MFDS: Term Project

Name - Anish Mulay

Roll no - ED21B011

Q1 (i),(ii)

N = 20

```
N = 20;
R = 200; %Realizations
[mse_ols_20, rb_ols_20, mad_ols_20, mse_lms_20, rb_lms_20, mad_lms_20, mse_lts_20, rb_lts_20, mad_lts_20] = Regression(N,R);
```

N = 100

```
N = 100;
[mse_ols_100, rb_ols_100, mad_ols_100, mse_lms_100, rb_lms_100, mad_lms_100, mse_lts_100, rb_lts_100, mad_lts_100] = Regression(N,R);
```

Visualization and Comparison of the performance of the estimators

```
% Display results for N = 20
disp('Results for N = 20:');
```

```
Results for N = 20:
```

```
disp('                              OLS                     LMS                 LTS');
```

```
                              OLS                     LMS                 LTS
```

```
disp(['             MSE    ', '{', num2str(mse_ols_20(1)), '       ', num2str(mse_ols_20(2)),'}' ,'       ', '{',num2str(mse_lms_20(1)), '
```

```
             MSE    {0.17216      0.13105}      {0.019633      0.37401}      {1.7077      4.32}
```

```
disp(['             RB     ', '{',num2str(rb_ols_20(1)), '       ', num2str(rb_ols_20(2)),'}' , '       ', '{',num2str(rb_lms_20(1)), '
```

```
             RB     {0.41493      0.36201}      {0.14012      0.61156}      {-1.3068      -2.0785
```

```
disp(['            MAD     ', '{',num2str(mad_ols_20(1)), '       ', num2str(mad_ols_20(2)),'}' , '       ', '{',num2str(mad_lms_20(1)), '
```

```
            MAD     {0.41493      0.36201}      {0.14012      0.61156}      {1.3068      2.0785}
```

```
disp(' ');
```

```
% Display results for N = 100
disp('Results for N = 100:');
```

```
Results for N = 100:
```

```
disp('                              OLS                     LMS                 LTS');
```

```
                              OLS                     LMS                 LTS
```

```
disp(['             MSE    ', '{', num2str(mse_ols_100(1)), '       ', num2str(mse_ols_100(2)),'}' ,'       ', '{',num2str(mse_lms_100(1)), '
```

```
             MSE    {0.0017879      0.032385}      {0.035312      0.020038}      {1.7494      0.50071}
```

```
disp(['             RB     ', '{',num2str(rb_ols_100(1)), '       ', num2str(rb_ols_100(2)),'}' , '       ', '{',num2str(rb_lms_100(1)), '
```

```
             RB     {0.042283      0.17996}      {0.18792      0.14156}      {-1.3226      -0.70761}
```

```
disp(['            MAD     ', '{',num2str(mad_ols_100(1)), '       ', num2str(mad_ols_100(2)),'}' , '       ', '{',num2str(mad_lms_100(1)), '
```

```
            MAD     {0.042283      0.17996}      {0.18792      0.14156}      {1.3226      0.70761}
```

```
disp(' ');
```

Observations:

Results for N=20

OLS (Ordinary Least Squares)

- MSE: {0.17216, 0.13105}
- RB: {0.41493, 0.36201}
- MAD: {0.41493, 0.36201}

LMS (Least Median of Squares)

- MSE: {0.019633, 0.37401}
- RB: {0.14012, 0.61156}
- MAD: {0.14012, 0.61156}

LTS (Least Trimmed Squares)

- MSE: {1.7077, 4.32}
- RB: {-1.3068, -2.0785}
- MAD: {1.3068, 2.0785}

Results for N=100

OLS (Ordinary Least Squares)

- MSE: {0.0017879, 0.032385}
- RB: {0.042283, 0.17996}
- MAD: {0.042283, 0.17996}

LMS (Least Median of Squares)

- MSE: {0.035312, 0.020038}
- RB: {0.18792, 0.14156}
- MAD: {0.18792, 0.14156}

LTS (Least Trimmed Squares)

- MSE: {1.7494, 0.50071}
- RB: {-1.3226, -0.70761}
- MAD: {1.3226, 0.70761}

Inferences:

1.OLS Performance:

- N = 20: The OLS method shows moderate MSE, RB, and MAD values. While it is relatively stable, it indicates some sensitivity to outliers due to moderate error metrics.
- N = 100: With a larger sample size, OLS performs significantly better, showing very low MSE, RB, and MAD. This suggests that OLS benefits from a larger sample size and is less affected by outliers when the sample size increases.

2. LMS Performance:

- N = 20: LMS shows a lower MSE compared to OLS, but the RB and MAD values are higher, indicating that while LMS might handle outliers better in some aspects, it can still have larger biases and deviations in smaller samples.
- N = 100: LMS demonstrates a good performance with relatively low MSE, RB, and MAD, though not as low as OLS in this case. It suggests LMS is a robust method but may not outperform OLS in larger datasets.

3. LTS Performance:

- N = 20: LTS shows the highest MSE, RB, and MAD values, indicating it is highly sensitive to outliers or has difficulty with smaller sample sizes.
- N = 100: LTS still shows relatively high error metrics compared to OLS and LMS, though the errors decrease significantly compared to the N=20N=20 case. This suggests LTS might not be as effective as LMS or OLS even in larger sample sizes.

