```
% Pierce Zhang, CMOR220, Fall 2023, Project 7: The Shape of Curves
% shapeAnalysis.m
% Classifies shapes using distance minimization calculation
% Last modified: 26 October 2023

% Driver function
function shapeAnalysis
```

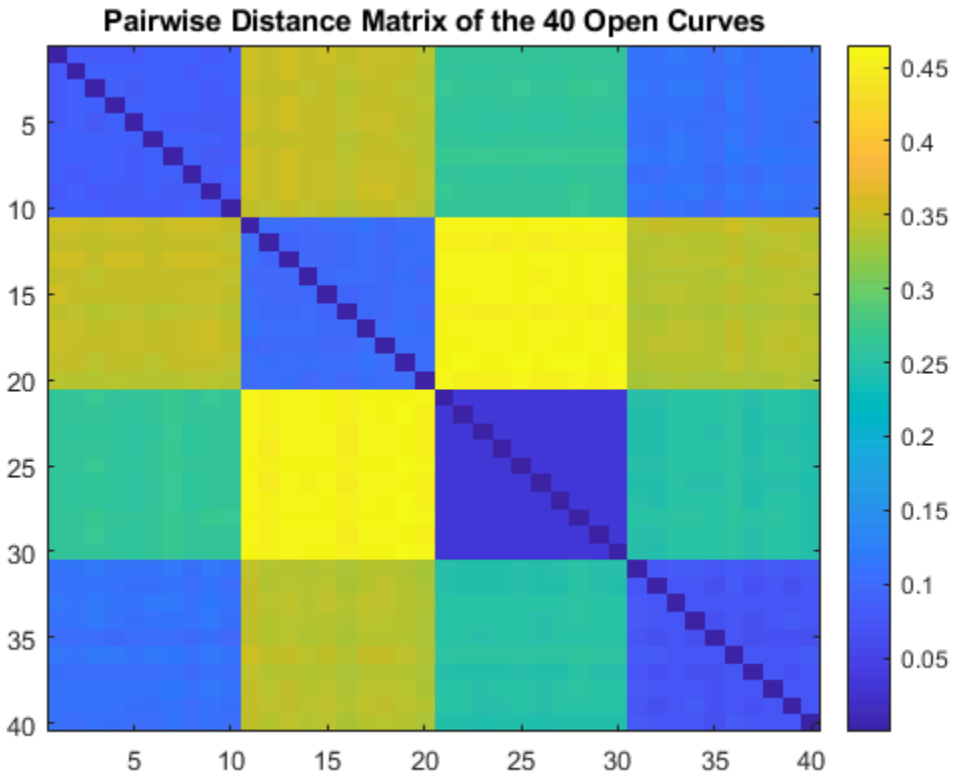
PART 1: OPEN CURVES

```
% TRAINING SECTION -- ESTABLISHES THE CURVE DATA TO "TRAIN" THE MODEL

% Define the range of x values
x_values = 0:0.02:pi;

% Create training curves: y1_train, y2_train, y3_train, y4_train
y1_train = [x_values; cos(x_values)];
y2_train = [x_values; sin(x_values)];
y3_train = [x_values; x_values.^2];
y4_train = [x_values; x_values];

% Initialize a matrix to store training data for 40 curves
training_data = zeros(2, length(x_values), 40);

% Assign training data based on curve type
for i = 1:40
    if i <= 10
        training_data(:,:,i) = y1_train;
    elseif i > 10 && i <= 20
        training_data(:,:,i) = y2_train;
    elseif i > 20 && i <= 30
        training_data(:,:,i) = y3_train;
    else
        training_data(:,:,i) = y4_train;
    end
end

% Add noise and preprocess the training data
for i = 1:40
    training_data(:,:,i) = addNoise(training_data(:,:,i));
    training_data(:,:,i) = preprocessData(training_data(:,:,i));
end

% Initialize a matrix to store pairwise distances between training curves
pairwise_distance_matrix = zeros(40, 40);

% Calculate distances between training curves
for n1 = 1:40
    for n2 = 1:40
        pairwise_distance_matrix(n1, n2) =
 calculateDistanceOpen(training_data(:,:,n1), training_data(:,:,n2));
```

```
    end
end

% Display the pairwise distance matrix as an image
figure();
imagesc(pairwise_distance_matrix);
colorbar;
title("Pairwise Distance Matrix of the 40 Open Curves");

% TESTING SECTION -- COMPARES TEST CURVES TO THE TRAINING DATA TO CREATE A
 VERDICT

% Create testing data: 25 test curves per type, 100 test curves in total
testing_data = zeros(2, length(x_values), 100);

% Assign testing data based on curve type
for i = 1:100
    if i <= 25
        testing_data(:,:,i) = y1_train;
    elseif i > 25 && i <= 50
        testing_data(:,:,i) = y2_train;
    elseif i > 50 && i <= 75
