Chang'an University

Exercise Report

"*K*-Means Clustering and Principal Component Analysis"

May 04, 2022

Submitted By:
Manik Debnath |2021124913
Computer Science and Technology
School of Information Engineering
Chang'an University
Statistical Machine Learning

Submitted To:
Prof. Qingsong Song
School of Information Engineering
Chang'an University

Machine Learning: Programming Exercise 7

K-Means Clustering and Principal Component Analysis

Introduction:

Machine learning algorithms can be broadly classified into two categories - supervised and unsupervised learning. There are other categories also like semi-supervised learning and reinforcement learning. But, most of the algorithms are classified as supervised or unsupervised learning. The difference between them happens because of presence of target variable. In unsupervised learning, there is no target variable. The dataset only has input variables which describe the data. This is called unsupervised learning.

Clustering is the method of dividing a set of abstract objects into groups. Points to Keep in Mind **a set of data objects can be viewed as a single entity**. When performing cluster analysis, we divide the data set into groups based on data similarity, then assign labels to the groups. The various types of clustering are Hierarchical clustering and Partitioning clustering. Partitioning clustering is further subdivided into K-Means clustering and Fuzzy C-Means clustering.

K-Means clustering is the most popular unsupervised learning algorithm. It is used when we have unlabeled data which is data without defined categories or groups. The algorithm follows an easy or simple way to classify a given data set through a certain number of clusters, fixed apriori. K-Means algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

K-Means clustering is used to find intrinsic groups within the unlabeled dataset and draw inferences from them. It is based on centroid-based clustering.

Principal Component Analysis (PCA) is a technique that is widely used for applications such as dimensionality reduction, visualization and lossy data compression. In this lab we will focus on the visualization aspect. PCA can be defined as the orthogonal linear transformation of the data to a lower dimensional linear space, known as the principal subspace, such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. Intuitively, PCA finds a meaningful coordinate basis to express our data.

In this exercise, you will implement the K-means clustering algorithm and apply it to compress an image. In the second part, you will use principal component analysis to find a low-dimensional representation of face images.

TABLE OF CONTENTS

Part 1: K-means Clustering **1-12**

Part 1.1: Implementing K-means……...…………………………………… 1

Part 1.1.1: Finding closest centroids…………………...…………………... 2

Part 1.1.2: Computing centroid means……………………………………... 4

Part 1.2: K-means on example dataset………………..…………………… 6

Part 1.3: Random initialization……...……………………………………... 8

Part 1.4: Image compression with K-means…………………...…………... 10

Part 1.4.1: K-means on pixels……………….……………………………... 11

Part 2: Principal Component Analysis **13-30**

Part 2.1: Example Dataset………………………………………………….. 13

Part 2.2: Implementing PCA.……………….……………………………... 14

Part 2.3: Dimensionality Reduction with PCA.…………………………… 17

Part 2.3.1: Projecting the data onto the principal components……………... 17

Part 2.3.2: Reconstructing an approximation of the data…….……………... 20

Part 2.3.3: Visualizing the projections.…………………….……………… 21

Part 2.4: Face Image Dataset……………... 22

Part 2.4.1: PCA on Faces…………………………….……………………. 23

Part 2.4.2: Dimensionality Reduction……………………….……………... 25

Part 2.5: PCA for visualization ……….…………………….…………… 28

LIST OF FIGURES

FIGURES	PAGE NO
Figure 1: K-means algorithm………………...………………...…………….....	1
Figure 2: Find closest centroids (ex7 p1)……………………..………………...	2
Figure 3: Implementation of finding the closest centroids…………………….	3
Figure 4: Output of the cluster center of each sample…………………………	3
Figure 5: Computing centroid means (ex7 p2)……………………….……..	4
Figure 6: Implementation of Computing centroid means………………………	5
Figure 7: Output of Computing centroid means…………..…………………..	5
Figure 8: K-means clustering (ex7 p3)……………………..……………….	6
Figure 9: Before iteration………………..……………………………………	7
Figure 10: Image after 10 iterations...	7
Figure 11: K-means clustering on pixel (ex7 p4 s-1)…………………………...	8
Figure 12: K-means clustering on pixel (ex7 p4 s-2)………………………….	9
Figure 13: Implementation of K-means clustering on pixel…...………………	10
Figure 14: Image Compression (ex7 p5)……………………………………	11
Figure 15: Output of K-means clustering on pixels……………..…………….	12
Figure 16: Original and reconstructed image (when using K-means to compress the image)……………………………………………………………...	12
Figure 17: Loading the example dataset (ex7_pca p1)………………………..	13
Figure 18: Sample dataset 1……………………………………….…………...	14
Figure 19: Loading the example dataset (ex7_pca p2)………………………..	15

Figure 20: Implementing PCA ……………………………………………….. 16

Figure 21: The eigenvectors of the computed dataset...………..…………….. 16

Figure 22: Dimension reduction s-1 (ex7_pca p3)…………………………… 17

Figure 23: Dimension reduction s-2 (ex7_pca p3)…………………………… 18

Figure 24: Implementation of project data sample…………………………... 19

Figure 25: Output of project data sample…...………………………………… 19

Figure 26: Implementation recover the data by projecting …………….…….. 20

Figure 27: Output of the data by projecting …………...…………….…….. 20

Figure 28: PCA After normalization and projection data ……………..……… 21

Figure 29: Visualizing face dataset (ex7_pca p4)…………………………….. 22

Figure 30: Face data sets…………...…………………………………………….. 23

Figure 31: PCA on face dataset (ex7_pca p5)………………………….……….. 24

Figure 32: The principal components on the face dataset ……………………. 25

Figure 33: Principal components on the face dataset ………………………… 25

Figure 34: Visualization of Faces after PCA Dimension Reduction ………… 26

Figure 35: Original images of faces and ones reconstructed from only the top
100 principal components……………………………………………………... 27

Figure 36: PCA for visualizing s-1 (ex7_pca p8)…………………………….. 28

Figure 37: PCA for visualizing s-2 (ex7_pca p8)………………………….…. 29

Figure 38: Raw data in dimensions………………..…………………...….. 30

Figure 39: Use PCA Produced 2D visualization……………………………... 30

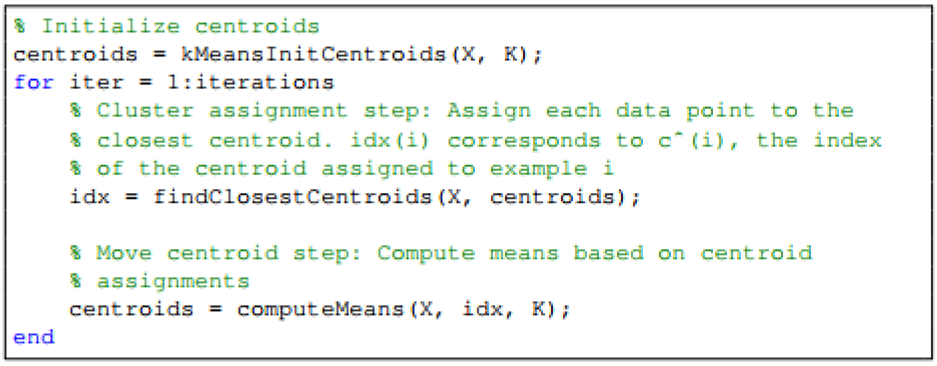
Part 1: K-means Clustering

In this this exercise, we will implement the *K*-means algorithm and use it for image compression. We will first start on an example 2D dataset that will help you gain an intuition of how the *K*-means algorithm works. After that, you will use the *K*-means algorithm for image compression by reducing the number of colors that occur in an image to only those that are most common in that image.

Part 1.1: Implementing K-means

The K-means algorithm is a method to automatically cluster similar data examples together. Concretely, you are given a training set $\{x^{(1)}, \dots x^{(m)}\}$ (where $x^{(i)} \in \mathbb{R}^n$), and want to group the data into a few cohesive 'clusters'. The intuition behind *K*-means is an iterative procedure that starts by guessing the initial centroids, and then refines this guess by repeatedly assigning examples to their closest centroids and then recomputing the centroids based on the assignments.

The *K*-means algorithm is as follows:



```
% Initialize centroids
centroids = kMeansInitCentroids(X, K);
for iter = 1:iterations
    % Cluster assignment step: Assign each data point to the
    % closest centroid. idx(i) corresponds to c^(i), the index
    % of the centroid assigned to example i
    idx = findClosestCentroids(X, centroids);

    % Move centroid step: Compute means based on centroid
    % assignments
    centroids = computeMeans(X, idx, K);
end
```

Figure 1: K-means algorithm

The inner-loop of the algorithm repeatedly carries out two steps:

1. Assigning each training example $x^{(i)}$ to its closest centroid
2. Recomputing the mean of each centroid using the points assigned to it.

The *K*-means algorithm will always converge to some final set of means for the centroids. Note that the converged solution may not always be ideal and depends on the initial setting of the centroids. Therefore, in practice the K-means algorithm is usually run a few times with different random initializations. One way to choose between these different solutions from different random initializations is to choose the one with the lowest cost function value (distortion). You will implement the two phases of the K-means algorithm separately in the next sections.

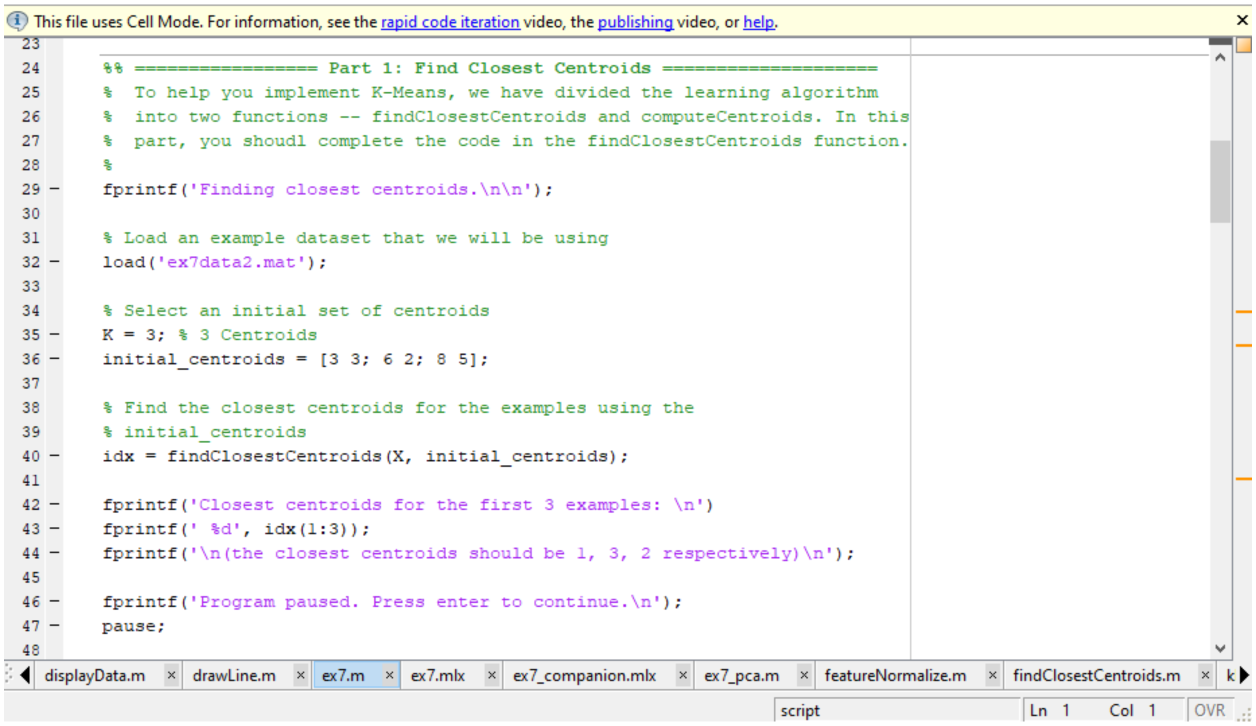
Part 1.1.1: Finding closest centroids

In the 'cluster assignment' phase of the K-means algorithm, the algorithm assigns every training example $x^{(i)}$ to its closest centroid, given the current positions of centroids. Specially, for every example i we set

$$c^{(i)} := j \quad \text{that minimizes} \quad ||x^{(i)} - \mu_j||^2,$$

where $c^{(i)}$ is the index of the centroid that is closest to $x^{(i)}$, and μ_j is the position (value) of the j-th centroid. Note that $c^{(i)}$ corresponds to idx (i) in the starter code.

ex7.m



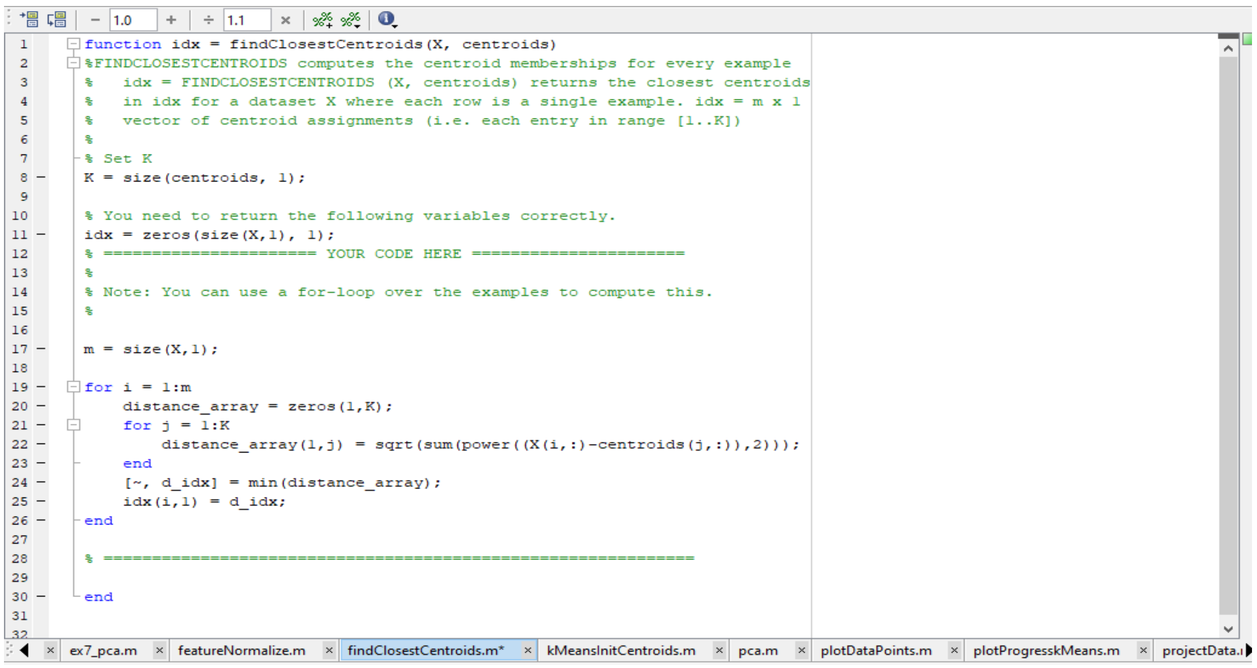
```
%% ================= Part 1: Find Closest Centroids ====================
%  To help you implement K-Means, we have divided the learning algorithm
%  into two functions -- findClosestCentroids and computeCentroids. In this
%  part, you shoudl complete the code in the findClosestCentroids function.
%
fprintf('Finding closest centroids.\n\n');

% Load an example dataset that we will be using
load('ex7data2.mat');

% Select an initial set of centroids
K = 3; % 3 Centroids
initial_centroids = [3 3; 6 2; 8 5];

% Find the closest centroids for the examples using the
% initial_centroids
idx = findClosestCentroids(X, initial_centroids);

fprintf('Closest centroids for the first 3 examples: \n')
fprintf(' %d', idx(1:3));
fprintf('\n(the closest centroids should be 1, 3, 2 respectively)\n');

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 2: Find closest centroids (ex7 p1)

Our task is to complete the code in **findClosestCentroids.m**. This function takes the data matrix X and the locations of all centroids inside centroids and should output a one-dimensional array idx that holds the index (a value in $\{1, \dots, K\}$, where K is total number of centroids) of the closest centroid to every training example. We can implement this using a loop over every training example and every centroid.

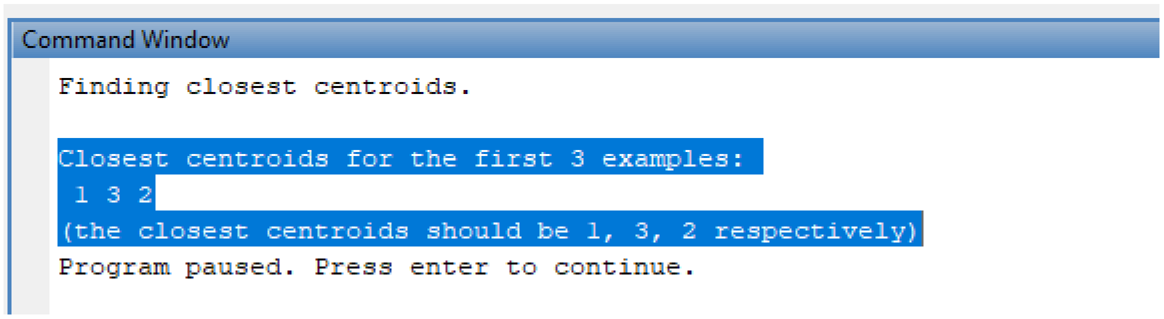
Once we have completed the code in **findClosestCentroids.m**, the code below will run our code and we should see the output [1 3 2] corresponding to the centroid assignments for the first 3 examples.

findClosestCentroids.m



```
function idx = findClosestCentroids(X, centroids)
%FINDCLOSESTCENTROIDS computes the centroid memberships for every example
%   idx = FINDCLOSESTCENTROIDS (X, centroids) returns the closest centroids
%   in idx for a dataset X where each row is a single example. idx = m x 1
%   vector of centroid assignments (i.e. each entry in range [1..K])
%
% Set K
K = size(centroids, 1);

% You need to return the following variables correctly.
idx = zeros(size(X,1), 1);
% ====================== YOUR CODE HERE ======================
%
% Note: You can use a for-loop over the examples to compute this.
%


m = size(X,1);

for i = 1:m
    distance_array = zeros(1,K);
    for j = 1:K
        distance_array(1,j) = sqrt(sum(power((X(i,:)-centroids(j,:)),2)));
    end
    [~, d_idx] = min(distance_array);
    idx(i,1) = d_idx;
end

% =============================================================

end
```

Figure 3: Implementation of finding the closest centroids



```
Finding closest centroids.

