Programming Exercise 7
K-means Clustering and Principal Component Analysis

Machine Learning (Stanford University)

September 2020

This exercise shows the implementation of:

(1) K-means algorithm and its application to image compression
(2) Principal Component Analysis (PCA) to find a low-dimensional representation of face images

1 K-means Clustering

Instruction:
In this this exercise, you will implement the K-means algorithm and use it for image compression.

(1) You will first start on an example 2D dataset that will help you gain an intuition of how the K-means algorithm works.
(2) 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.

1.1 Implementing K-means

The K-means algorithm

```
% 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 cˆ(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
```

Implementation details:

- 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.

- 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)

1.1.1 Finding closest centroids (Cluster Assignment Step)

Cluster assignment is given as, 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 $x^{(i)}$

μ_j is the position (value) of the j'th centroid

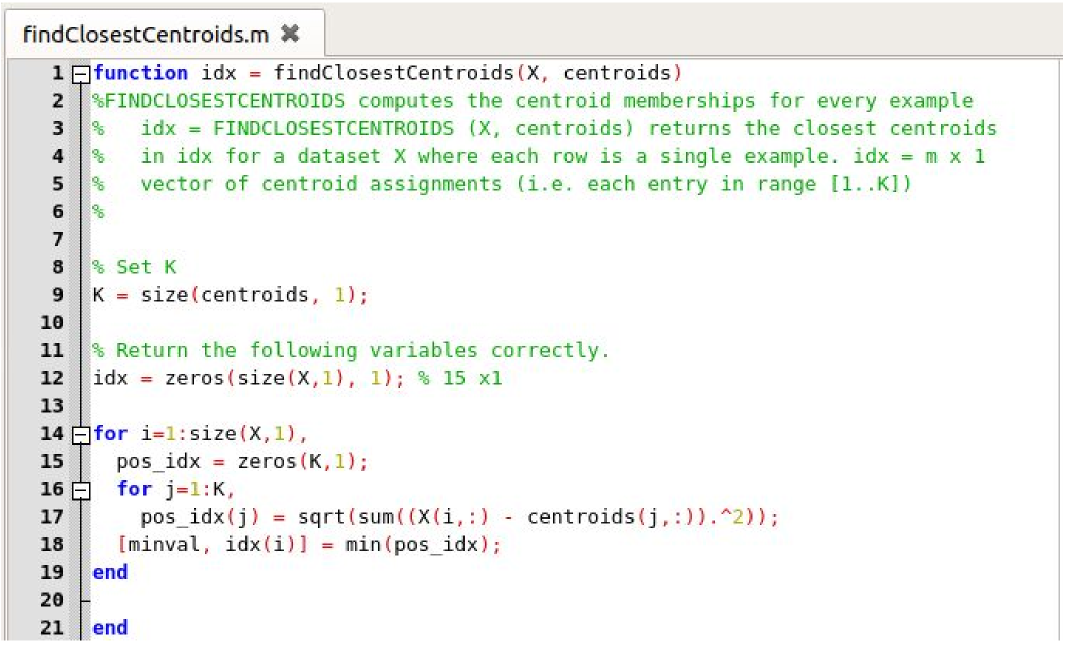
Note: $c^{(i)}$ corresponds to `idx(i)` in the starter code

Instruction:

Your task is to complete the code in findClosestCentroids.m.

- This function takes the data matrix X and the locations of all centroids inside centroids
- It should output a 1D array idx that holds index (a value in {1, …, K} where K is the total number of centroids) of the closest centroid to every training example

Implementation:



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);

% Return the following variables correctly.
idx = zeros(size(X,1), 1); % 15 x1

for i=1:size(X,1),
  pos_idx = zeros(K,1);
  for j=1:K,
    pos_idx(j) = sqrt(sum((X(i,:) - centroids(j,:)).^2));
  [minval, idx(i)] = min(pos_idx);
end

