```
clear; clc; close all;

%% Question 1 %% Maximum Likelihood

rng('default')  % For reproducibility

mu1     = [-.75 .5];
Sigma1  = [.5 .3 ; .3 .8];


X_10 = mvnrnd(mu1,Sigma1,10);

% % Visualize the data
figure;
plot(X_10(:,1),X_10(:,2),'+')
title('Scatter Plot of Normal Distribution for 10 samples');
xlabel('x1')
ylabel('x2')

% Maximum likelihood estimation for 10 samples

N_10                = size(X_10,1);      % Number of samples
mean_estimator_10   = sum(X_10,1)/N_10;  % Summing along the first dimension (rows)

% Initialize outer product
outer_product1      = zeros(size(2, 2));  % Initialize to the correct size

for i=1:N_10
    outer_product1 = outer_product1 + (transpose(X_10(i,:)) - transpose
(mean_estimator_10))*transpose(transpose(X_10(i,:)) - transpose
(mean_estimator_10));
end

varience_estimator_10 = outer_product1/N_10;

% Maximum likelihood estimation for 1000 samples
X_1000 = mvnrnd(mu1,Sigma1,1000);

% Visualize the data
figure;
plot(X_1000(:,1),X_1000(:,2),'+')
title('Scatter Plot of Normal Distribution for 1000 samples');
xlabel('x1')
ylabel('x2')

N_1000                = size(X_1000,1);     % Number of samples
mean_estimator_1000   = sum(X_1000,1)/N_1000;    % Summing along the first dimension
(rows)

% Initialize outer product
outer_product2 = zeros(size(2, 2));  % Initialize to the correct size

for i=1:N_1000
```

```
    outer_product2  = outer_product2 + (X_1000(i,:)' - mean_estimator_1000')*
(transpose(X_1000(i,:)' - mean_estimator_1000'));
end

varience_estimator_1000 = outer_product2/N_1000;
clear; clc; close all;

%% Question 2 %% Bayesian Parameter Estimation
rng('default') % For reproducibility

sigma    = 0.7;
% i)
mu1      = 3;
N_25     = 25;
x_i      = normrnd(mu1,sqrt(sigma),[N_25,1]);


% define values for y-axis
y        = zeros(length(x_i),1);
group    = ones(length(x_i), 1);   % All samples in one group
% Plot
figure;
gscatter(x_i, y, group, 'br', '.',18);
title('Scatter Plot of Normal Distribution for 25 samples');
xlabel("x")

% Maximum Likelihood Estimation
mean_estimator_i = sum(x_i)/N_25;

%ii)
mu_mu        = 2.8; % mean parameter of random variable mean
sigma_mu     = .8; % variance parameter of random variable mean

w1_ii        = (N_25*sigma_mu )/(N_25*sigma_mu  + sigma );
w2_ii        = (sigma )/(N_25*sigma_mu  + sigma );

mu_map_ii    = w1_ii * mean_estimator_i + w2_ii * mu_mu;

% iii)

N_1000   = 1000;
x_iii    = normrnd(mu1,sigma,[N_1000,1]);

% define values for y-axis
y        = zeros(length(x_iii),1);
group    = ones(length(x_iii), 1);   % All samples in one group
% Plot
figure;
gscatter(x_iii, y, group, 'br', '.',18);
title('Scatter Plot of Normal Distribution for 1000 samples');
xlabel("x")

% Maximum Likelihood Estimation
```

```
mean_estimator_iii = sum(x_iii)/N_1000;

% MAP Estimation
w1_iii          = (N_1000*sigma_mu )/(N_1000*sigma_mu  + sigma );
w2_iii          = (sigma )/(N_1000*sigma_mu  + sigma );

mu_map_iii      = w1_iii*mean_estimator_iii + w2_iii * mu_mu; % Prior infoya daha↙
yakın yorum yazarken aklında bulundurursun kıps :D


%% Question 3 %% Minimum error-rate classifier
rng(55)  % For reproducibility

% Mean vectors of classes
mu1     = [-1, -1]';
mu2     = [1, 1]';

% Covarience matrix
sigma   = [1.4 .2; .2 .28];