Closest centroids for the first 3 examples:
 1 3 2
(the closest centroids should be 1, 3, 2 respectively)
Program paused. Press enter to continue.
```

Figure 4: Output of the cluster center of each sample

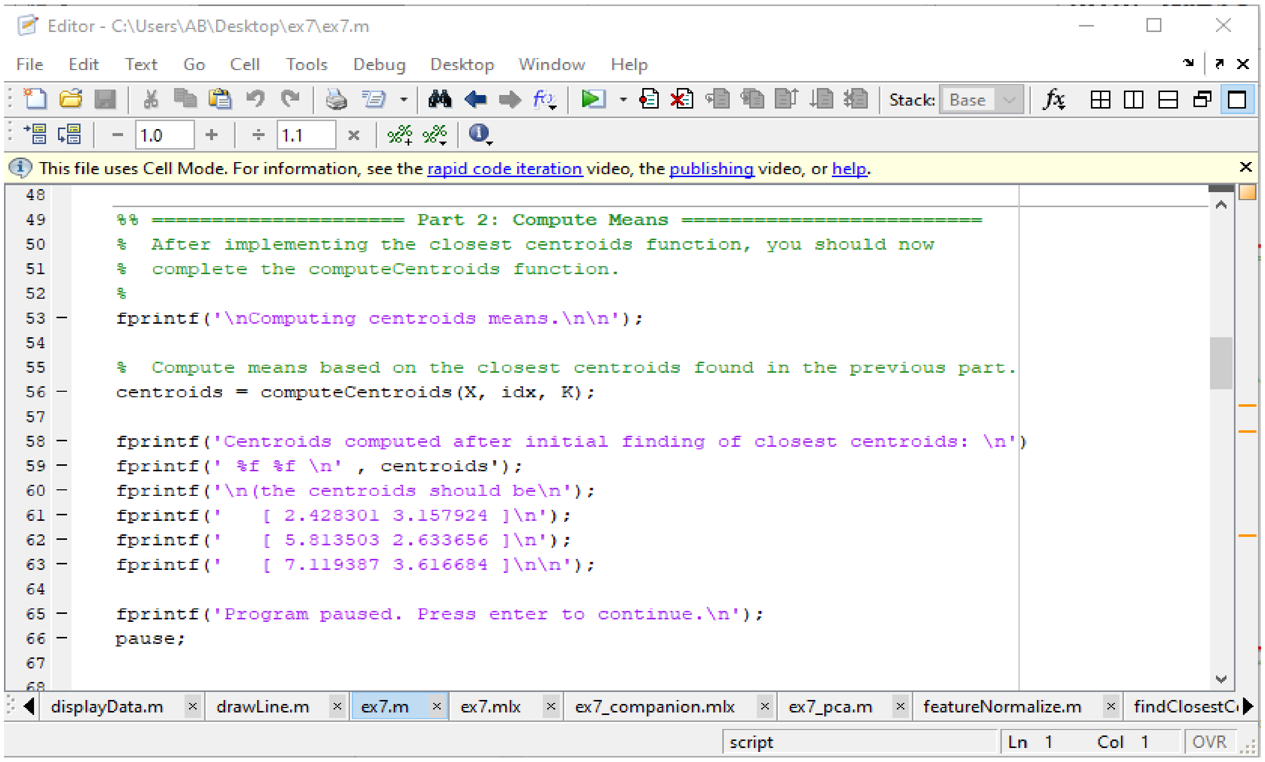
Part 1.1.2: Computing centroid means

Given assignments of every point to a centroid, the second phase of the algorithm recomputes, for each centroid, the mean of the points that were assigned to it. Specially, for every centroid k we set

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

where C_k is the set of examples that are assigned to centroid k. Concretely, if two examples say $x^{(3)}$ and $x^{(5)}$ are assigned to centroid $k = 2$, then we should update $\mu_2 = \frac{1}{2}(x^{(3)} + x^{(5)})$. We should now complete the code in **computeCentroids.m**. We can implement this function using a loop over the centroids. We can also use a loop over the examples; but if we can use a vectorized implementation that does not use such a loop, our code may run faster.

ex7.m



Editor - C:\Users\AB\Desktop\ex7\ex7.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1
This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.
displayData.m drawLine.m ex7.m ex7.mlx ex7_companion.mlx ex7_pca.m featureNormalize.m findClosestC
script Ln 1 Col 1 OVR

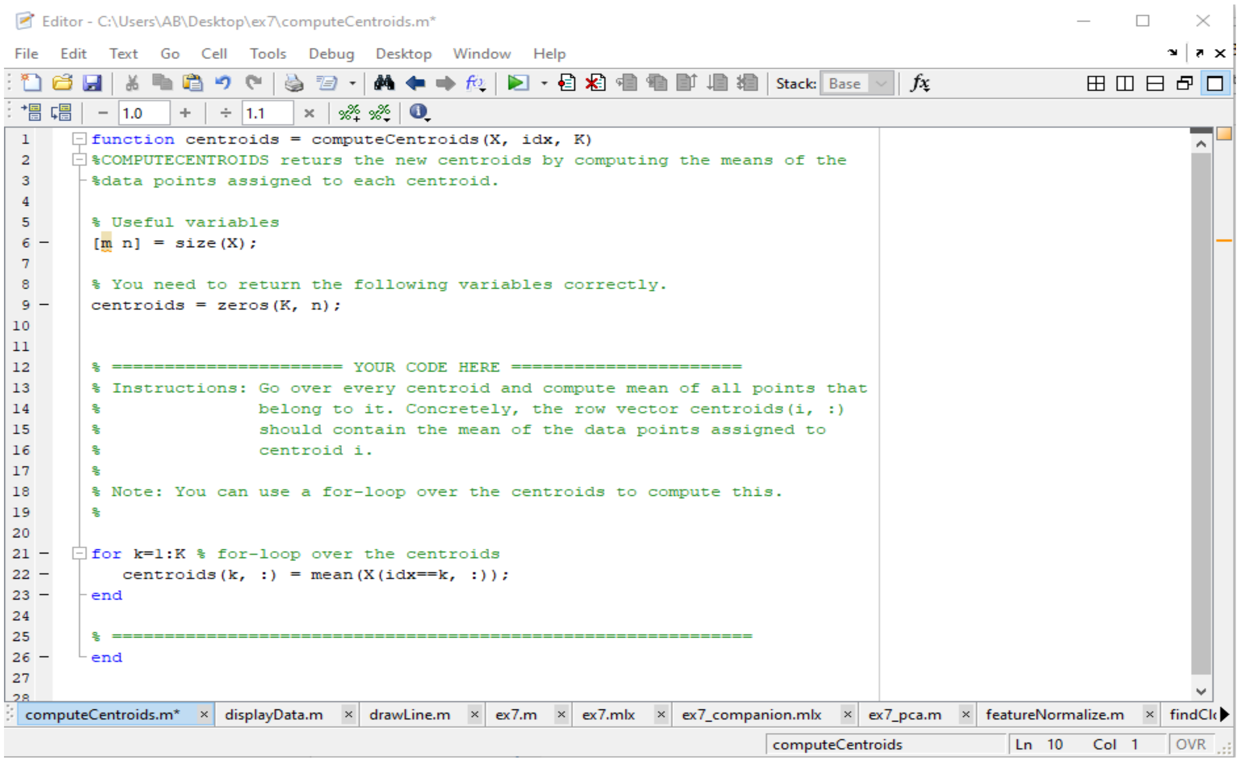
```
%% ===================== Part 2: Compute Means =========================
%  After implementing the closest centroids function, you should now
%  complete the computeCentroids function.
%
fprintf('\nComputing centroids means.\n\n');

%  Compute means based on the closest centroids found in the previous part.
centroids = computeCentroids(X, idx, K);

fprintf('Centroids computed after initial finding of closest centroids: \n')
fprintf(' %f %f \n' , centroids');
fprintf('\n(the centroids should be\n');
fprintf('   [ 2.428301 3.157924 ]\n');
fprintf('   [ 5.813503 2.633656 ]\n');
fprintf('   [ 7.119387 3.616684 ]\n\n');

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 5: Computing centroid means (ex7 p2)

Once we have completed the code in **computeCentroids.m**, the code below will run our code and output the centroids after the first step of *K*-means. The centroids should be:

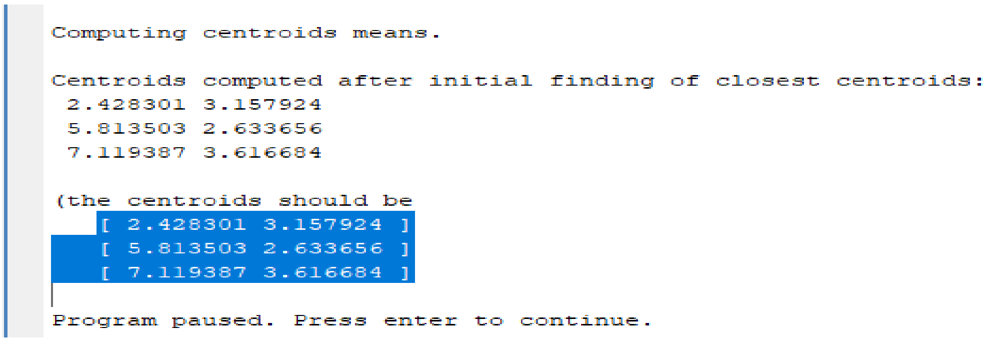
- [2.428301 3.157924]
- [5.813503 2.633656]
- [7.119387 3.616684]

computeCentroids.m


Editor - C:\Users\AB\Desktop\ex7\computeCentroids.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
computeCentroids.m* displayData.m drawLine.m ex7.m ex7.mlx ex7_companion.mlx ex7_pca.m featureNormalize.m findClo
computeCentroids Ln 10 Col 1 OVR

```
function centroids = computeCentroids(X, idx, K)
%COMPUTECENTROIDS returs the new centroids by computing the means of the
%data points assigned to each centroid.

% Useful variables
[m n] = size(X);

% You need to return the following variables correctly.
centroids = zeros(K, n);


% ====================== YOUR CODE HERE ======================
% Instructions: Go over every centroid and compute mean of all points that
%               belong to it. Concretely, the row vector centroids(i, :)
%               should contain the mean of the data points assigned to
%               centroid i.
%
% Note: You can use a for-loop over the centroids to compute this.
%

for k=1:K % for-loop over the centroids
    centroids(k, :) = mean(X(idx==k, :));
end

% =============================================================
end
```

Figure 6: Implementation of computing centroid means



```
Computing centroids means.

Centroids computed after initial finding of closest centroids:
 2.428301 3.157924
 5.813503 2.633656
 7.119387 3.616684

(the centroids should be
   [ 2.428301 3.157924 ]
   [ 5.813503 2.633656 ]
   [ 7.119387 3.616684 ]

Program paused. Press enter to continue.
```

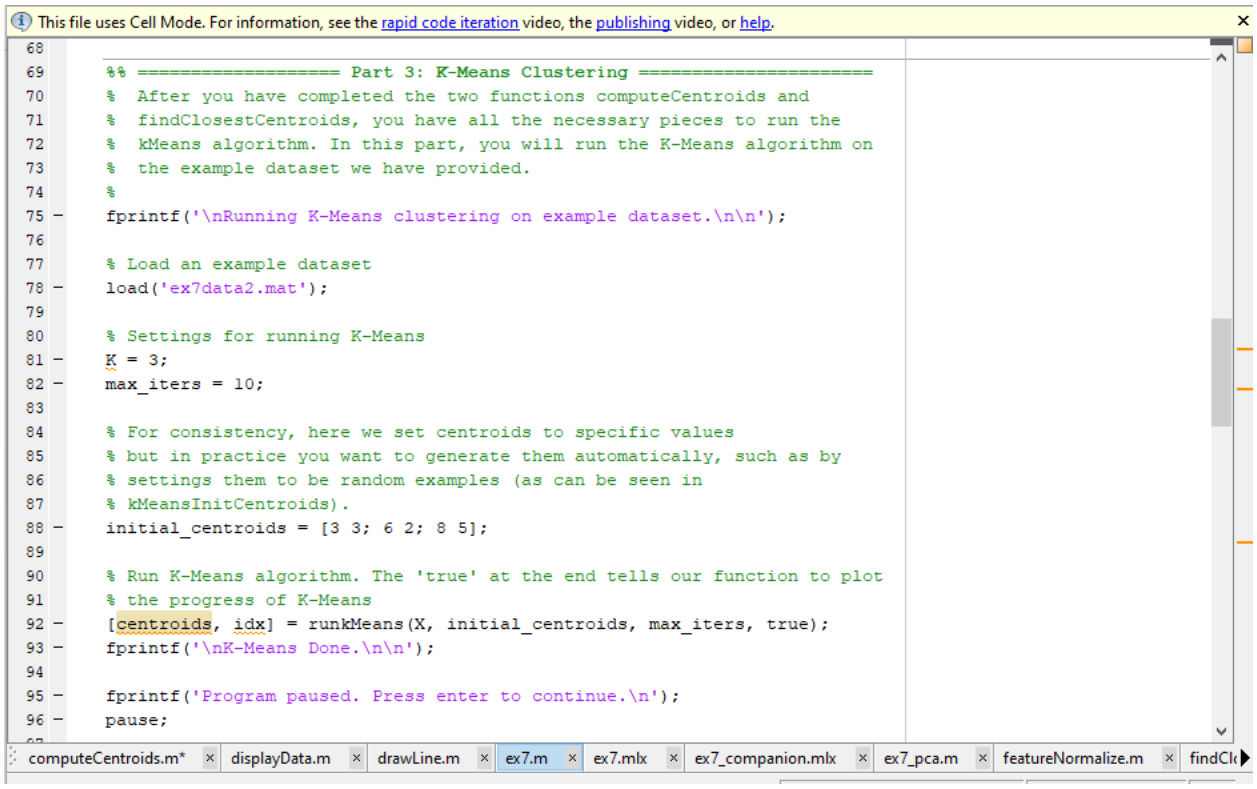
Figure 7: Output of computing centroid means

Part 1.2: K-means on example dataset

After we have completed the two functions (findClosestCentroids and computeCentroids), the code below will run the *K*-means algorithm on a toy 2D dataset to help us understand how *K*-means works. Our functions are called from inside the **runKmeans.m** script. We encourage to take a look at the function to understand how it works. Notice that the code calls the two functions we implemented in a loop.

As before, we have to divide the points into three categories with 10 iterations. The center points of all three classes are initialized.

ex7.m

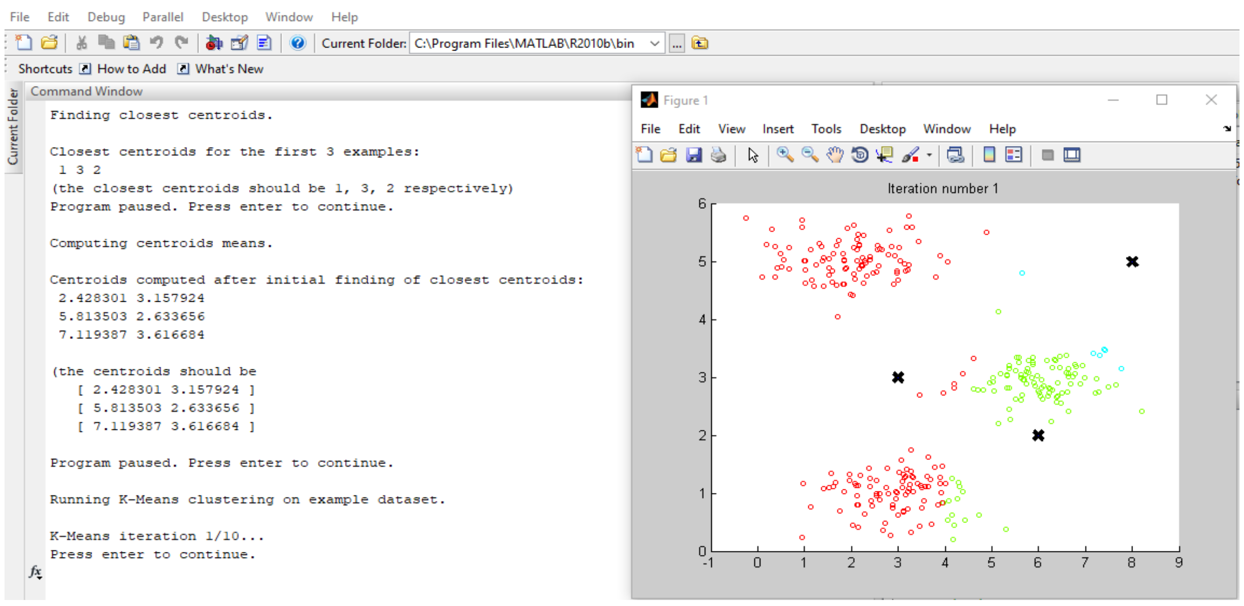


This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% =================== Part 3: K-Means Clustering ======================
%  After you have completed the two functions computeCentroids and
%  findClosestCentroids, you have all the necessary pieces to run the
%  kMeans algorithm. In this part, you will run the K-Means algorithm on
%  the example dataset we have provided.
%
fprintf('\nRunning K-Means clustering on example dataset.\n\n');

% Load an example dataset
load('ex7data2.mat');

% Settings for running K-Means
K = 3;
max_iters = 10;

% For consistency, here we set centroids to specific values
% but in practice you want to generate them automatically, such as by
% settings them to be random examples (as can be seen in
% kMeansInitCentroids).
initial_centroids = [3 3; 6 2; 8 5];

% Run K-Means algorithm. The 'true' at the end tells our function to plot
% the progress of K-Means
[centroids, idx] = runkMeans(X, initial_centroids, max_iters, true);
fprintf('\nK-Means Done.\n\n');

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 8: K-means clustering (ex7 p3)

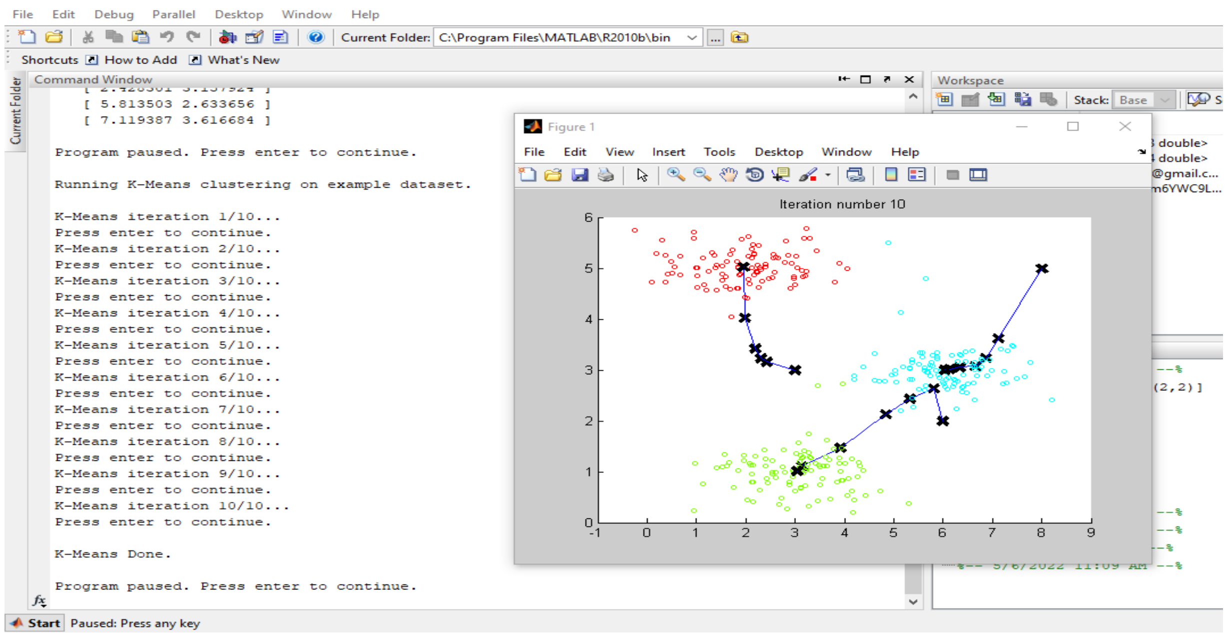
We have to divide the points into three categories, with 10 iterations.