end
```

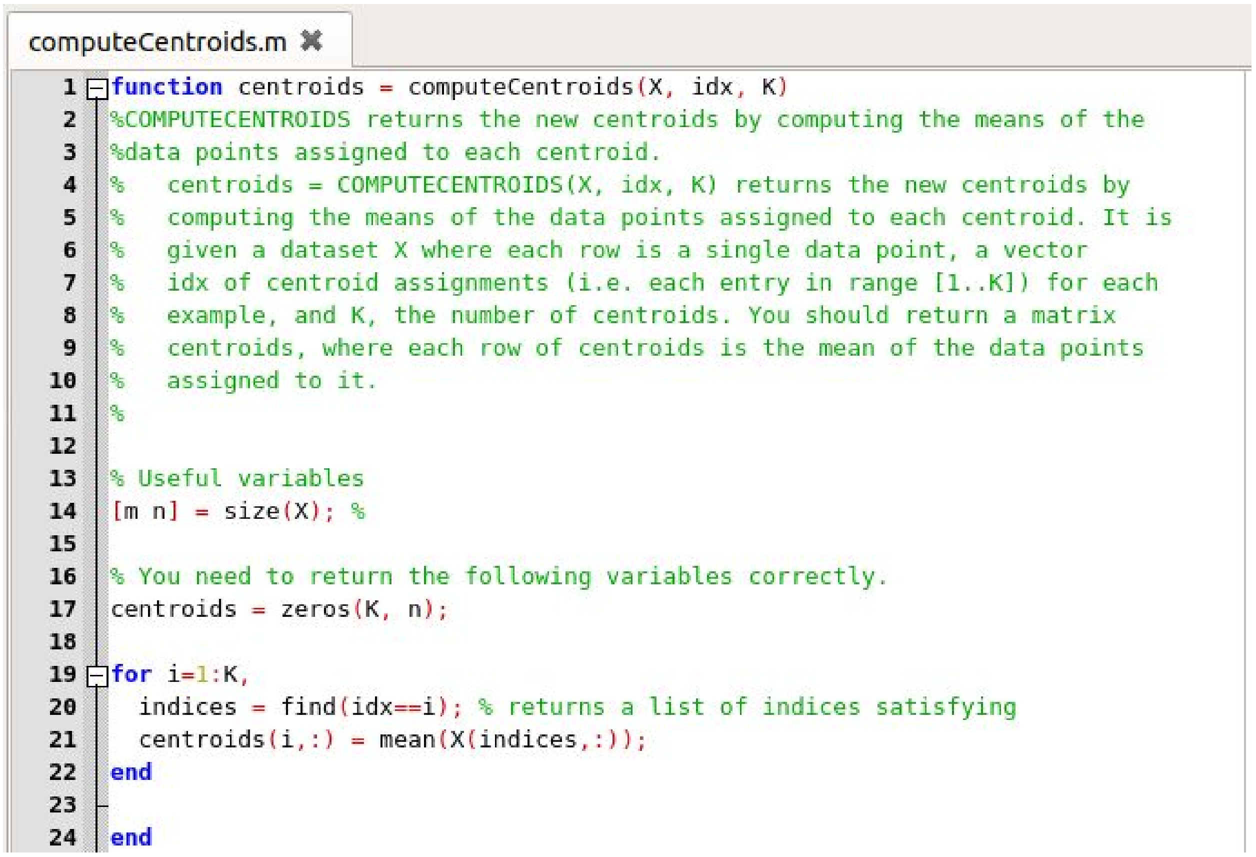
1.1.2 Computing centroid means

Instruction:
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. Specifically, for every centroid k we set

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

*This is your common mean. What $|C_k|$ means is the size or number of elements from that cluster.

Implementation:



computeCentroids.m

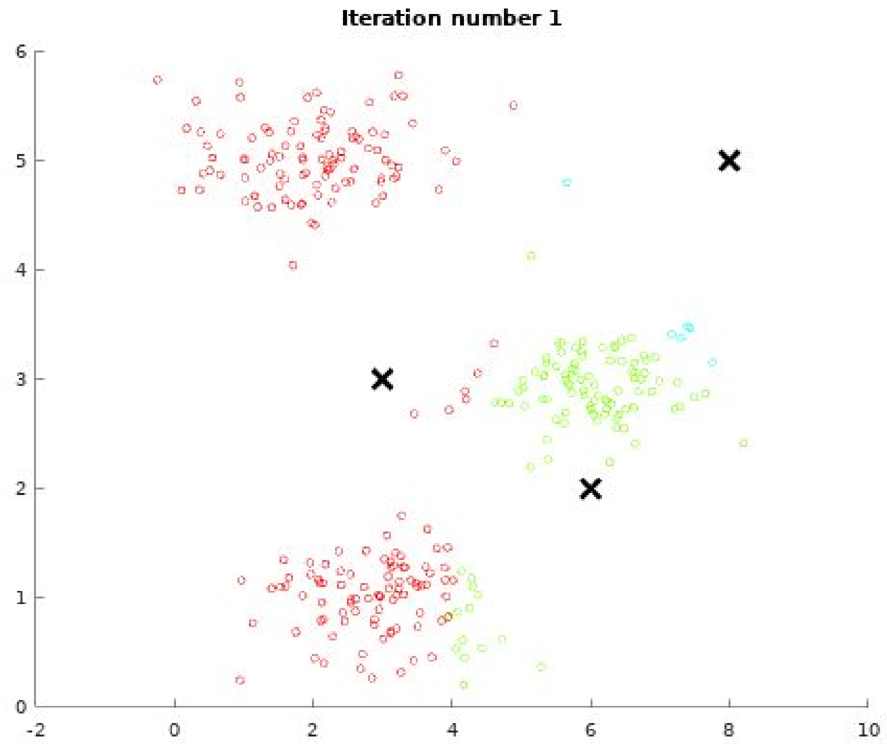
```
function centroids = computeCentroids(X, idx, K)
%COMPUTECENTROIDS returns the new centroids by computing the means of the
%data points assigned to each centroid.
%   centroids = COMPUTECENTROIDS(X, idx, K) returns the new centroids by
%   computing the means of the data points assigned to each centroid. It is
%   given a dataset X where each row is a single data point, a vector
%   idx of centroid assignments (i.e. each entry in range [1..K]) for each
%   example, and K, the number of centroids. You should return a matrix
%   centroids, where each row of centroids is the mean of the data points
%   assigned to it.
%

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

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

for i=1:K,
  indices = find(idx==i); % returns a list of indices satisfying
  centroids(i,:) = mean(X(indices,:));
end