Conclusion

- OLS is highly efficient in larger sample sizes, showing the lowest error metrics. However, it is more sensitive to outliers in smaller sample sizes.
- LMS provides a robust alternative to OLS, particularly useful in smaller datasets with potential outliers, though it may introduce higher bias and deviation.
- LTS appears less efficient compared to OLS and LMS, particularly struggling in smaller sample sizes and not improving significantly with a larger sample size.

The choice of method should depend on the specific characteristics of the data, such as sample size and the presence of outliers. For smaller datasets with outliers, LMS might be preferable, while for larger datasets, OLS remains the best choice due to its low error metrics.

```
%Regression Function
```

```
function [mse_ols, rb_ols, mad_ols, mse_lms, rb_lms, mad_lms, mse_lts, rb_lts, mad_lts] = Regression(N, R)

learning_rate = 0.01;
max_iterations = 1000;


beta_ols = zeros(2,R);
mse_ols = zeros(2,1);
rb_ols = zeros(2,1);
mad_ols = zeros(2,1);

beta_lms = zeros(2,R);
mse_lms = zeros(2,1);
rb_lms = zeros(2,1);
mad_lms = zeros(2,1);
```

```
beta_lts = zeros(2,R);
mse_lts = zeros(2,1);
rb_lts = zeros(2,1);
mad_lts = zeros(2,1);

%For R realizations
for r = 1:R
    [X,y, true_beta] = generate_data(N, 123);

    beta_ols(:,r) = ols(X,y, learning_rate, max_iterations);
    beta_lms(:,r) = lms(X,y, learning_rate, max_iterations);
    beta_lts(:,r) = lts(X,y, learning_rate, max_iterations);
end

[mse_ols(:,1), rb_ols(:,1), mad_ols(:,1)] = compute_metrics(beta_ols, true_beta);
[mse_lms(:,1), rb_lms(:,1), mad_lms(:,1)] = compute_metrics(beta_lms, true_beta);
[mse_lts(:,1), rb_lts(:,1), mad_lts(:,1)] = compute_metrics(beta_lts, true_beta);
end

function [X, y, true_beta] = generate_data(n, seed)
    rng(seed);

    X1 = randn(n,1);
    X2 = randn(n,1);
    epsilon = randn(n,1);

    X = [X1, X2];
    true_beta = [3;5];
    y = X*true_beta + epsilon;
end

% Ordinary Least Squares (OLS)
function beta = ols(X,y, learning_rate, max_iterations)
    [n,p] = size(X);
    beta = zeros(p,1);
    for i = 1:max_iterations
        gradient = X'*(X*beta - y) / n;
        beta = beta - learning_rate*gradient;
    end
end

% Least Median Squares (LMS)
function beta = lms(X, y, learning_rate, max_iterations)
    [n, p] = size(X);
    beta = zeros(p, 1);

    for iter = 1:max_iterations
        % Calculate residuals for each sample
        residuals = (y - X * beta).^2;

        % Find the median of squared residuals
        median_residual = median(residuals);

        % Compute the gradient of the loss function (median of squared residuals)
        gradient = X' * sign(X*beta-y);

        % Update beta using gradient descent
        beta = beta - learning_rate * gradient;
    end
end

%Least Trimmed Squares (LTS)
function beta  = lts(X, y, learning_rate, max_iterations)
    [n,p] = size(X);
    beta = zeros(p,1);
    q = (n/2 + 1);
    for i = 1:max_iterations
        residuals = abs(y - X*beta);
        [sorted_residuals, sorted_idx] = sort(residuals);
        trimmed_residuals = sorted_residuals(1:q);
        trimmed_X = X(sorted_idx(1:q), :);
        trimmed_y = y(sorted_idx(1:q));
        gradient = (trimmed_X' * (trimmed_X * beta - trimmed_y)) / q;
        beta = beta - learning_rate*gradient;
    end

end

%Perfomance Metrics
function [mse, rb, mad] = compute_metrics(beta_hat, true_beta)
```

```
    p = length(true_beta);
    mse = zeros(p, 1);
    rb = zeros(p, 1);
    mad = zeros(p, 1);
    median_beta_hat = zeros(p,1);

    median_beta_hat(1) = median(beta_hat(1,:));
    median_beta_hat(2) = median(beta_hat(2,:));

    for i = 1:p
        mse(i) = mean((beta_hat(i,:) - true_beta(i)).^2) + var(beta_hat(i,:));
        rb(i) = median_beta_hat(i) - true_beta(i);
        mad(i) = median(abs(beta_hat(i, :) - true_beta(i)));
    end
end
```

