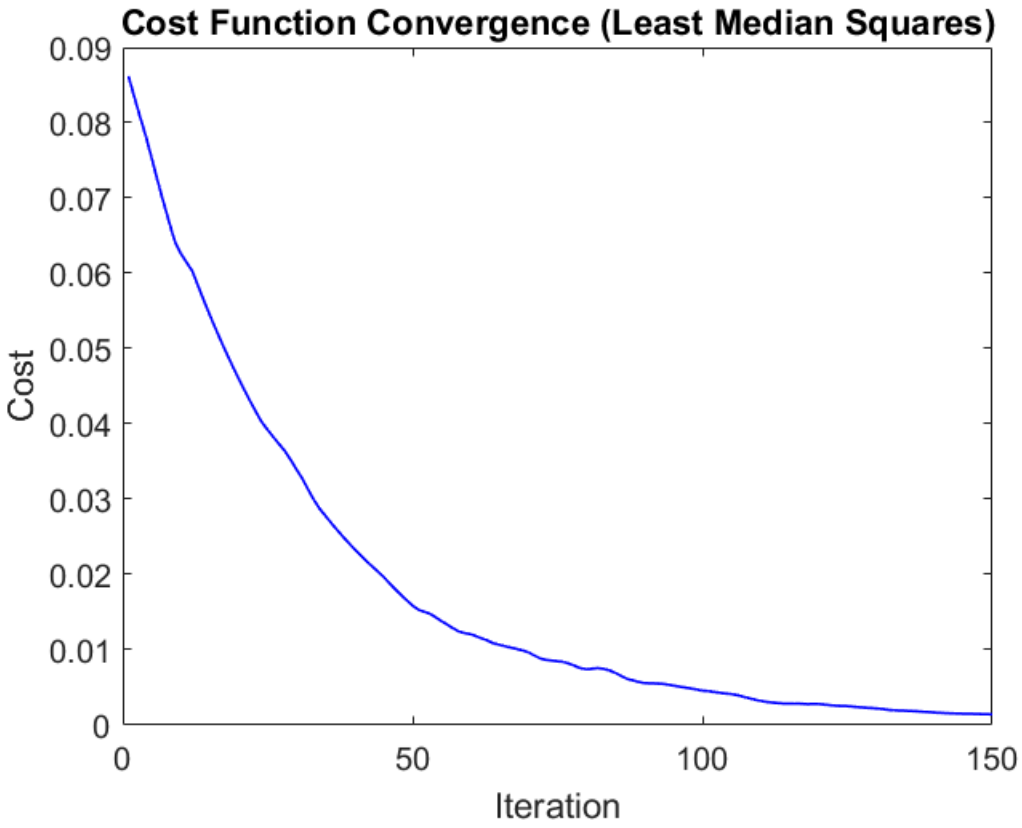
```
plotresult(X, y, best_estimates(:,1), epochs(1), cost_history_ols, "Ordinary Least Squares")
```

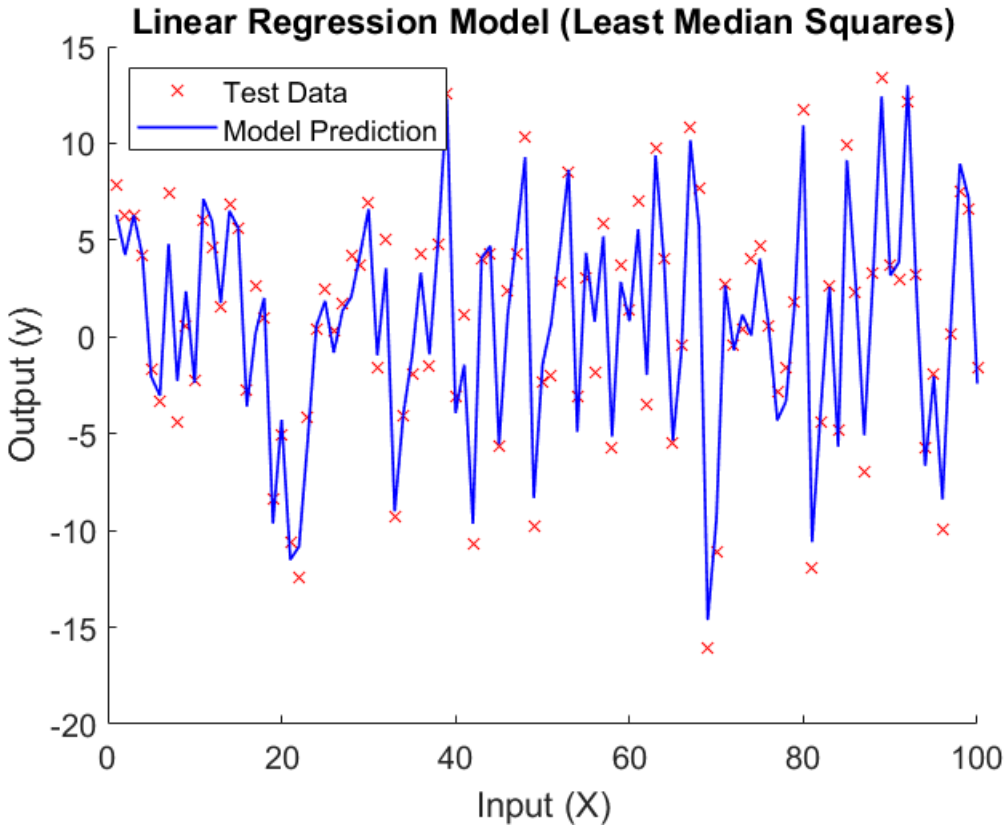




Least Median Squares

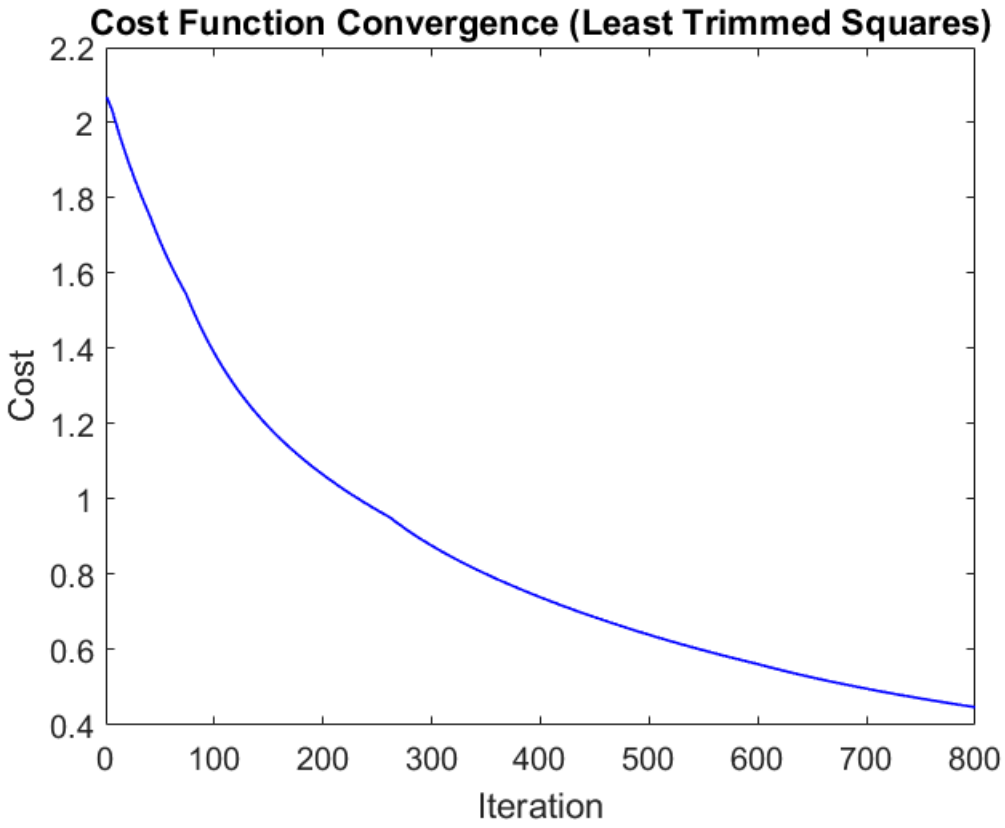
```
plotresult(X, y, best_estimates(:,2), epochs(2), cost_history_lms, "Least Median Squares")
```

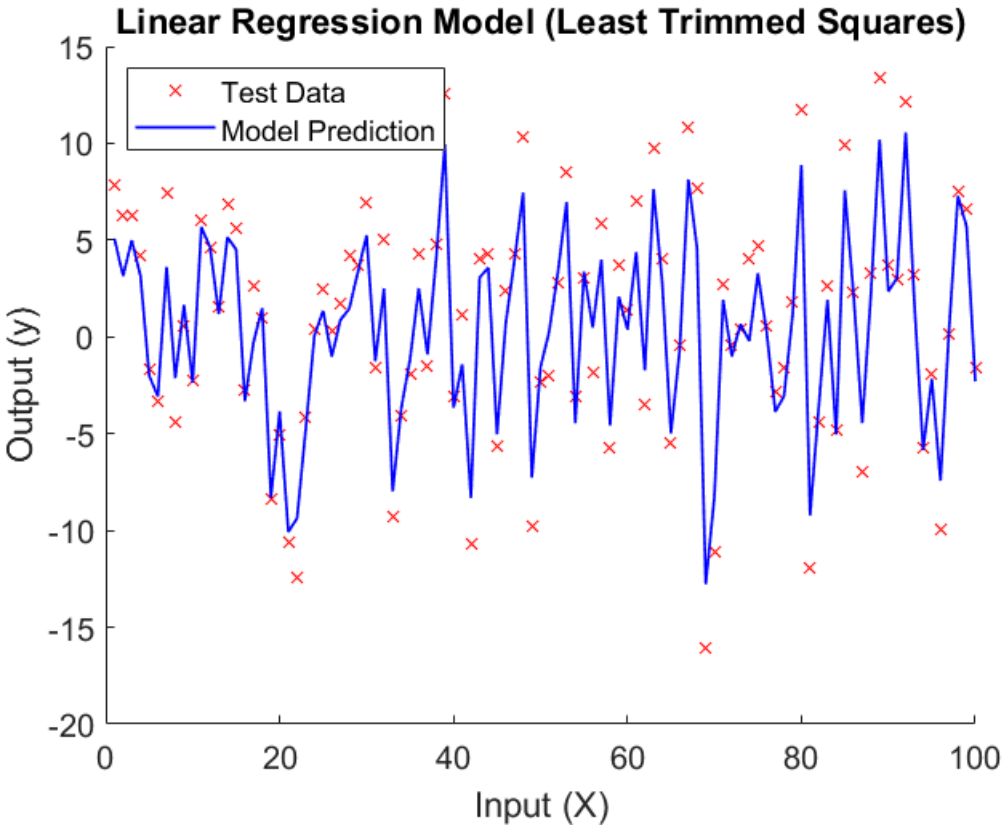




Least Trimmed Squares

```
plotresult(X, y, best_estimates(:,3), epochs(3), cost_history_lts, "Least Trimmed Squares")
```





Model Comparison

```
% Calculate metrics
metrics_ols = metrics(beta, beta_estimates_ols);
metrics_lms = metrics(beta, beta_estimates_lms);
metrics_lts = metrics(beta, beta_estimates_lts);

% Comparison Metrics
cmetrics = {'MSE', 'RB', 'MAD'};

% Display metrics
TM = table(metrics_ols', metrics_lms', metrics_lts', 'VariableNames', methods);
TMD = table(cmetrics', TM, 'VariableNames', {'Parameters', 'Metrics'});
disp(TMD);
```

Parameters	Metrics		
	OLS	LMS	LTS
'MSE'	0.034016	0.01881	0.26578
'RB'	6.7453	8.1251	3.9865
'MAD'	18.367	22.303	10.385

Question - 1 (PART - III)

```
% Clear workspace and command window
clear; clc; close all;
% For reproducibility
rng(27);
```

Load data

```
% Load data
raw_data = readtable("../../dataset/medical_insurance.csv");
age = table2array(raw_data(:,1));
[~, ~, sex] = unique(raw_data(:, 2));
bmi = table2array(raw_data(:,3));
children = table2array(raw_data(:,4));
[~, ~, smoker] = unique(raw_data(:, 5));
[~, ~, region] = unique(raw_data(:, 6));
charges = table2array(raw_data(:,7));

% Combine the encoded data
encoded_data = [age sex bmi children smoker region charges];
% Separate dependent and independent variables
X = encoded_data(:, 1:6);
Y = encoded_data(:, 7);
```

Splitting the data

```
cv = cvpartition(numel(Y), 'HoldOut', 0.2); % 20% for testing
idx_train = training(cv);
idx_test = test(cv);
X_train = normalize(X(idx_train, :));
Y_train = Y(idx_train);
X_test = normalize(X(idx_test, :));
Y_test = Y(idx_test);
```

Initialize variables

```
alpha = 0.005; % Learning rate
epochs = [450, 150, 1000]; % Epochs or Number of Iterations
```

Perform Regression

```
[beta_hat_ols, cost_history_ols] = ordinaryLeastSquares(X_train, Y_train, alpha, epochs(1));
[beta_hat_lms, cost_history_lms] = leastMedianSquares(X_train, Y_train, alpha, epochs(2));
[beta_hat_lts, cost_history_lts] = leastTrimmedSquares(X_train, Y_train, alpha, epochs(3));
```

Display best parameters

```
% Model parameter names
parameters = {'intercept', 'beta1', 'beta2', 'beta3', 'beta4', 'beta5', 'beta6'};
methods = {'OLS', 'LMS', 'LTS'};
```

