```
clear; clc; close all;

% Load Fisher's iris data set. Use the petal lengths and widths as predictors.
load fisheriris
X = meas(:,3:4);

figure;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title 'Fisher''s Iris Data';
xlabel 'Petal Lengths (cm)';
ylabel 'Petal Widths (cm)';

rng(1); % For reproducibility

C = rand(3,2); % Randomly initialized centroids
x1 = min(X(:,1)):0.01:max(X(:,1));
x2 = min(X(:,2)):0.01:max(X(:,2));
[x1G,x2G] = meshgrid(x1,x2);
XGrid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot

% K-Means with random centroids
idx2Region = kmeans(XGrid,3,'MaxIter',25,'Start',C);
[idx,C_1,sumd] = kmeans(X,3,"Distance","sqeuclidean","Start",C);

% Assigns each node in the grid to the closest centroid
figure;
gscatter(XGrid(:,1),XGrid(:,2),idx2Region,...
    [0,0.75,0.75;0.75,0,0.75;0.75,0.75,0],'..');
hold on;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title(['Fisher''s Iris Data Random Mean Values (C = [', num2str(C(:)'), '])']);
xlabel 'Petal Lengths (cm)';
ylabel 'Petal Widths (cm)';
legend('Region 1','Region 2','Region 3','Data','Location','SouthEast');
hold off;

% Measure Clustering Quality
Quality = class_quality(X,idx,C);

% K-Means with better initial centroids
C_better = [4.5, 1.5; 6, 2; 1.5, 0.25];
idx2Region_better = kmeans(XGrid,3,'MaxIter',1,'Start',C_better);
[idx2,C_better1,sumd2] = kmeans(X,3,'Start',C_better,'Distance','sqeuclidean');

% Assigns each node in the grid to the closest centroid
figure;
gscatter(XGrid(:,1),XGrid(:,2),idx2Region_better,...
    [0,0.75,0.75;0.75,0,0.75;0.75,0.75,0],'..');
hold on;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title(['Fisher''s Iris Data Better Initialized Mean Values (C_{better} = [',
num2str(C_better(:)'), '])']);
xlabel 'Petal Lengths (cm)';
```

```
ylabel 'Petal Widths (cm)';
legend('Region 1','Region 2','Region 3','Data','Location','SouthEast');
hold off;

% Measure Clustering Quality
Quality_better = class_quality(X,idx2,C_better1);

% K-Means with centroids based on class means
C_better2 = [mean(X(1:50,:)); mean(X(51:100,:)); mean(X(101:end,:))];
[idx3,C_better3,sumd3] = kmeans(X,3,'Start',C_better2,'Distance','sqeuclidean');
idx2Region_better2 = kmeans(XGrid,3,'MaxIter',1,'Start',C_better2);

% Assigns each node in the grid to the closest centroid
figure;
gscatter(XGrid(:,1),XGrid(:,2),idx2Region_better2,...
    [0,0.75,0.75;0.75,0,0.75;0.75,0.75,0],'..');
hold on;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title(['Fisher''s Iris Data Better Initialized Mean Values 2 (C_{better2} = [',
num2str(C_better2(:)'), '])']);
xlabel 'Petal Lengths (cm)';
ylabel 'Petal Widths (cm)';
legend('Region 1','Region 2','Region 3','Data','Location','SouthEast');
hold off;

% Measure Clustering Quality
Quality_better2 = class_quality(X,idx3,C_better3);
Quality_random = class_quality(X,idx3,C_1);

clear; clc; close all;
load fisheriris
classes = unique(species);

% Use principal component analysis to reduce the dimension of the data to two
dimensions for visualization.