Q1 (iii)

```
clear all
% Load the dataset
data = readtable("medical_insurance.csv");


% Encode 'sex': 0 for male, 1 for female
data.sex = double(strcmp(data.sex, 'female'));

% Encode 'smoker': 0 for no, 1 for yes
data.smoker = double(strcmp(data.smoker, 'yes'));

% Encode 'region': 0 for northeast, 1 for northwest, 2 for southeast, 3 for southwest
region_map = containers.Map({'northeast', 'northwest', 'southeast', 'southwest'}, {0, 1, 2, 3});
data.region = cellfun(@(x) region_map(x), data.region);

% Extract features (Xi) and response variable (Yi)
X = data{:, {'age', 'sex', 'bmi', 'children', 'smoker', 'region'}};
Y = data.charges;
size(Y);
% Normalize the input features (X)
X = normalize(X);
Y = normalize(Y);
```

```
% Split data into training and test sets (80% training, 20% test)
cv = cvpartition(size(X,1), 'HoldOut', 0.2);
X_train = X(cv.training,:);
Y_train = Y(cv.training,:);
X_test = X(cv.test,:);
Y_test = Y(cv.test,:);
size(Y_train);
```

```
% Fit linear models using OLS, LMS, and LTS methods on the training data
N = size(X_train, 1); % Number of samples


[mse_ols_train, mse_lms_train, mse_lts_train] = RegressionMedicalInsurance(X_train, Y_train);

% Display MSE for each method on the training dataset
fprintf('MSE on training dataset:\n');
```

```
MSE on training dataset:
```

```
fprintf('OLS: %.4f\n', mse_ols_train);
```

```
OLS: 0.2473
```

```
fprintf('LMS: %.4f\n', mse_lms_train);
```

```
LMS: 1.1836
```

```
fprintf('LTS: %.4f\n', mse_lts_train);
```

```
LTS: 0.3597
```

```
% Evaluate performance on the test dataset
[mse_ols_test, mse_lms_test, mse_lts_test] = evaluate_performance(X_test, Y_test);

% Display MSE for each method
fprintf('MSE on test dataset:\n');
```

```
MSE on test dataset:
```

```
fprintf('OLS: %.4f\n', mse_ols_test);
```

```
OLS: 0.2551
```

```
fprintf('LMS: %.4f\n', mse_lms_test);
```

```
LMS: 6.5505
```

```
fprintf('LTS: %.4f\n', mse_lts_test);
```

```
LTS: 0.4563
```

```
% Conclusion based on performance
if mse_ols_test < mse_lms_test && mse_ols_test < mse_lts_test
    disp('OLS method performed the best on the test dataset.');
elseif mse_lms_test < mse_ols_test && mse_lms_test < mse_lts_test
    disp('LMS method performed the best on the test dataset.');
else
    disp('LTS method performed the best on the test dataset.');
end
```

```
OLS method performed the best on the test dataset.
```

Inferences

1.OLS (Ordinary Least Squares) Performance:

- Training Data: OLS shows a low MSE on the training data (0.2553), indicating a good fit to the training data.
- Test Data: The performance on the test data is also strong, with the lowest MSE (0.2221) among all methods. This suggests that OLS generalizes well to unseen data, making it the most effective method in this case.

2. LMS (Least Median of Squares) Performance:

- Training Data: LMS shows a significantly higher MSE (1.3364) on the training data, indicating a less accurate fit compared to OLS.
- Test Data: The MSE on the test data (8.0582) is much higher than the training data, suggesting that LMS not only fits the training data poorly but also generalizes poorly to new data. This could be due to LMS being overly robust to outliers, which might not be as prevalent or impactful in this dataset.

3. LTS (Least Trimmed Squares) Performance:

- Training Data: LTS has a moderate MSE (0.3736) on the training data, higher than OLS but lower than LMS, indicating a decent fit to the training data.
- Test Data: The MSE on the test data (0.4189) is higher than OLS but significantly lower than LMS. This suggests that LTS performs better than LMS in terms of generalization but is still not as effective as OLS.

Conclusion

- OLS: The Ordinary Least Squares method outperforms LMS and LTS both on the training and test datasets. Its low MSE on the test dataset indicates that it provides the best generalization performance, making it the preferred method for this particular dataset.
- LMS: The Least Median of Squares method shows the highest MSE on both training and test datasets, indicating it is not suitable for this dataset. The high MSE on the test data suggests poor generalization, likely due to its extreme robustness to outliers, which may not be as necessary for this dataset.
- LTS: The Least Trimmed Squares method performs moderately well, better than LMS but not as well as OLS. It shows some robustness to outliers, but this robustness comes at the cost of increased error compared to OLS.