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command Window
Figure 1
Iteration number 1

```
Finding closest centroids.

Closest centroids for the first 3 examples:
 1 3 2
(the closest centroids should be 1, 3, 2 respectively)
Program paused. Press enter to continue.

Computing centroids means.

Centroids computed after initial finding of closest centroids:
 2.428301 3.157924
 5.813503 2.633656
 7.119387 3.616684

(the centroids should be
   [ 2.428301 3.157924 ]
   [ 5.813503 2.633656 ]
   [ 7.119387 3.616684 ]

Program paused. Press enter to continue.

Running K-Means clustering on example dataset.

K-Means iteration 1/10...
Press enter to continue.
```

Figure 9: Before iteration

After 10 iterations the image can see that the three piles of points are well divided into three categories. The picture also shows the movement track of the center point.


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command Window
Workspace
Stack: Base
Figure 1
Iteration number 10
Start Paused: Press any key

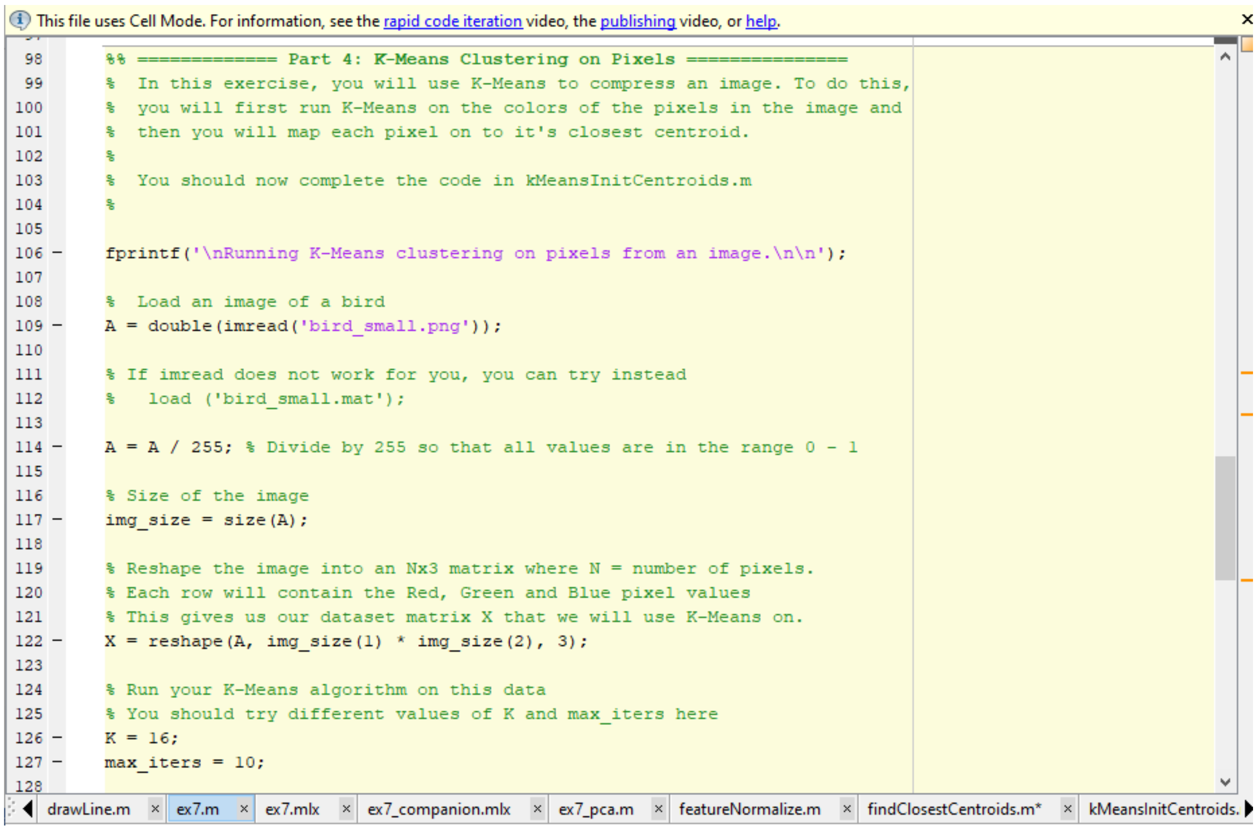
```
   [ 2.428301 3.157924 ]
   [ 5.813503 2.633656 ]
   [ 7.119387 3.616684 ]

Program paused. Press enter to continue.

Running K-Means clustering on example dataset.

K-Means iteration 1/10...
Press enter to continue.
K-Means iteration 2/10...
Press enter to continue.
K-Means iteration 3/10...
Press enter to continue.
K-Means iteration 4/10...
Press enter to continue.
K-Means iteration 5/10...
Press enter to continue.
K-Means iteration 6/10...
Press enter to continue.
K-Means iteration 7/10...
Press enter to continue.
K-Means iteration 8/10...
Press enter to continue.
K-Means iteration 9/10...
Press enter to continue.
K-Means iteration 10/10...
Press enter to continue.

K-Means Done.

Program paused. Press enter to continue.
```

Figure 10: Image after 10 iterations

Part 1.3: Random initialization

This part mainly realizes the selection of initial center points. We need to randomly select K points from the training set, and then use these points as the positions of K center points. A randomly selected code has been given in the test.

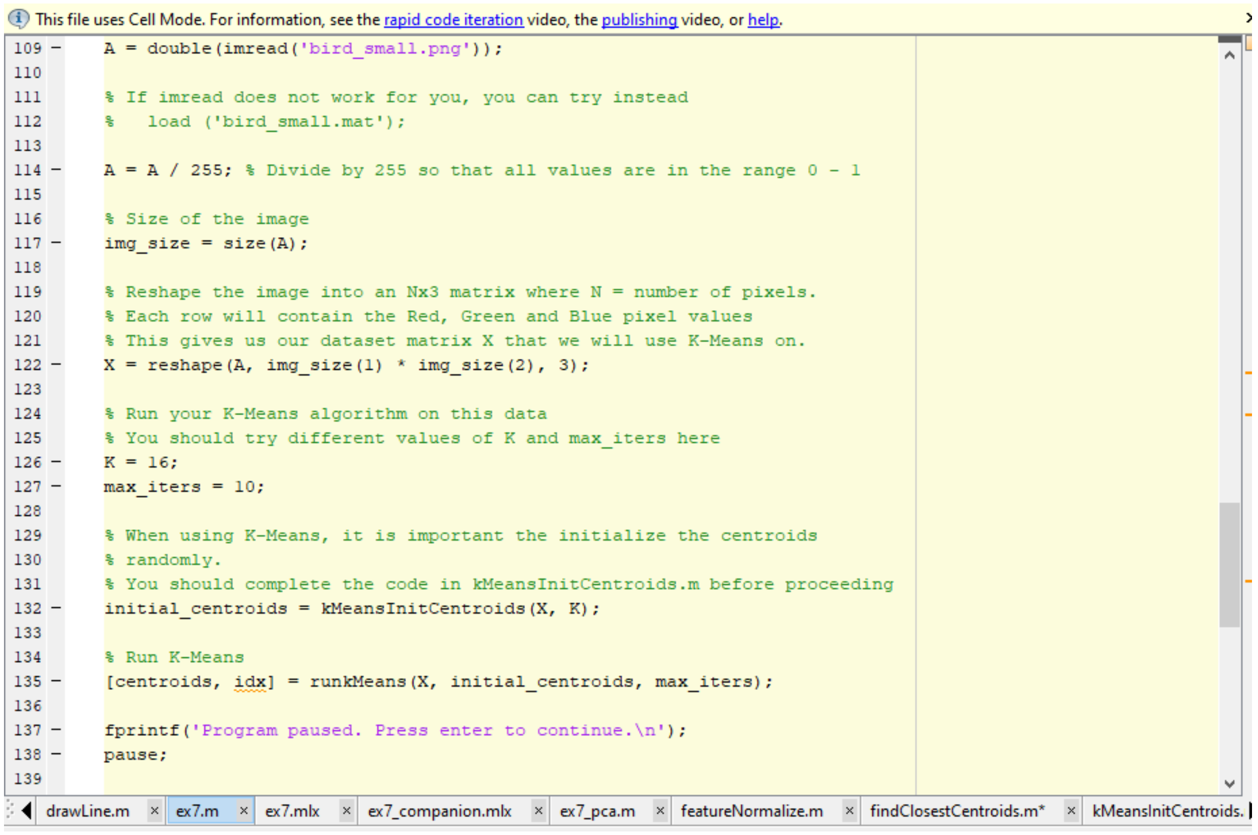
ex7.m



This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% ============= Part 4: K-Means Clustering on Pixels ===============
%  In this exercise, you will use K-Means to compress an image. To do this,
%  you will first run K-Means on the colors of the pixels in the image and
%  then you will map each pixel on to it's closest centroid.
%
%  You should now complete the code in kMeansInitCentroids.m
%

fprintf('\nRunning K-Means clustering on pixels from an image.\n\n');

%  Load an image of a bird
A = double(imread('bird_small.png'));

% If imread does not work for you, you can try instead
%   load ('bird_small.mat');

A = A / 255; % Divide by 255 so that all values are in the range 0 - 1

% Size of the image
img_size = size(A);

% Reshape the image into an Nx3 matrix where N = number of pixels.
% Each row will contain the Red, Green and Blue pixel values
% This gives us our dataset matrix X that we will use K-Means on.
X = reshape(A, img_size(1) * img_size(2), 3);

% Run your K-Means algorithm on this data
% You should try different values of K and max_iters here
K = 16;
max_iters = 10;
```

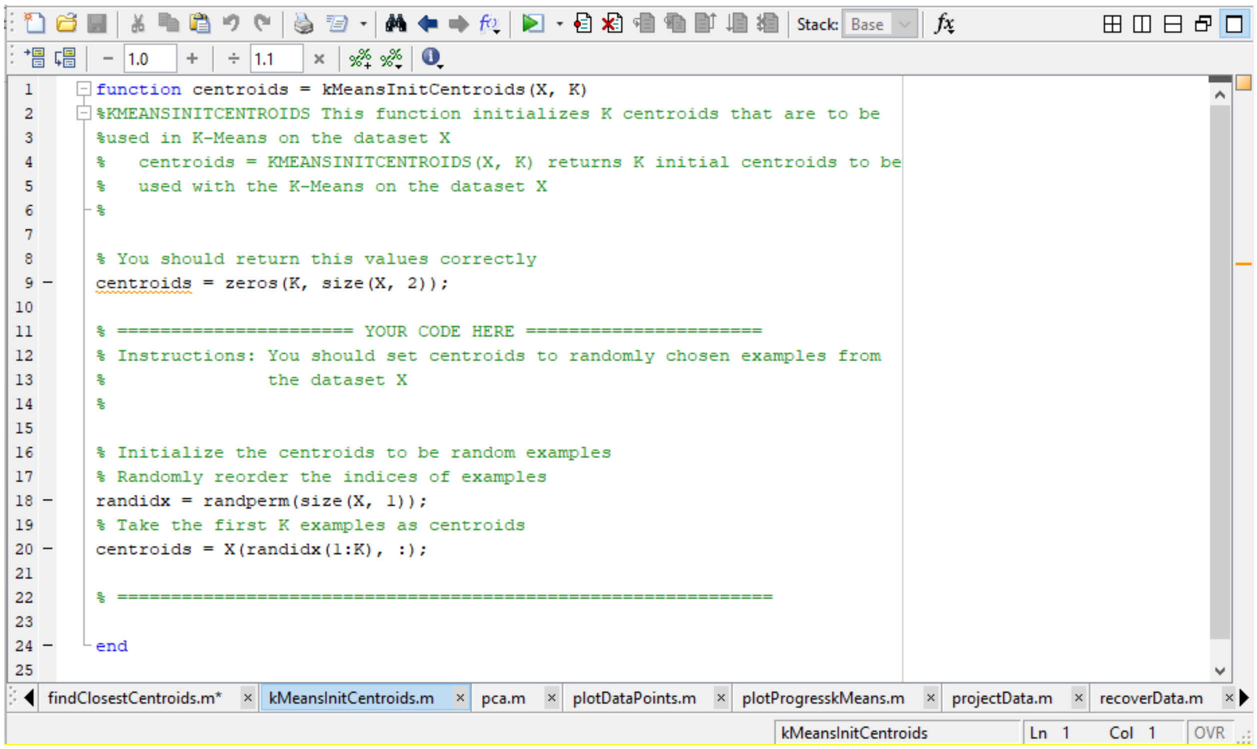
Figure 11: K-means clustering on pixel (ex7 p4 s-1)


This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
A = double(imread('bird_small.png'));

% If imread does not work for you, you can try instead
%   load ('bird_small.mat');

A = A / 255; % Divide by 255 so that all values are in the range 0 - 1

% Size of the image
img_size = size(A);

% Reshape the image into an Nx3 matrix where N = number of pixels.
% Each row will contain the Red, Green and Blue pixel values
% This gives us our dataset matrix X that we will use K-Means on.
X = reshape(A, img_size(1) * img_size(2), 3);

% Run your K-Means algorithm on this data
% You should try different values of K and max_iters here
K = 16;
max_iters = 10;

% When using K-Means, it is important the initialize the centroids
% randomly.
% You should complete the code in kMeansInitCentroids.m before proceeding
initial_centroids = kMeansInitCentroids(X, K);

% Run K-Means
[centroids, idx] = runkMeans(X, initial_centroids, max_iters);

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 12: K-means clustering on pixel (ex7 p4 s-2)

KMeansInitCentroids.m



```
function centroids = kMeansInitCentroids(X, K)
%KMEANSINITCENTROIDS This function initializes K centroids that are to be
%used in K-Means on the dataset X
%   centroids = KMEANSINITCENTROIDS(X, K) returns K initial centroids to be
%   used with the K-Means on the dataset X
%

% You should return this values correctly
centroids = zeros(K, size(X, 2));

% ====================== YOUR CODE HERE ======================
% Instructions: You should set centroids to randomly chosen examples from
%               the dataset X
%

% Initialize the centroids to be random examples
% Randomly reorder the indices of examples
randidx = randperm(size(X, 1));
% Take the first K examples as centroids
centroids = X(randidx(1:K), :);