        testing_data(:,:,i) = y3_train;
    else
        testing_data(:,:,i) = y4_train;
    end
end

% Add noise and preprocess the testing data
for i = 1:100
    testing_data(:,:,i) = addNoise(testing_data(:,:,i));
    testing_data(:,:,i) = preprocessData(testing_data(:,:,i));
end

% Initialize a count for correct classifications
correct_count = 0;

% Compare test curves to training data
for n1 = 1:100 % n1 refers to test data index
    test_distances = zeros(1, 40);

    % Calculate the true class of the test data
    true_class = ceil(n1/25);

    % Use training data to classify the test data
    for n2 = 1:40
        test_distances(n2) = calculateDistanceOpen(testing_data(:,:,n1),
 training_data(:,:,n2));
    end

    % Find the index with the minimum distance
    [~, index] = min(test_distances);

    % Calculate the observed class based on the index
```

```
    observed_class = ceil(index/10);

    % Check if the observed class matches the true class
    if observed_class == true_class
        correct_count = correct_count + 1;
    end
end

% Calculate the percentage of correct classifications
percent_correct_open = (correct_count/100) * 100

% Clear the workspace
clear;
```

```
percent_correct_open =

   100
```

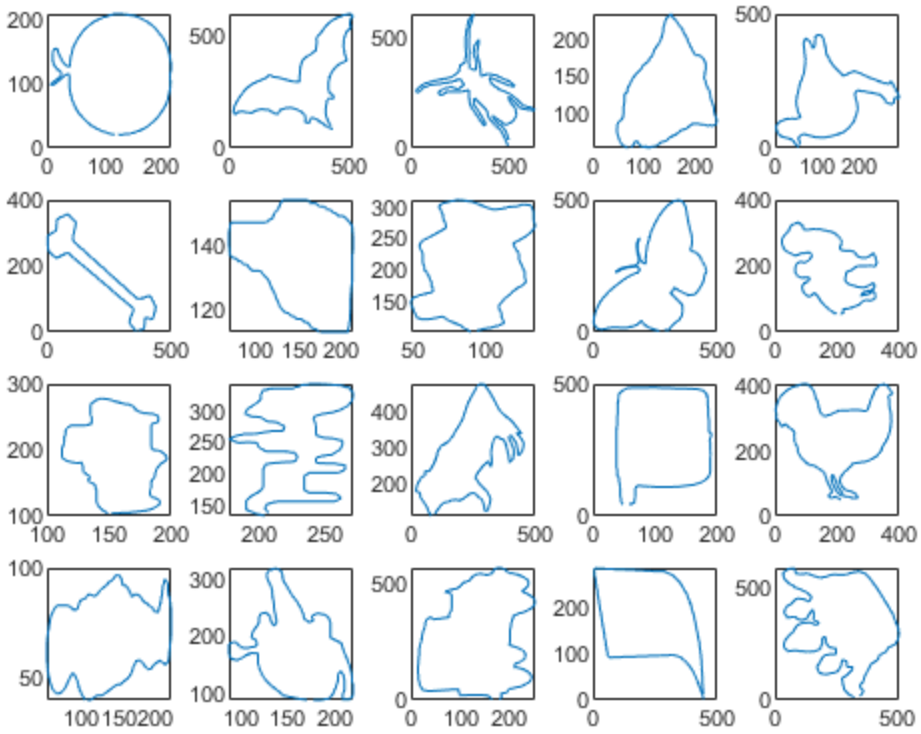


PART TWO: CLOSED CURVES

```
% LOAD DATA AND PLOT
load("DataClassification.mat")
figure();
```

```
% Plot training data for each cluster
for n = 1:20
    subplot(4, 5, n);
    plot(trainingdata(1, :, (n-1)*15+1), trainingdata(2, :, (n-1)*15+1))
end

% Preprocess testing and training data
for n = 1:100
    testdata(:, :, n) = preprocessData(testdata(:, :, n));
end

for n = 1:300
    trainingdata(:, :, n) = preprocessData(trainingdata(:, :, n));
end

% Initialize a count for correct classifications
Cnt=0;

% Compare test curves to training data for closed curves
for n=1:100
    TDist=zeros(300,100);

    % Calculate the true cluster of the test data
    TrClst=ceil(n/5);

    % Calculate distances between test data and training data
    for m=1:300
        for k=1:100
            TstCv=[testdata(:,k:100,n),testdata(:,1:(k-1),n)];