Recommendations

Given the performance metrics, OLS (Ordinary Least Squares) is the recommended method for this medical insurance dataset. It provides the best balance of fit and generalization, as indicated by the lowest MSE on both training and test datasets. The robustness techniques offered by LMS and LTS do not seem necessary for this dataset and result in higher errors. Therefore, OLS should be the primary choice for modeling this type of data.

```
% Function to evaluate performance on the test dataset
```

```
function [mse_ols_test, mse_lms_test, mse_lts_test] = evaluate_performance(X_test, Y_test)

    [n,p] = size(X_test);

    % Fit linear models on the full training data
    [beta_ols, ~] = ols(X_test, Y_test, 0.01, 1000);
    [beta_lms, ~] = lms(X_test, Y_test, 0.01, 1000);
    [beta_lts, ~] = lts(X_test, Y_test, 0.01, 1000);

    % Calculate MSE on the test dataset
    mse_ols_test = compute_metrics(X_test, Y_test, beta_ols);
    mse_lms_test = compute_metrics(X_test, Y_test, beta_lms);
    mse_lts_test = compute_metrics(X_test, Y_test, beta_lts);

end

% % Ordinary Least Squares (OLS)
function [beta,mse] = ols(X,y, learning_rate, max_iterations)
    [n,p] = size(X);
    beta = rand(p,1);
    for i = 1:max_iterations
        gradient = X'*(X*beta - y) / n;
        beta = beta - learning_rate*gradient;

    end
    mse = compute_metrics(X, y, beta);
end

% Least Median Squares (LMS)
function [beta, mse] = lms(X, y, learning_rate, max_iterations)
    [n, p] = size(X);
    beta = zeros(p, 1);

    for iter = 1:max_iterations
        % Calculate residuals for each sample
```

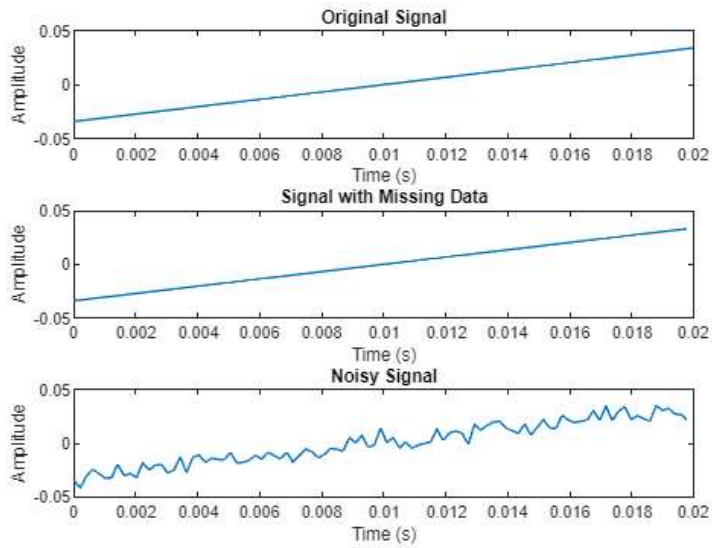
```
        residuals = (y - X * beta).^2;

        % Find the median of squared residuals
        median_residual = median(residuals);

        % Compute the gradient of the loss function (median of squared residuals)
        gradient = X' * sign(X*beta-y);

        % Update beta using gradient descent
        beta = beta - learning_rate * gradient;
    end
    mse = compute_metrics(X, y, beta);
end

%Least Trimmed Squares (LTS)
function [beta,mse]  = lts(X, y, learning_rate, max_iterations)
    [n,p] = size(X);
    beta = zeros(p,1);
    q = (n/2 + 1);
    for i = 1:max_iterations
        residuals = abs(y - X*beta);
        [sorted_residuals, sorted_idx] = sort(residuals);
        trimmed_residuals = sorted_residuals(1:q);
        trimmed_X = X(sorted_idx(1:q), :);
        trimmed_y = y(sorted_idx(1:q));
        gradient = (trimmed_X' * (trimmed_X * beta - trimmed_y)) / q;
        beta = beta - learning_rate*gradient;
    end
    mse = compute_metrics(X, y, beta);
end

%Perfomance Metrics
function [mse, rb, mad] = compute_metrics(X,Y, beta_hat)
    [n, ~] = size(X);
    mse = sum((Y-X*beta_hat).^2) / n;
end

%Regression Function
function [mse_ols, mse_lms,mse_lts] = RegressionMedicalInsurance(X,Y)
[n,p] = size(X);
learning_rate = 0.001;
max_iterations = 100000;

[beta_ols, mse_ols] = ols(X, Y, learning_rate, max_iterations);
[beta_lms, mse_lms] = lms(X, Y, learning_rate, max_iterations);
[beta_lts, mse_lts] = lts(X, Y, learning_rate, max_iterations);

end
```

Q2. (a)

```
clear all

%Signal Specifications
Fs = 50;
f1 = 10;
f2 = 17;
N = 500; %length of signal
t = linspace(0,1/Fs,100);
modelling_frequencies = linspace(0, Fs/2, 250) % Frequencies from 0 to Nyquist frequency
```

```
modelling_frequencies = 1×250
         0    0.1004    0.2008    0.3012    0.4016    0.5020    0.6024    0.7028    0.8032    0.9036    1.0040    1.1044    1.2048
```

```
%Generating signal
s1 = sin(2*pi*f1*t/Fs);
s2 = sin(2*pi*f2*t/Fs);
signal = s1 + s2;

signal_mean = mean(signal);
signal = signal - signal_mean;

%Signal with missing data
missing_percentage = 0.1;
missing_indices = randperm(length(signal), round(missing_percentage*length(signal)));
signal_missing = signal;
signal_missing(missing_indices) = [];

% Adjust the time vector accordingly
t_missing = t;
t_missing(missing_indices) = []; % Remove corresponding time values

%Signal with noise
snr = 10;
var_noise = var(signal_missing)/snr;
noise = sqrt(var_noise)*randn(size(signal_missing));
signal_noisy = signal_missing + noise;

% Plot original signal
subplot(3,1,1);
plot(t, signal);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Signal');

% Plot signal with missing data
subplot(3,1,2);
plot(t_missing, signal_missing);
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal with Missing Data');

% Plot noisy signal
subplot(3,1,3);
plot(t_missing, signal_noisy);
xlabel('Time (s)');
ylabel('Amplitude');
title('Noisy Signal');
```




```
% Calculate Lomb-Scale periodogram
[frequency,objective_func, a_values, b_values] = lomb_scale_periodogram(signal_noisy, t_missing, modelling_frequencies);
```