% =============================================================

end

```

Figure 13: Implementation of K-means clustering on pixel

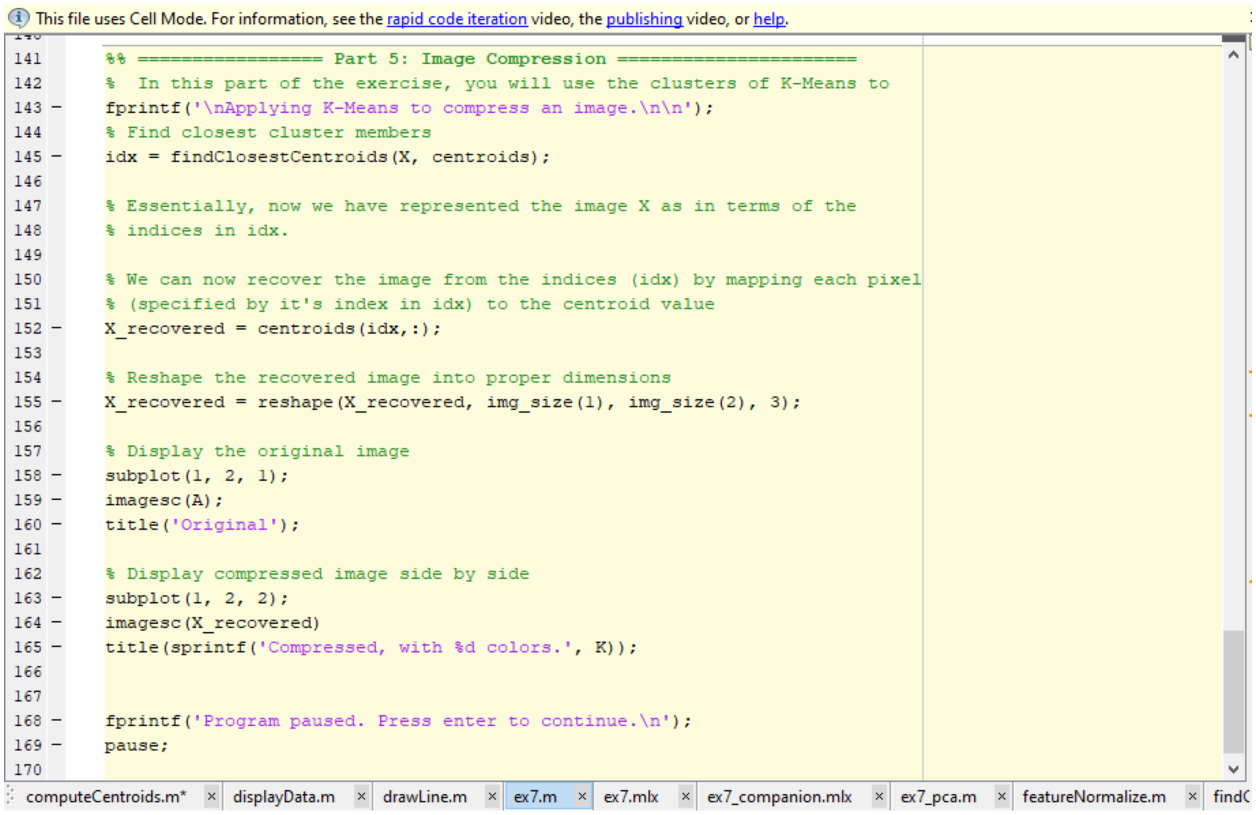
Part 1.4: Image compression with K-means

In this section, K-means clustering is used for image compression. In the direct 24-bit color representation of the image, each pixel is represented as three 8-bit unsigned integers (ranging from 0 to 255), which are the intensity values of red, green, and blue. This kind of coding is also called RGB coding.

Purpose: Compress thousands of colors of the original image into 16 colors.

The RGB values of all pixels in the original image are used as data samples, and K-means clustering is used to cluster the points in the three-dimensional RGB space into 16 types.

ex7.m



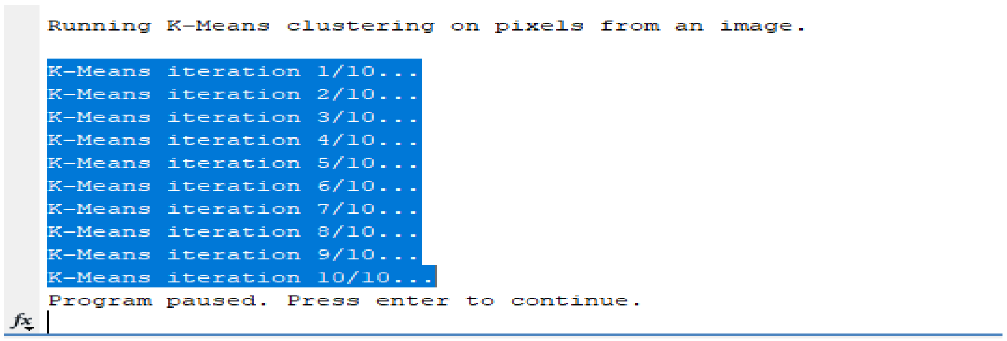
This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% ================== Part 5: Image Compression ======================
%  In this part of the exercise, you will use the clusters of K-Means to
fprintf('\nApplying K-Means to compress an image.\n\n');
% Find closest cluster members
idx = findClosestCentroids(X, centroids);

% Essentially, now we have represented the image X as in terms of the
% indices in idx.

% We can now recover the image from the indices (idx) by mapping each pixel
% (specified by it's index in idx) to the centroid value
X_recovered = centroids(idx,:);

% Reshape the recovered image into proper dimensions
X_recovered = reshape(X_recovered, img_size(1), img_size(2), 3);

% Display the original image
subplot(1, 2, 1);
imagesc(A);
title('Original');

% Display compressed image side by side
subplot(1, 2, 2);
imagesc(X_recovered)
title(sprintf('Compressed, with %d colors.', K));


fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 14: Image Compression (ex7 p5)

Part 1.4.1: K-means on pixels

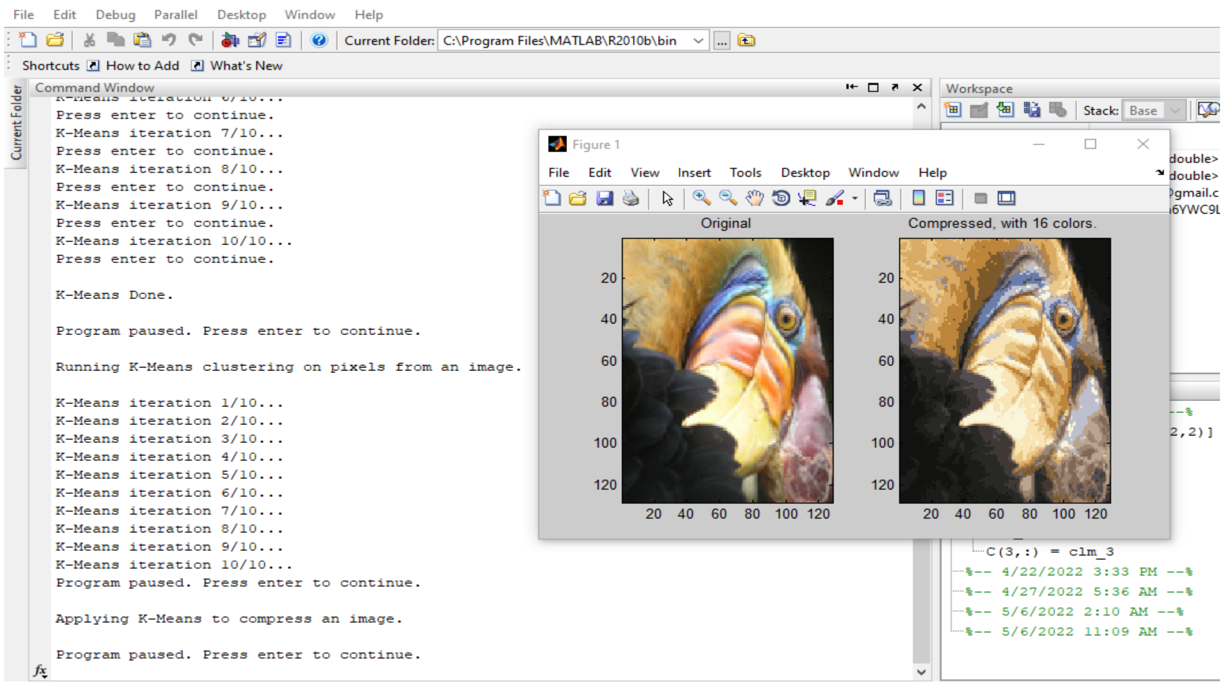
After completing the previous tasks, we can directly achieve this goal. As shown in the figure below, the left is the original image, and the right is the compressed image (16 colors).



```
Running K-Means clustering on pixels from an image.

K-Means iteration 1/10...
K-Means iteration 2/10...
K-Means iteration 3/10...
K-Means iteration 4/10...
K-Means iteration 5/10...
K-Means iteration 6/10...
K-Means iteration 7/10...
K-Means iteration 8/10...
K-Means iteration 9/10...
K-Means iteration 10/10...
Program paused. Press enter to continue.
```

Figure 15: Output of K-means clustering on pixels

Use K-means for image compression. Taking a picture of 128\times 128 as an example, using RGB, a total of 128\times 128/times 24 = 393216 bits are required. Here we compress it, divide all colors into 16 categories, and replace the colors in the entire category with the colors corresponding to the centroid. The space can be compressed to 16\times 24 + 128\times 128/times 4 = 65920 bits. Using the example provided in the title, the effect is probably as follows:



File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command Window
Workspace
Stack: Base
Figure 1
File Edit View Insert Tools Desktop Window Help
Original
Compressed, with 16 colors.
20 40 60 80 100 120
20 40 60 80 100 120

```
Press enter to continue.
K-Means iteration 7/10...
Press enter to continue.
K-Means iteration 8/10...
Press enter to continue.
K-Means iteration 9/10...
Press enter to continue.
K-Means iteration 10/10...
Press enter to continue.

K-Means Done.

Program paused. Press enter to continue.

Running K-Means clustering on pixels from an image.

K-Means iteration 1/10...
K-Means iteration 2/10...
K-Means iteration 3/10...
K-Means iteration 4/10...
K-Means iteration 5/10...
K-Means iteration 6/10...
K-Means iteration 7/10...
K-Means iteration 8/10...
K-Means iteration 9/10...
K-Means iteration 10/10...
Program paused. Press enter to continue.

Applying K-Means to compress an image.

Program paused. Press enter to continue.
```

```
C(3,:) = clm_3
%-- 4/22/2022 3:33 PM --%
%-- 4/27/2022 5:36 AM --%
%-- 5/6/2022 2:10 AM --%
%-- 5/6/2022 11:09 AM --%
```

Figure 16: Original and reconstructed image (when using K-means to compress the image)

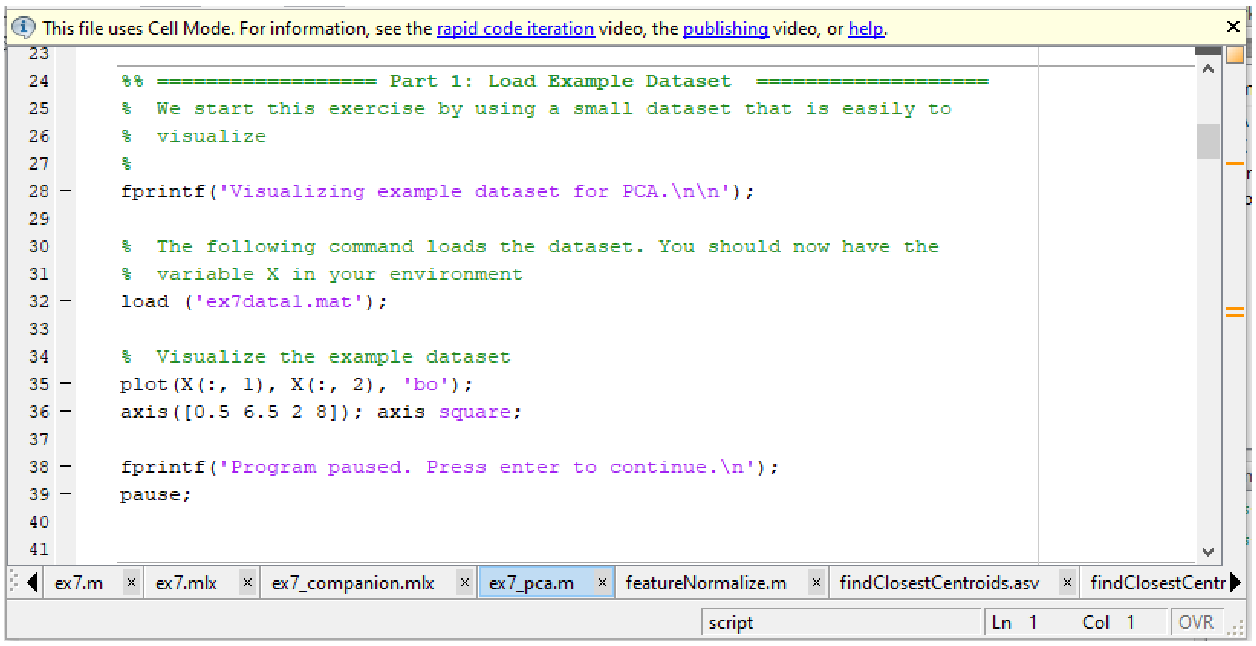
Part 2: Principal Component Analysis

In this exercise, we will use principal component analysis (PCA) to perform dimensionality reduction. We will first experiment with an example 2D dataset to get intuition on how PCA works, and then use it on a bigger dataset of 5000 face image dataset. The code provided, will help us step through the first half of the exercise.

Part 2.1: Example Dataset

First reduce the two-dimensional vector in the example to one dimension.

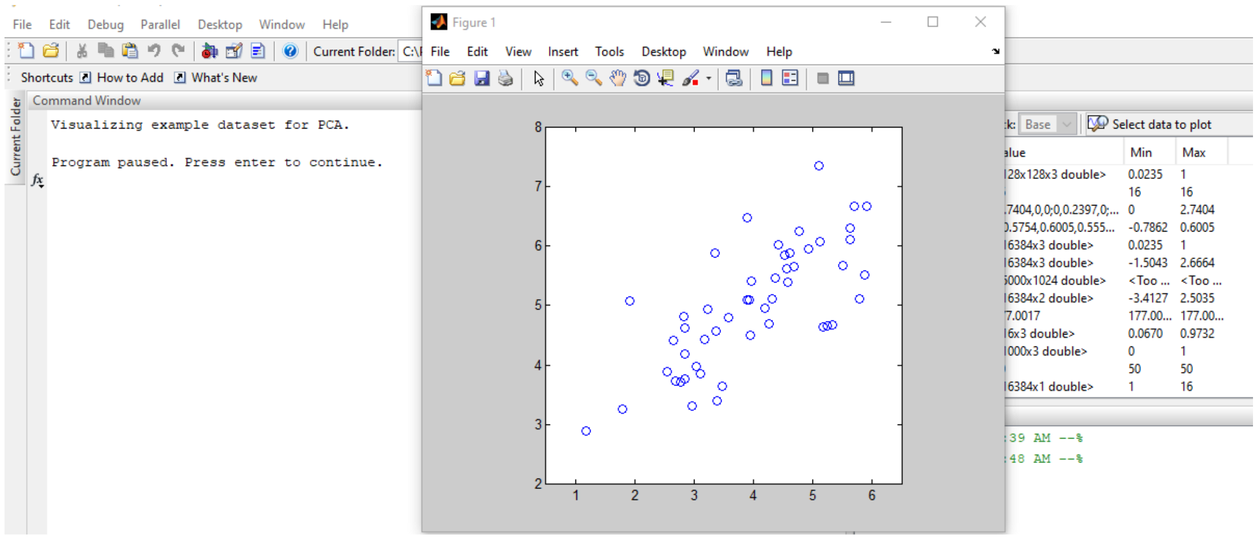
ex7_pca.m



This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% ================== Part 1: Load Example Dataset  ===================
%  We start this exercise by using a small dataset that is easily to
%  visualize
%
fprintf('Visualizing example dataset for PCA.\n\n');

%  The following command loads the dataset. You should now have the
%  variable X in your environment
load ('ex7data1.mat');

%  Visualize the example dataset
plot(X(:, 1), X(:, 2), 'bo');
axis([0.5 6.5 2 8]); axis square;

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 17: Loading the example dataset (ex7_pca p1)



```
Visualizing example dataset for PCA.

Program paused. Press enter to continue.
```

alue	Min	Max
128x128x3 double>	0.0235	1
	16	16
.7404,0,0;0,0.2397,0;...	0	2.7404
0.5754,0.6005,0.555...	-0.7862	0.6005
16384x3 double>	0.0235	1
16384x3 double>	-1.5043	2.6664
5000x1024 double>	<Too ...	<Too ...
16384x2 double>	-3.4127	2.5035
7.0017	177.00...	177.00...
16x3 double>	0.0670	0.9732
1000x3 double>	0	1
	50	50
16384x1 double>	1	16