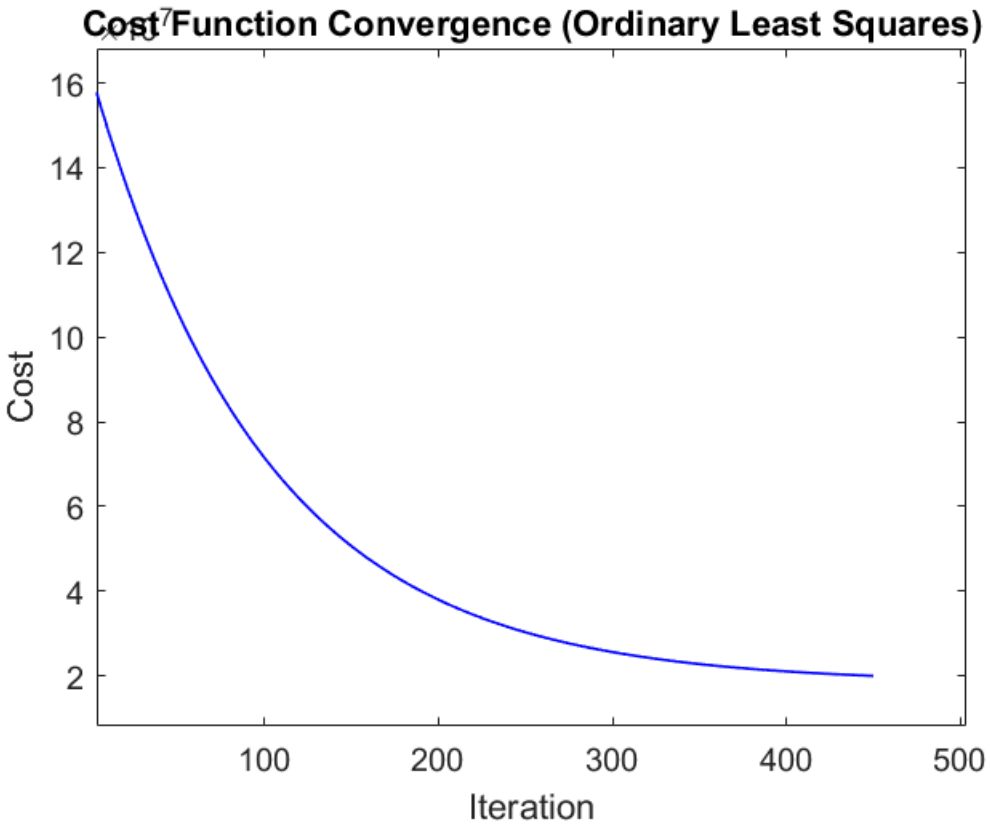
```
TP = table(beta_hat_ols, beta_hat_lms, beta_hat_lts, 'VariableNames', methods);
TPD = table(parameters', TP, 'VariableNames', {'Parameters', 'Optimum'});
disp(TPD);
```

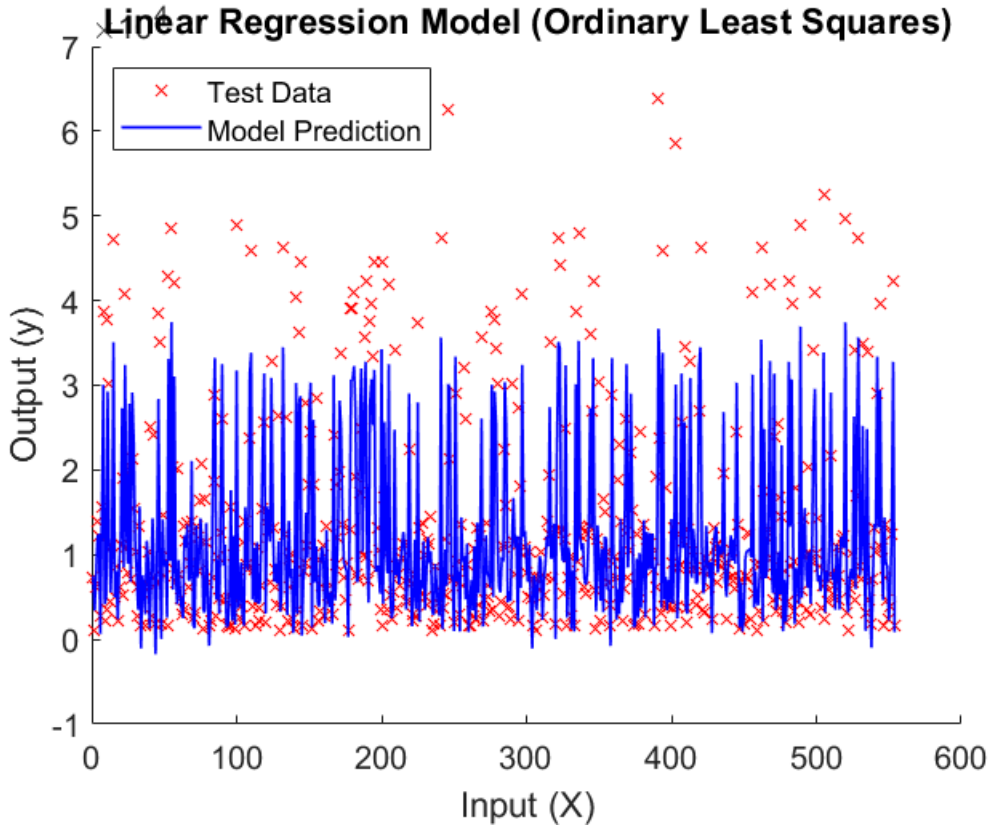
Parameters	Optimum		
	OLS	LMS	LTS
'intercept'	11905	10354	4676.8
'beta1'	3195.8	2756	2265.1
'beta2'	185.05	273.62	-297.75
'beta3'	1817.7	1620.4	52.252
'beta4'	663.02	598.23	998.51
'beta5'	8549.9	7415.1	-2385.2
'beta6'	-394.55	-307.36	-216.83

Model Testing

Ordinary Least Squares

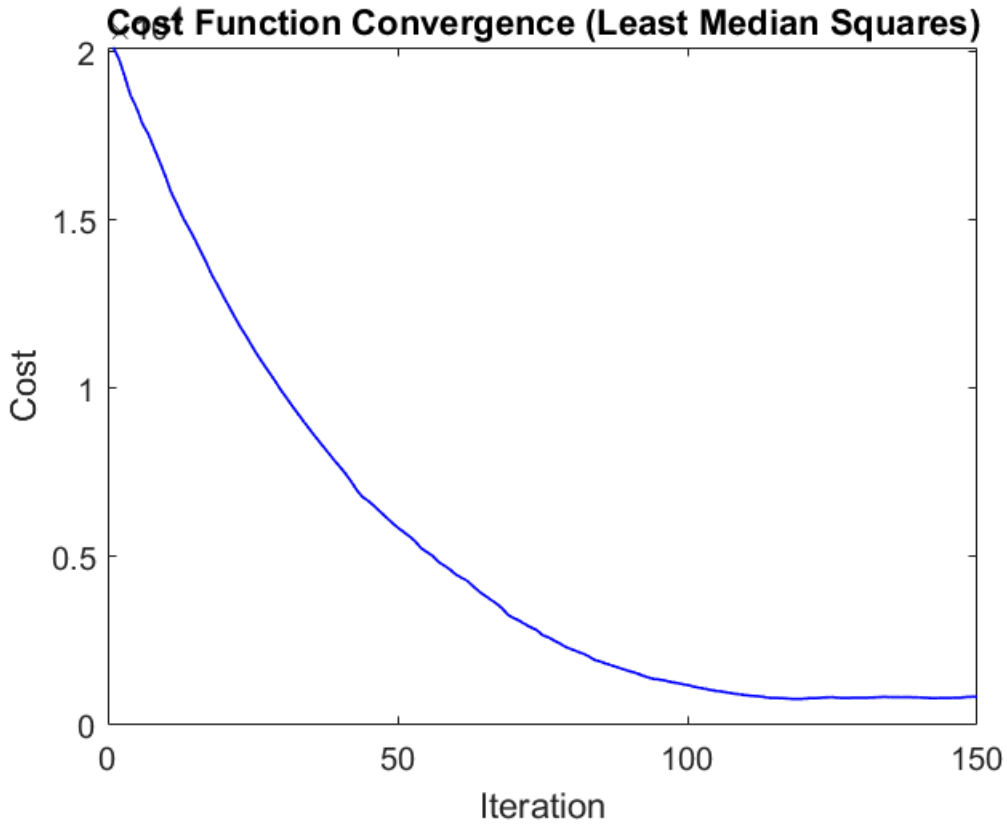
```
plotresult(X_test, Y_test, beta_hat_ols, epochs(1), cost_history_ols, "Ordinary Least Squares")
```

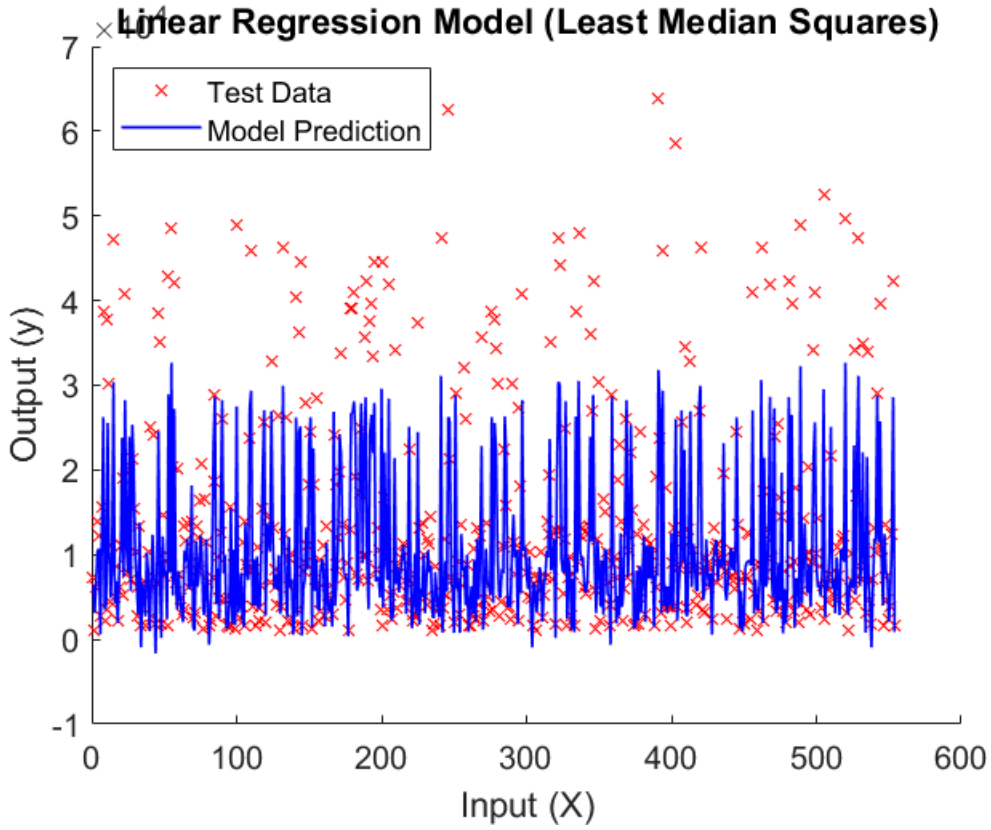




Least Median Squares

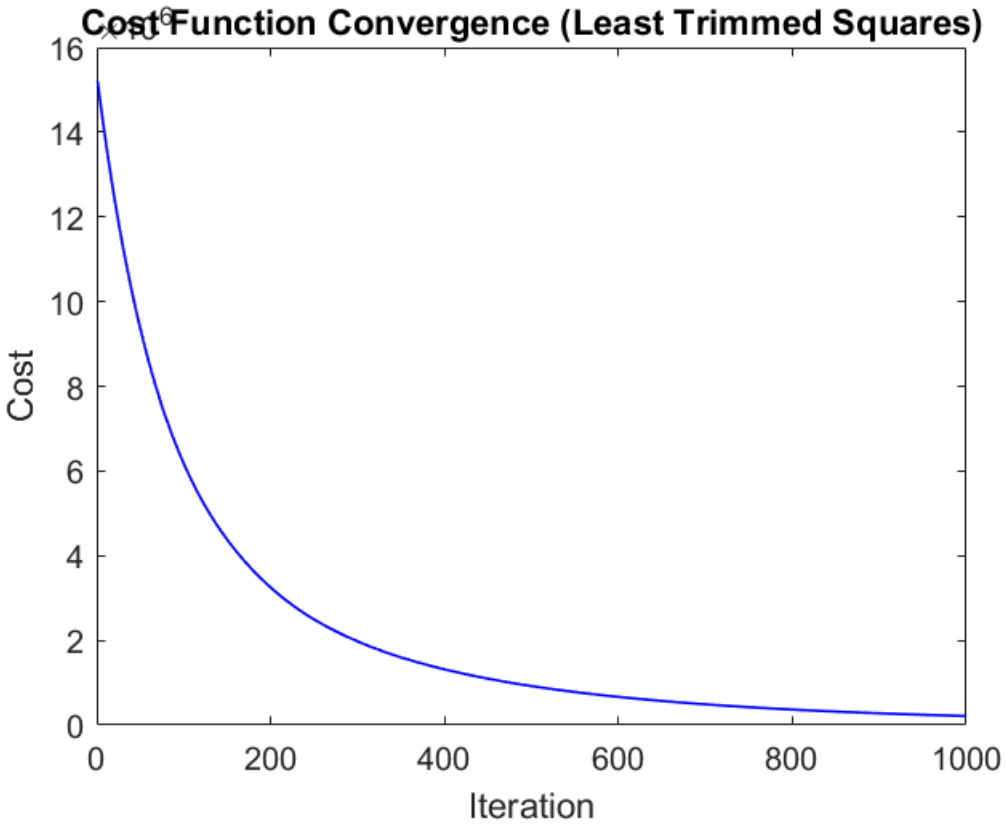
```
plotresult(X_test, Y_test, beta_hat_lms, epochs(2), cost_history_lms, "Least Median Squares")
```