end
```

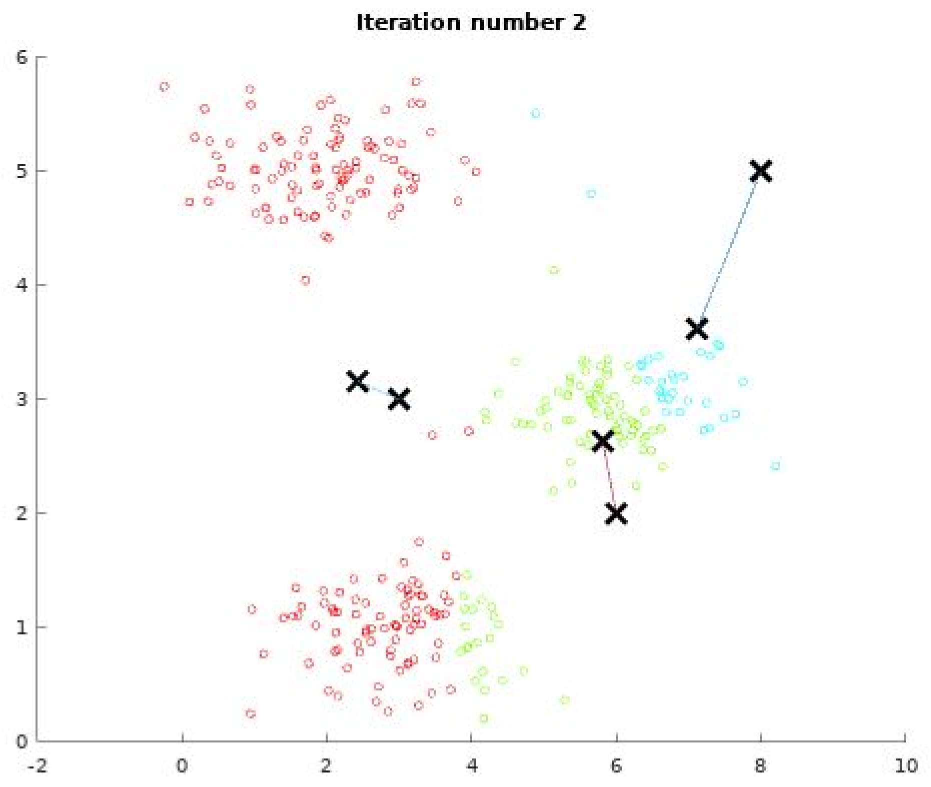
1.2 K-means on example dataset

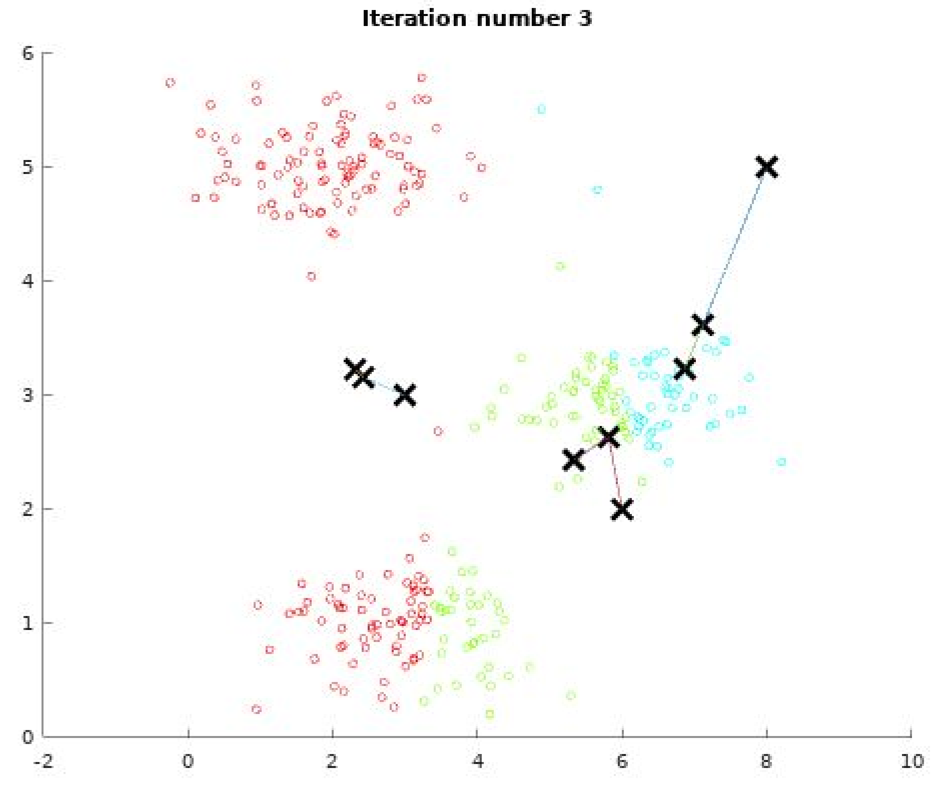
Instruction:
After you have completed the two functions (findClosestCentroids and computeCentroids), the next step in ex7.m will run the K-means algorithm on a toy 2D dataset to help you understand

how K-means works. Your functions are called from inside the runKmeans.m script. We encourage you to take a look at the function to understand how it works. Notice that the code calls the two functions you implemented in a loop.

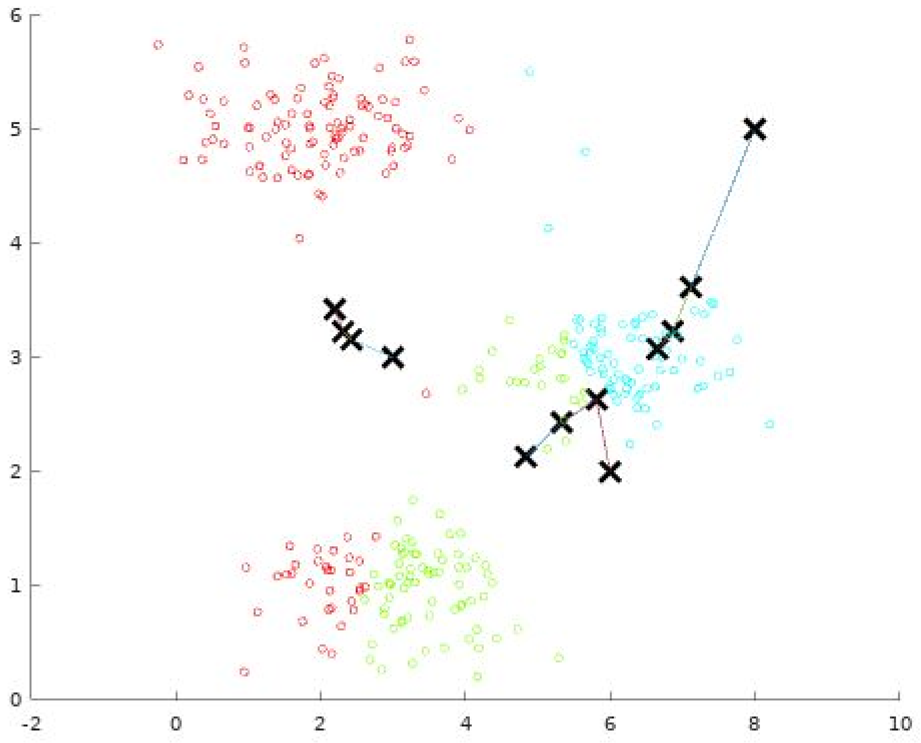