```
:39 AM --%
:48 AM --%
```

Figure 18: Sample dataset 1

Part 2.2: Implementing PCA

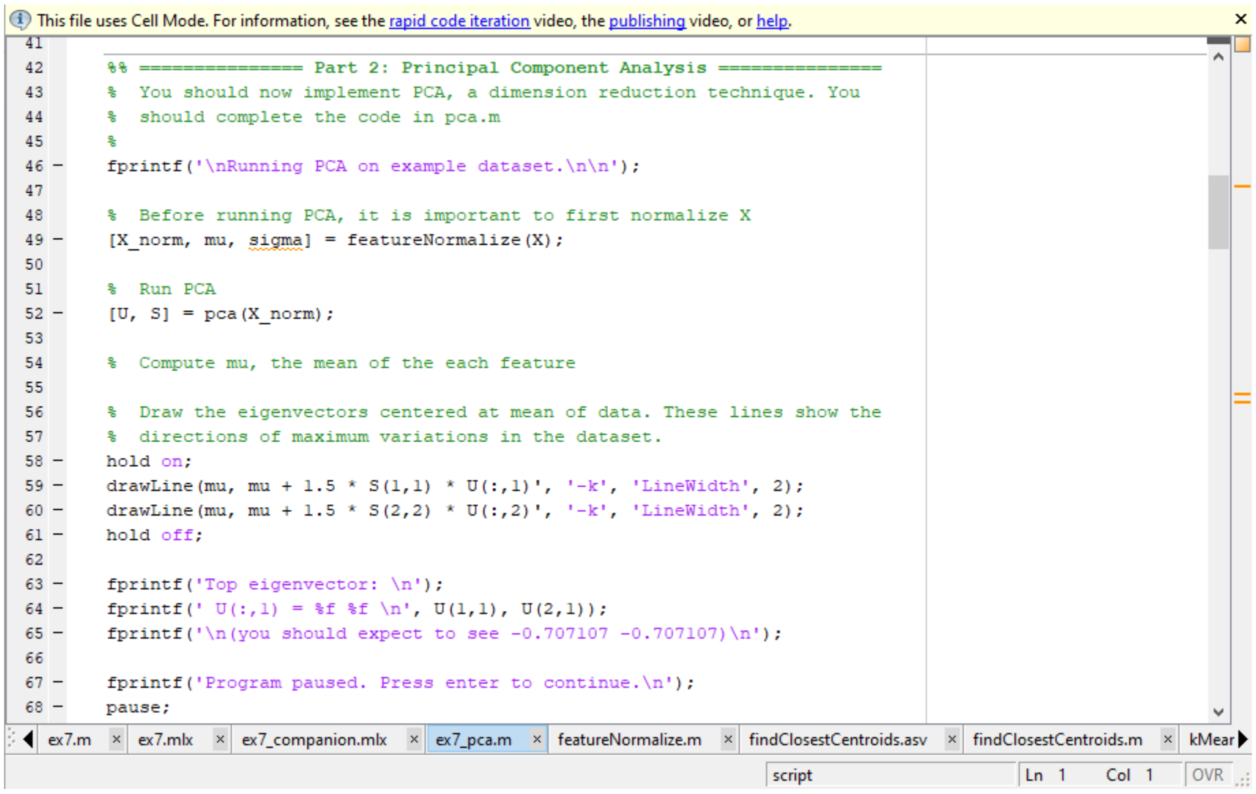
In this part of the exercise, we will implement PCA. PCA consists of two computational steps: First, we compute the covariance matrix of the data. Then, we use MATLAB's svd function to compute the eigenvectors $U_1, U_2, \ldots, U_n$. These will correspond to the principal components of variation in the data. Before using PCA, it is important to first normalize the data by subtracting the mean value of each feature from the dataset, and scaling each dimension so that they are in the same range. In the code below, this normalization has been performed for us using the **featureNormalize function**.

After normalizing the data, we can run PCA to compute the principal components. Our task is to complete the code in **pca.m** to compute the principal components of the dataset. First, we should compute the covariance matrix of the data, which is given by:

$$\Sigma = \frac{1}{m} X^T X$$

where X is the data matrix with examples in rows, and m is the number of examples. Note that Σ is a $n \times n$ matrix and not the summation operator.

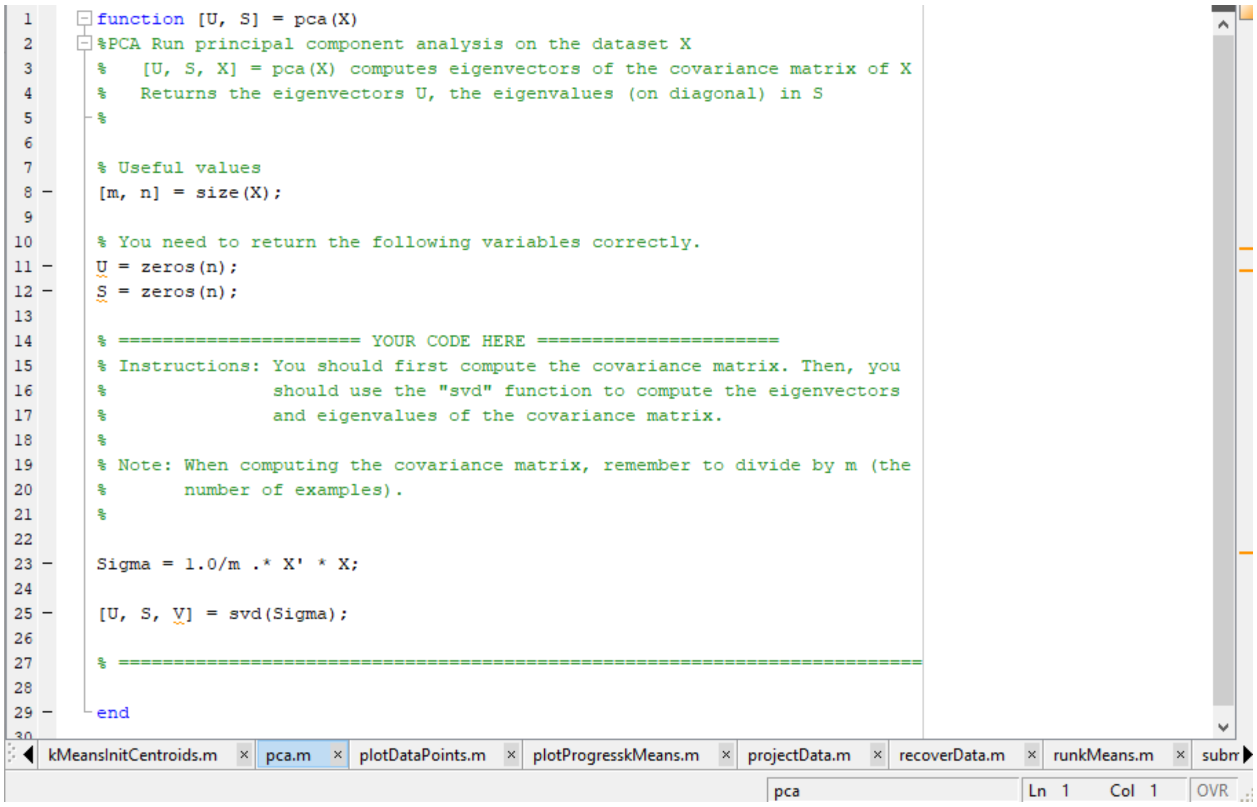
ex7_pca.m


This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% =============== Part 2: Principal Component Analysis ===============
%  You should now implement PCA, a dimension reduction technique. You
%  should complete the code in pca.m
%
fprintf('\nRunning PCA on example dataset.\n\n');

%  Before running PCA, it is important to first normalize X
[X_norm, mu, sigma] = featureNormalize(X);

%  Run PCA
[U, S] = pca(X_norm);

%  Compute mu, the mean of the each feature

%  Draw the eigenvectors centered at mean of data. These lines show the
%  directions of maximum variations in the dataset.
hold on;
drawLine(mu, mu + 1.5 * S(1,1) * U(:,1)', '-k', 'LineWidth', 2);
drawLine(mu, mu + 1.5 * S(2,2) * U(:,2)', '-k', 'LineWidth', 2);
hold off;

fprintf('Top eigenvector: \n');
fprintf(' U(:,1) = %f %f \n', U(1,1), U(2,1));
fprintf('\n(you should expect to see -0.707107 -0.707107)\n');

fprintf('Program paused. Press enter to continue.\n');
pause;
```

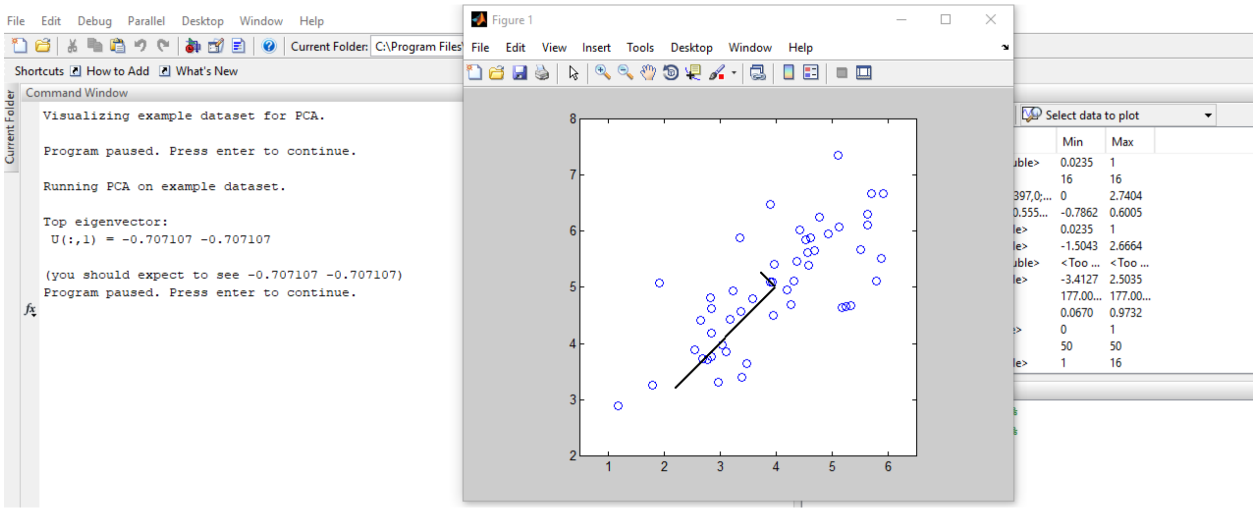
Figure 19: Loading the example dataset (ex7_pca p2)

pca.m



```
function [U, S] = pca(X)
%PCA Run principal component analysis on the dataset X
%   [U, S, X] = pca(X) computes eigenvectors of the covariance matrix of X
%   Returns the eigenvectors U, the eigenvalues (on diagonal) in S
%

% Useful values
[m, n] = size(X);

% You need to return the following variables correctly.
U = zeros(n);
S = zeros(n);

% ====================== YOUR CODE HERE ======================
% Instructions: You should first compute the covariance matrix. Then, you
%               should use the "svd" function to compute the eigenvectors
%               and eigenvalues of the covariance matrix.
%
% Note: When computing the covariance matrix, remember to divide by m (the
%       number of examples).
%

Sigma = 1.0/m .* X' * X;

[U, S, V] = svd(Sigma);

% =========================================================================

end
```

Figure 20: Implementing PCA



```
Visualizing example dataset for PCA.

Program paused. Press enter to continue.

Running PCA on example dataset.

Top eigenvector:
 U(:,1) = -0.707107 -0.707107

(you should expect to see -0.707107 -0.707107)
Program paused. Press enter to continue.
```

	Min	Max
ıble>	0.0235	1
	16	16
397,0;...	0	2.7404
0.555...	-0.7862	0.6005
le>	0.0235	1
le>	-1.5043	2.6664
ıble>	<Too ...	<Too ...
le>	-3.4127	2.5035
	177.00...	177.00...
	0.0670	0.9732
:>	0	1
	50	50
le>	1	16

File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files
Shortcuts How to Add What's New
Command Window
Current Folder
Figure 1
File Edit View Insert Tools Desktop Window Help
Select data to plot
8
7
6
5
4
3
2
1
2
3
4
5
6

Figure 21: The eigenvectors of the computed dataset

Part 2.3: Dimensionality Reduction with PCA

After obtaining the principal components, we use them to reduce the dimensionality of the data and project the data to a lower-dimensional space.

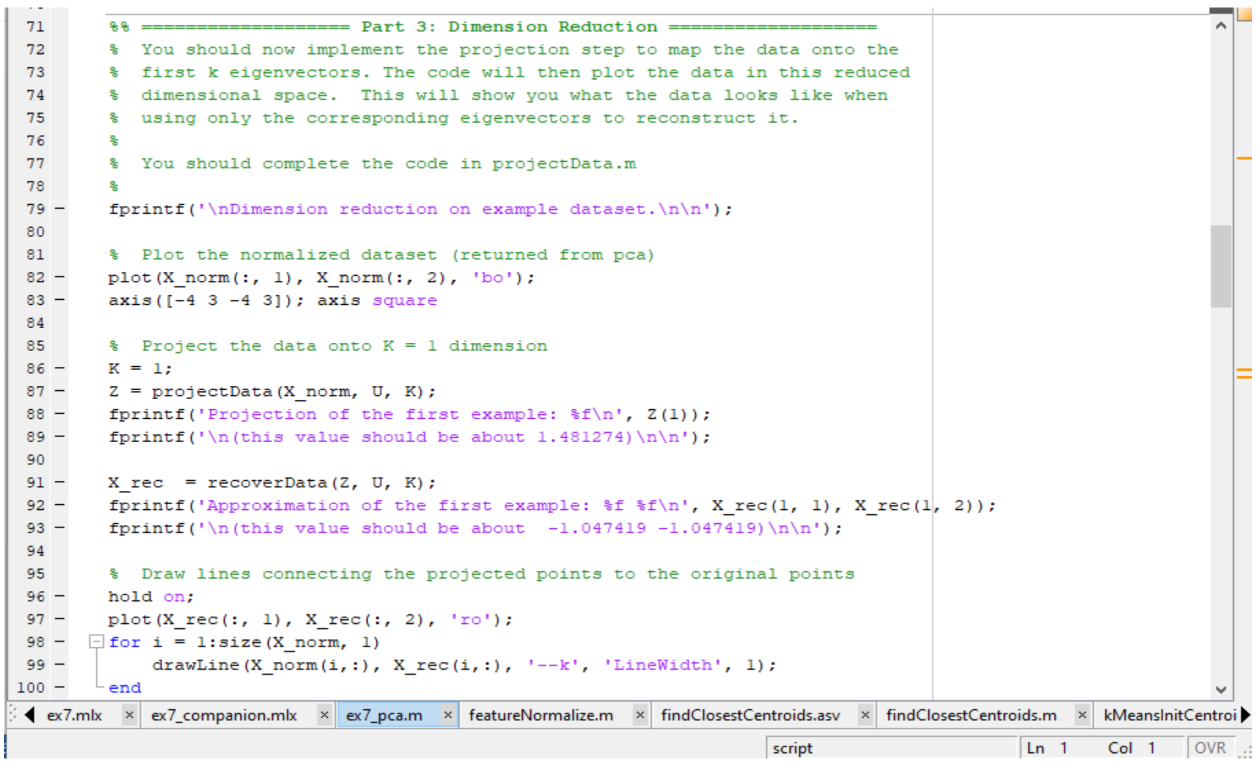
In practical applications, if we are using linear regression or neural network algorithms, we can use dimensionality reduction data to replace the original data. This can make the model train faster because the input value has a lower dimensionality.

Part 2.3.1: Projecting the data onto the principal components

We should now complete the code in **projectData.m**. Specially, we are given a dataset X, the principal components U, and the desired number of dimensions to reduce to K. We should project each example in X onto the top K components in U. Note that the top K components in U are given by the first K columns of U, that is U_reduce = U(:, 1:K).

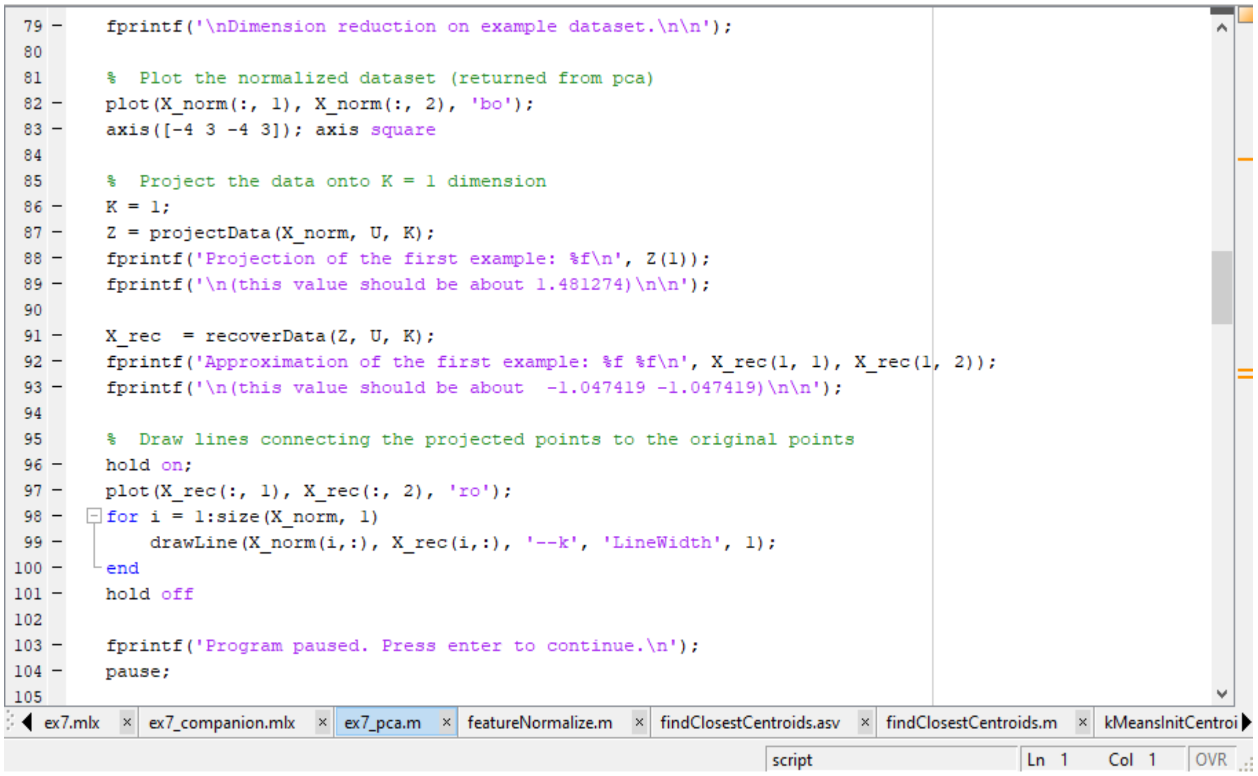
Once we have completed the code in **projectData.m**, run the code below to project the first example onto the first dimension and we should see a value of about 1.481 (or possibly -1.481, if you got $-U_1$ instead of U_1).

ex7_pca.m



```
%% =================== Part 3: Dimension Reduction ===================
%  You should now implement the projection step to map the data onto the
%  first k eigenvectors. The code will then plot the data in this reduced
%  dimensional space.  This will show you what the data looks like when
%  using only the corresponding eigenvectors to reconstruct it.
%
%  You should complete the code in projectData.m
%
fprintf('\nDimension reduction on example dataset.\n\n');

%  Plot the normalized dataset (returned from pca)
plot(X_norm(:, 1), X_norm(:, 2), 'bo');
axis([-4 3 -4 3]); axis square

%  Project the data onto K = 1 dimension
K = 1;
Z = projectData(X_norm, U, K);
fprintf('Projection of the first example: %f\n', Z(1));
fprintf('\n(this value should be about 1.481274)\n\n');

X_rec  = recoverData(Z, U, K);
fprintf('Approximation of the first example: %f %f\n', X_rec(1, 1), X_rec(1, 2));
fprintf('\n(this value should be about  -1.047419 -1.047419)\n\n');

%  Draw lines connecting the projected points to the original points
hold on;
plot(X_rec(:, 1), X_rec(:, 2), 'ro');
for i = 1:size(X_norm, 1)
    drawLine(X_norm(i,:), X_rec(i,:), '--k', 'LineWidth', 1);
end
```