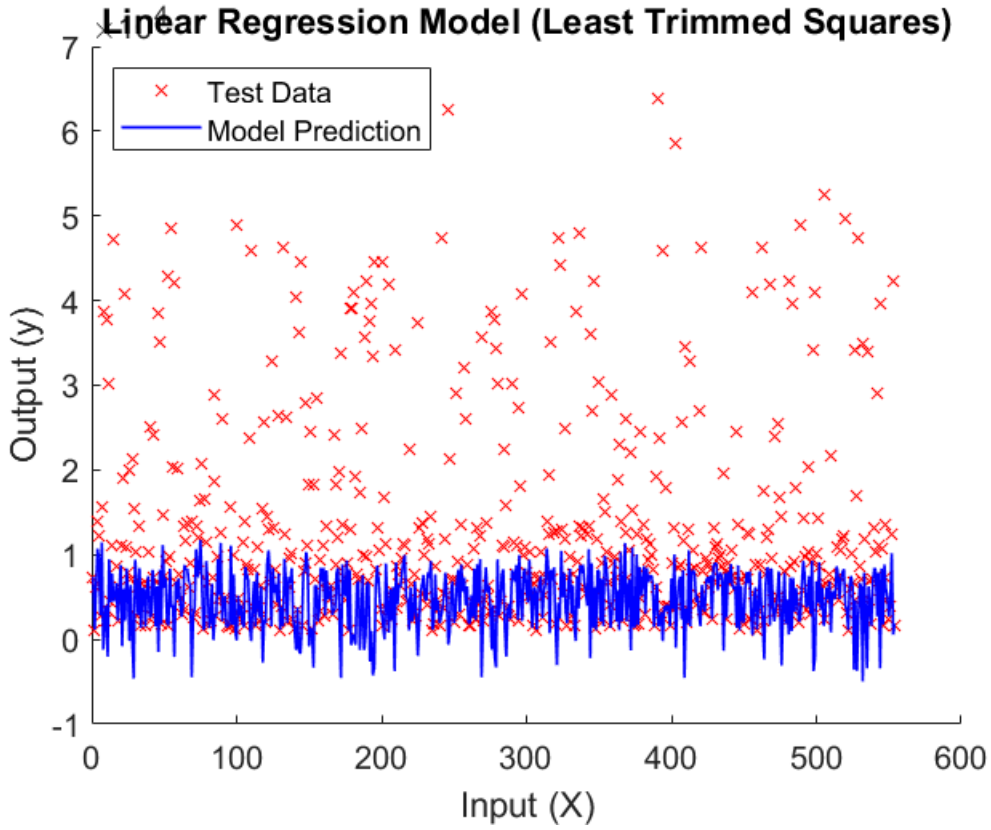




Least Trimmed Squares

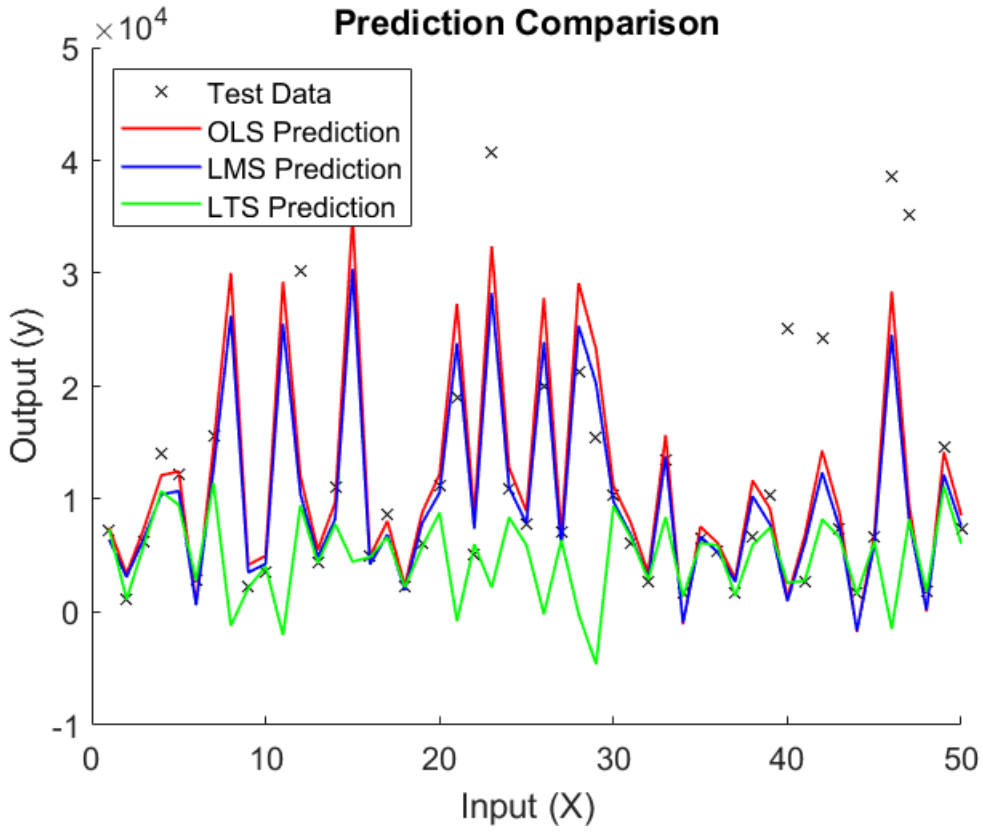
```
plotresult(X_test, Y_test, beta_hat_lts, epochs(3), cost_history_lts, "Least Trimmed Squares")
```





Compare Models

```
m = length(Y_test);
Xt = [ones(m, 1) X_test];
y_ols = Xt*beta_hat_ols;
y_lms = Xt*beta_hat_lms;
y_lts = Xt*beta_hat_lts;

%% Linear regression model plot

figure;
n = 50;
scatter(1:n, Y_test(1:n), 'kx', 'MarkerFaceColor', 'r'); % Exclude bias term from X
hold on;
plot(1:n, y_ols(1:n), 'r-', 'LineWidth', 1);
plot(1:n, y_lms(1:n), 'b-', 'LineWidth', 1);
plot(1:n, y_lts(1:n), 'g-', 'LineWidth', 1);
legend('Test Data', 'OLS Prediction', 'LMS Prediction', 'LTS Prediction', 'Location', 'NorthWes
xlabel('Input (X)', 'FontSize', 14);
ylabel('Output (y)', 'FontSize', 14);
title('Prediction Comparison', 'FontSize', 14);
set(gca, 'FontSize', 12); % Set font size for axis labels and ticks
hold off;
```




```
mse_ols = mean((Y_test-y_ols).^2);
mse_lms = mean((Y_test-y_lms).^2);
mse_lts = mean((Y_test-y_lts).^2);

% Display metrics
TM = table(mse_ols, mse_lms, mse_lts, 'VariableNames', methods);
TMD = table({'MSE'}, TM, 'VariableNames', {'Parameters', 'Metrics'});
disp(TMD);
```

Parameters	Metrics		
	OLS	LMS	LTS
'MSE'	3.9839e+07	5.0851e+07	2.664e+08

Q2 - Unevenly Sampled Time Series Analysis

Implementation

The question is to perform Unevenly Sampled Time Series Analysis using Lomb Scale Periodogram and ARIMA model. Here, Lomb Scale Periodogram is implemented from scratch and ARIMA is implemented using the builtin routine of MATLAB. The `matlab\Q2` folder has all the MATLAB files related to this question.

Lomb Scale Periodogram

```
%% Lomb Scale Periodogram using Gradient Descent
function [frequencies, powers, cost_history, a, b] = lomb_scale_periodogram(time, signal,
    %% Input Arguments
    %   time: Time instances of the signal
    %   signal: Signal sampled at the instants specified in time
    %   alpha: Learning rate (scalar)
    %   epochs: Number of epochs (scalar)

    %% Output Arguments
    %   Returns the power spectral density estimate
    %       frequencies: List of frequncies
    %       powers: Powers at each frequency
    %   cost_history: Cost at every epoch using the LS method
    %   LS Periodogram Estimates
    %       a: Coefficients of cosine term
    %       b: Coefficients of sine term

    %% Initialize variables
    N = length(time); % Length of the signal
    if ~isempty(varargin)
        K = varargin{1}; % User-defined Resolution of the frequency range considered for
    else
        K = 1000; % Resolution of the frequency range considered for analysis
    end
    % Compute the frequency range for analysis
    lower_bound = 0; % Lower bound
    upper_bound = floor((N/time(end))/2); % Upper bound (Fs/2)
    frequencies = lower_bound:(upper_bound/K):upper_bound; % Construct frequency range
    m = length(frequencies); % Number of frequencies in the analysis range
    powers = zeros(m, 1); % Power of each signal in the frequency range
    a = zeros(m, 1); % Coefficient for cosine term
    b = zeros(m, 1); % Coefficient for sine term
    cost_history = zeros(epochs, 1); % Cost history
```

```
    %% Pre-process signal
    missing_indices = isnan(signal); % Identify the indices of missing data
    non_missing_signal = signal(~missing_indices); % Construct the signal omitting the mi
    mean_value = mean(non_missing_signal); % Compute mean
    centered_non_missing_signal = non_missing_signal - mean_value; % Mean center the sigr
    % Processed signal
    y = signal;
    y(~missing_indices) = centered_non_missing_signal; %For non-missing indice,use the me
    y(missing_indices) = mean_value; % For missing indices, use the mean value

    %% Gradient Descent
    for epoch = 1:epochs
        % Compute y_hat and cost
        y_hat = zeros(N, 1);
        for i = 1:m
           w = 2*pi*frequencies(i); % w=2?f
           y_hat = y_hat + (a(i)*cos(w*time) + b(i)*sin(w*time)); % construct y_hat
        end
        residual = y-y_hat; % Residual
        cost_history(epoch) = (sum(residual.^2)/2*N); % Oridinary Least Squares

        % Compute gradients
        for i = 1:m
            w = 2*pi*frequencies(i);
            grad_a = -sum(residual.*cos(w*time))/N; % Gradient for a
            grad_b = -sum(residual.*sin(w*time))/N; % Gradient for b

            % Parameter updates
            a(i) = a(i) - alpha*grad_a;
            b(i) = b(i) - alpha*grad_b;
        end
    end

    %% Compute powers of each signal frequency
    for i = 1:m
        w = 2*pi*frequencies(i);
        s = a(i)*cos(w*time) + b(i)*sin(w*time);
        powers(i) = sum(s.^2) / N;
    end
end
```

The above implementation uses Gradient Descent for optimization. The number of frequencies to search for is inferred from the data (analysing the sample frequency of the data). The number of divisions between the range can be specified. The higher the number of divisions, the better the output. Other helper functions used in the above analysis can be found in the `matlab\Q2` folder.

Synthetic Data

As per the question, synthetic data is generated as a summation of two sine waves. Lomb Scale Periodogram is applied and the predictions are made. Here is the summary of the analysis,

Metric	Original Signal vs Reconstructed Signal	Noisy Signal vs Reconstructed Signal
NMSE	0.61553	0.0027936
MAPE	95.218	4.6254

From the above table, we can infer that while the Lomb Scale Periodogram works well in cleaning up noise and missing values (as indicated by the low NMSE and MAPE for the noisy signal), it may not be as effective in accurately replicating the original signal (as indicated by the high NMSE and MAPE for the original signal).

Real Dataset - Tesla Stocks

For the real dataset, Tesla stock price from 2011 to 2020 is used. The first 80% data is used for training and remaining 20% is used for testing. Data is mean centered before computing the periodogram. Both ARIMA and Lomb Scale Periodogram are used to predict the test data in this question. The following is the analysis results:

Metric	LSP	ARIMA
NMSE	1.3658	1.2682
MAPE	20.479	22.55

From the above data we can infer that,

- The NMSE values for both models are **relatively close**, indicating that both LSP and ARIMA perform similarly in terms of squared error between predicted and actual stock prices. Lower NMSE values indicate better performance in terms of minimizing squared errors.
- The MAPE values for LSP and ARIMA differ slightly, with **LSP having a lower MAPE**. Lower MAPE values indicate better accuracy in predicting stock prices, as they represent a smaller percentage of error relative to the actual prices.

Both models, LSP and ARIMA, have similar NMSE values, suggesting comparable performance in terms of squared error between predicted and actual stock prices. LSP outperforms ARIMA slightly in terms of MAPE, indicating slightly better accuracy in predicting stock prices with lower percentage error. When considering both NMSE and MAPE, LSP appears to be slightly more accurate than ARIMA in predicting stock prices for the given task.