When you run the next step, the K-means code will produce a visualization that steps you through the progress of the algorithm at each iteration. Press enter multiple times to see how each step of the K-means algorithm changes the centroids and cluster assignments



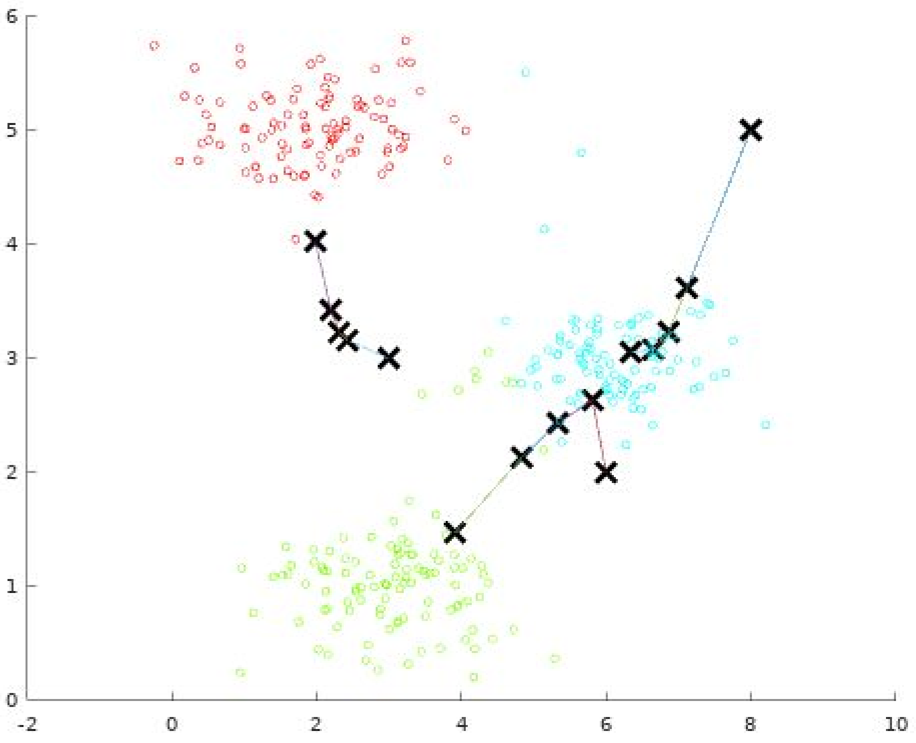

Iteration number 2
6
5
4
3
2
1
0
-2
0
2
4
6
8
10

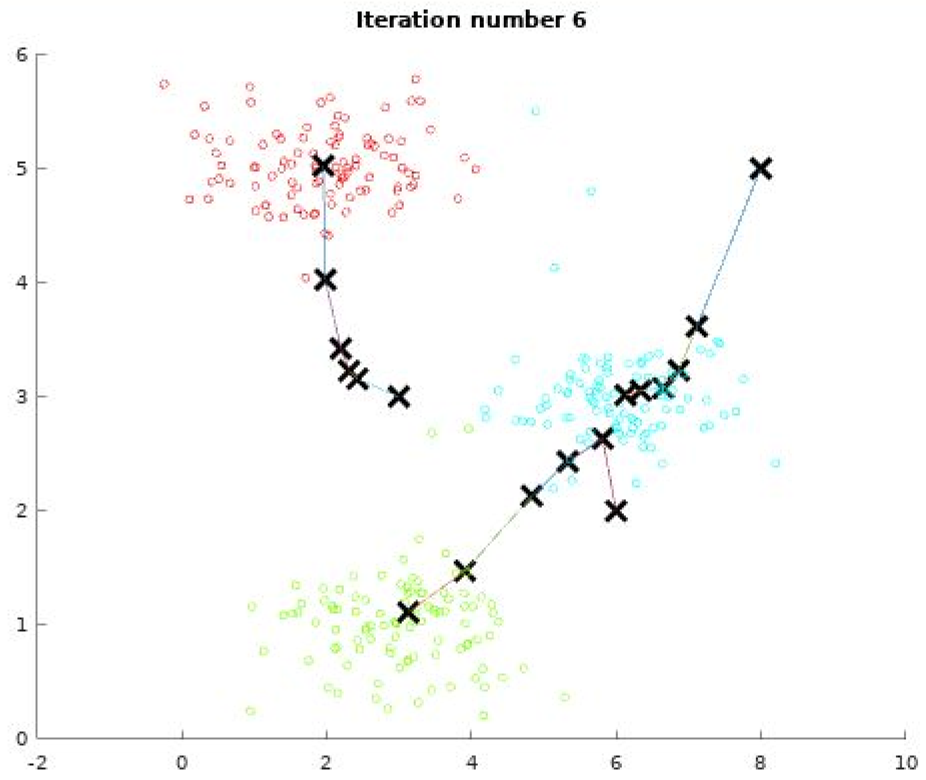

Iteration number 3
6
5
4
3
2
1
0
-2
0
2
4
6
8
10

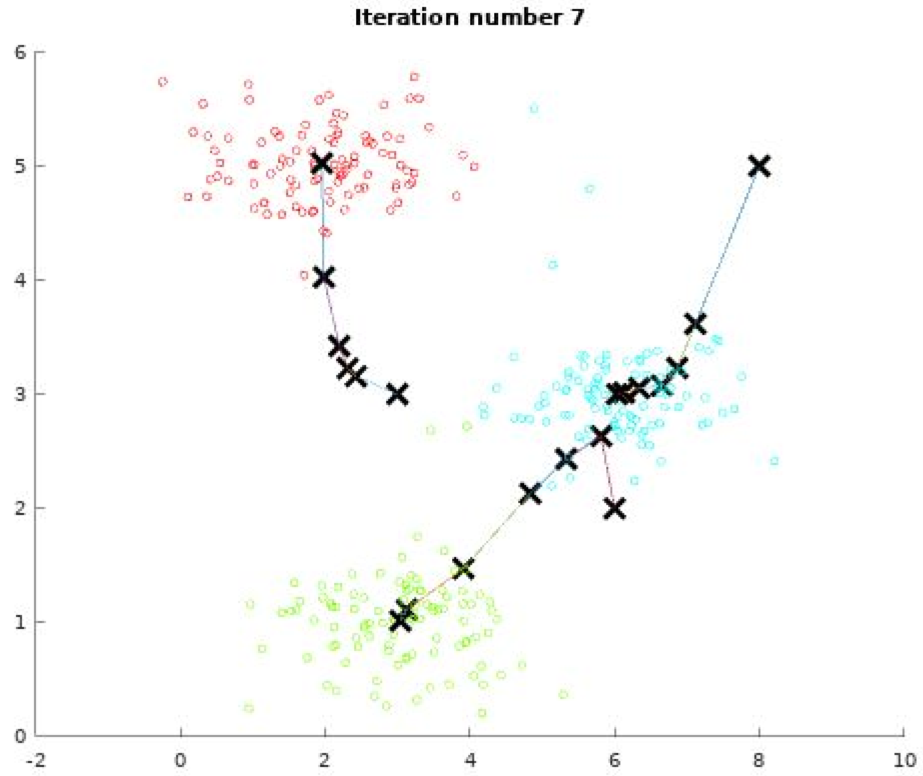
Iteration number 4



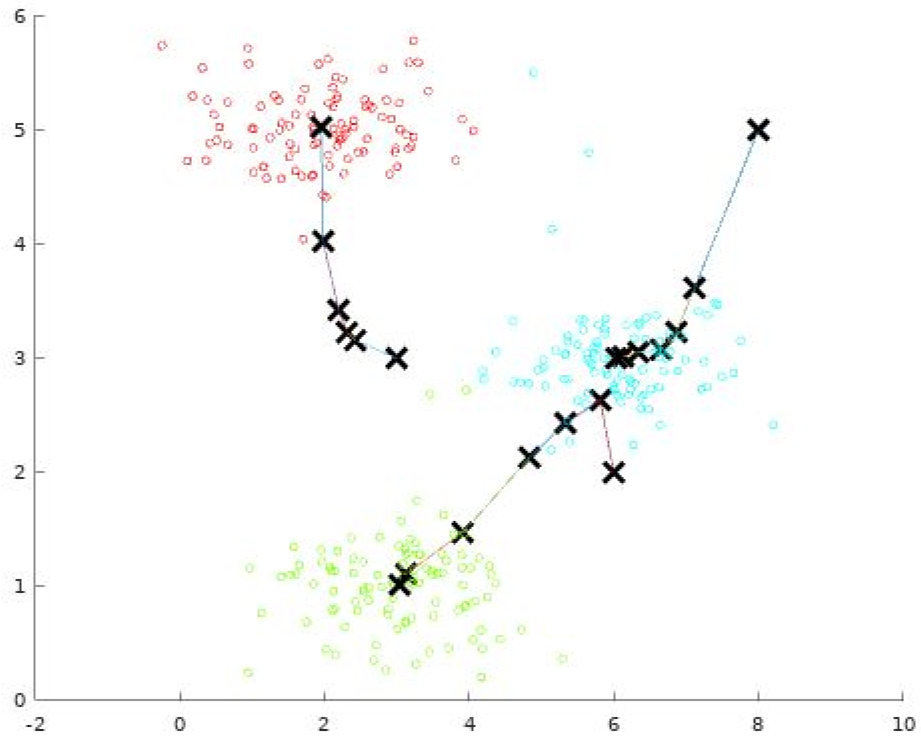