            TDist(m,k)=calculateDistanceClosed(TstCv,trainingdata(:,:,m));
        end
    end

    % Find the minimum distance
    Val=min(min(TDist));
    [Idx,~]=find(TDist==Val);

    % Calculate the observed cluster based on the index
    ObsClst=ceil(Idx/15);

    % Check if the observed cluster matches the true cluster
    if TrClst==ObsClst
        Cnt=Cnt+1;
    end
end

% Calculate the percentage of correct classifications
percent_correct_closed = (Cnt/100)*100
```

```
percent_correct_closed =

    98
```



```
end

% Function to add random noise to a curve
function noisyCurve = addNoise(curve)
    for n = 1:length(curve(1, :, :))
        % Add random noise to the x-coordinate
        noisyCurve(1, n, :) = curve(1, n, :) + randn() * 0.05;
        % Add random noise to the y-coordinate
        noisyCurve(2, n, :) = curve(2, n, :) + randn() * 0.05;
    end
end

% Function to preprocess a curve
function preprocessedCurve = preprocessData(curve)
    % Calculate the mean value of the curve
    mean_value = mean(curve, 2);
    % Center the curve by subtracting the mean
    centeredCurve = curve - mean_value;
    % Normalize the centered curve
    preprocessedCurve = centeredCurve / norm(centeredCurve, 'fro');
end

% Function to calculate the distance between open curves
function distance = calculateDistanceOpen(curve1, curve2)
    % Calculate the dot product between the curves
    product = curve1 * curve2';
```

```
    % Perform singular value decomposition on the product
    [U, ~, V] = svd(product);
    if det(U * V') > 0
        % Calculate the Frobenius norm distance
        distance = norm(curve1 - (U * V') * curve2, 'fro');
    else
        % Calculate the Frobenius norm distance with sign correction
        distance = norm(curve1 - (U * [1 0; 0 -1] * V') * curve2, 'fro');
    end
end

% Function to calculate the distance between closed curves
function [D]=calculateDistanceClosed(Cv1,Cv2)
    A=Cv1*Cv2';
    [U,~,V]=svd(A);
    if det(U*V')>0
        D=norm(Cv1-(U*V')*Cv2,'fro');
    else
        D=norm(Cv1-(U*[1 0; 0 -1]*V')*Cv2,'fro');
    end
end
```

Published with MATLAB® R2023a