Question - 2 (PART- I)

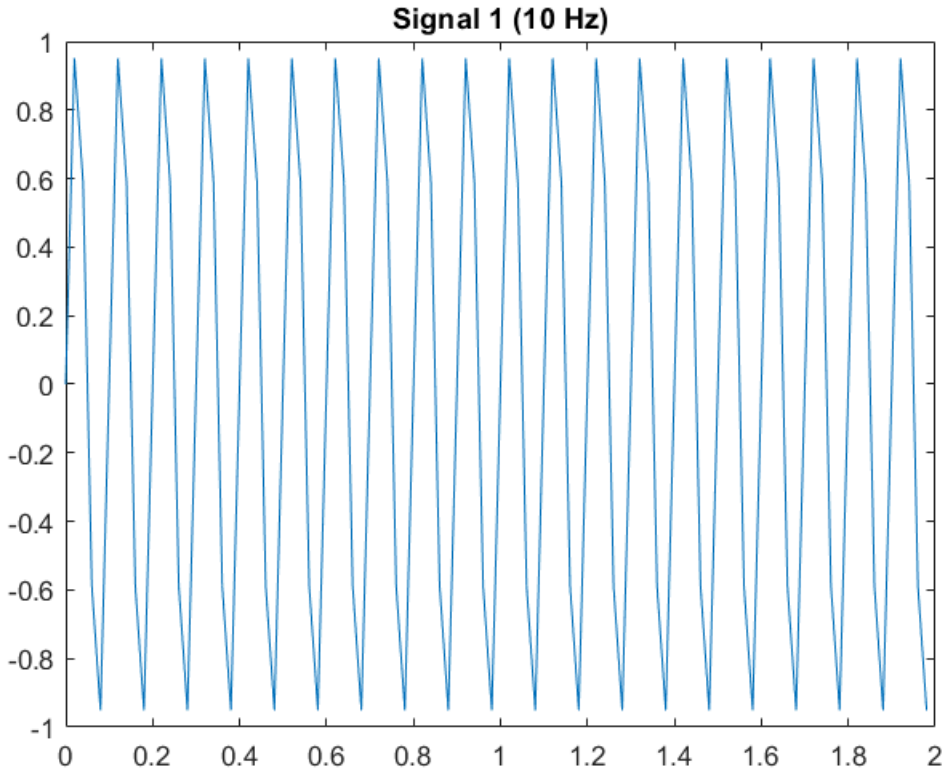
```
% Clear workspace and command window
clear; clc; close all;
% For reproducibility
rng(27);
```

Given Data

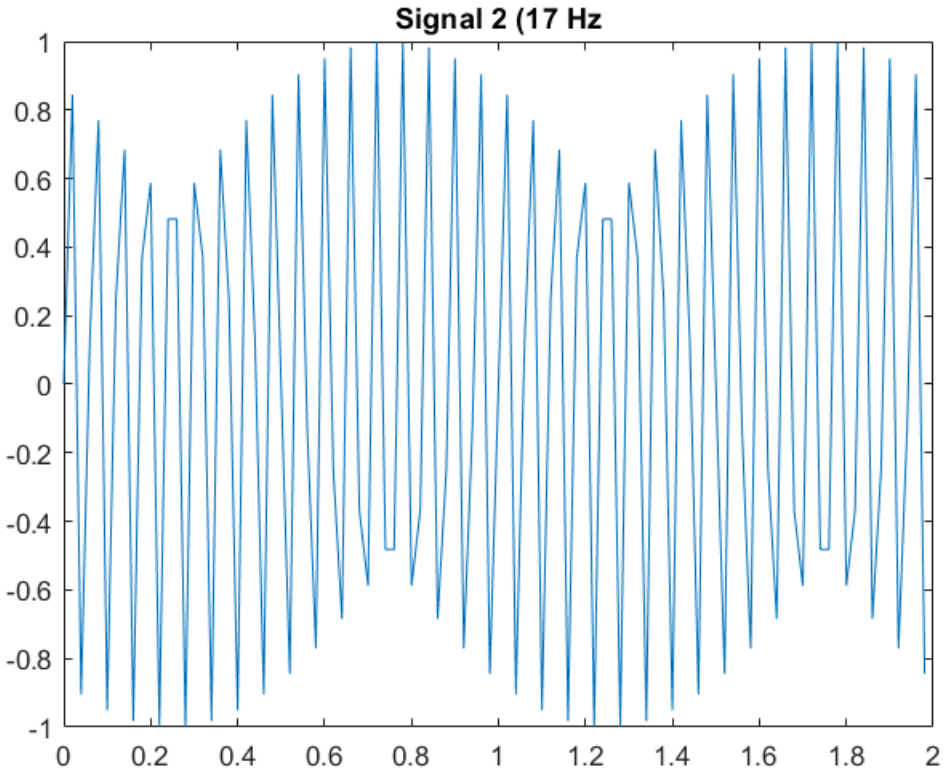
```
N = 500; % Number of observations
Fs = 50; % Sampling frequency (Hz)

f1 = 10; % Frequency 1 (Hz)
f2 = 17; % Frequency 2 (Hz)

missing_percentage = 0.1; % Percentage of missing data (10%)

target_snr = 10; % Target Signal-to-Noise Ratio (SNR)
```

Generate Data

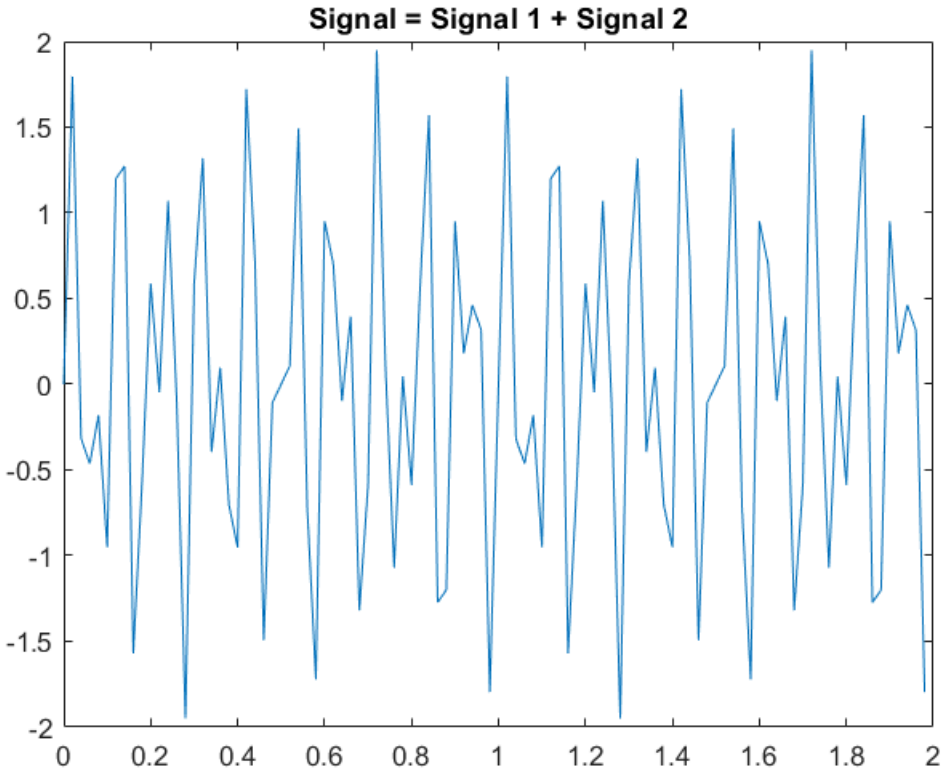
```
% Generate time vector
time = (0:N-1)'/Fs;

% Signal-1 10 Hz
signal1 = sin(2*pi*f1*time);
plot(time(1:100), signal1(1:100));
title("Signal 1 (10 Hz)");
```




```
% Signal-2 17 Hz
signal2 = sin(2*pi*f2*time);
plot(time(1:100), signal2(1:100));
title("Signal 2 (17 Hz");
```

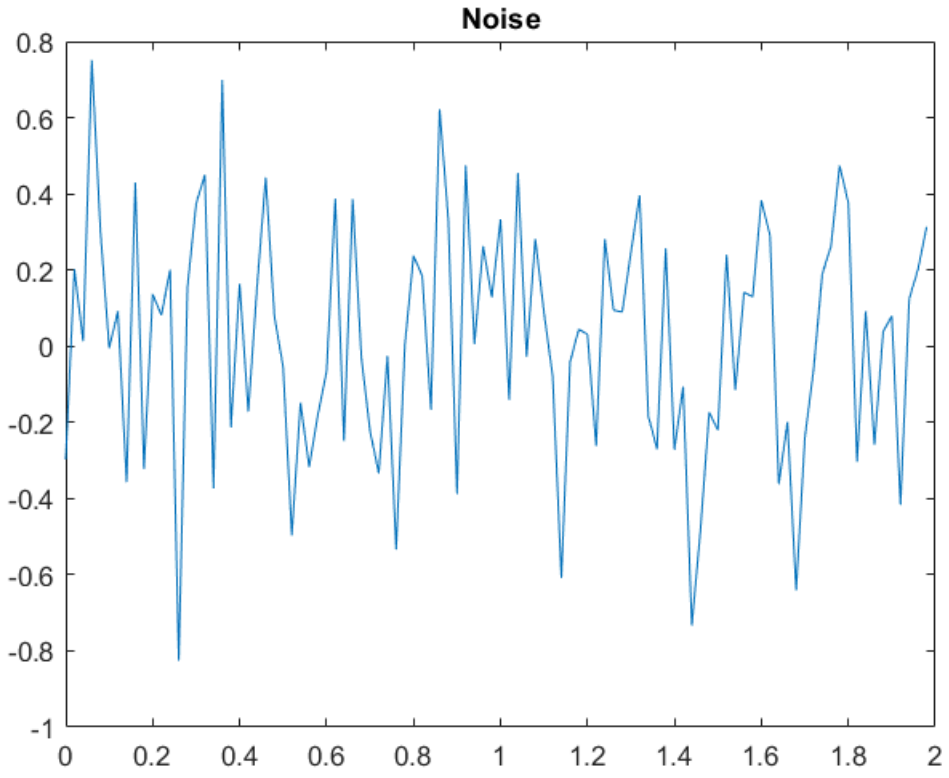



```
% Signal for analysis
signal =  signal1 + signal2;
plot(time(1:100), signal(1:100));
title("Signal = Signal 1 + Signal 2");
```

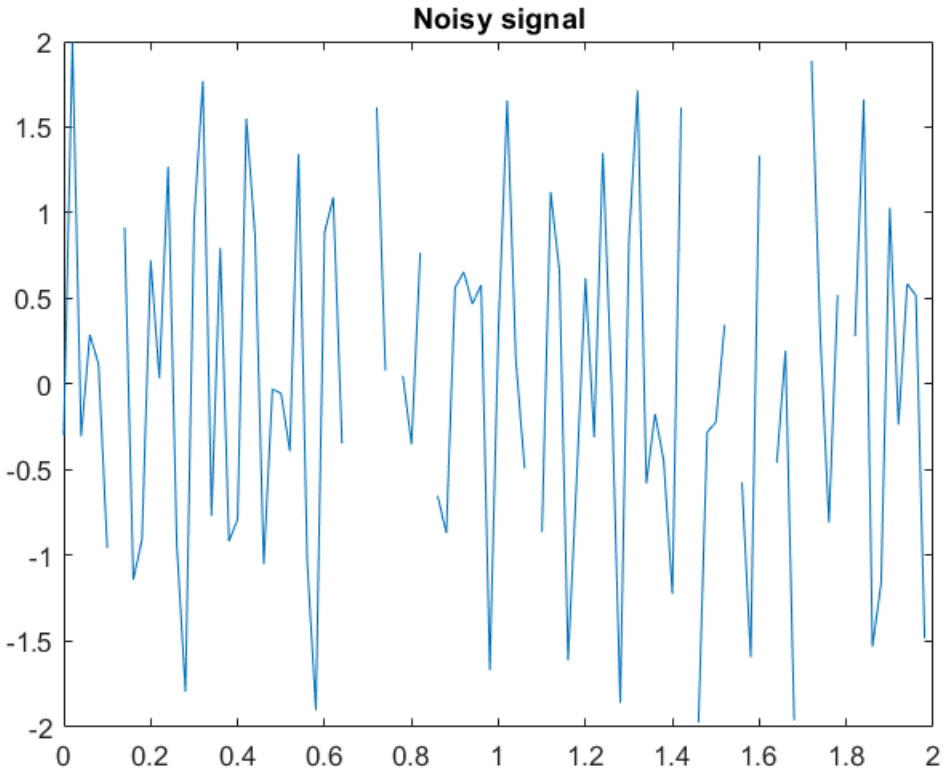



```
% Construct signal with Missing indices
missing_indices = sort(randperm(N, round(N*missing_percentage)));
missing_signal = signal;
missing_signal(missing_indices) = nan;
plot(time(1:100), missing_signal(1:100));
title("Signal with Missing data");
```


```
% Generate Noise Signal
signal_power = sum(signal.^2)/N;
noise_power = signal_power / (10^(target_snr/10));
noise = sqrt(noise_power) * randn(N, 1);
plot(time(1:100), noise(1:100))
title("Noise");
```

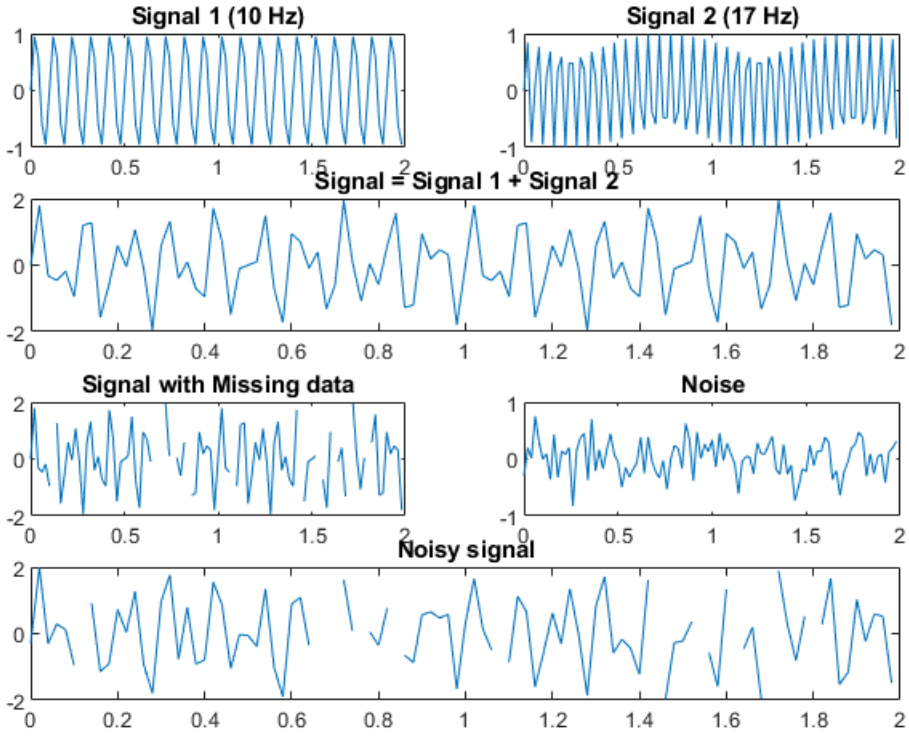



```
% Final signal with Noise and Missing data
noisy_signal = missing_signal + noise;
plot(time(1:100), noisy_signal(1:100));
title("Noisy signal");
```