% Generate sets
omega1  = mvnrnd(mu1,sigma,500);
omega2  = mvnrnd(mu2,sigma,500);


% Randomly select 250 indices for training
indices1     = randperm(500, 250); % Randomly select 250 indices from 1 to 500
indices2     = randperm(500, 250); % For omega2 as well

% Split train-test sets using the random indices
omega1_train = omega1(indices1, :);
omega1_test  = omega1(setdiff(1:500, indices1), :); % Remaining points for testing
omega2_train = omega2(indices2, :);
omega2_test  = omega2(setdiff(1:500, indices2), :);

% Create a new figure for the plot
figure;

% Plot training data
scatter(omega1_train(:,1), omega1_train(:,2), 25, 'r', 'filled'); % Class 1↙
training data
hold on;
scatter(omega2_train(:,1), omega2_train(:,2), 25, 'b', 'filled'); % Class 2↙
training data
hold off;
title('Training Data (Class 1 and Class 2)');
xlabel('x1');
ylabel('x2');
legend('Class 1 Training', 'Class 2 Training');
grid on;

% Plot test data
figure;
```

```
scatter(omega1_test(:,1), omega1_test(:,2), 25, 'r', 'o'); % Class 1 test data
hold on
scatter(omega2_test(:,1), omega2_test(:,2), 25, 'b', 'o'); % Class 2 test data
hold off

title('Test Data');
xlabel('x1');
ylabel('x2');
legend( 'Class 1 Test', 'Class 2 Test');
grid on;


% Decision boundary from eq REFF
n   = (mu1 - mu2)'; % normal of the boundary line
x0  = (mu1 + mu2)/2;

% Decision boundary function
decision_boundary  = @(x1, x2) n * inv(sigma) * ([x1; x2] - x0);

% Classification and Error Calculation for Train Data
train_data            = [omega1_train; omega2_train];
train_labels          = [ones(250, 1); 2 * ones(250, 1)]; % 1 for class 1, 2 for
class 2
predicted_labels1     = zeros(size(train_labels));

for i = 1:length(train_data)
    x = train_data(i, :)';
    if decision_boundary(x(1), x(2)) > 0
        predicted_labels1(i) = 1; % Assign to class 1
    else
        predicted_labels1(i) = 2; % Assign to class 2
    end
end

% Calculate error rate
error_rate1 = sum(predicted_labels1 ~= train_labels) / length(train_labels);
fprintf('Error Rate for Train Data: %.2f%%\n', error_rate1 * 100);

% Classification and Error Calculation for Test Data
test_data            = [omega1_test; omega2_test];
test_labels          = [ones(250, 1); 2 * ones(250, 1)]; % 1 for class 1, 2 for
class 2
predicted_labels     = zeros(size(test_labels));

for i = 1:length(test_data)
    x = test_data(i, :)';
    if decision_boundary(x(1), x(2)) > 0
        predicted_labels(i) = 1; % Assign to class 1
    else
        predicted_labels(i) = 2; % Assign to class 2
    end
end
```

```
% Calculate error rate
error_rate = sum(predicted_labels ~= test_labels) / length(test_labels);
fprintf('Error Rate for Test Data: %.2f%%\n', error_rate * 100);

% Plot decision boundary on the train data
figure;
scatter(omega1_train(:, 1), omega1_train(:, 2), 25, 'r', 'filled'); % Class 1 test
data
hold on;
scatter(omega2_train(:, 1), omega2_train(:, 2), 25, 'b', 'filled'); % Class 2 test
data
fimplicit(@(x1, x2) decision_boundary(x1, x2), [-4 4 -4 4], 'k', 'LineWidth', 1.5);
hold off;

title(sprintf('Train Data with Decision Boundary (Error Rate: %.2f%%)', error_rate1
* 100));
xlabel('x1');
ylabel('x2');
legend('Class 1 Train', 'Class 2 Train', 'Decision Boundary');
grid on;