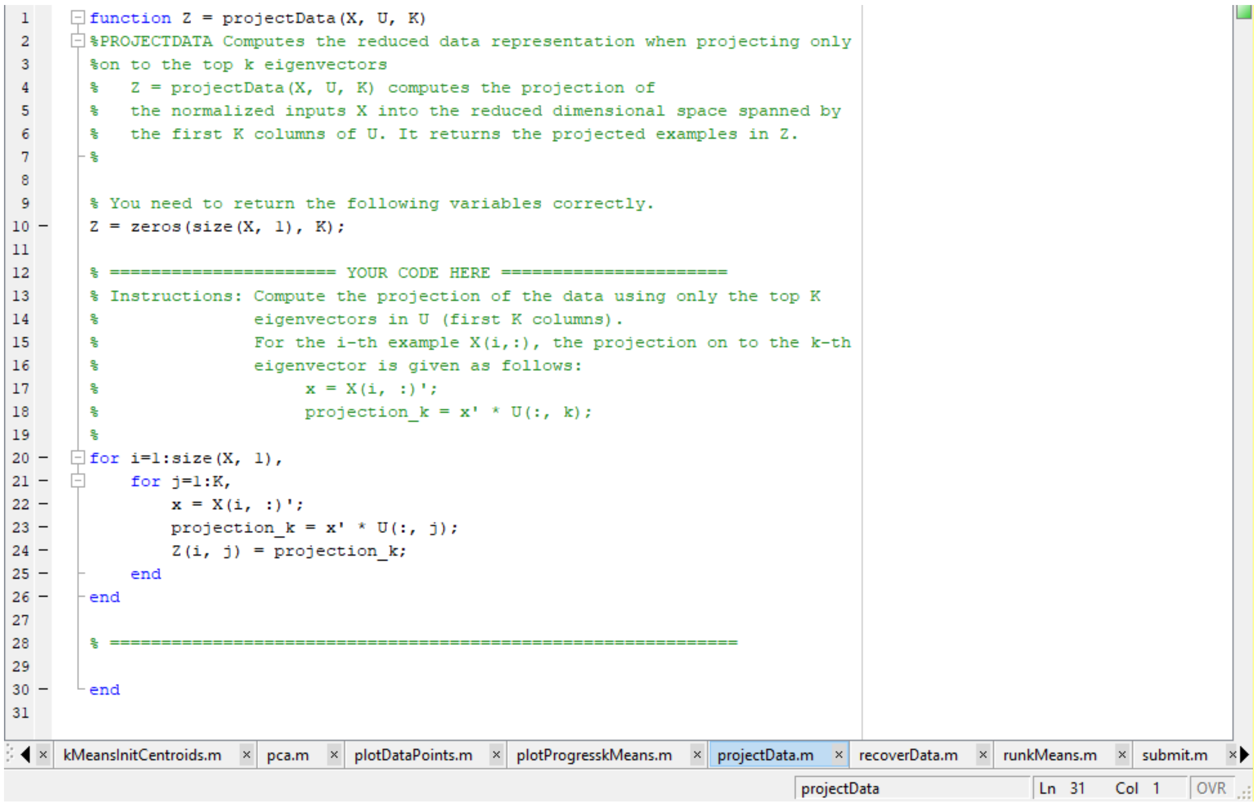
Figure 22: Dimension reduction s-1 (ex7_pca p3)



```matlab
fprintf('\nDimension reduction on example dataset.\n\n');

%  Plot the normalized dataset (returned from pca)
plot(X_norm(:, 1), X_norm(:, 2), 'bo');
axis([-4 3 -4 3]); axis square

%  Project the data onto K = 1 dimension
K = 1;
Z = projectData(X_norm, U, K);
fprintf('Projection of the first example: %f\n', Z(1));
fprintf('\n(this value should be about 1.481274)\n\n');

X_rec  = recoverData(Z, U, K);
fprintf('Approximation of the first example: %f %f\n', X_rec(1, 1), X_rec(1, 2));
fprintf('\n(this value should be about  -1.047419 -1.047419)\n\n');

%  Draw lines connecting the projected points to the original points
hold on;
plot(X_rec(:, 1), X_rec(:, 2), 'ro');
for i = 1:size(X_norm, 1)
    drawLine(X_norm(i,:), X_rec(i,:), '--k', 'LineWidth', 1);
end
hold off

fprintf('Program paused. Press enter to continue.\n');
pause;
```

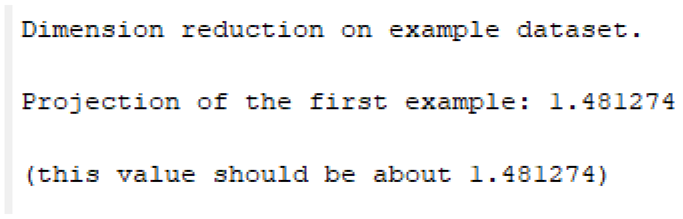
Figure 23: Dimension reduction s-2 (ex7_pca p3)

projectData.m



```
function Z = projectData(X, U, K)
%PROJECTDATA Computes the reduced data representation when projecting only
%on to the top k eigenvectors
%   Z = projectData(X, U, K) computes the projection of
%   the normalized inputs X into the reduced dimensional space spanned by
%   the first K columns of U. It returns the projected examples in Z.
%

% You need to return the following variables correctly.
Z = zeros(size(X, 1), K);

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the projection of the data using only the top K
%               eigenvectors in U (first K columns).
%               For the i-th example X(i,:), the projection on to the k-th
%               eigenvector is given as follows:
%                    x = X(i, :)';
%                    projection_k = x' * U(:, k);
%
for i=1:size(X, 1),
    for j=1:K,
        x = X(i, :)';
        projection_k = x' * U(:, j);
        Z(i, j) = projection_k;
    end
end

% =============================================================

end
```

Figure 24: Implementation of project data sample



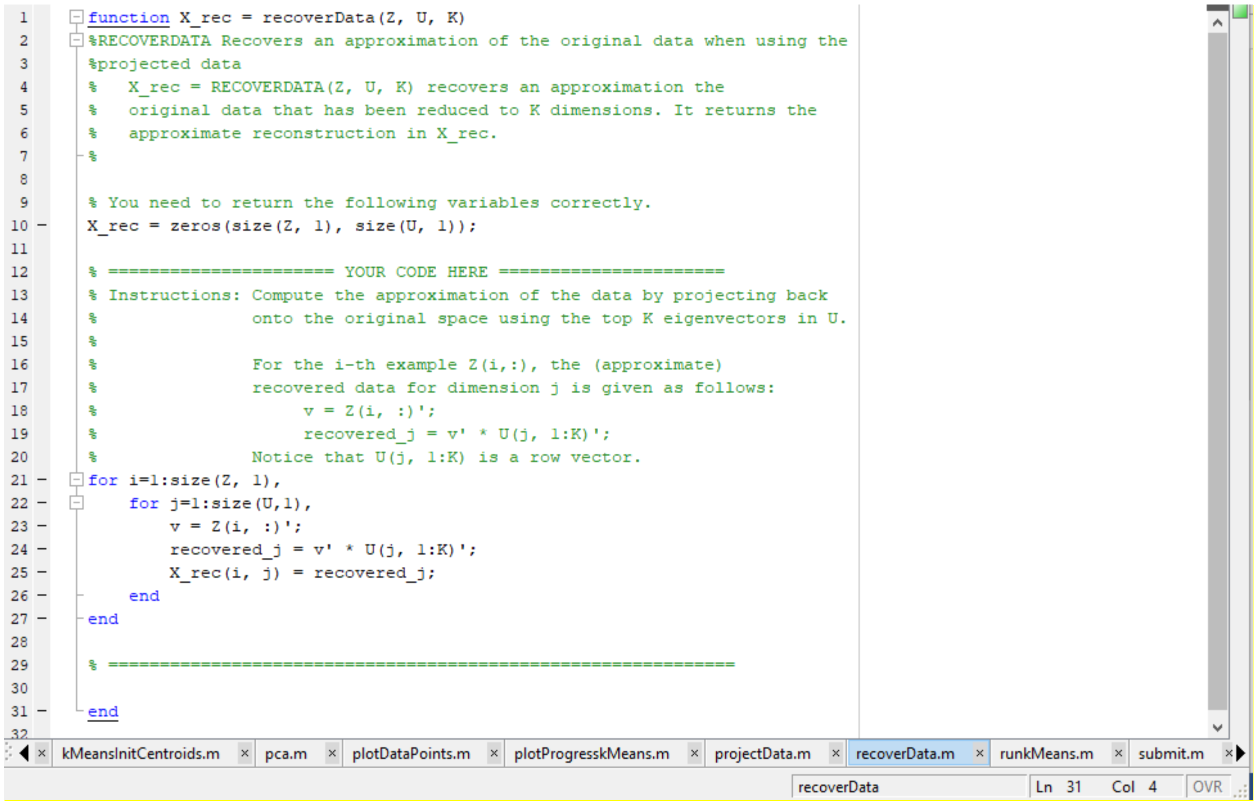
```
Dimension reduction on example dataset.

Projection of the first example: 1.481274

(this value should be about 1.481274)
```

Figure 25: Output of project data sample

Part 2.3.2: Reconstructing an approximation of the data

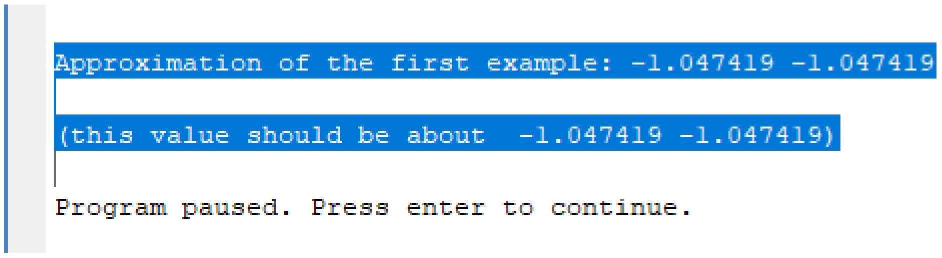
Recovering the high dimensionality from the projected low dimensionality.

recoverData.m



```
function X_rec = recoverData(Z, U, K)
%RECOVERDATA Recovers an approximation of the original data when using the
%projected data
%   X_rec = RECOVERDATA(Z, U, K) recovers an approximation the
%   original data that has been reduced to K dimensions. It returns the
%   approximate reconstruction in X_rec.
%

% You need to return the following variables correctly.
X_rec = zeros(size(Z, 1), size(U, 1));

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the approximation of the data by projecting back
%               onto the original space using the top K eigenvectors in U.
%
%               For the i-th example Z(i,:), the (approximate)
%               recovered data for dimension j is given as follows:
%                    v = Z(i, :)';
%                    recovered_j = v' * U(j, 1:K)';
%               Notice that U(j, 1:K) is a row vector.
for i=1:size(Z, 1),
    for j=1:size(U,1),
        v = Z(i, :)';
        recovered_j = v' * U(j, 1:K)';
        X_rec(i, j) = recovered_j;
    end
end

% =============================================================

end
```

Figure 26: Implementation recover the data by projecting

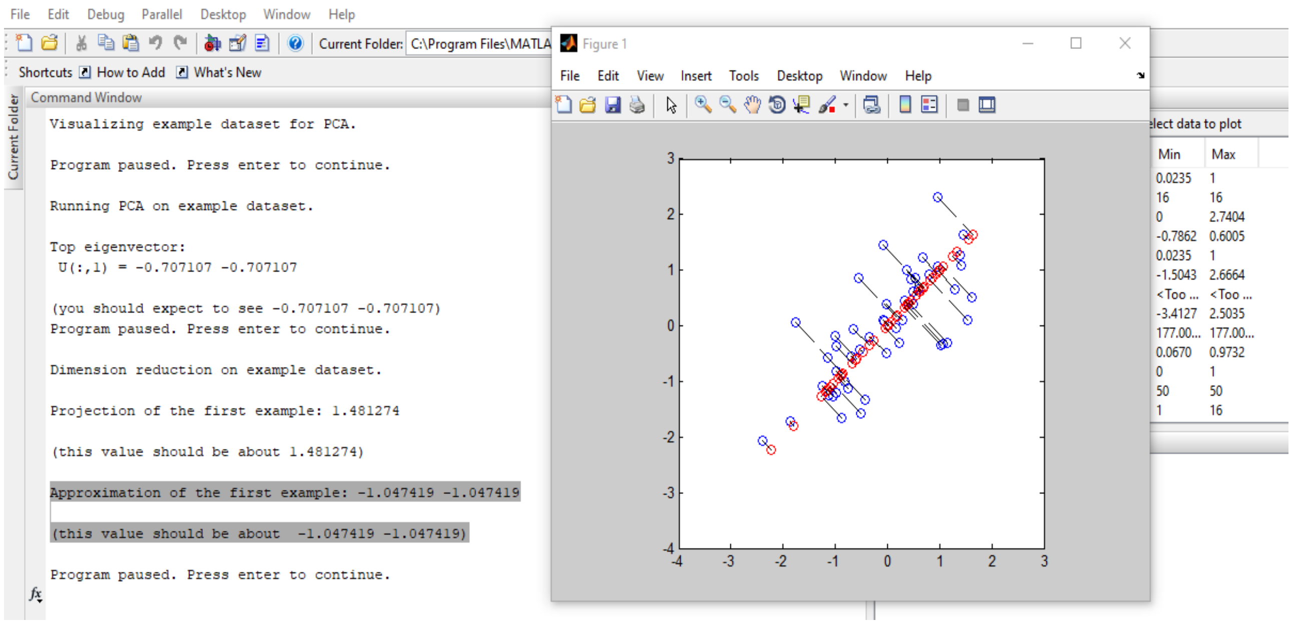


```
Approximation of the first example: -1.047419 -1.047419

(this value should be about  -1.047419 -1.047419)

Program paused. Press enter to continue.
```

Figure 27: Output of the data by projecting

Part 2.3.3: Visualizing the projections

The figure below shows how the original data is projected onto the low-dimensional hyperplane. The blue circle represents the original data, and the red circle represents the data after dimensionality reduction. The projection only effectively retains U 1 U_1 U1Information in a given direction.



```
Visualizing example dataset for PCA.

Program paused. Press enter to continue.

Running PCA on example dataset.

Top eigenvector:
 U(:,1) = -0.707107 -0.707107

(you should expect to see -0.707107 -0.707107)
Program paused. Press enter to continue.

Dimension reduction on example dataset.

Projection of the first example: 1.481274

(this value should be about 1.481274)

Approximation of the first example: -1.047419 -1.047419

(this value should be about  -1.047419 -1.047419)

Program paused. Press enter to continue.
```

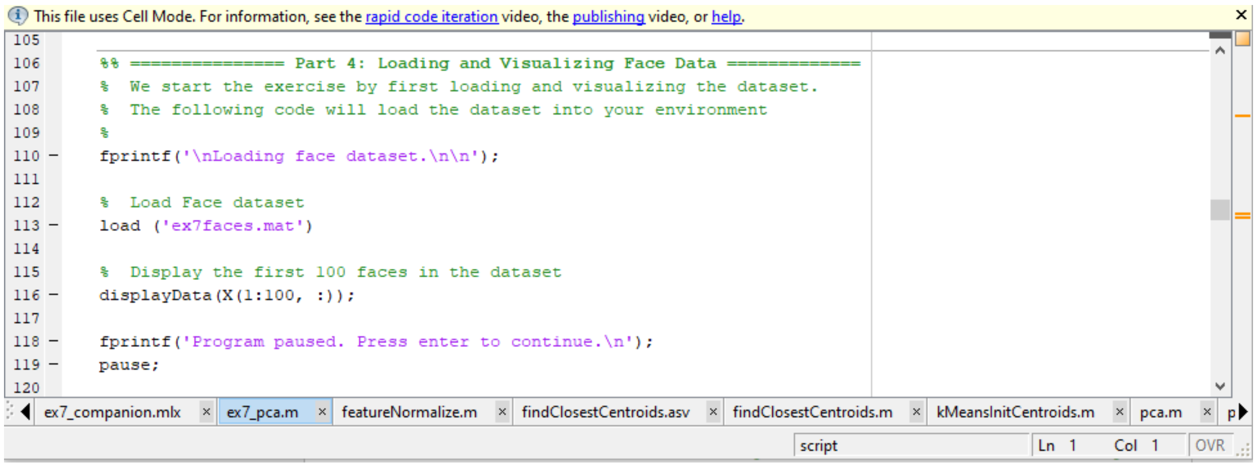
Min	Max
0.0235	1
16	16
0	2.7404
-0.7862	0.6005
0.0235	1
-1.5043	2.6664
<Too ...	<Too ...
-3.4127	2.5035
177.00...	177.00...
0.0670	0.9732
0	1
50	50
1	16

Figure 28: PCA after normalization and projection data

Part 2.4: Face Image Dataset

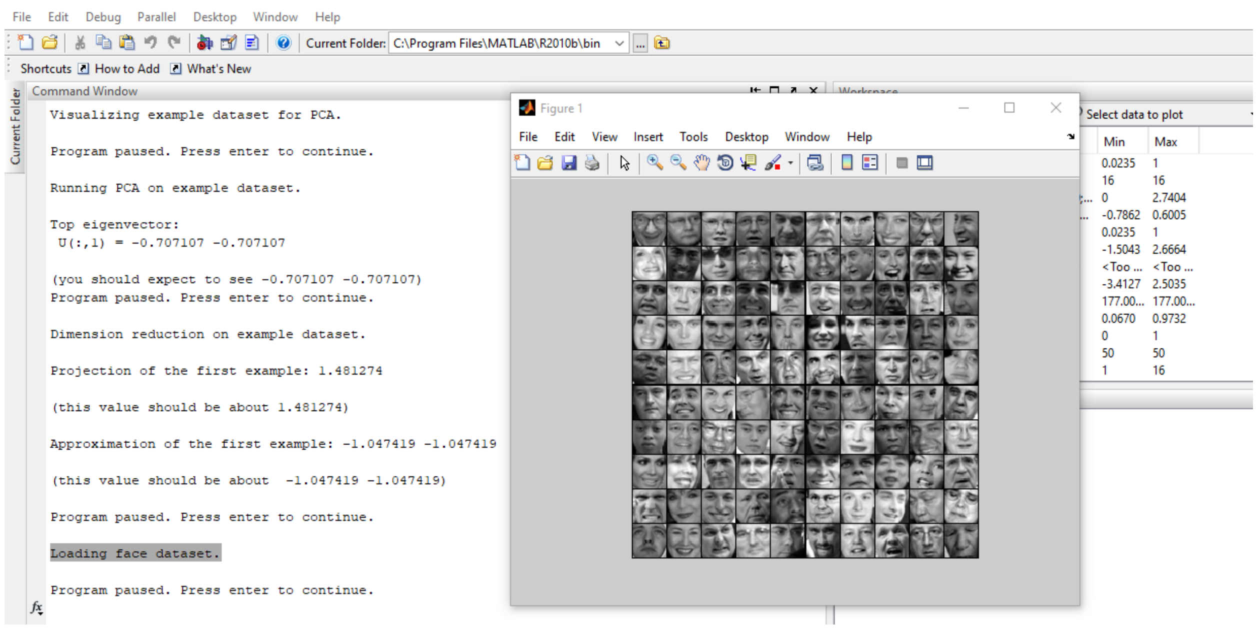
This section we will apply PCA to face images to reduce dimensionality. Perform dimension reduction on face pictures. There are a large number of grayscale images of human faces (32/times 32) in **ex7faces.mat**, so the dimension of each vector is 32/times 32 = 1024.