Iteration number 5



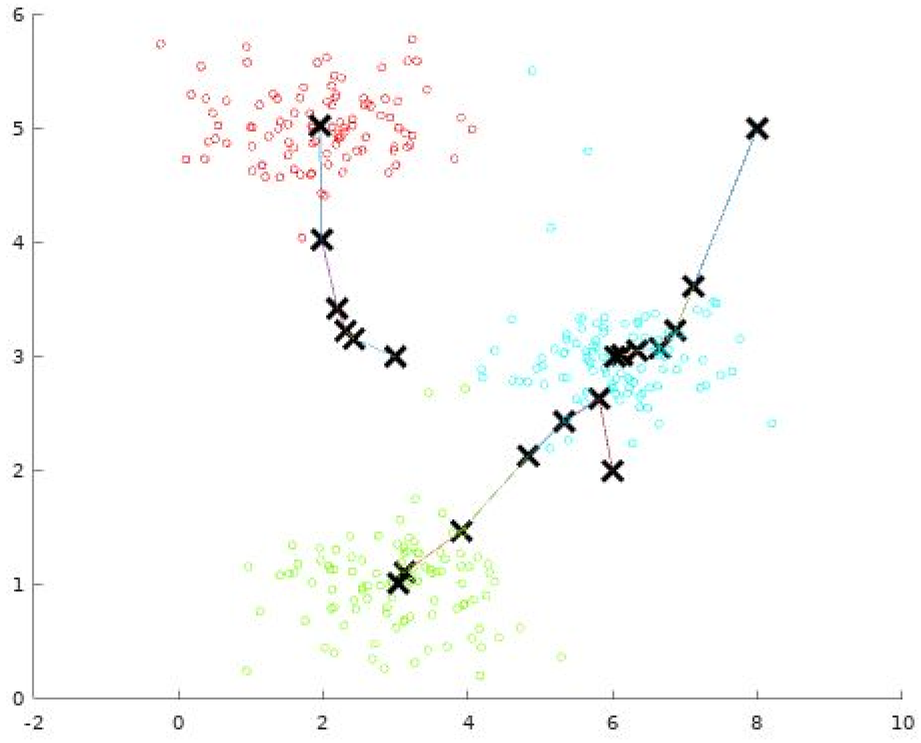

Iteration number 6
6
5
4
3
2
1
0
-2
0
2
4
6
8
10

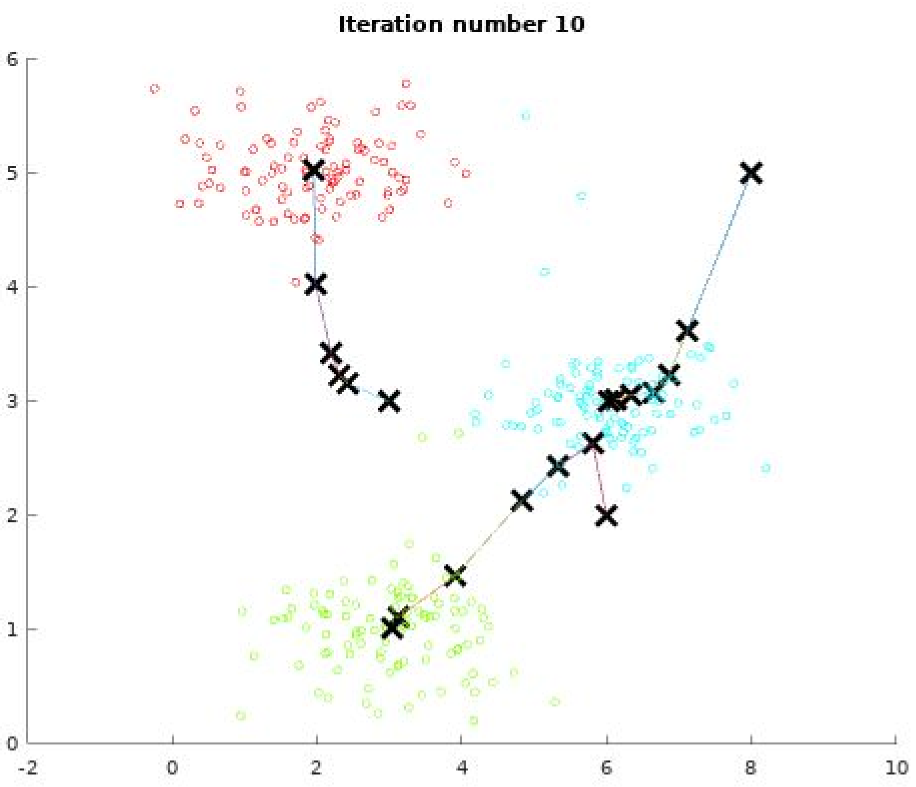

Iteration number 7
6
5
4
3
2
1
0
-2
0
2
4
6
8
10

Iteration number 8



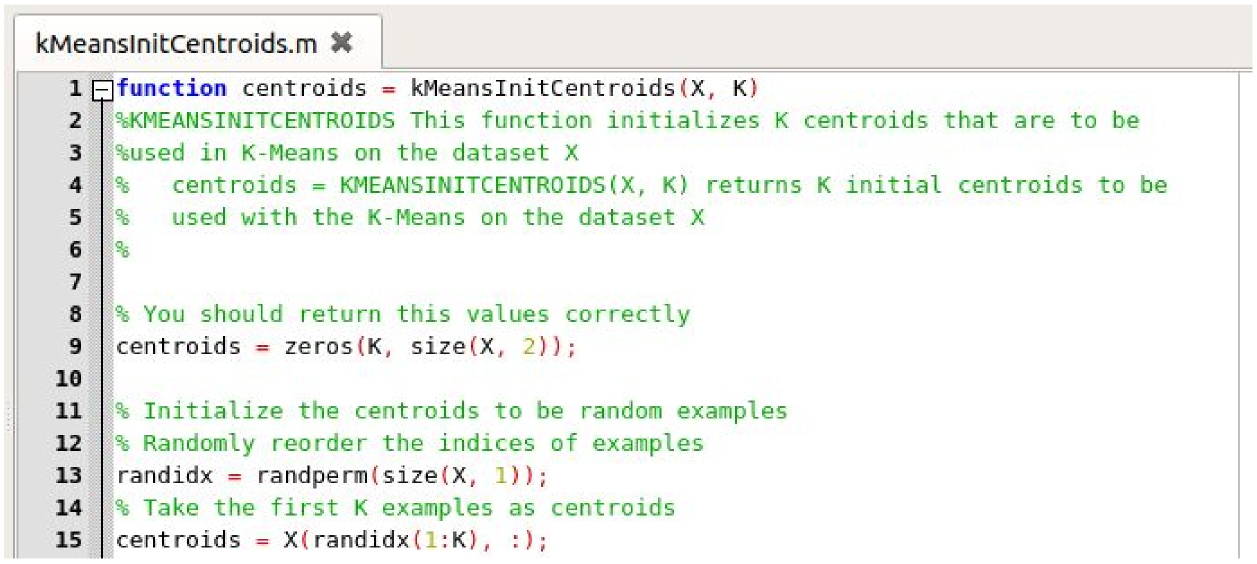
Iteration number 9





1.3 Random Initialization

Instruction:
In practice, a good strategy for initializing the centroids is to select random examples from the training set.



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));

% 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), :);
```