% Plot decision boundary on the test data
figure;
scatter(omega1_test(:, 1), omega1_test(:, 2), 25, 'r', 'o'); % Class 1 test data
hold on;
scatter(omega2_test(:, 1), omega2_test(:, 2), 25, 'b', 'o'); % Class 2 test data
fimplicit(@(x1, x2) decision_boundary(x1, x2), [-4 4 -4 4], 'k', 'LineWidth', 1.5);
hold off;

title(sprintf('Test Data with Decision Boundary (Error Rate: %.2f%%)', error_rate *
100));
xlabel('x1');
ylabel('x2');
legend('Class 1 Test', 'Class 2 Test', 'Decision Boundary');
grid on;

%% Question 3 iii)

% Maximum Likelihood estimator
N1               = size(omega1_train,1);
N2               = size(omega2_train,1);

mu1_estimator    = sum(omega1_train,1)/N1;
mu2_estimator    = sum(omega2_train,1)/N2;

% Initialize outer product

outer_product1 = zeros(size(2, 2));  % Initialize to the correct size

for i=1:N1
    outer_product1 = outer_product1 + (omega1_train(i,:)' - mu1_estimator')*
(transpose(omega1_train(i,:)' - mu1_estimator'));
end
```

```
variance_estimator1 = outer_product1/N1;

outer_product2 = zeros(size(2, 2));  % Initialize to the correct size

for i=1:N2
    outer_product2  = outer_product2 + (omega2_train(i,:)' - mu2_estimator')* ↙
(transpose(omega2_train(i,:)' - mu2_estimator'));
end

variance_estimator2 = outer_product2/N2;

% Decision boundary from eq REFF
n_2   = (mu1_estimator - mu2_estimator); % normal of the boundary line
x0_2  = (mu1_estimator + mu2_estimator)'/2;

% Decision boundary function
decision_boundary_2  = @(x1_2, x2_2) n_2 * inv(variance_estimator1) * ([x1_2; x2_2] ↙
- x0_2);

% Classification and Error Calculation for Train Data
train_data_2           = [omega1_train; omega2_train];
train_labels_2         = [ones(250, 1); 2 * ones(250, 1)]; % 1 for class 1, 2 for ↙
class 2
predicted_labels1_2     = zeros(size(train_labels));

for i = 1:length(train_data_2)
    x = train_data_2(i, :)';
    if decision_boundary_2(x(1), x(2)) > 0
        predicted_labels1_2(i) = 1; % Assign to class 1
    else
        predicted_labels1_2(i) = 2; % Assign to class 2
    end
end

% Calculate error rate
error_rate1_2 = sum(predicted_labels1_2 ~= train_labels_2) / length ↙
(train_labels_2);
fprintf('Error Rate for Train Data ML: %.2f%%\n', error_rate1_2 * 100);

% Classification and Error Calculation for Test Data
test_data_2            = [omega1_test; omega2_test];
test_labels_2          = [ones(250, 1); 2 * ones(250, 1)]; % 1 for class 1, 2 for ↙
class 2
predicted_labels_2     = zeros(size(test_labels));

for i = 1:length(test_data_2)
    x = test_data_2(i, :)';
    if decision_boundary_2(x(1), x(2)) > 0
        predicted_labels_2(i) = 1; % Assign to class 1
    else
        predicted_labels_2(i) = 2; % Assign to class 2
    end
```

```
end

% Calculate error rate
error_rate_2 = sum(predicted_labels_2 ~= test_labels_2) / length(test_labels_2);
fprintf('Error Rate for Test Data ML: %.2f%%\n', error_rate_2 * 100);

% Plot decision boundary on the train data
figure;
scatter(omega1_train(:, 1), omega1_train(:, 2), 25, 'r', 'filled'); % Class 1 test
data
hold on;
scatter(omega2_train(:, 1), omega2_train(:, 2), 25, 'b', 'filled'); % Class 2 test
data
fimplicit(@(x1_2, x2_2) decision_boundary_2(x1_2, x2_2), [-4 4 -4 4], 'k',
'LineWidth', 1.5);
hold off;