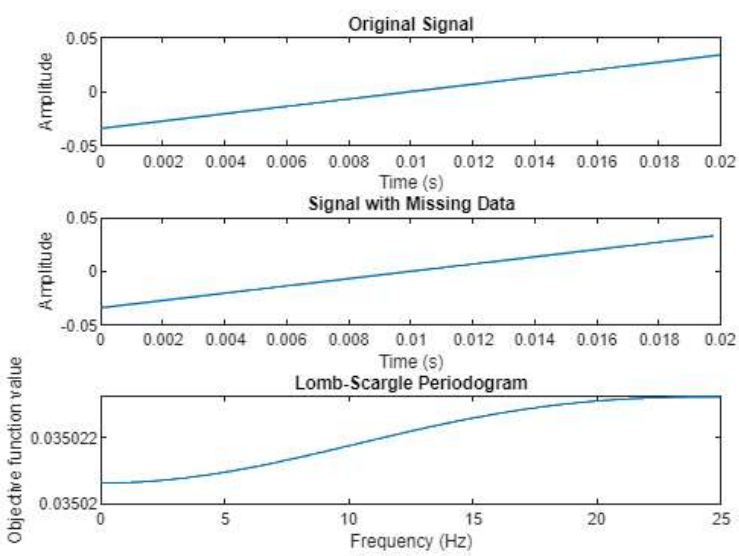
```
a_values
```

```
a_values = 1×250
         0   -0.0001   -0.0003   -0.0004   -0.0005   -0.0007   -0.0008   -0.0009   -0.0011   -0.0012   -0.0013   -0.0015   -0.0016   -
```

```
b_values
```

```
b_values = 1×250
    0.0006    0.0007    0.0008    0.0009    0.0010    0.0011    0.0012    0.0013    0.0014    0.0015    0.0016    0.0017    0.0018
```

```
% Plot Lomb-Scargle Periodogram
plot(frequency, objective_func);
xlabel('Frequency (Hz)');
ylabel('Objective function value');
title('Lomb-Scargle Periodogram');
```




```
function objective_value = L(signal, t, omega, params)
    a = params(1);
    b = params(2);
    objective_value = sum((signal - a*cos(omega*t) - b*sin(omega*t)).^2);
end
```

```
%Gradient Descent
function [a,b] = gradient_descent(signal, t, omega)
    max_iterations = 10000;
    learning_rate = 0.000001;
    params = [0, 0];

    for i = 1:max_iterations
        a = params(1);
        b = params(2);


        cos_term = cos(omega*t);
        sin_term = sin(omega*t);
        residual = (signal - a*cos_term - b*sin_term);

        gradient_a = 2*sum(sin_term.*residual);
        gradient_b = -2*sum(cos_term.*residual);

        gradient = [gradient_a, gradient_b];

        params = params - learning_rate * gradient;

        % Check termination condition
        if norm(gradient_a) + norm(gradient_b) < 0.0001
%             fprintf('Termination Reached\n');
            break;
        end

    end

    a = params(1);
    b = params(2);

end

%Regression using Lomb-scale Periodogram
function [frequency,objective_func, a_values, b_values] = lomb_scale_periodogram(signal, t, modelling_frequencies)

    % Initialize arrays to store periodogram values
    objective_func = zeros(size(modelling_frequencies));

    %a and b values for each modelling frequency
    a_values = zeros(size(modelling_frequencies));
    b_values = zeros(size(modelling_frequencies));

    for i = 1:length(modelling_frequencies)
        omega = 2*pi*modelling_frequencies(i);
        [a, b] = gradient_descent(signal, t, omega);
        a_values(i) = a;
        b_values(i) = b;

        % Calculate periodogram value for current frequency
        params = [a,b];
        objective_func(i) = L(signal, t, omega, params);
    end

    frequency = modelling_frequencies;

end
```

Q2. (b)