[~,score] = pca(meas(:,3:4),'NumComponents',1);

GMModels = cell(3,1); % Preallocation
options = statset('MaxIter',1000);
rng(1); % For reproducibility

for j = 1:4
    GMModels{j} = fitgmdist(score,j,'Options',options);
    fprintf('\n GM Mean for %i Component(s)\n',j)
    Mu = GMModels{j}.mu;
end

figure;
for j = 1:4
    subplot(2,2,j)
    h1 = gscatter(score,zeros(size(score)),species);
    h = gca;
    hold on
```

```
    gmPDF = @(x) arrayfun(@(x0) pdf(GMModels{j},x0),x);
    fplot(gmPDF, [h.XLim(1), h.XLim(2)]);
    % fcontour(gmPDF,[h.XLim h.YLim],'MeshDensity',100)
    title(sprintf('GM Model - %i Component(s)',j));
    xlabel('1st principal component');
    ylabel('2nd principal component');
    if(j ~= 3)
        legend off;
    end
    hold off
end
g = legend(h1);
g.Position = [0.7 0.25 0.1 0.1];


clear; clc; close all;
% Define the data matrix (each row is an observation, each column is a variable)
X = [0 1 2 3;
     1 0 4 5;
     2 4 0 6;
     3 5 6 0];

Z = linkage(X, 'centroid');
% Mahalanobis Distance
D_mahalanobis = pdist(X, 'mahalanobis'); % Precompute Mahalanobis distances
Z_mahalanobis = linkage(D_mahalanobis, 'centroid'); % Perform hierarchical
clustering

% Minkowski Distance (with p = 3)
D_minkowski = pdist(X, 'minkowski', 1); % p = 1
Z_minkowski = linkage(D_minkowski, 'centroid');

% Standardized Euclidean Distance
D_seuclidean = pdist(X, 'seuclidean'); % Precompute Standardized Euclidean
distances
Z_seuclidean = linkage(D_seuclidean, 'centroid');

% cosine Distance
D_cosine = pdist(X, 'cosine'); % Precompute Standardized Euclidean distances
Z_cosine = linkage(D_cosine, 'centroid');

% correlation Distance
D_corr = pdist(X, 'correlation'); % Precompute Standardized Euclidean distances
Z_correlation = linkage(D_corr, 'centroid');

% Display the linkage matrices
disp('Linkage Matrix (Mahalanobis):');
disp(Z_mahalanobis);
disp('Linkage Matrix (Minkowski):');
disp(Z_minkowski);
disp('Linkage Matrix (Standardized Euclidean):');
disp(Z_seuclidean);
```

```
% Plot dendrograms for each distance metric

figure;
subplot(2,3,1);
dendrogram(Z);
title('Dendrogram (Euclidean)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,2);
dendrogram(Z_mahalanobis);
title('Dendrogram (Mahalanobis)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,3);
dendrogram(Z_minkowski);
title('Dendrogram (Minkowski, p=1)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,4);
dendrogram(Z_seuclidean);
title('Dendrogram (Standardized Euclidean)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,5);
dendrogram(Z_cosine);
title('Dendrogram (Cosine)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,6);
dendrogram(Z_correlation);
title('Dendrogram (Correlation)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

clear; clc; close all;

% Load Fisher Iris Dataset
load fisheriris
X = meas(:,3:4);

% Scatter plot of the original data
figure;
gscatter(X(:,1), X(:,2), species);
title('Original Data Scatter Plot');
xlabel('Petal Length (cm)');
ylabel('Petal Width (cm)');

% Compute distance matrix and similarity matrix
```

```
dist_temp = pdist(X);
dist = squareform(dist_temp);

% Similarity matrix
S = exp(-dist.^2);
disp(['Is Similarity Matrix Symmetric: ', num2str(issymmetric(S))]);
% Compute degree matrix
D = diag(sum(S, 2));

% Compute Laplacian Matrices
% 1. Unnormalized Laplacian
L_unnormalized = D - S;

% 2. Symmetric Normalized Laplacian
L_sym = eye(size(S)) - D^(-1/2) * S * D^(-1/2);

% Visualize Laplacian Matrices
figure;
imagesc(L_unnormalized);
colorbar;
title('Unnormalized Laplacian Matrix (L)');
xlabel('Nodes');
ylabel('Nodes');

figure;
imagesc(L_sym);
colorbar;
title('Symmetric Normalized Laplacian Matrix (L_{sym})');
xlabel('Nodes');
ylabel('Nodes');

figure;
imagesc(S);
colorbar;
title('Similarity Matrix (S)');
xlabel('Nodes');
ylabel('Nodes');

% Spectral clustering
k = 3; % Number of clusters
rng('default') % For reproducibility
idx = spectralcluster(S, k, 'Distance', 'precomputed', 'LaplacianNormalization',↙
'symmetric');

% Spectral clustering with unnormalized Laplacian
idx_unnormalized = spectralcluster(S, k, 'Distance', 'precomputed',↙