title(sprintf('Train Data with Decision Boundary ML (Error Rate: %.2f%%)',
error_rate1_2 * 100));
xlabel('x1');
ylabel('x2');
legend('Class 1 Train', 'Class 2 Train', 'Decision Boundary');
grid on;

% Plot decision boundary on the test data
figure;
scatter(omega1_test(:, 1), omega1_test(:, 2), 25, 'r', 'o'); % Class 1 test data
hold on;
scatter(omega2_test(:, 1), omega2_test(:, 2), 25, 'b', 'o'); % Class 2 test data
fimplicit(@(x1_2, x2_2) decision_boundary_2(x1_2, x2_2), [-4 4 -4 4], 'k',
'LineWidth', 1.5);
hold off;

title(sprintf('Test Data with Decision Boundary ML (Error Rate: %.2f%%)',
error_rate_2 * 100));
xlabel('x1');
ylabel('x2');
legend('Class 1 Test', 'Class 2 Test', 'Decision Boundary');
grid on;

%% Question 4 %%  Non-parametric Density Estimation
% Parameters of normal distributed feature vector
mu          = [-1.5; 1.5];
sigma       = [.8 .2; .2 .6];

% Number of samples
N           = 50;  % Reduced sample size for faster computation

% Grid for plotting the Gaussian distribution
[X1, X2]    = meshgrid(linspace(-6, 6, 500), linspace(-6, 6, 500));
X           = [X1(:) X2(:)];

% True distribution
```

```
gaus_true   = mvnpdf([X1(:) X2(:)], mu', sigma);
gaus_true   = reshape(gaus_true, length(X2), length(X1));

% Plot true distribution
figure;
surf(X1, X2, gaus_true, 'EdgeColor', 'interp');
title('True Distribution')
xlabel('X1')
ylabel('X2')
zlabel('PDF')

% Generate samples from the distribution
X_samples = mvnrnd(mu, sigma, N);

% Different h1 values for testing
h1_values = [0.5, 1, 2, 3, 5, 10];
num_h1 = length(h1_values);

% Loop over each h1 value and perform Parzen window density estimation

% Gaussian Parzen window estimation
figure;
for idx = 1:num_h1
    h1  = h1_values(idx);
    % Initial volume
    Vo  = h1^3;
    % Adaptive volume and h
    V   = Vo / sqrt(N);
    h   = V^(1/3);

    % Estimate density with Gaussian Parzen window
    p_gaussian = zeros(size(X,1), 1);    % Initialize density function
    for k = 1:size(X,1)
        % Take one feature from true distribution
        x   = X(k, :);

        sum = 0;
        for n = 1:N
            % Calculate each samples effect
            xk  = X_samples(n, :);
            sum = sum + parzen_window((x - xk) / h);

        end
        p_gaussian(k) = sum / (N * V);
    end
    p_gaussian = reshape(p_gaussian, length(X2), length(X1));

    % Plot the Gaussian Parzen window estimate for the current h1
    subplot(2, 3, idx);  % Arrange in a 2x3 grid
    surf(X1, X2, p_gaussian, 'EdgeColor', 'interp');
    title(['Gaussian Parzen (h1 = ', num2str(h1), ')(V = ', num2str(V), ')']);
    xlabel('X1');
    ylabel('X2');
```

```
    zlabel('PDF');
end

% Cubic Parzen window estimation
figure;
for idx = 1:num_h1
    h1 = h1_values(idx);
    Vo = h1^3;
    V = Vo / sqrt(N);
    h = V^(1/3);

    % Estimate density with cubic Parzen window
    p_cubic = zeros(size(X,1), 1);
    for k = 1:size(X,1)
        x = X(k, :);
        sum = 0;
        for n = 1:N
            xk = X_samples(n, :);
            sum = sum + parzen_window_cub((x - xk) / h);
        end
        p_cubic(k) = sum / (N * V);
    end
    p_cubic = reshape(p_cubic, length(X2), length(X1));

    % Plot the Cubic Parzen window estimate for the current h1