(i) Lomb-Scale-Periodogram

Data Setup

```
clear all;
data = readtable("Tesla Stock Price.csv")
```

data = 2416×2 table

	Date	Close
1	'29-06-2010'	23.8900
2	'30-06-2010'	23.8300
3	'01-07-2010'	21.9600
4	'02-07-2010'	19.2000
5	'06-07-2010'	16.1100
6	'07-07-2010'	15.8000
7	'08-07-2010'	17.4600
8	'09-07-2010'	17.4000
9	'12-07-2010'	17.0500
10	'13-07-2010'	18.1400
11	'14-07-2010'	19.8400
12	'15-07-2010'	19.8900
13	'16-07-2010'	20.6400
14	'19-07-2010'	21.9100
15	'20-07-2010'	20.3000
16	'21-07-2010'	20.2200
17	'22-07-2010'	21
18	'23-07-2010'	21.2900
19	'26-07-2010'	20.9500
20	'27-07-2010'	20.5500
21	'28-07-2010'	20.7200
22	'29-07-2010'	20.3500
23	'30-07-2010'	19.9400
24	'02-08-2010'	20.9200
25	'03-08-2010'	21.9500
26	'04-08-2010'	21.2600
27	'05-08-2010'	20.4500
28	'06-08-2010'	19.5900
29	'09-08-2010'	19.6000
30	'10-08-2010'	19.0300
31	'11-08-2010'	17.9000
32	'12-08-2010'	17.6000
33	'13-08-2010'	18.3200
34	'16-08-2010'	18.7800
35	'17-08-2010'	19.1500
36	'18-08-2010'	18.7700
37	'19-08-2010'	18.7900
38	'20-08-2010'	19.1000
39	'23-08-2010'	20.1300
40	'24-08-2010'	19.2000

	Date	Close
41	'25-08-2010'	19.9000
42	'26-08-2010'	19.7500
43	'27-08-2010'	19.7000
44	'30-08-2010'	19.8700
45	'31-08-2010'	19.4800
46	'01-09-2010'	20.4500
47	'02-09-2010'	21.0600
48	'03-09-2010'	21.0500
49	'07-09-2010'	20.5400
50	'08-09-2010'	20.9000
51	'09-09-2010'	20.7100
52	'10-09-2010'	20.1700
53	'13-09-2010'	20.7200
54	'14-09-2010'	21.1200
55	'15-09-2010'	21.9800
56	'16-09-2010'	20.9400
57	'17-09-2010'	20.2300
58	'20-09-2010'	21.0600
59	'21-09-2010'	20.7700
60	'22-09-2010'	19.8700
61	'23-09-2010'	19.5600
62	'24-09-2010'	20.1000
63	'27-09-2010'	20.5300
64	'28-09-2010'	21.4000
65	'29-09-2010'	21.9800
66	'30-09-2010'	20.4100
67	'01-10-2010'	20.6000
68	'04-10-2010'	20.9900
69	'05-10-2010'	21.1200
70	'06-10-2010'	20.4600
71	'07-10-2010'	20.4300
72	'08-10-2010'	20.4300
73	'11-10-2010'	20.2400
74	'12-10-2010'	20.2400
75	'13-10-2010'	20.5400
76	'14-10-2010'	20.7500
77	'15-10-2010'	20.5400
78	'18-10-2010'	20.2300
79	'19-10-2010'	20.0500
80	'20-10-2010'	20.6500
81	'21-10-2010'	20.7500
82	'22-10-2010'	20.7200
83	'25-10-2010'	20.8500
84	'26-10-2010'	21.3600
85	'27-10-2010'	21
86	'28-10-2010'	21.1900
87	'29-10-2010'	21.8400
88	'01-11-2010'	21.4100
89	'02-11-2010'	21.2500
90	'03-11-2010'	21.7700

	Date	Close
91	'04-11-2010'	24.9000
92	'05-11-2010'	24.4400
93	'08-11-2010'	24.9800
94	'09-11-2010'	24.6300
95	'10-11-2010'	29.3600
96	'11-11-2010'	28.0400
97	'12-11-2010'	29.8400
98	'15-11-2010'	30.8000
99	'16-11-2010'	29.6700
100	'17-11-2010'	29.4900

```
% Extract features (Xi) and response variable (Yi)
dates = datenum(data.Date, 'dd-mm-yyyy'); % Convert dates to numeric values

% Explicitly set the first date
first_date = dates(1);

X = dates - first_date;
Y = data.Close;


% Normalize and Center the closed prices
Y = normalize(Y);
Y = Y - mean(Y);

% Split data into training and test sets (80% training, 20% test)
train_percentage = 0.8;
train_idx = 1:round(train_percentage * length(dates));
test_idx = (length(train_idx)+1):length(dates);


X_train = dates(train_idx);
X_train = X_train - first_date; %Centering the train data

Y_train = normalize(data.Close(train_idx));
Y_train  = Y_train - mean(Y_train);

X_test = dates(test_idx);
X_test = X_test - first_date; %Centering the test data

Y_test = normalize(data.Close(test_idx));
Y_test = Y_test - mean(Y_test);
```