ex7_pca.m


This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% =============== Part 4: Loading and Visualizing Face Data =============
%  We start the exercise by first loading and visualizing the dataset.
%  The following code will load the dataset into your environment
%
fprintf('\nLoading face dataset.\n\n');

%  Load Face dataset
load ('ex7faces.mat')

%  Display the first 100 faces in the dataset
displayData(X(1:100, :));

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 29: Visualizing face dataset (ex7_pca p4)


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Current Folder
Command Window
Figure 1
File Edit View Insert Tools Desktop Window Help
Select data to plot

```
Visualizing example dataset for PCA.

Program paused. Press enter to continue.

Running PCA on example dataset.

Top eigenvector:
 U(:,1) = -0.707107 -0.707107

(you should expect to see -0.707107 -0.707107)
Program paused. Press enter to continue.

Dimension reduction on example dataset.

Projection of the first example: 1.481274

(this value should be about 1.481274)

Approximation of the first example: -1.047419 -1.047419

(this value should be about  -1.047419 -1.047419)

Program paused. Press enter to continue.

Loading face dataset.

Program paused. Press enter to continue.
```

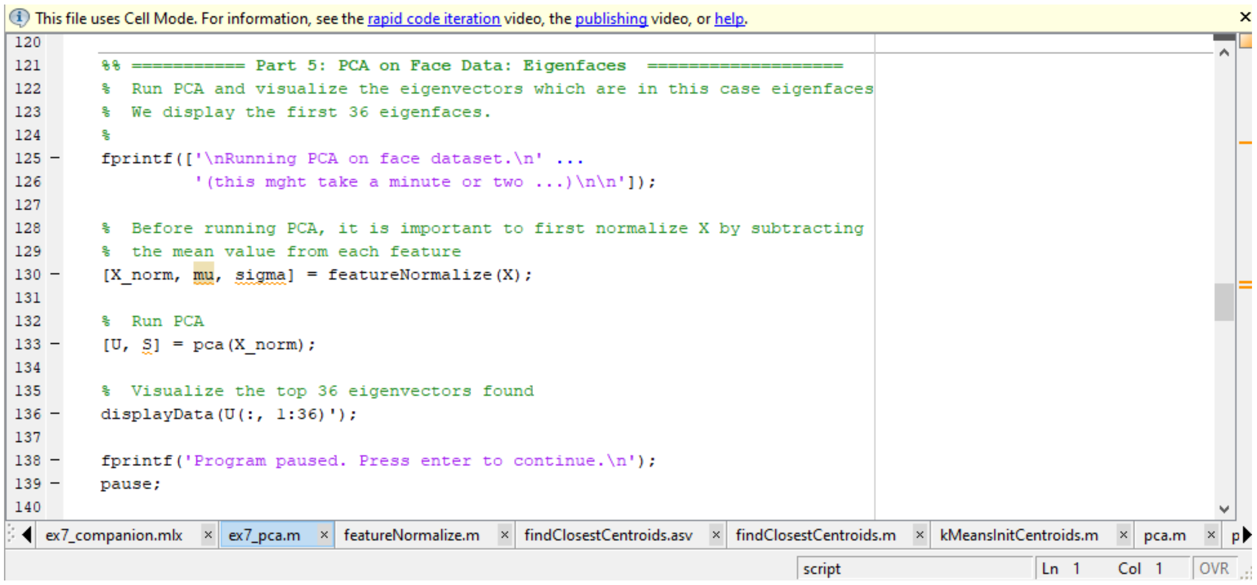
Min	Max
0.0235	1
16	16
0	2.7404
-0.7862	0.6005
0.0235	1
-1.5043	2.6664
<Too ...	<Too ...
-3.4127	2.5035
177.00...	177.00...
0.0670	0.9732
0	1
50	50
1	16

Figure 30: Face data sets

Part 2.4.1: PCA on Faces

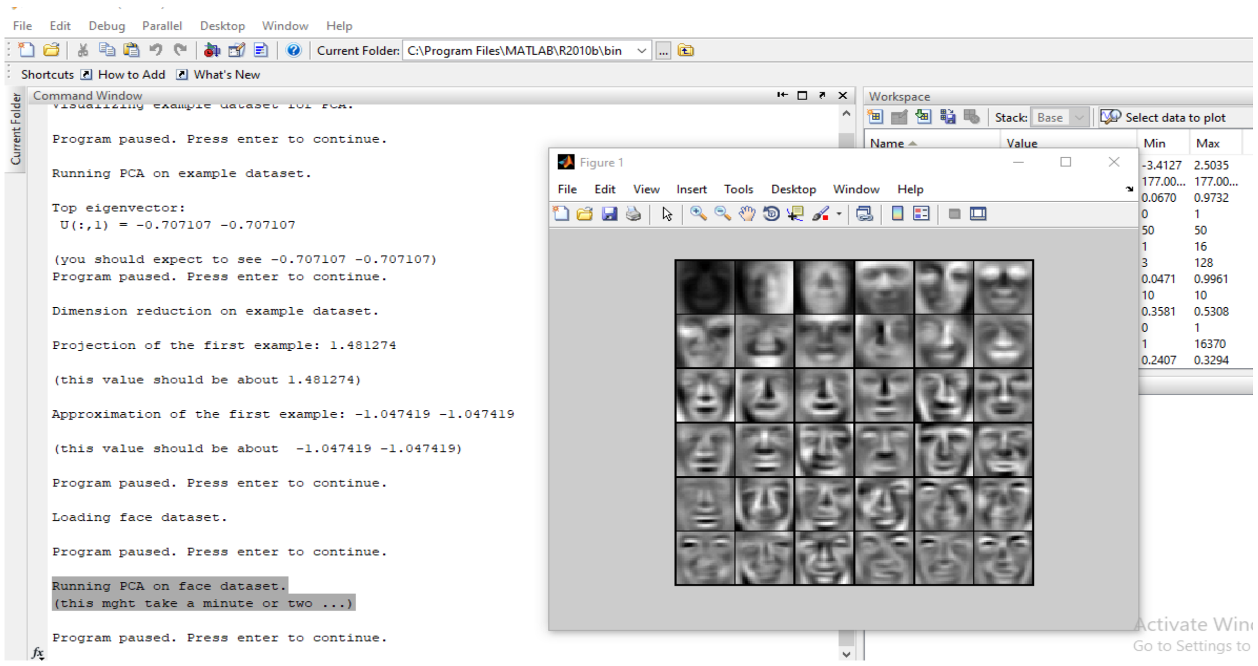
We can do this by reshaping each of them into 32×32 Matrix to visualize these principal components, the matrix corresponds to the pixels in the original dataset. Script **ex7_pca.m** Show the front that describes the biggest change 36 A principal component. If necessary, we can also change the code to show more principal components, to see how they capture more and more details.

ex7_pca.m


This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% =========== Part 5: PCA on Face Data: Eigenfaces  ===================
%  Run PCA and visualize the eigenvectors which are in this case eigenfaces
%  We display the first 36 eigenfaces.
%
fprintf(['\nRunning PCA on face dataset.\n' ...
         '(this mght take a minute or two ...)\n\n']);

%  Before running PCA, it is important to first normalize X by subtracting
%  the mean value from each feature
[X_norm, mu, sigma] = featureNormalize(X);

%  Run PCA
[U, S] = pca(X_norm);

%  Visualize the top 36 eigenvectors found
displayData(U(:, 1:36)');

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 31: PCA on face dataset (ex7_pca p5)



```
visualizing example dataset for PCA.

Program paused. Press enter to continue.

Running PCA on example dataset.

Top eigenvector:
 U(:,1) = -0.707107 -0.707107

(you should expect to see -0.707107 -0.707107)
Program paused. Press enter to continue.

Dimension reduction on example dataset.

Projection of the first example: 1.481274

(this value should be about 1.481274)

Approximation of the first example: -1.047419 -1.047419

(this value should be about  -1.047419 -1.047419)

Program paused. Press enter to continue.

Loading face dataset.

Program paused. Press enter to continue.

Running PCA on face dataset.
(this mght take a minute or two ...)

Program paused. Press enter to continue.
```

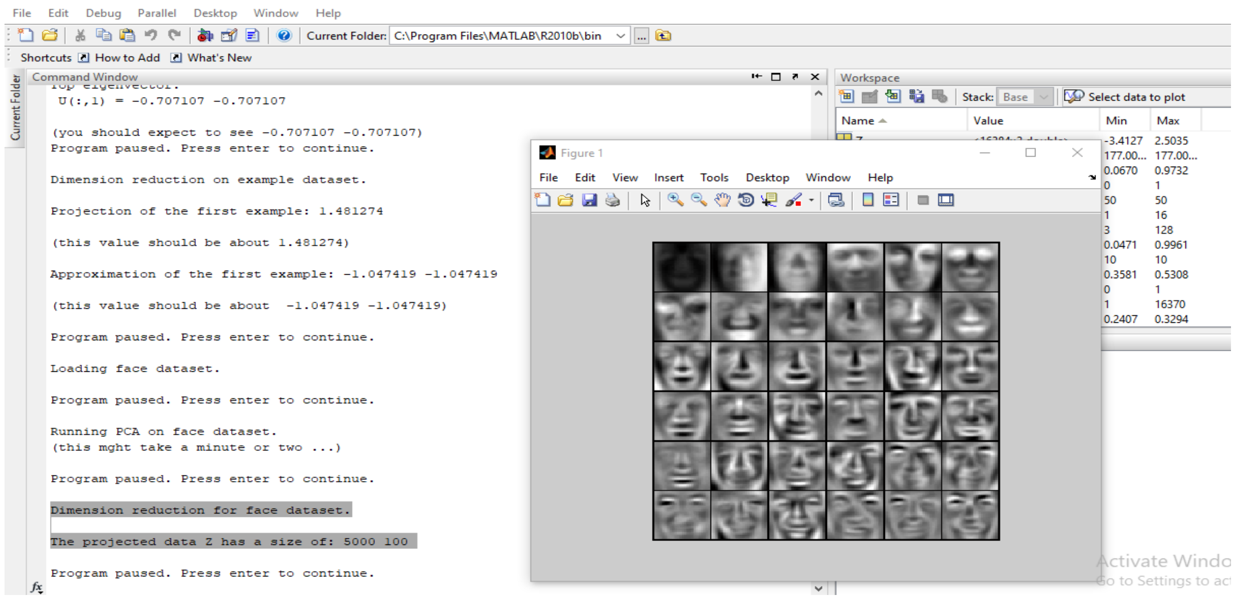
File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command Window
Workspace
Stack: Base
Select data to plot
Figure 1
File Edit View Insert Tools Desktop Window Help
Name Value Min Max

Min	Max
-3.4127	2.5035
177.00...	177.00...
0.0670	0.9732
0	1
50	50
1	16
3	128
0.0471	0.9961
10	10
0.3581	0.5308
0	1
1	16370
0.2407	0.3294

Figure 32: The principal components on the face dataset

Part 2.4.2: Dimensionality Reduction

Take the first 100 feature vectors for projection



```
Top eigenvector:
 U(:,1) = -0.707107 -0.707107

(you should expect to see -0.707107 -0.707107)
Program paused. Press enter to continue.

Dimension reduction on example dataset.

Projection of the first example: 1.481274

(this value should be about 1.481274)

Approximation of the first example: -1.047419 -1.047419

(this value should be about  -1.047419 -1.047419)

Program paused. Press enter to continue.

Loading face dataset.

Program paused. Press enter to continue.

Running PCA on face dataset.
(this mght take a minute or two ...)

Program paused. Press enter to continue.

Dimension reduction for face dataset.

The projected data Z has a size of: 5000 100

Program paused. Press enter to continue.
```

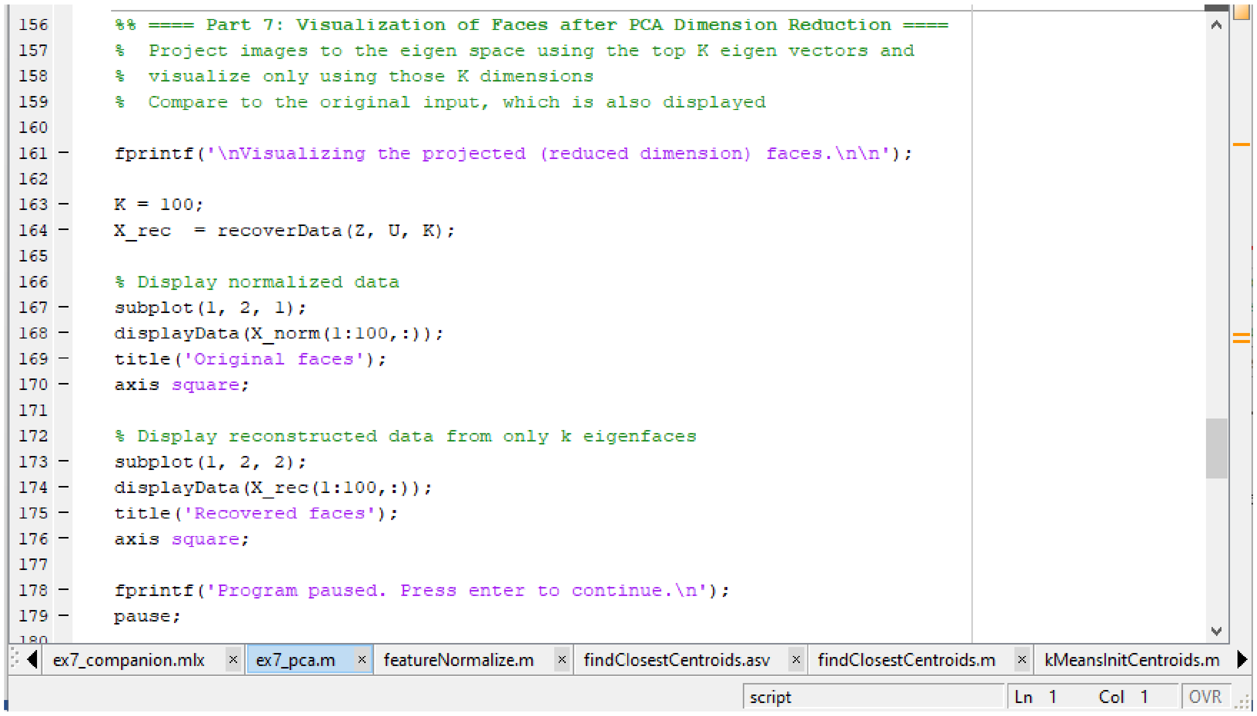
File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command Window
Workspace
Stack: Base
Select data to plot
Figure 1
File Edit View Insert Tools Desktop Window Help
Name Value Min Max

Min	Max
-3.4127	2.5035
177.00...	177.00...
0.0670	0.9732
0	1
50	50
1	16
3	128
0.0471	0.9961
10	10
0.3581	0.5308
0	1
1	16370
0.2407	0.3294

Figure 33: Principal components on the face dataset

It can be seen that after the dimensionality is reduced, the general frame of the face is still retained, but some details are lost. The inspiration for us is that when we are training face recognition with neural networks, we can sometimes use this method to increase the speed

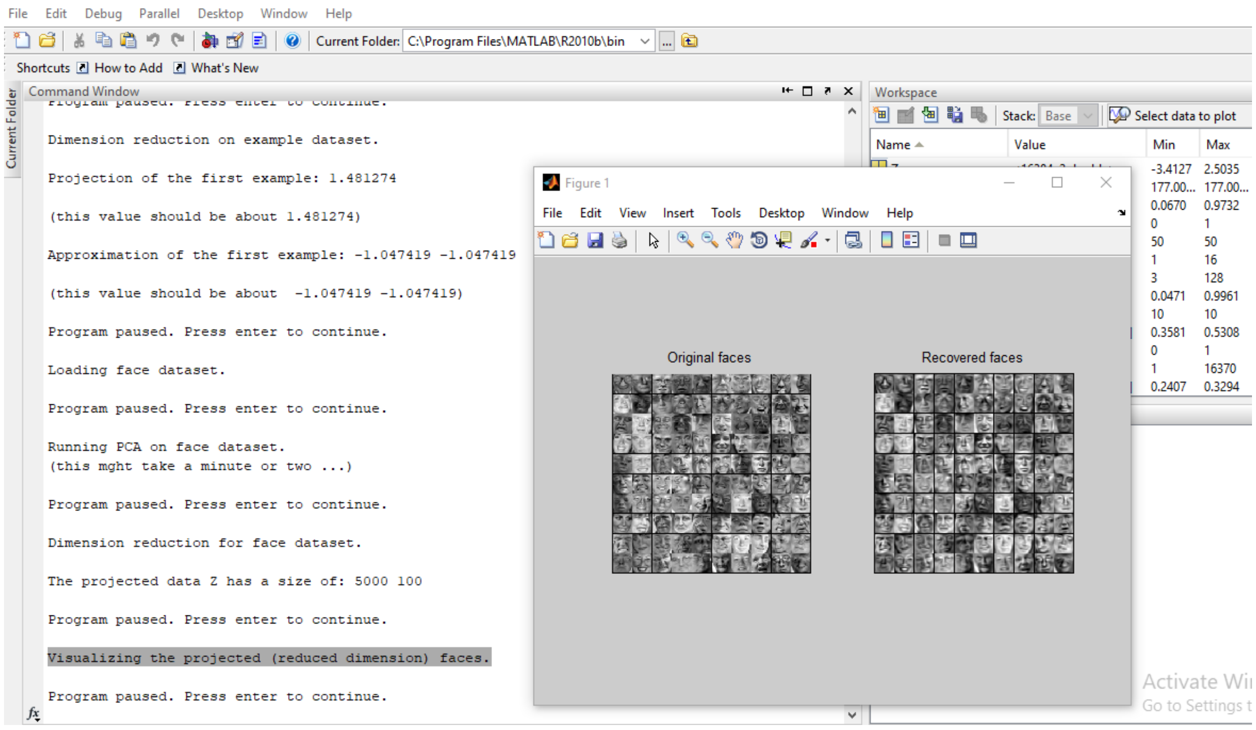
ex7_pca.m



```
%% ==== Part 7: Visualization of Faces after PCA Dimension Reduction ====
%  Project images to the eigen space using the top K eigen vectors and
%  visualize only using those K dimensions
%  Compare to the original input, which is also displayed

fprintf('\nVisualizing the projected (reduced dimension) faces.\n\n');

K = 100;
X_rec  = recoverData(Z, U, K);

% Display normalized data
subplot(1, 2, 1);
displayData(X_norm(1:100,:));
title('Original faces');
axis square;

% Display reconstructed data from only k eigenfaces
subplot(1, 2, 2);
displayData(X_rec(1:100,:));
title('Recovered faces');
axis square;

fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 34: Visualization of Faces after PCA Dimension Reduction

The image on the left is the original image, and the image on the right is the image reconstructed after dimensionality reduction.


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command Window
Workspace
Stack: Base
Select data to plot
Figure 1
File Edit View Insert Tools Desktop Window Help
Original faces
Recovered faces

```
Program paused. Press enter to continue.