1.4 Image compression with K-means

Instruction:
In this exercise, you will apply K-means to image compression. In a straightforward 24-bit color representation of an image, 2 each pixel is represented as three 8-bit unsigned integers ranging from 0 to 255) that specify the red, green and blue intensity values. This encoding is often referred to as the RGB encoding. Our image contains thousands of colors, and **in this part of the exercise, you will reduce the number of colors to 16 colors.**

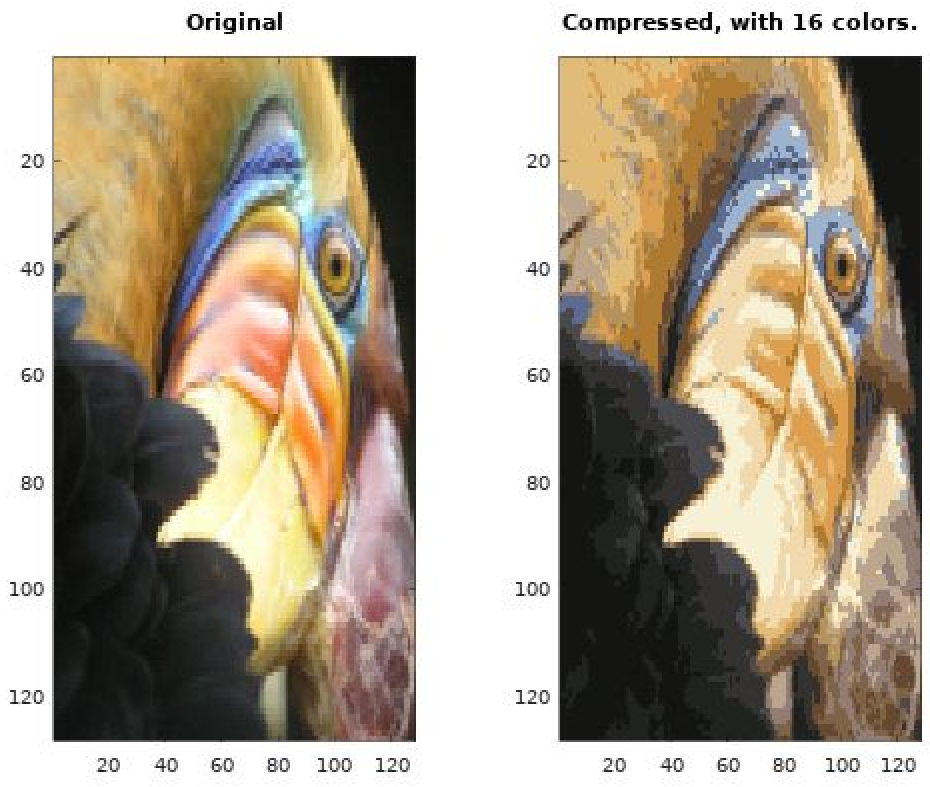
By making this reduction, it is possible to represent (compress) the photo in an efficient way. Specifically, you only need to store the RGB values of the 16 selected colors, and for each pixel in the image you now need to only store the index of the color at that location (where only 4 bits are necessary to represent 16 possibilities).

In this exercise, you will use the K-means algorithm to select the 16 colors that will be used to represent the compressed image. Concretely, you will treat every pixel in the original image as a data example and use the K-means algorithm to find the 16 colors that best group (cluster) the pixels in the 3-dimensional RGB space. Once you have computed the cluster centroids on the image, you will then use the 16 colors to replace the pixels in the original image.

1.4.1 K-means on pixels

The steps are as follows:
1. first loads the image
2. reshapes it to create an m × 3 matrix of pixel colors (where m = 16384 = 128 × 128)
3. Calls your K-means function on it
4. After finding the top K = 16 colors to represent the image, you can now assign each pixel position to its closest centroid using the findClosestCentroidsfunction.
5. Finally, you can view the effects of the compression by reconstructing the image based only on the centroid assignments.

Remarks:
- Notice that you have significantly reduced the number of bits that are required to describe the image.
    - The original image required 24 bits for each one of the 128×128 pixel locations, resulting in a total size of 128 × 128 × 24 = 393, 216 bits.
    - The new representation requires some overhead storage in form of a dictionary of 16 colors, each of which require 24 bits, but the image itself then only requires 4 bits per pixel location. The final number of bits used is therefore 16 × 24 + 128 × 128 × 4 = 65, 920 bits, which corresponds to compressing the original image by about a factor of 6.



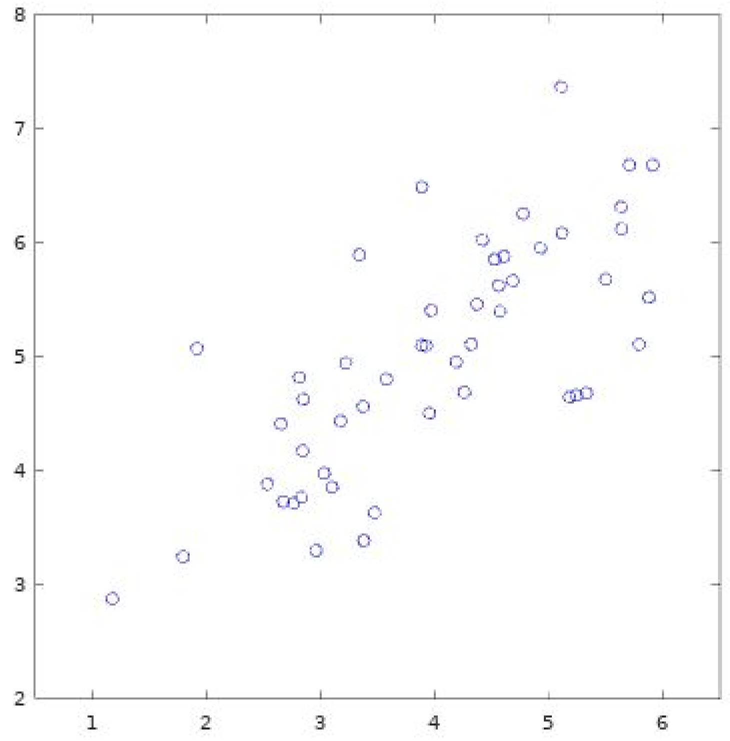
2 Principal Component Analysis

Instruction:
In this exercise, you will use principal component analysis (PCA) to perform dimensionality reduction. You 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 datasets.

2.1 Example Dataset

Instruction:
To help you understand how PCA works, you will first start with a 2D dataset which has one direction of large variation and one of smaller variation.