Data Generating Process - Summary

```
subplot(4,2,1);
plot(time(1:100), signal1(1:100));
title("Signal 1 (10 Hz)");

subplot(4,2,2);
plot(time(1:100), signal2(1:100));
title("Signal 2 (17 Hz)");

subplot(4,2,[3,4]);
plot(time(1:100), signal(1:100));
title("Signal = Signal 1 + Signal 2");

subplot(4,2,5);
plot(time(1:100), missing_signal(1:100));
title("Signal with Missing data");

subplot(4,2,6);
plot(time(1:100), noise(1:100))
title("Noise");

subplot(4,2,[7,8]);
plot(time(1:100), noisy_signal(1:100));
title("Noisy signal");
```

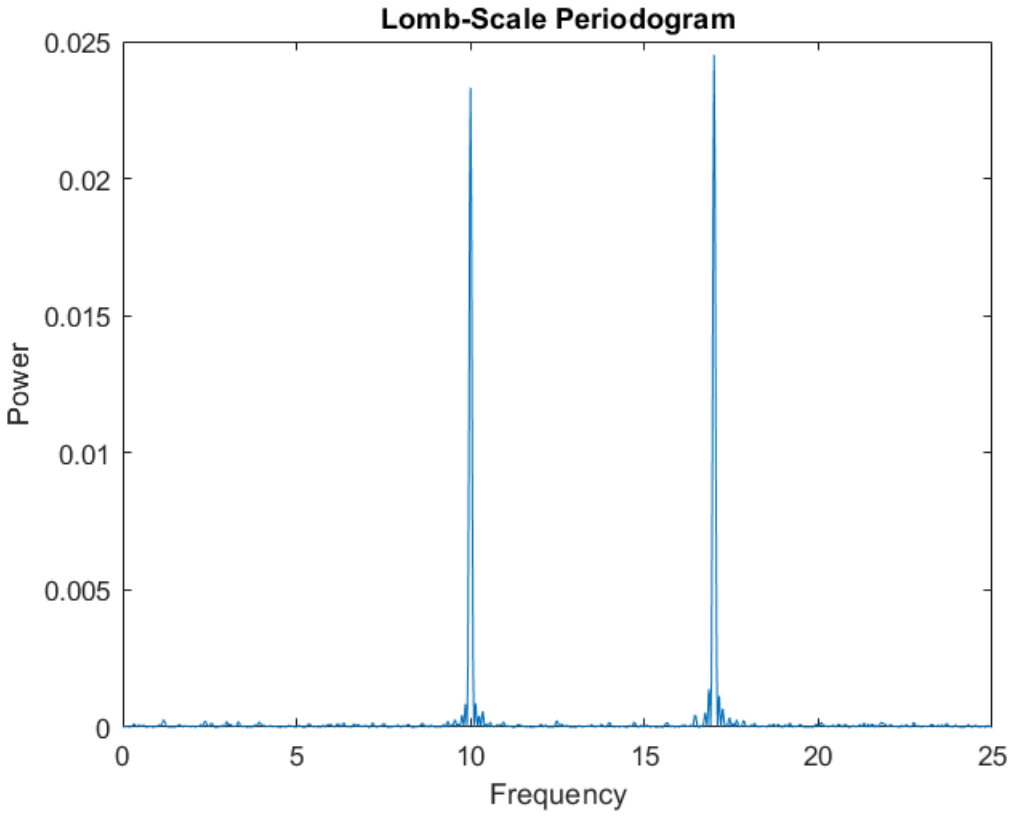


Lomb-Scale Periodogram

```
learning_rate = 4e-3;
epochs = 400;
[F, P, C, A, B] = lomb_scale_periodogram(time, noisy_signal, learning_rate, epochs);
```

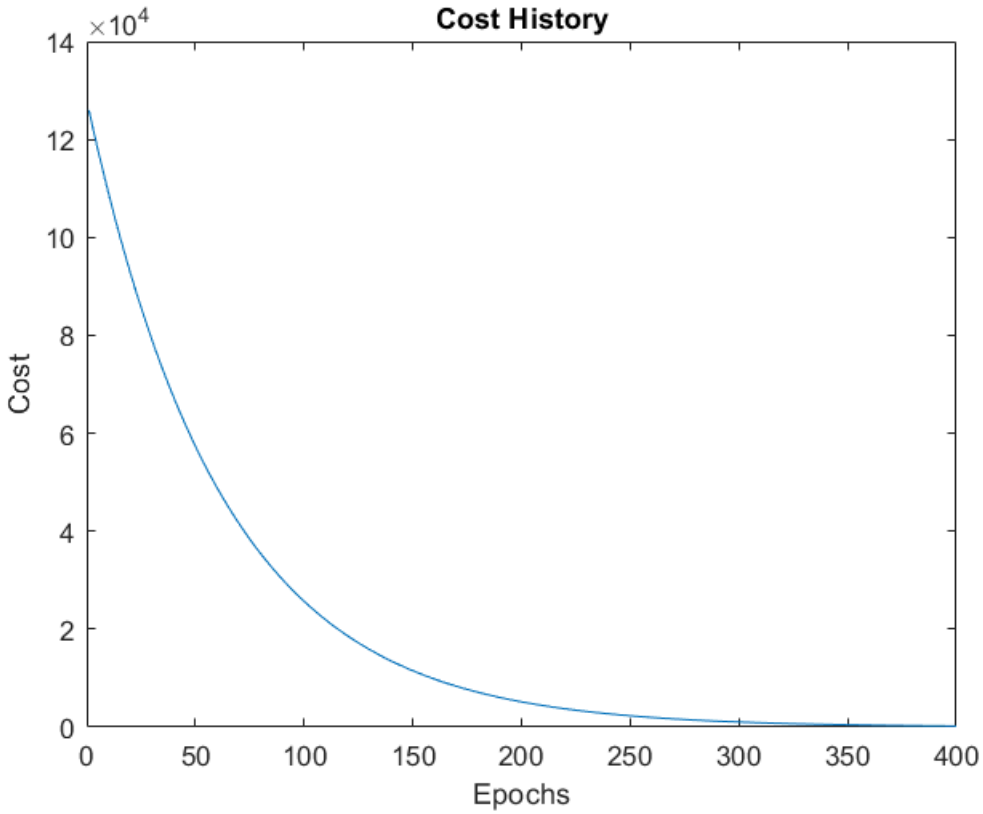
Periodogram

```
figure;
plot(F, P);
xlabel('Frequency');
ylabel('Power');
title('Lomb-Scale Periodogram');
```



Cost History

```
figure;
plot(1:epochs, C);
xlabel('Epochs');
ylabel('Cost');
title('Cost History');
```

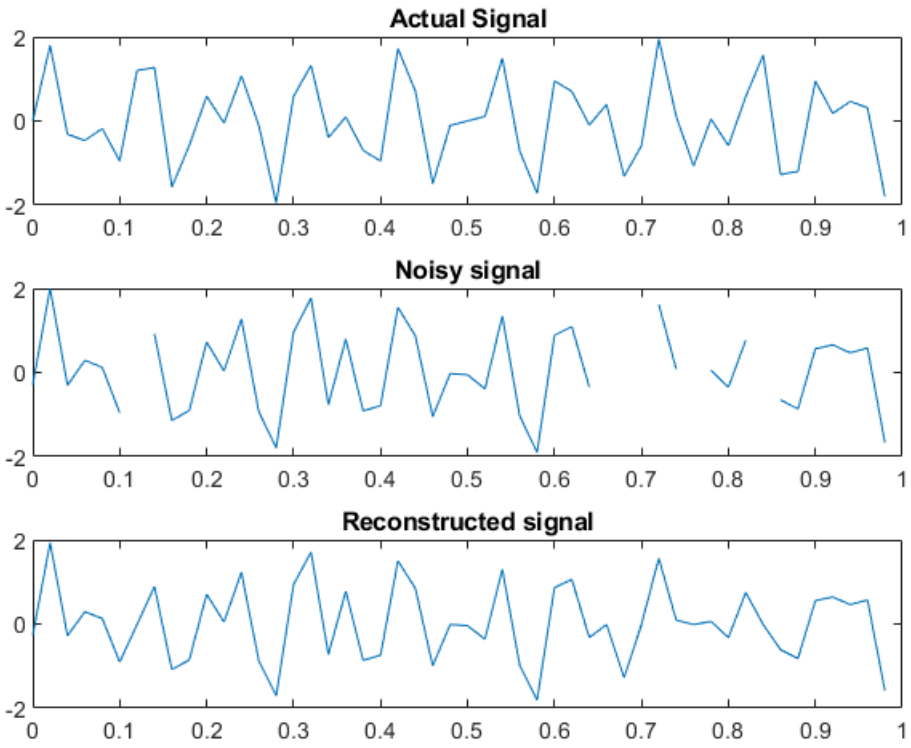


Predictions

```
reconstructed_signal = zeros(N, 1);
for i = 1:length(F)
    w = 2*pi*F(i);
    reconstructed_signal = reconstructed_signal + A(i)*cos(w*time) + B(i)*sin(w*time);
end
```

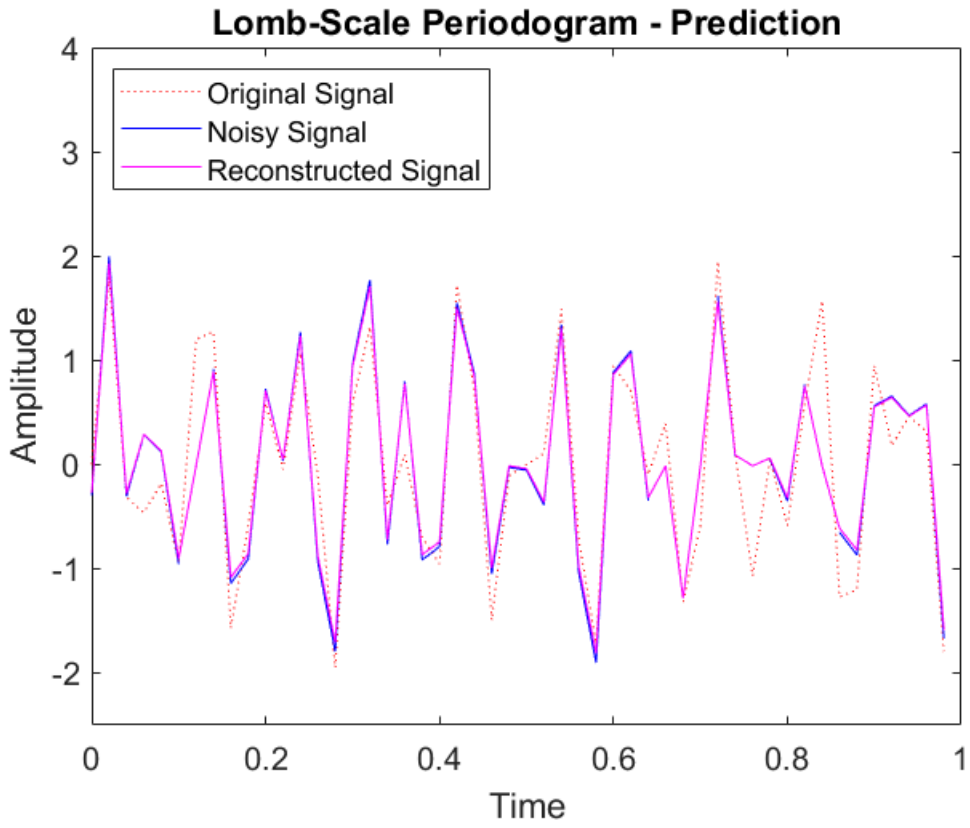
Compare Signals

```
figure
subplot(3,1,1)
plot(time(1:50), signal(1:50));
title("Actual Signal");
subplot(3,1,2)
plot(time(1:50), noisy_signal(1:50));
title("Noisy signal");
subplot(3,1,3)
plot(time(1:50), reconstructed_signal(1:50));
title("Reconstructed signal");
```



Model Prediction Comparison

```
figure;
plot(time(1:50), signal(1:50), 'r:');
hold on;
plot(time(1:50), noisy_signal(1:50), 'b-');
hold on;
plot(time(1:50), reconstructed_signal(1:50), 'm-');
legend('Original Signal', 'Noisy Signal', 'Reconstructed Signal', 'Location', 'NorthWest');
xlabel('Time', 'FontSize', 14);
ylabel('Amplitude', 'FontSize', 14);
title('Lomb-Scale Periodogram - Prediction', 'FontSize', 14);
set(gca, 'FontSize', 12); % Set font size for axis labels and ticks
ylim([-2.5, 4]);
hold off;
```



Metrics

```
[nmse1, mape1] = metrics(signal, reconstructed_signal);
[nmse2, mape2] = metrics(noisy_signal, reconstructed_signal);
```

Original Signal vs Reconstructed Signal

```
disp(table(nmse1, mape1, 'VariableNames', {'NMSE', 'MAPE'}));
```

NMSE	MAPE
0.61553	95.218

Noisy Signal vs Reconstrcuted Signal

```
disp(table(nmse2, mape2, 'VariableNames', {'NMSE', 'MAPE'}));
```

NMSE	MAPE
0.0027936	4.6254

Question - 2 (PART- II)

```
% Clear workspace and command window
clear; clc; close all;
% For reproducibility
rng(27);
```

Load data

```
% Load data from the CSV file
raw_data = readtable("../../dataset/tesla_stock_price.csv");

% Input data
time_stamps = datetime(table2array(raw_data(:,1)),  "InputFormat", "dd-MM-yyyy"); % Timestamps
stock_price = table2array(raw_data(:,2)); % Stock price
```