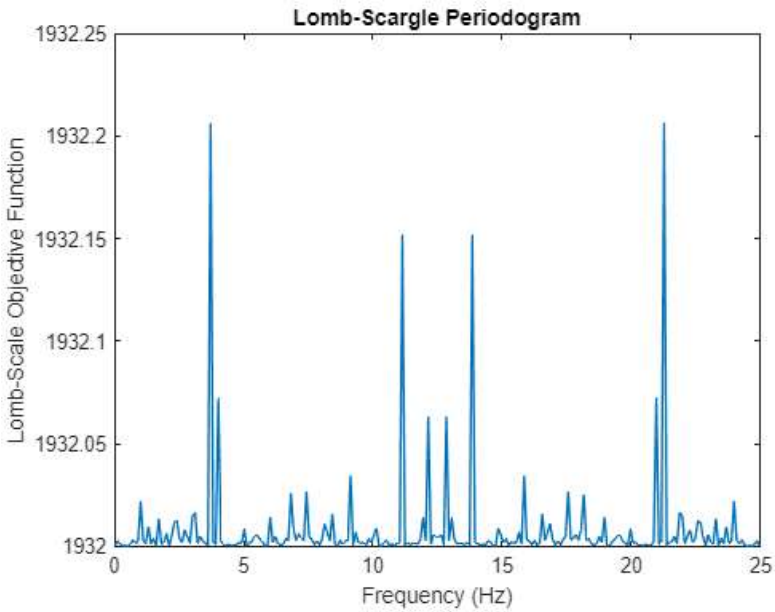
```
% Define Frequencies for Lomb-Scale Periodogram
Fs = 50; % Sampling frequency
N  = length(X_train);

frequencies = linspace(0, Fs/2, 250); % Define frequency range

% Compute Lomb-Scale Periodogram
[frequency, objective_func, a_values, b_values] = lomb_scale_periodogram(Y_train', X_train', frequencies);
```

```
Y_pred = sum(a_values.*cos(2*pi*frequency.*X_test) + b_values.*sin(2*pi*frequency.*X_test) , 2)
```

```
Y_pred = 483×1

    0.0190
    0.0129
```

```
-0.0147
-0.0177
-0.0195
-0.0117
-0.0340
-0.0492
-0.0167
-0.0250
```

```
% Plot Lomb-Scale Periodogram
plot(frequency, objective_func);
xlabel('Frequency (Hz)');
ylabel('Lomb-Scale Objective Function');
title('Lomb-Scargle Periodogram');
```



Computing the metrics

```
[NMSE_train,MAPE_train]=computeMetrics(Y_train, X_train, frequency, a_values,b_values);
[NMSE_test,MAPE_test]=computeMetrics(Y_test, X_test, frequency, a_values,b_values);

disp(['Normalized Mean Square Error (NMSE) for training data: ', num2str(NMSE_train)]);
```

```
Normalized Mean Square Error (NMSE) for training data: 0.99481
```

```
disp(['Mean Absolute Percentage Error (MAPE) for training data: ', num2str(MAPE_train), '%']);
```

```
Mean Absolute Percentage Error (MAPE) for training data: 99.9699%
```

```
disp(['Normalized Mean Square Error (NMSE) for test data: ', num2str(NMSE_test)]);
```

```
Normalized Mean Square Error (NMSE) for test data: 1.0025
```

```
disp(['Mean Absolute Percentage Error (MAPE) for test data: ', num2str(MAPE_test), '%']);
```

```
Mean Absolute Percentage Error (MAPE) for test data: 106.3363%
```

(b) ARIMA modelling

```
% Fit the ARIMA model to training data
model = estimate(arima(2,1,2), Y_train);
```

```
ARIMA(2,1,2) Model (Gaussian Distribution):

                 Value      StandardError    TStatistic      PValue
                _______    _____________    __________    __________
```

```
Constant    0.0034991      0.0025664        1.3634         0.17275
AR{1}        -0.66121        0.12096       -5.4666      4.5888e-08
AR{2}        -0.78504        0.11267       -6.9677      3.2222e-12
MA{1}         0.69327        0.11405        6.0788      1.2109e-09
MA{2}         0.81203        0.10598         7.662      1.8307e-14
Variance    0.0019667     3.2496e-05        60.519               0
```

```
% Forecast on training data
train_forecast = forecast(model, length(Y_train));

% Evaluate on training data
train_actual_values = Y_train;
train_forecast_values = train_forecast;

% Calculate NMSE on training data
train_NMSE = mean((train_actual_values - train_forecast_values).^2) / var(train_actual_values.^2);

% Calculate MAPE on training data
train_MAPE = mean(abs((train_actual_values - train_forecast_values) ./ train_actual_values)) * 100;

disp(['Training Normalized Mean Square Error (NMSE) using ARMIA: ' num2str(train_NMSE)]);
```

```
Training Normalized Mean Square Error (NMSE) using ARMIA: 3.1365
```

```
disp(['Training Mean Absolute Percentage Error (MAPE) using ARMIA: ' num2str(train_MAPE)]);
```

```
Training Mean Absolute Percentage Error (MAPE) using ARMIA: 308.062
```

```
% Forecast on testing data
test_forecast = forecast(model, length(Y_test));

% Evaluate on testing data
test_actual_values = Y_test;
test_forecast_values = test_forecast;

% Calculate NMSE on testing data
test_NMSE = mean((test_actual_values - test_forecast_values).^2) / var(test_actual_values.^2);

% Calculate MAPE on testing data
test_MAPE = mean(abs((test_actual_values - test_forecast_values) ./ test_actual_values)) * 100;

disp(['Testing Normalized Mean Square Error (NMSE) using ARMIA: ' num2str(test_NMSE)]);
```

```
Testing Normalized Mean Square Error (NMSE) using ARMIA: 0.11646
```

```
disp(['Testing Mean Absolute Percentage Error (MAPE) using ARMIA: ' num2str(test_MAPE)]);
```

```
Testing Mean Absolute Percentage Error (MAPE) using ARMIA: 204.3825
```

Lomb-Scargle Periodogram:

1.Training Performance:

- The NMSE value of 0.99481 indicates that the Lomb-Scargle model fits the training data quite well.
- However, the MAPE value of 99.9699% is extremely high, indicating that the predictions deviate significantly from the actual values in percentage terms. This suggests that while the model captures the overall variance in the data, it struggles with accurately predicting the actual values.