Dimension reduction on example dataset.

Projection of the first example: 1.481274

(this value should be about 1.481274)

Approximation of the first example: -1.047419 -1.047419

(this value should be about  -1.047419 -1.047419)

Program paused. Press enter to continue.

Loading face dataset.

Program paused. Press enter to continue.

Running PCA on face dataset.
(this mght take a minute or two ...)

Program paused. Press enter to continue.

Dimension reduction for face dataset.

The projected data Z has a size of: 5000 100

Program paused. Press enter to continue.

Visualizing the projected (reduced dimension) faces.

Program paused. Press enter to continue.
```

Name	Value	Min	Max
		-3.4127	2.5035
		177.00...	177.00...
		0.0670	0.9732
		0	1
		50	50
		1	16
		3	128
		0.0471	0.9961
		10	10
		0.3581	0.5308
		0	1
		1	16370
		0.2407	0.3294

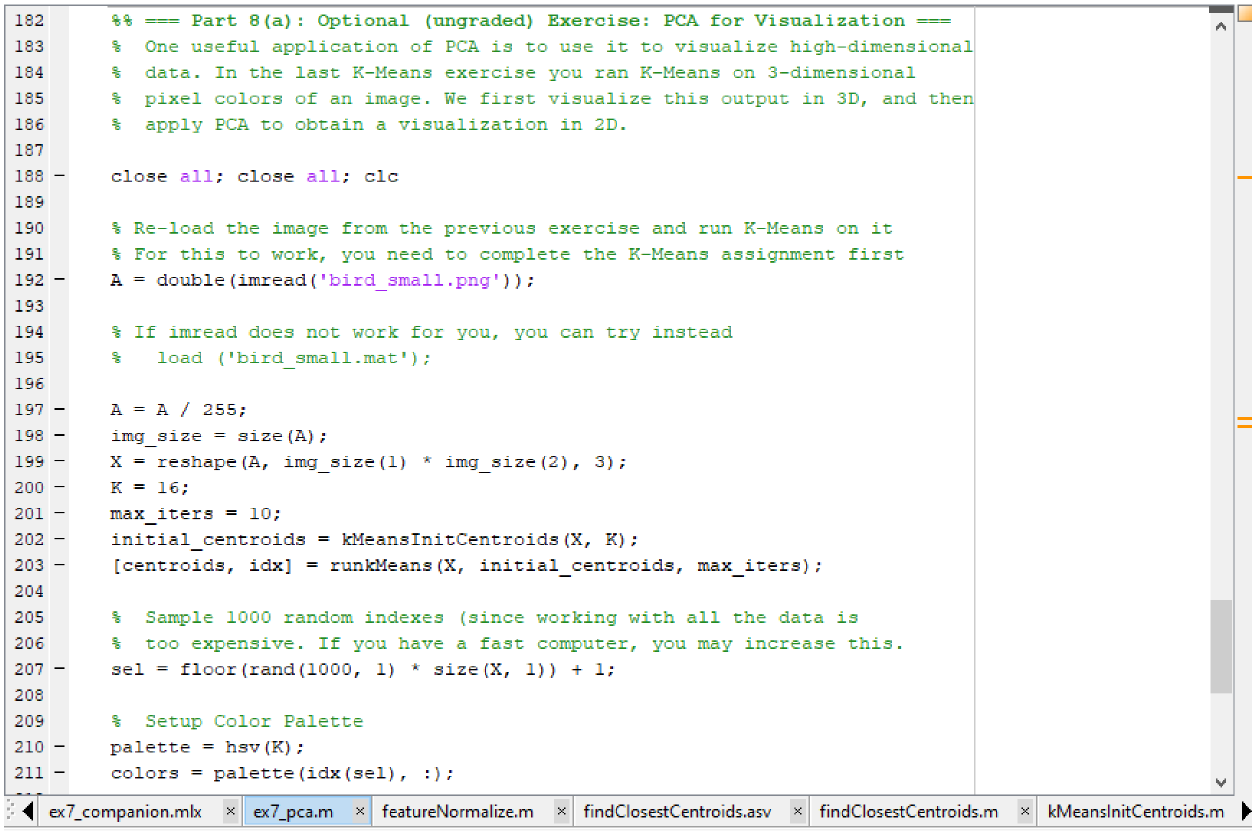
Figure 35: Original images of faces and ones reconstructed from only the top 100 principal components.

It can be seen that after reducing the dimensions, the general framework of the human face is still preserved, but some details are lost. This inspired us that when we are training facial recognition with neural networks, we can sometimes use this method to increase speed.

Part 2.5: PCA for visualization

PCA Commonly used for high-dimensional vector display. We will use PCA implementation to 3D data, reduce it to a 2D space, and visualize the result as a 2D scatter plot. The PCA projection can be thought of as a rotation, which selects the view that maximizes the data, which usually corresponds to the "best" view.

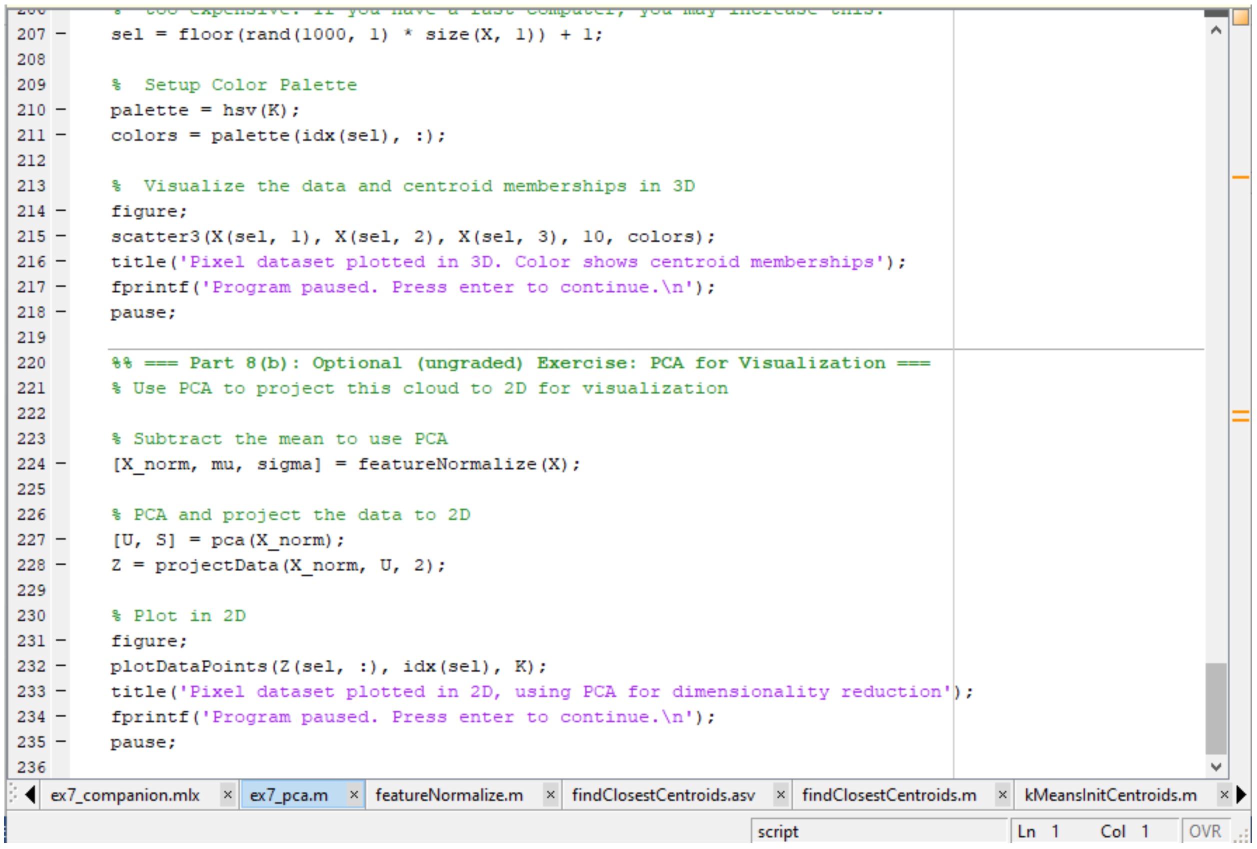
ex7_pca.m



```
%% === Part 8(a): Optional (ungraded) Exercise: PCA for Visualization ===
%  One useful application of PCA is to use it to visualize high-dimensional
%  data. In the last K-Means exercise you ran K-Means on 3-dimensional
%  pixel colors of an image. We first visualize this output in 3D, and then
%  apply PCA to obtain a visualization in 2D.

close all; close all; clc

% Re-load the image from the previous exercise and run K-Means on it
% For this to work, you need to complete the K-Means assignment first
A = double(imread('bird_small.png'));

% If imread does not work for you, you can try instead
%   load ('bird_small.mat');

A = A / 255;
img_size = size(A);
X = reshape(A, img_size(1) * img_size(2), 3);
K = 16;
max_iters = 10;
initial_centroids = kMeansInitCentroids(X, K);
[centroids, idx] = runkMeans(X, initial_centroids, max_iters);

%  Sample 1000 random indexes (since working with all the data is
%  too expensive. If you have a fast computer, you may increase this.
sel = floor(rand(1000, 1) * size(X, 1)) + 1;

%  Setup Color Palette
palette = hsv(K);
colors = palette(idx(sel), :);
```

Figure 36: PCA for visualizing s-1 (ex7_pca p8)

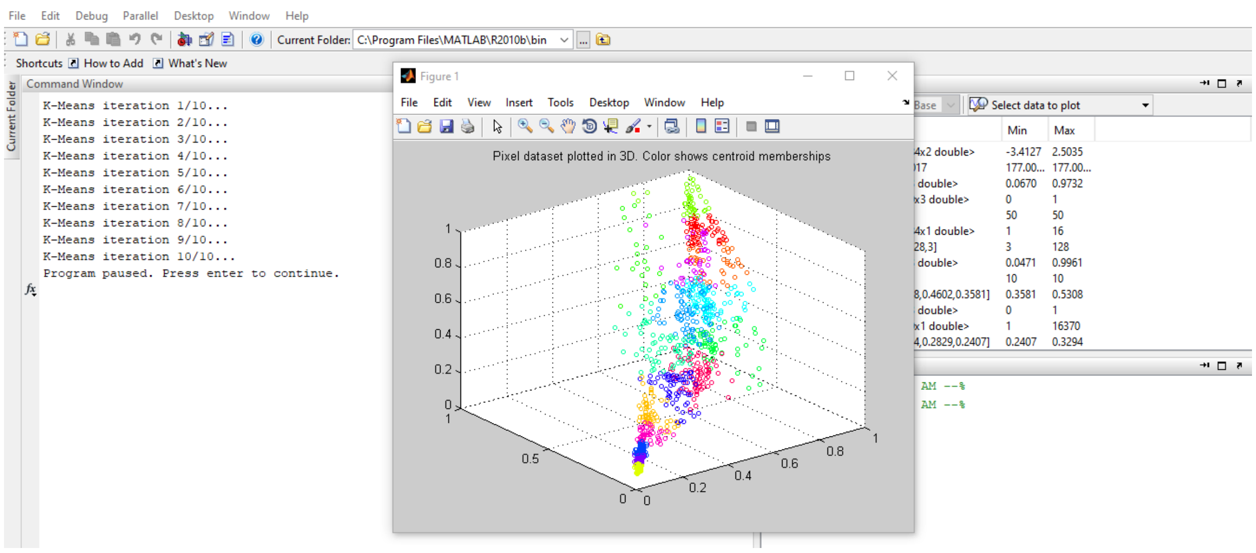
ex7_pca.m



```
sel = floor(rand(1000, 1) * size(X, 1)) + 1;

%  Setup Color Palette
palette = hsv(K);
colors = palette(idx(sel), :);

%  Visualize the data and centroid memberships in 3D
figure;
scatter3(X(sel, 1), X(sel, 2), X(sel, 3), 10, colors);
title('Pixel dataset plotted in 3D. Color shows centroid memberships');
fprintf('Program paused. Press enter to continue.\n');
pause;

%% === Part 8(b): Optional (ungraded) Exercise: PCA for Visualization ===
% Use PCA to project this cloud to 2D for visualization

% Subtract the mean to use PCA
[X_norm, mu, sigma] = featureNormalize(X);

% PCA and project the data to 2D
[U, S] = pca(X_norm);
Z = projectData(X_norm, U, 2);

% Plot in 2D
figure;
plotDataPoints(Z(sel, :), idx(sel), K);
title('Pixel dataset plotted in 2D, using PCA for dimensionality reduction');
fprintf('Program paused. Press enter to continue.\n');
pause;
```

Figure 37: PCA for visualizing s-2 (ex7_pca p8)

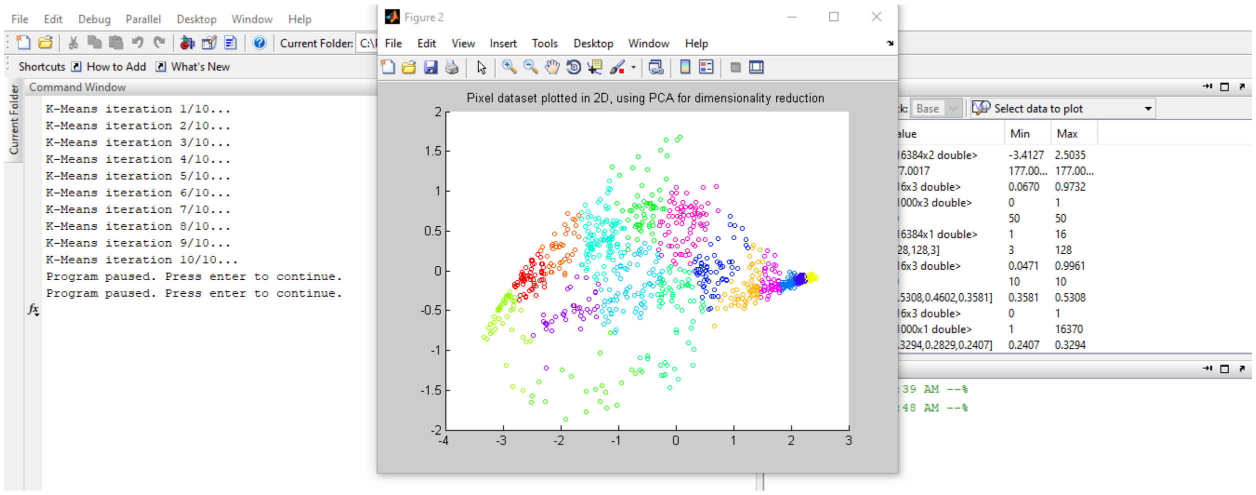
PCA Commonly used for high-dimensional vector display. As shown below, use **K-means** to classify points in three-dimensional space.


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Shortcuts How to Add What's New
Command Window
Current Folder
Figure 1
File Edit View Insert Tools Desktop Window Help
Pixel dataset plotted in 3D. Color shows centroid memberships
Select data to plot
Min
Max

```
K-Means iteration 1/10...
K-Means iteration 2/10...
K-Means iteration 3/10...
K-Means iteration 4/10...
K-Means iteration 5/10...
K-Means iteration 6/10...
K-Means iteration 7/10...
K-Means iteration 8/10...
K-Means iteration 9/10...
K-Means iteration 10/10...
Program paused. Press enter to continue.
```

	Min	Max
4x2 double>	-3.4127	2.5035
)17	177.00...	177.00...
double>	0.0670	0.9732
x3 double>	0	1
	50	50
4x1 double>	1	16
28,3]	3	128
double>	0.0471	0.9961
	10	10
8,0.4602,0.3581]	0.3581	0.5308
double>	0	1
x1 double>	1	16370
4,0.2829,0.2407]	0.2407	0.3294

Figure 38: Raw data in dimensions


File Edit Debug Parallel Desktop Window Help
Current Folder: C:\
Shortcuts How to Add What's New
Command Window
Current Folder
Figure 2
File Edit View Insert Tools Desktop Window Help
Pixel dataset plotted in 2D, using PCA for dimensionality reduction
Select data to plot
alue
Min
Max

```
K-Means iteration 1/10...
K-Means iteration 2/10...
K-Means iteration 3/10...
K-Means iteration 4/10...
K-Means iteration 5/10...
K-Means iteration 6/10...
K-Means iteration 7/10...
K-Means iteration 8/10...
K-Means iteration 9/10...
K-Means iteration 10/10...
Program paused. Press enter to continue.
Program paused. Press enter to continue.
```

	Min	Max
6384x2 double>	-3.4127	2.5035
7.0017	177.00...	177.00...
6x3 double>	0.0670	0.9732
000x3 double>	0	1
	50	50
6384x1 double>	1	16
28,128,3]	3	128
6x3 double>	0.0471	0.9961
	10	10
5308,0.4602,0.3581]	0.3581	0.5308
6x3 double>	0	1
000x1 double>	1	16370
3294,0.2829,0.2407]	0.2407	0.3294

Figure 39: Use PCA Produced 2D visualization