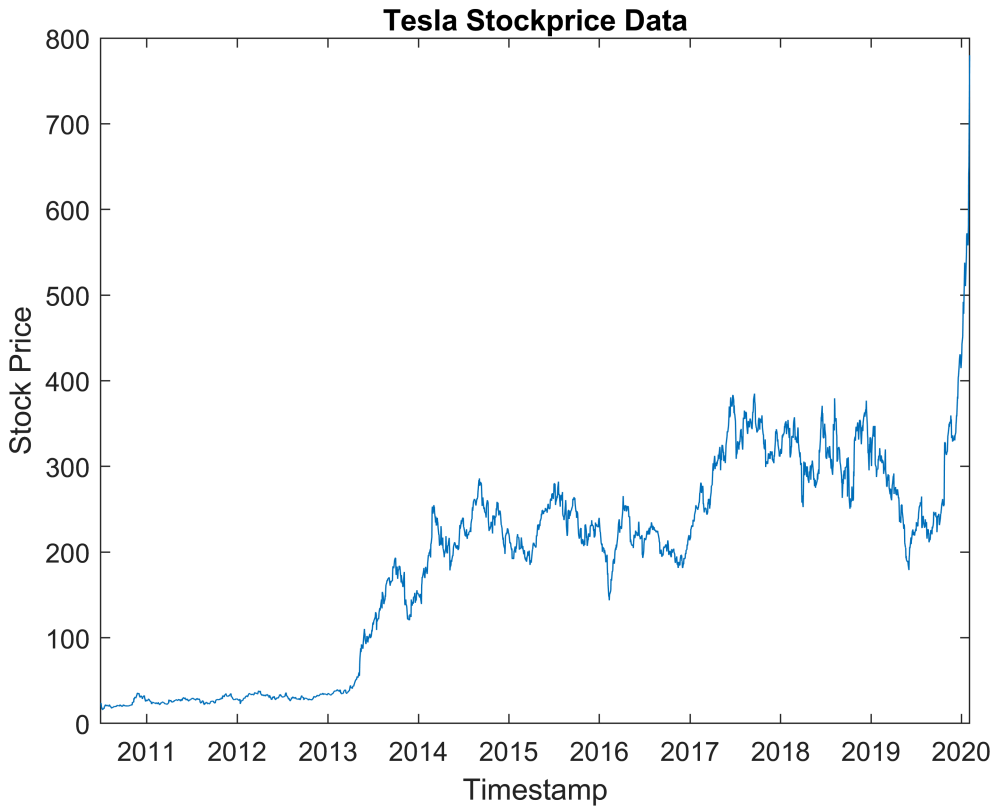
Splitting the data

Since, the input is a Timeseries data, we consider the first 80% data for training and the last 20% data for testing

```
N = length(time_stamps);
% Compute the split index
split_index = N-floor(N*0.2);
% Training data
X_train = time_stamps(1:split_index);
Y_train = stock_price(1:split_index);
% Testing data
X_test = time_stamps(split_index+1:N);
Y_test = stock_price(split_index+1:N);
```

Visualize Data

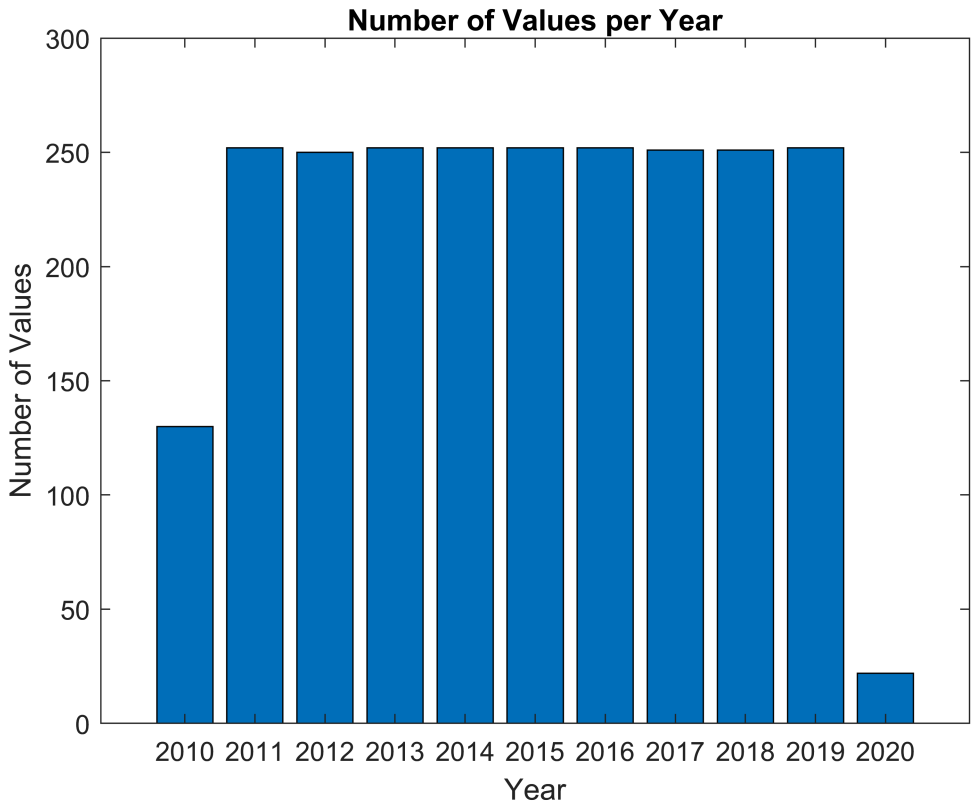
```
figure;
plot(time_stamps, stock_price)
title("Tesla Stockprice Data")
xlabel("Timestamp")
ylabel("Stock Price")
```



Period Identification

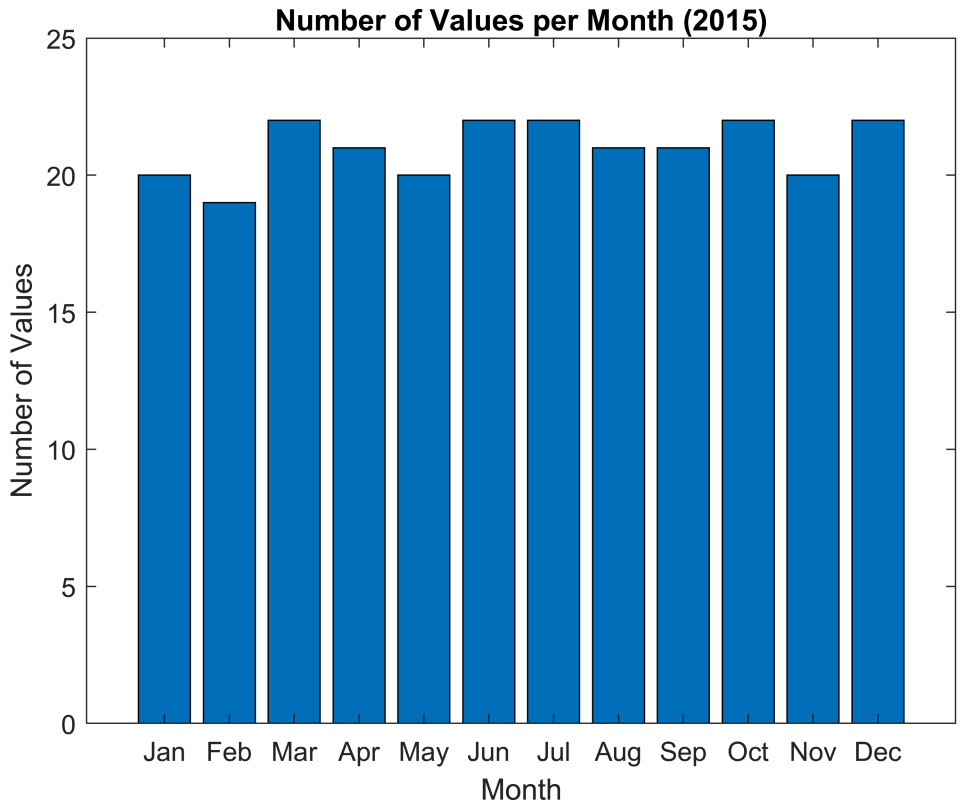
We need to identify the period of the data to fix the frequency range for the Lomb Scale Periodogram. Let's us check the number of values collected for every year in the dataset.

```
% Extract the year from the timestamps array
years = year(time_stamps);
% Get the unique years present in the data
unique_years = unique(years);
% Initialize a vector to store the counts for each year
year_counts = zeros(size(unique_years));
% Count the number of values for each year
for i = 1:length(unique_years)
    year_counts(i) = sum(years == unique_years(i));
end
% Create a bar graph
figure;
bar(unique_years, year_counts);
xlabel('Year');
ylabel('Number of Values');
title('Number of Values per Year');
```



Similarly for number of values per month in a year.

```
% Year of interest
year_of_interest = 2015;
% Extract the year from the timestamps array
years = year(time_stamps);
% Filter the data for the specified year
idx = years == year_of_interest;
timestamps_filtered = time_stamps(idx);
% Count the number of values per month
months = month(timestamps_filtered);
counts = accumarray(months, 1);
% Create a bar graph
figure;
bar(counts);
xlabel('Month');
ylabel('Number of Values');
title(sprintf('Number of Values per Month (%d)', year_of_interest));
set(gca, 'XTick', 1:12, 'XTickLabel', {'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
```



Based on the above graphs, we can infer that the data is collected on the daily basis. The given dataset contains missing values for a few days in between. But almost all the years have about 70% of the data (i.e) nearly 250 data points per year. Therefore, the sampling is done daily. **Fs/2 = 365.**

Preprocess Data

```
% Convert the DateTime array into double. We will consider the first
% timestamp as t=0
X_diff = convertTo(X_train, 'datenum')-convertTo(X_train(1), 'datenum');
fprintf("Series Length: %d", length(X_diff));
```

```
Series Length: 1933
```

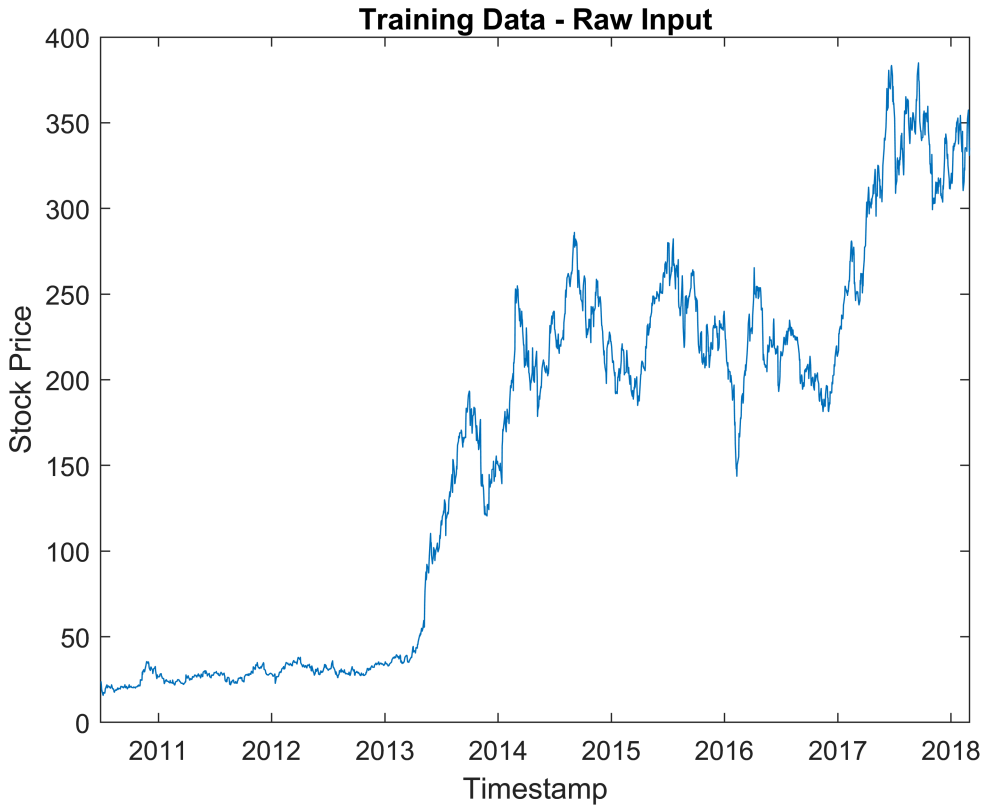
```
fprintf("Final Value of the Series: %d", X_diff(end));
```

```
Final Value of the Series: 2803
```

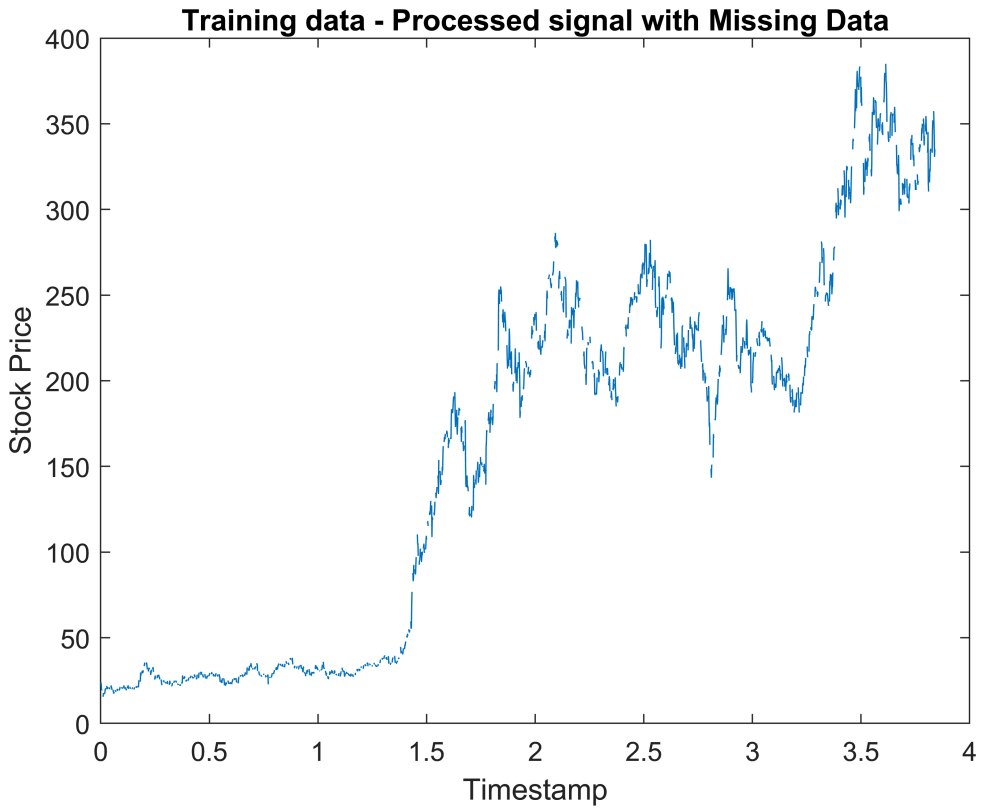
As predicted the dataset has missing values. Let's construct a dataset with all time indices with the missing values as NaN.

```
% Sampling frequency (Nyquist Rate, 2*F)
Fs = 730; % 365*2
M = X_diff(end)+1; % Length of the training data
time = (0:M-1)'/Fs; % Time Vector
signal = zeros(M, 1); % Signal placeholder
signal(:) = NaN;
signal(X_diff+1) = Y_train; % Assign the value of the dataset
```

```
% Given Input training data
figure;
plot(X_train, Y_train);
title("Training Data - Raw Input");
xlabel("Timestamp");
ylabel("Stock Price");
```




```
% Processed training data
figure;
plot(time, signal);
title("Training data - Processed signal with Missing Data");
xlabel("Timestamp");
ylabel("Stock Price");
```