    subplot(2, 3, idx);  % Arrange in a 2x3 grid
    surf(X1, X2, p_cubic, 'EdgeColor', 'interp');
    title(['Cubic Parzen (h1 = ', num2str(h1), ')(V = ', num2str(V), ')']);
    xlabel('X1');
    ylabel('X2');
    zlabel('PDF');
end

%% Question 5
% Load dataset
load fisheriris

% Extract feature and labels
X = meas(:,1:2);     % Features, take length and width
Y = species;         % Labels

figure;
gscatter(X(:,1),X(:,2),species,'rgb','osd');
xlabel('Sepal length');
ylabel('Sepal width');
% PCA
[coeff, score, eigenvalues] = pca(X);

% Take basis and normal vectors
basis   = coeff(:,1);
normal  = coeff(:,2);

% Mean vector of data
```

```
m       = mean(X,1);

% Scores give the projection of each point onto the line
[n,p]   = size(X);

% Fitting lines
Xfit    = repmat(m,n,1) + score(:,1)*coeff(:,1)'; % Most principle
Xfit2   = repmat(m,n,1) + score(:,2)*coeff(:,2)'; % Least principle


% Error
error   = abs((X - repmat(m,n,1))*normal);
sse     = sum(error.^2);

% Plot PCA
figure;
t = [min(score(:,1)) - 0.2, max(score(:,1)) + 0.2];
endpts = [m + t(1) * basis'; m + t(2) * basis'];
plot(endpts(:,1), endpts(:,2), 'k-', 'LineWidth', 1.5);
hold on;
colors = 'rgb';
classes = unique(Y);

idx1 = strcmp(Y, classes{1});
idx2 = strcmp(Y, classes{2});
idx3 = strcmp(Y, classes{3});

plot(X(idx1,1), X(idx1,2), 'ro', 'MarkerFaceColor', 'none');  % Blue diamond
outline with no fill
plot(X(idx2,1), X(idx2,2), 'gs', 'MarkerFaceColor', 'none');
plot(X(idx3,1), X(idx3,2), 'bd', 'MarkerFaceColor', 'none');


for i = 1:length(classes)
    % Select data points for the current class
    idx = strcmp(Y, classes{i});
    % Plot projections on the PCA line
    X1 = [X(idx,1) Xfit(idx,1) nan*ones(sum(idx),1)];
    X2 = [X(idx,2) Xfit(idx,2) nan*ones(sum(idx),1)];
    plot(X1', X2', '-', 'Color', colors(i));
end

hold off;
legend('Fitting Line','Setosa','versicolor','virginica', 'Location', 'best');
xlabel('Sepal length');
ylabel('Sepal width');
title('Orthogonal Regression using PCA with Projections by Class');


% Plot of most principle direction
X_setosa        = Xfit(1:50,1);
X_versicolor    = Xfit(50:100,1);
X_virginica     = Xfit(100:150,1);
```

```
figure;
subplot(1,2,1)
histogram(X_setosa, 10); % 10 bins for Setosa
hold on;
histogram(X_versicolor, 10); % 10 bins for Versicolor
histogram(X_virginica, 10); % 10 bins for Virginica
hold off;
title("Histogram of of the PCA-reduced 1D data on most principle direction")
xlabel('Principal Component 1');
ylabel('Frequency');

% Plot of least principle direction
X_setosa2        = Xfit2(1:50,1);
X_versicolor2    = Xfit2(50:100,1);
X_virginica2     = Xfit2(100:150,1);

subplot(1,2,2)
histogram(X_setosa2, 10); % 10 bins for Setosa
hold on;
histogram(X_versicolor2, 10); % 10 bins for Versicolor
histogram(X_virginica2, 10); % 10 bins for Virginica
hold off;
title("Histogram of of the PCA-reduced 1D data on least principle direction")
xlabel('Principal Component 2');
ylabel('Frequency');
%% Parzen Functions
% Gaussian shape Parzen window function
function f = parzen_window(u)
    u = norm(u);
    f = (1/sqrt(2 * pi)) * exp(-0.5 * u^2);
end

% Cubic Parzen window function
function f_cub = parzen_window_cub(u)
    if norm(u) <= 1/2
        f_cub = 1;
    else
        f_cub = 0;
    end
end
```