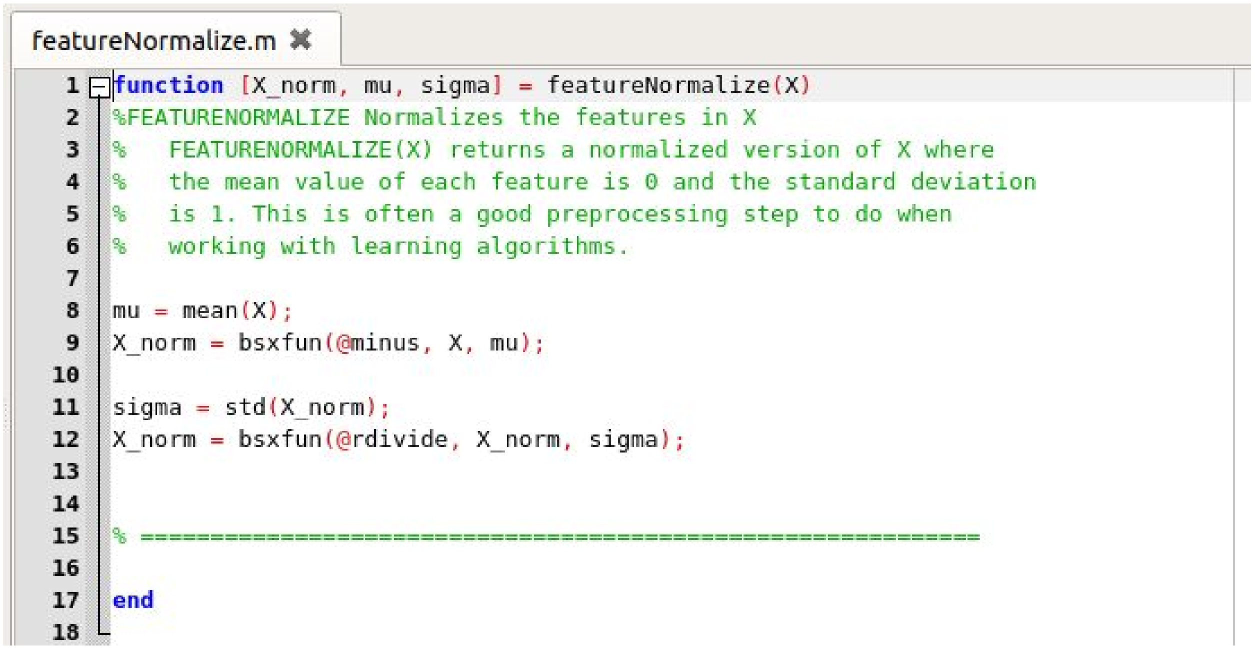
In this part of the exercise, you will visualize what happens when you use PCA to reduce the data from 2D to 1D. In practice, you might want to reduce data from 256 to 50 dimensions, say; but using lower dimensional data in this example allows us to visualize the algorithms better.



2.2 Implementing PCA

Instruction:
In this part of the exercise, you will implement PCA.

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 provided script ex7 pca.m, this normalization has been performed for you using the featureNormalize function.


featureNormalize.m

```
function [X_norm, mu, sigma] = featureNormalize(X)
%FEATURENORMALIZE Normalizes the features in X
%   FEATURENORMALIZE(X) returns a normalized version of X where
%   the mean value of each feature is 0 and the standard deviation
%   is 1. This is often a good preprocessing step to do when
%   working with learning algorithms.

mu = mean(X);
X_norm = bsxfun(@minus, X, mu);

sigma = std(X_norm);
X_norm = bsxfun(@rdivide, X_norm, sigma);


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

end

```

PCA consists of two computational steps:

1. compute the covariance matrix of the data

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

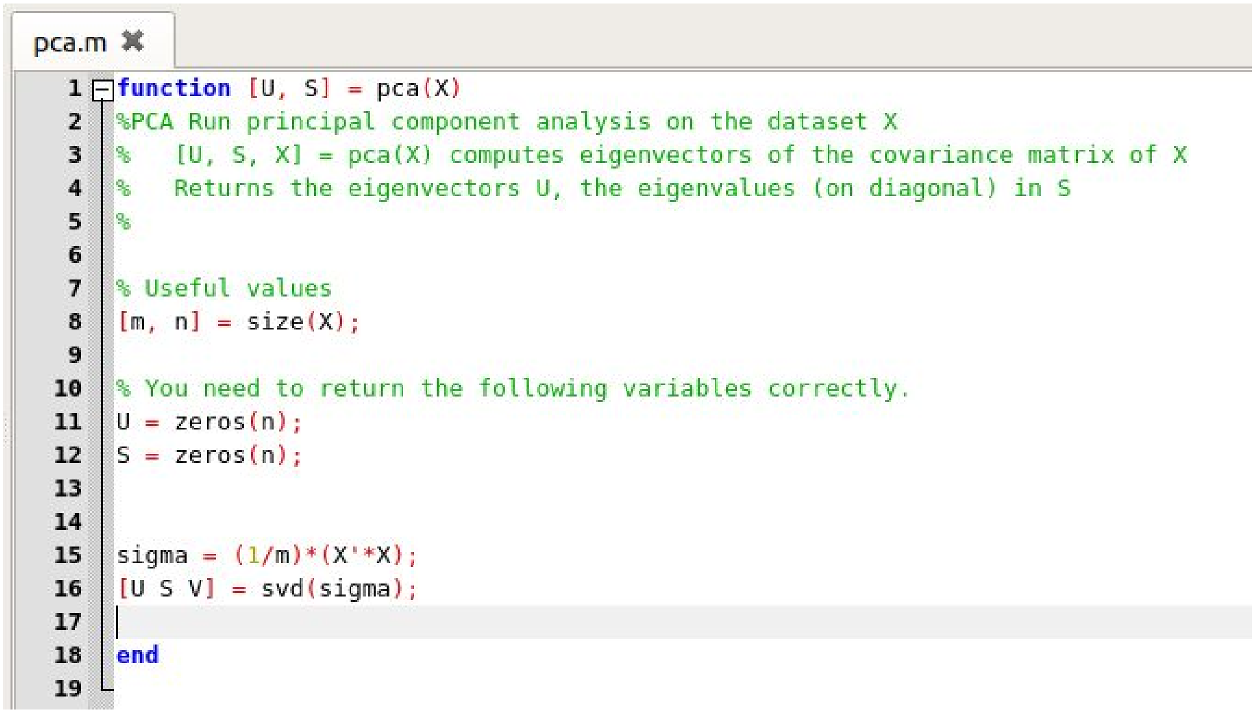
   Where

   X is the data matrix with examples in rows
   m is the number of examples

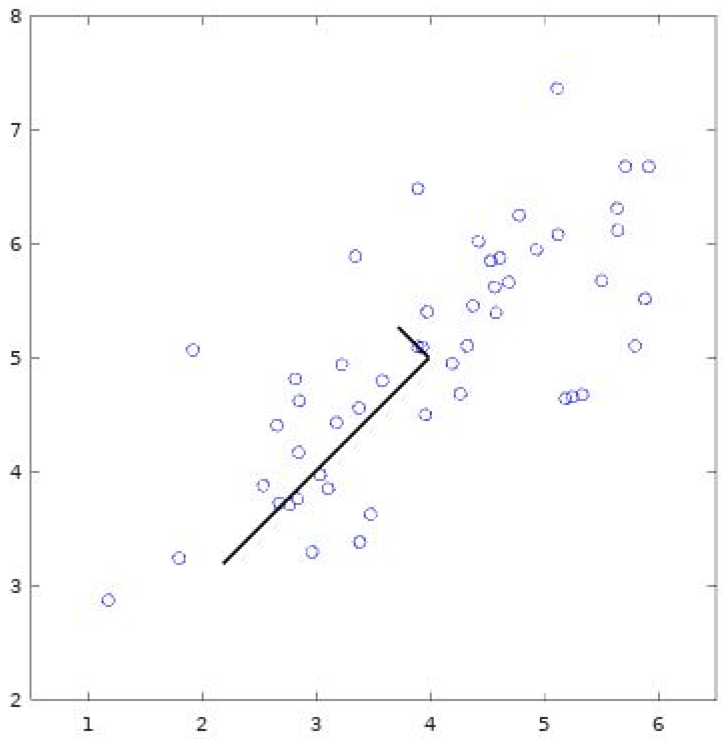
2. you use Octave/MATLAB's SVD function to compute the eigenvectors U_1 , U_2, . . . , U_n . These will correspond to the principal components of variation in the data.