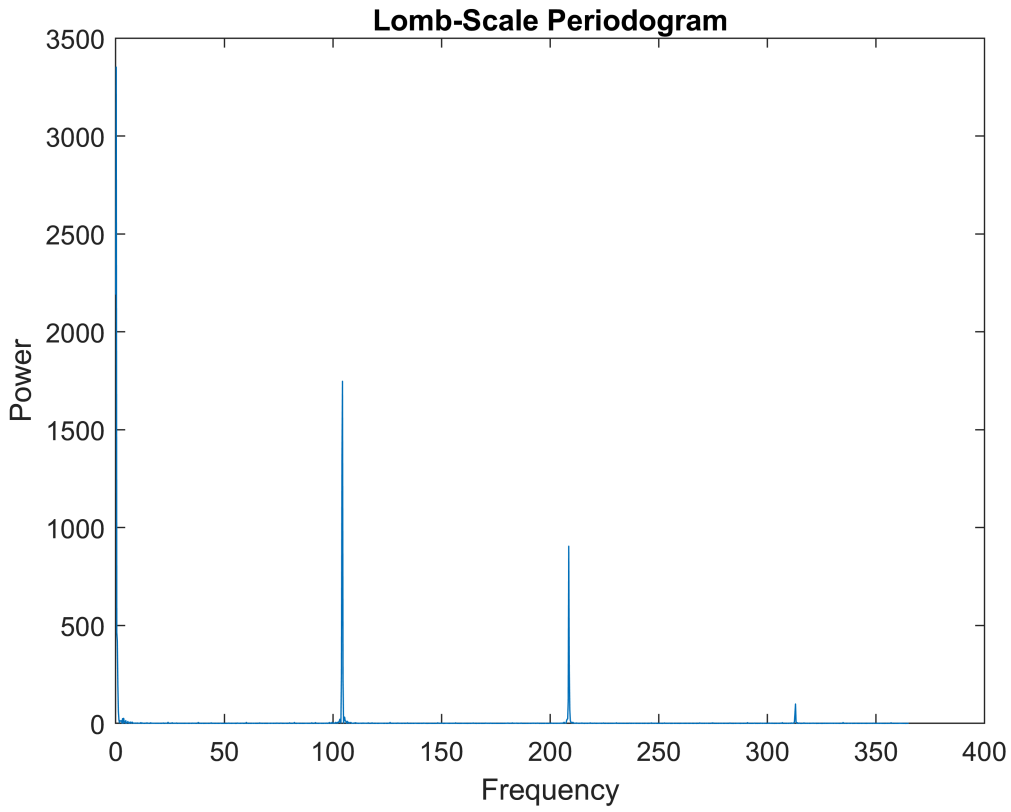


Lomb Scale Periodogram

```
learning_rate = 3e-2; % Learning rate
epochs = 150; % Epochs
% Compute Lomb Scale Periodogram
[F, P, C, A, B] = lomb_scale_periodogram(time, signal, learning_rate, epochs, 1500);
```

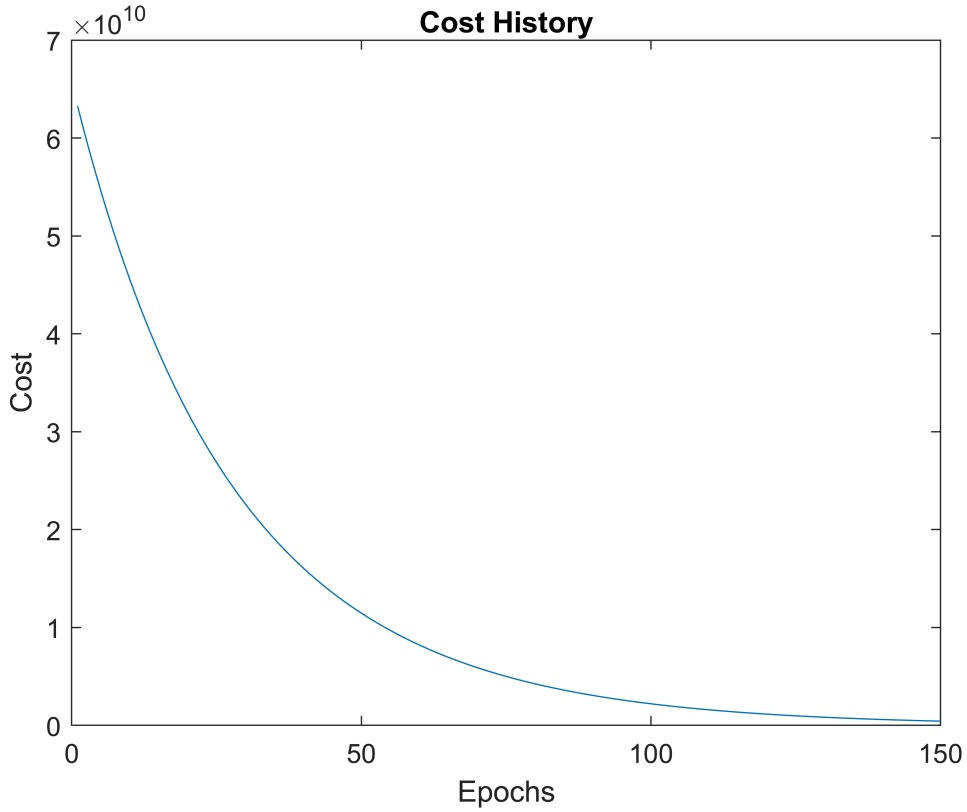
Periodogram

```
% Plot the periodogram
figure;
plot(F, P);
xlabel('Frequency');
ylabel('Power');
title('Lomb-Scale Periodogram');
```



Cost History

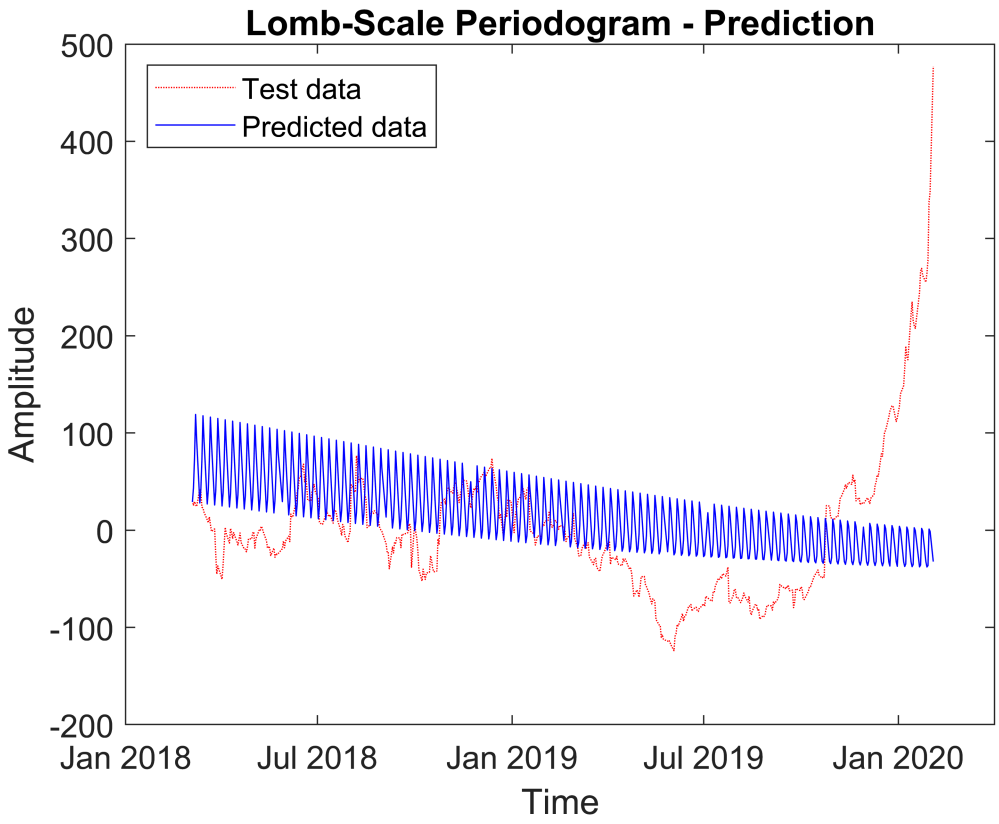
```
% Plot the cost history
figure;
plot(1:epochs, C);
xlabel('Epochs');
ylabel('Cost');
title('Cost History');
```



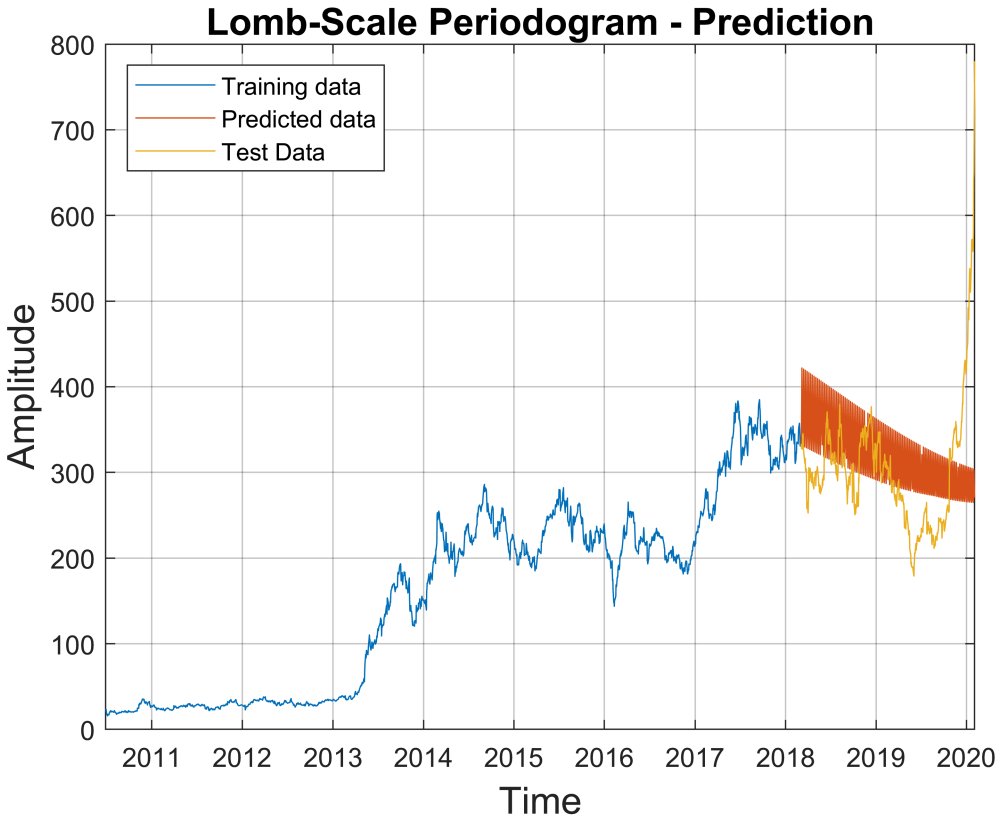
Predictions

```
% Convert DateTime and double
X_range = convertTo(X_test, 'datenum')-convertTo(X_train(1), 'datenum');
time_t = X_range/Fs;

% Predict the signal with the periodogram outputs
% Only consider the frequencies for which the power is greater than 1
frequency = F(P>500);
y_hat_lsp = zeros(N-split_index, 1);
for i = 1:length(frequency)
    w = 2*pi*frequency(i);
    y_hat_lsp = y_hat_lsp + A(i)*cos(w*time_t) + B(i)*sin(w*time_t);
end

% Mean center the Test data
mean_value = mean(Y_test); % Compute mean
y = Y_test - mean_value; % Mean center the signal

% Plot the comparison
figure;
plot(X_test, y, 'r:');
hold on;
plot(X_test, y_hat_lsp, 'b-');
hold on;
legend('Test data', 'Predicted data', 'Location', 'NorthWest');
xlabel('Time', 'FontSize', 14);
ylabel('Amplitude', 'FontSize', 14);
```