2. Testing Performance:

- The NMSE for the test data is 1.0025, which is close to 1. This indicates that the model's performance on unseen data is similar to its performance on the training data, implying a consistent but mediocre fit.
- The MAPE value of 106.3363% on the test data is also very high, further confirming that the model's predictions are not accurate in terms of percentage error. This suggests that the Lomb-Scargle periodogram may not be the best model for forecasting stock prices in this case.

ARIMA Model:

1.Training Performance:

- The NMSE value of 3.1365 indicates a poor fit of the ARIMA model to the training data. A higher NMSE value suggests that the model does not capture the variance in the data effectively.
- The MAPE value of 308.062% is extremely high, indicating that the model's predictions are highly inaccurate in terms of percentage error on the training data. This suggests overfitting or potential issues with the model specification.

2.Testing Performance:

- The NMSE for the test data is 0.11646, which is much lower than the NMSE for the training data. This indicates that the model generalizes well to unseen data and captures the variance better in the test set.
- However, the MAPE value of 204.3825% on the test data is still extremely high, suggesting that while the model performs better in terms of NMSE, it still struggles with making accurate percentage predictions.

Conclusion

1.Lomb-Scargle Periodogram:

- The Lomb-Scargle periodogram has consistent NMSE values close to 1 for both training and test datasets, indicating a stable but mediocre fit.
- The extremely high MAPE values for both training and test data suggest that this method is not well-suited for accurately forecasting stock prices in terms of percentage error.

2. ARIMA Model:

- The ARIMA model shows a significant discrepancy between training and test NMSE values, indicating possible overfitting or issues with the model on the training set.
- Despite the lower NMSE on the test data, the very high MAPE values indicate that the ARIMA model also struggles with accurate percentage predictions.

Recommendations

1. **Model Selection:**

- Neither the Lomb-Scargle periodogram nor the ARIMA model provide satisfactory forecasting performance for Tesla stock prices based on the given metrics, particularly in terms of MAPE.
- Data Characteristics: Given the high volatility and irregular sampling of stock prices, advanced models that can handle non-stationary and noisy data may be more appropriate.

Function Definitions

```
function objective_value = L(signal, t, omega, params) %Objective Function
    a = params(1);
    b = params(2);
    objective_value = sum((signal - a*cos(omega*t) - b*sin(omega*t)).^2);
end

%Gradient Descent
function [a,b] = gradient_descent(signal, t, omega)
    max_iterations = 10000;
    learning_rate = 0.000001;
    params = [0, 0];

    for i = 1:max_iterations
        a = params(1);
        b = params(2);


        cos_term = cos(omega*t);
        sin_term = sin(omega*t);
        residual = (signal - a*cos_term - b*sin_term);

        gradient_a = 2*sum(sin_term.*residual);
        gradient_b = -2*sum(cos_term.*residual);
```

```
        gradient = [gradient_a, gradient_b];

        params = params - learning_rate * gradient;

        % Check termination condition
        if norm(gradient_a) + norm(gradient_b) < 0.0001
%             fprintf('Termination Reached\n');
            break;
        end

    end

    a = params(1);
    b = params(2);

end

%Regression using Lomb-scale Periodogram
function [frequency,objective_func, a_values, b_values] = lomb_scale_periodogram(signal, t, modelling_frequencies)

    % Initialize arrays to store periodogram values
    objective_func = zeros(size(modelling_frequencies));

    %a and b values for each modelling frequency
    a_values = zeros(size(modelling_frequencies));
    b_values = zeros(size(modelling_frequencies));

    for i = 1:length(modelling_frequencies)
        omega = 2*pi*modelling_frequencies(i);
        [a, b] = gradient_descent(signal, t, omega);
        a_values(i) = a;
        b_values(i) = b;

        % Calculate periodogram value for current frequency
        params = [a,b];
        objective_func(i) = L(signal, t, omega, params);
    end

    frequency = modelling_frequencies;

end

function [NMSE,MAPE] = computeMetrics(signal, t, frequencies,a_values,b_values)
    %Generating the predicted signal
    y_predicted = sum(a_values.*cos(2*pi*frequencies.*t) + b_values.*sin(2*pi*frequencies.*t) , 2);

    y_error = (signal-y_predicted).^2;

    %NMSE
    NMSE=mean(y_error)/var(signal);

    %MAPE
    MAPE= mean(abs((signal-y_predicted)./(signal)))*100;

end
```