'LaplacianNormalization', 'none');

% Plot results with symmetric Laplacian
figure;
gscatter(X(:,1), X(:,2), idx);
title('Spectral Clustering with Normalized Laplacian');
xlabel('Petal Length (cm)');
```

```
ylabel('Petal Width (cm)');

% Plot results with unnormalized Laplacian
figure;
gscatter(X(:,1), X(:,2), idx_unnormalized);
title('Spectral Clustering with Unnormalized Laplacian');
xlabel('Petal Length (cm)');
ylabel('Petal Width (cm)');

% Mahalanobis Distance
[idx_mahalanobis, ~] = spectralcluster(X, k,'NumNeighbors', size(X,1), 'Distance',↙
'mahalanobis');
figure;
gscatter(X(:,1), X(:,2), idx_mahalanobis);
title('Spectral Clustering with Mahalanobis Distance');

% KernelScale = 0.1
idx1 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 0.1,↙
'LaplacianNormalization', 'symmetric');

% KernelScale = 1
idx2 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 1,↙
'LaplacianNormalization', 'symmetric');

% KernelScale = 10
idx3 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 10,↙
'LaplacianNormalization', 'symmetric');

% KernelScale = 15
idx4 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 15,↙
'LaplacianNormalization', 'symmetric');

% Visualize the results
figure;

gscatter(X(:,1), X(:,2), idx1);
title('KernelScale = 0.1');
xlabel('X1');
ylabel('X2');
figure;
gscatter(X(:,1), X(:,2), idx2);
title('KernelScale = 1');
xlabel('X1');
ylabel('X2');
figure;
gscatter(X(:,1), X(:,2), idx3);
title('KernelScale = 10');
xlabel('X1');
ylabel('X2');

figure;
gscatter(X(:,1), X(:,2), idx4);
title('KernelScale = 15');
```

```
xlabel('X1');
ylabel('X2');

% Convert species to numerical labels for comparison
true_labels = grp2idx(species); % 1 = setosa, 2 = versicolor, 3 = virginica

% Q_default = correct_classification(true_labels,idx,k);
% Q_unnormalized = correct_classification(true_labels,idx_unnormalized,k);
% Q_mahalanobis = correct_classification(true_labels,idx_mahalanobis,k);
% Q_kernel01 = correct_classification(true_labels,idx1,k);
% Q_kernel1 = correct_classification(true_labels,idx2,k);
% Q_kernel10 = correct_classification(true_labels,idx3,k);


% Compute distance matrix
dist_temp = pdist(X);
dist = squareform(dist_temp);

% Define different KernelScale values
kernel_scales = [0.1, 1, 10, 15];

% Plot similarity matrices for each KernelScale
figure;
for i = 1:length(kernel_scales)
    % Compute similarity matrix with given KernelScale
    kernel_scale = kernel_scales(i);
    S = exp(-dist.^2 / (2 * kernel_scale^2));

    % Plot the similarity matrix
    subplot(2, 2, i);
    imagesc(S);
    colorbar;
    title(['Similarity Matrix (KernelScale = ', num2str(kernel_scale), ')']);
    xlabel('Data Points');
    ylabel('Data Points');
end


function [correct_class1, correct_class2,correct_class3] = correct_classification↙
(true_labels,idx,k)

    % Initialize counts
    correct_class1 = 0;
    correct_class2 = 0;
    correct_class3 = 0;

    % Compare true labels with predicted clusters
    for i = 1:k
        % Find majority cluster for each class
        class_indices = (true_labels == i);
        predicted_clusters = idx(class_indices);

        % Majority voting: most frequent cluster in true class
```

```
        most_frequent_cluster = mode(predicted_clusters);
        correct_count = sum(predicted_clusters == most_frequent_cluster);

        % Store correct counts
        if i == 1
            correct_class1 = correct_count;
        elseif i == 2
            correct_class2 = correct_count;
        elseif i == 3
            correct_class3 = correct_count;
        end
    end

    % Display results
    fprintf('Correct classifications:\n');
    fprintf('Class 1 (Setosa): %d / %d\n', correct_class1, sum(true_labels == 1));
    fprintf('Class 2 (Versicolor): %d / %d\n', correct_class2, sum(true_labels ==
2));
    fprintf('Class 3 (Virginica): %d / %d\n', correct_class3, sum(true_labels ==
3));
end



% Function to compute clustering quality
function Quality = class_quality(X,idx,C)
Quality = 0;
    for i = 1:length(X)
        Quality = Quality + norm(X(i,:) - C(idx(i),:)).^2;
    end
end
```