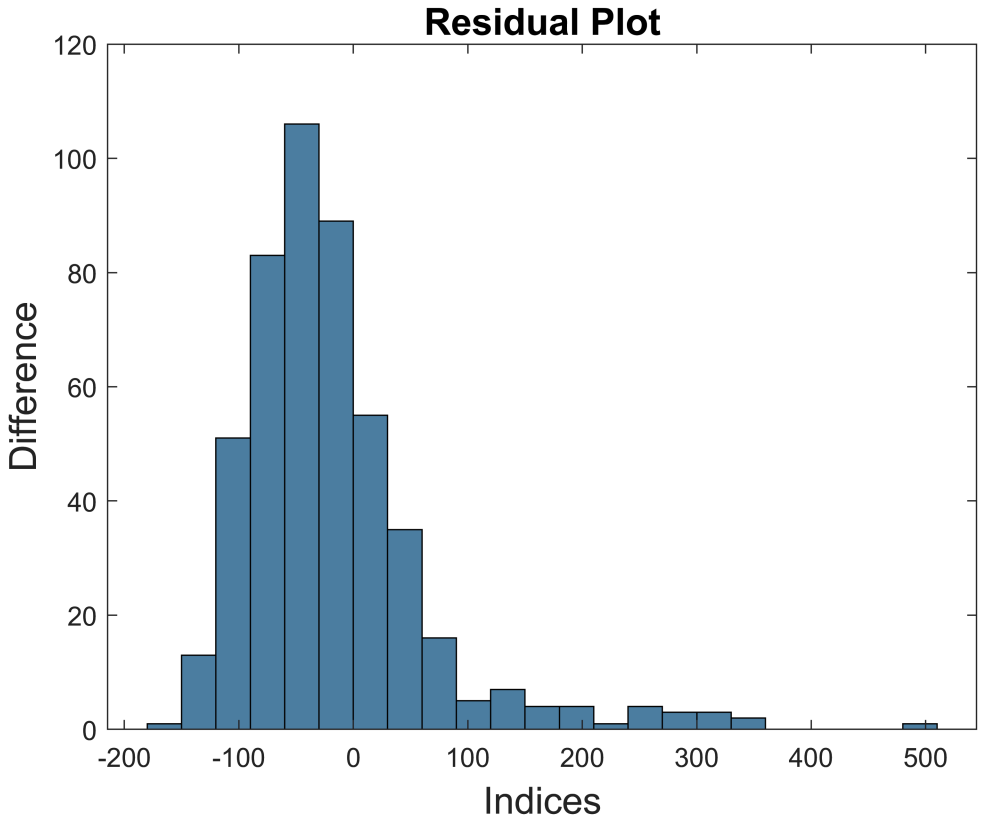
```
title('Lomb-Scale Periodogram - Prediction', 'FontSize', 14);
set(gca, 'FontSize', 12); % Set font size for axis labels and ticks
hold off;
```




```
% Plot the whole series
figure
plot(X_train,Y_train);
hold on
plot(X_test,y_hat_lsp+mean(Y_test));
plot(X_test,Y_test);
legend('Training data', 'Predicted data', 'Test Data', 'Location', 'NorthWest');
xlabel('Time', 'FontSize', 14);
ylabel('Amplitude', 'FontSize', 14);
title('Lomb-Scale Periodogram - Prediction', 'FontSize', 14);
grid on
hold off
```




```
% Residual Plot
histogram(y-y_hat_lsp);
xlabel('Indices', 'FontSize', 14);
ylabel('Difference', 'FontSize', 14);
title('Residual Plot', 'FontSize', 14);
```



ARIMA

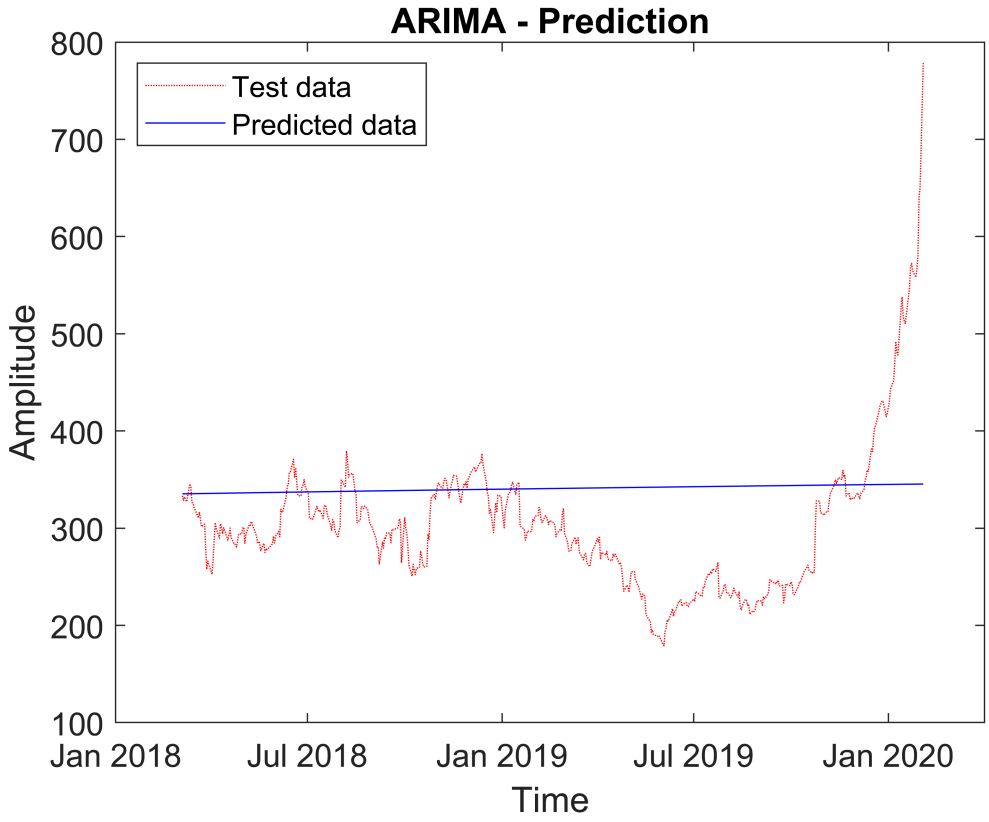
```
% Order of the time series model
P = 2;
D = 0;
Q = 0;
% Initiate the model
model = arima(P,D,Q);

% Estimate coefficients for the model
[estModel,~,loglikehood, info] = estimate(model, signal)
```

```
    ARIMA(2,0,0) Model (Gaussian Distribution):

                  Value      StandardError    TStatistic      PValue
                 _________    _____________    __________    _________

    Constant      0.27206       0.39017        0.69727       0.48563
    AR{1}          1.0334      0.016587         62.301             0
    AR{2}       -0.034103      0.016578        -2.0571      0.039675
    Variance       24.094       0.40489         59.509             0
estModel =
  arima with properties:

     Description: "ARIMA(2,0,0) Model (Gaussian Distribution)"
    Distribution: Name = "Gaussian"
               P: 2
               D: 0
               Q: 0
```

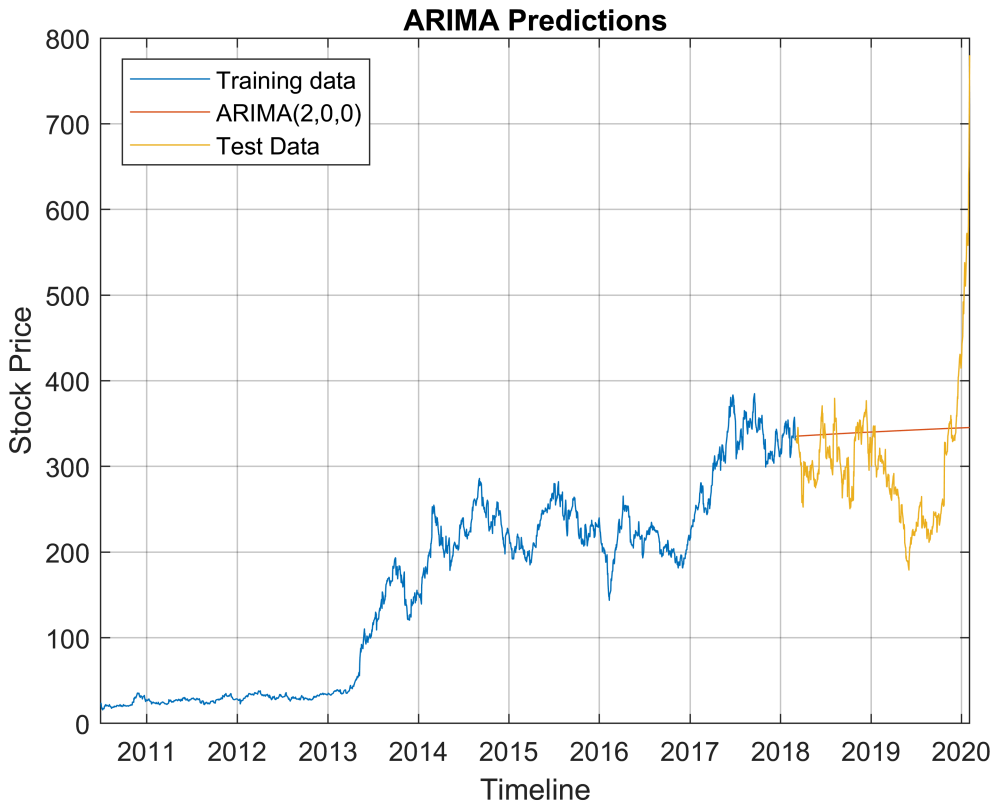
```
        Constant: 0.272057
              AR: {1.03336 -0.034103} at lags [1 2]
             SAR: {}
              MA: {}
             SMA: {}
     Seasonality: 0
            Beta: [1×0]
        Variance: 24.0943
loglikehood = -5.8167e+03
info = struct with fields:
    exitflag: 2
     options: [1×1 optim.options.Fmincon]
           X: [4×1 double]
          X0: [4×1 double]
```

Predictions

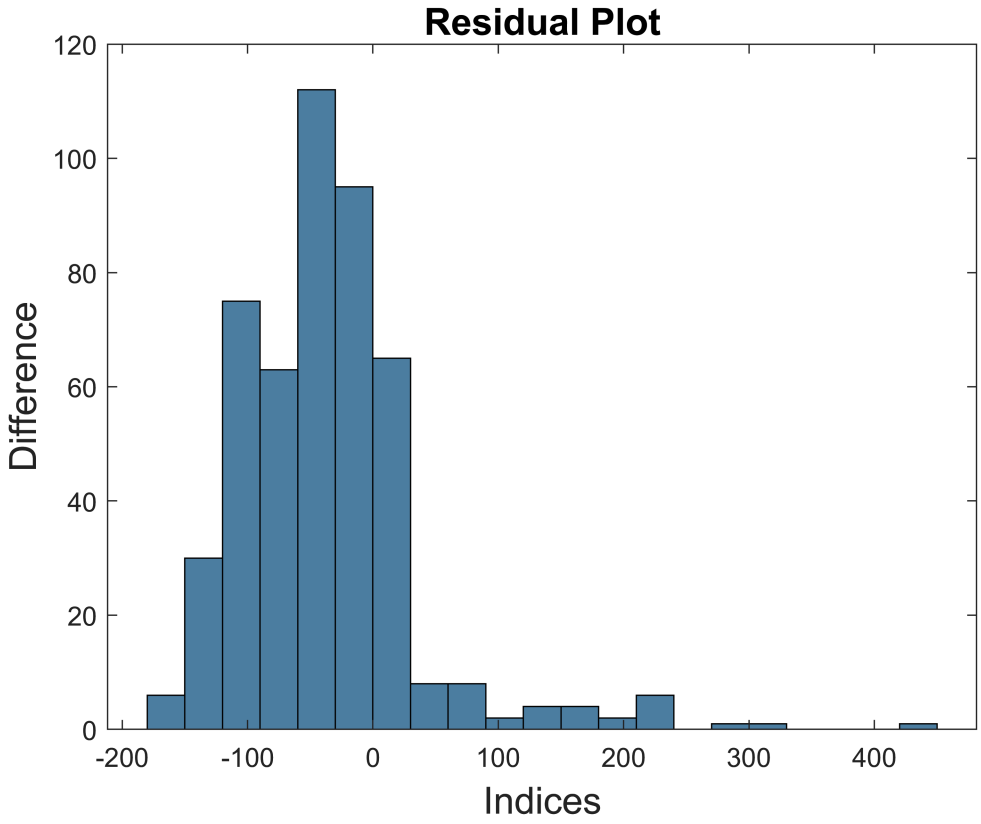
```
% Forecast the test data
len=length(time_t);
y_hat_ari = forecast(estModel,len,'Y0',signal);

% Plot the comparison
figure;
plot(X_test, Y_test, 'r:');
hold on;
plot(X_test, y_hat_ari, 'b-');
hold on;
legend('Test data', 'Predicted data', 'Location', 'NorthWest');
xlabel('Time', 'FontSize', 14);
ylabel('Amplitude', 'FontSize', 14);
title('ARIMA - Prediction', 'FontSize', 14);
set(gca, 'FontSize', 12); % Set font size for axis labels and ticks
hold off;
```




```
figure
plot(X_train,Y_train);
hold on
plot(X_test,y_hat_ari);
plot(X_test,Y_test);
title('ARIMA Predictions');
ylabel('Stock Price');
xlabel('Timeline');
legend('Training data','ARIMA(2,0,0)','Test Data','Location','northwest');
grid on
hold off
```




```
histogram(Y_test-y_hat_ari)
xlabel('Indices', 'FontSize', 14);
ylabel('Difference', 'FontSize', 14);
title('Residual Plot', 'FontSize', 14);
```



Metrics

```
% Compute Metrics
[nmse1, mape1] = metrics(Y_test, y_hat_lsp+mean(Y_test));
[nmse2, mape2] = metrics(Y_test, y_hat_ari);
```

Original Signal vs Reconstructed Signal

```
% Comparison Metrics
cmetrics = {'NMSE', 'MAPE'};
methods = {'LSP', 'ARIMA'};

% Display metrics
TM = table([nmse1 mape1]', [nmse2, mape2]', 'VariableNames', methods);
TMD = table(cmetrics', TM, 'VariableNames', {'Parameters', 'Metrics'});
disp(TMD);
```

```
    Parameters        Metrics
                    LSP      ARIMA
    __________    _________________

      'NMSE'       1.3658    1.2682
      'MAPE'       20.479     22.55
```