Your task is to complete the code in pca.m to compute the principal components of the dataset.

Implementation:



```
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);


sigma = (1/m)*(X'*X);
[U S V] = svd(sigma);

end

```



2.3 Dimensionality Reduction with PCA

Instruction:

After computing the principal components, you can use them to reduce the feature dimension of your dataset by projecting each example onto a lower dimensional space, x (i) → z (i) (e.g., projecting the data from 2D to 1D). In this part of the exercise, you will use the eigenvectors returned by PCA and project the example dataset into a 1-dimensional space.

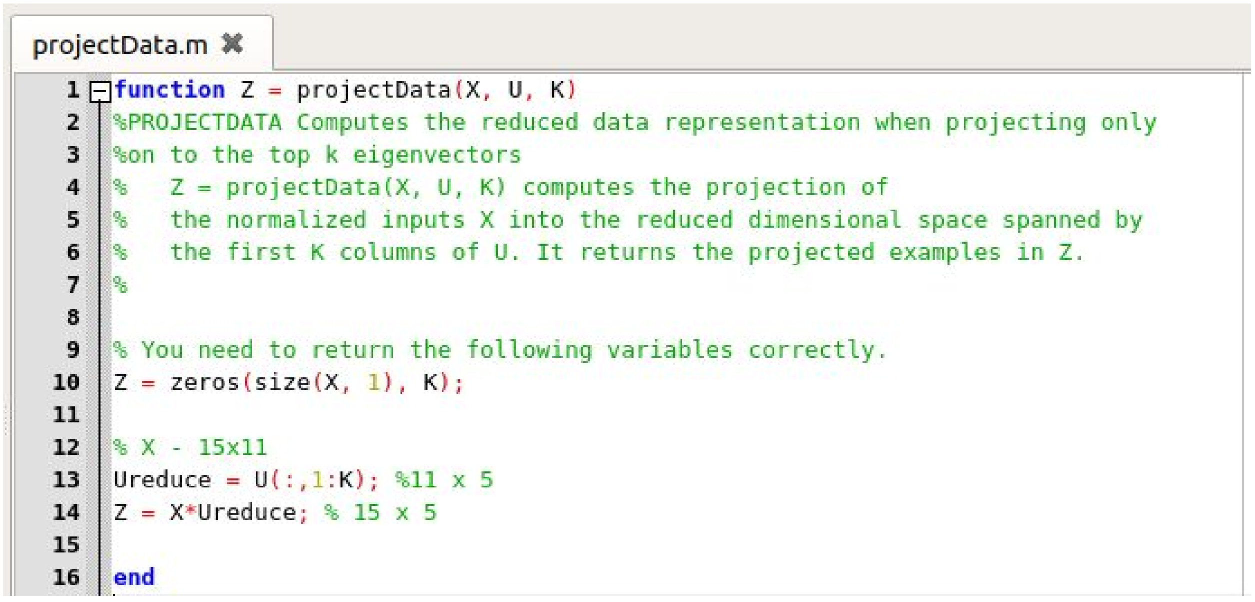
Remark:
In practice, if you were using a learning algorithm such as linear regression or perhaps neural networks, you could now use the projected data instead of the original data. By using the projected data, you can train your model faster as there are less dimensions in the input.

2.3.1 Projecting the data onto the principal components

Instruction:
You should now complete the code in projectData.m. Specifically, you are given a dataset X, the principal components U, and the desired number of dimensions to reduce to K. You 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 you have completed the code in projectData.m, ex7 pca.m will project the first example onto the first dimension.

Implementation:



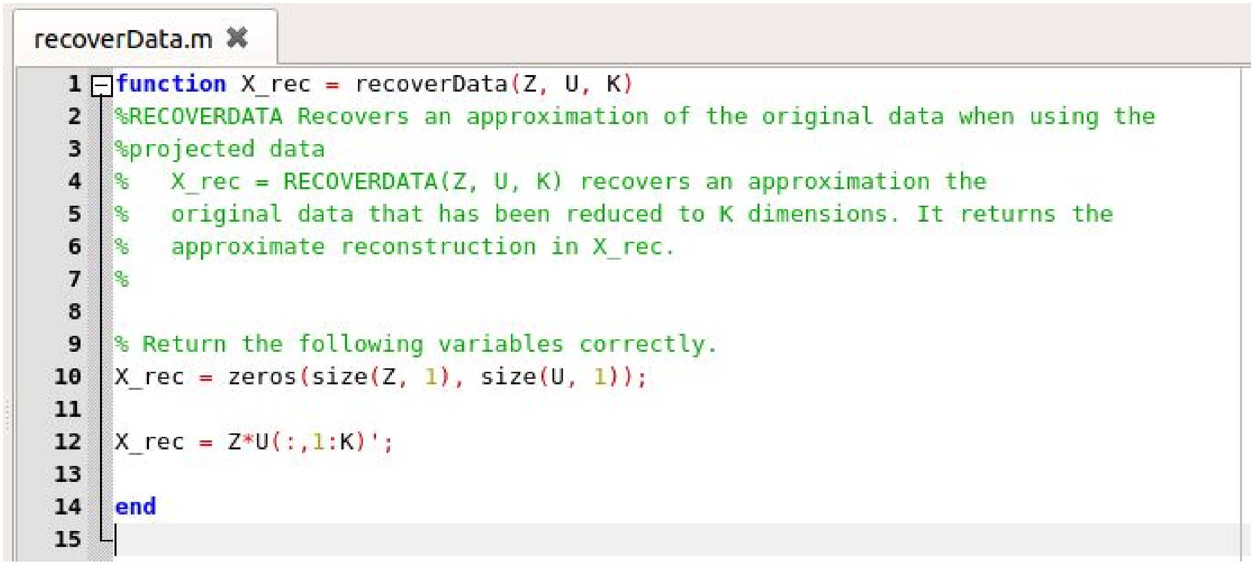
```
function Z = projectData(X, U, K)
%PROJECTDATA Computes the reduced data representation when projecting only
%on to the top k eigenvectors
%   Z = projectData(X, U, K) computes the projection of
%   the normalized inputs X into the reduced dimensional space spanned by
%   the first K columns of U.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

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

% X - 15x11
Ureduce = U(:,1:K); %11 x 5
Z = X*Ureduce; % 15 x 5

end
```

2.3.2 Reconstructing an approximation of the data

Instruction:

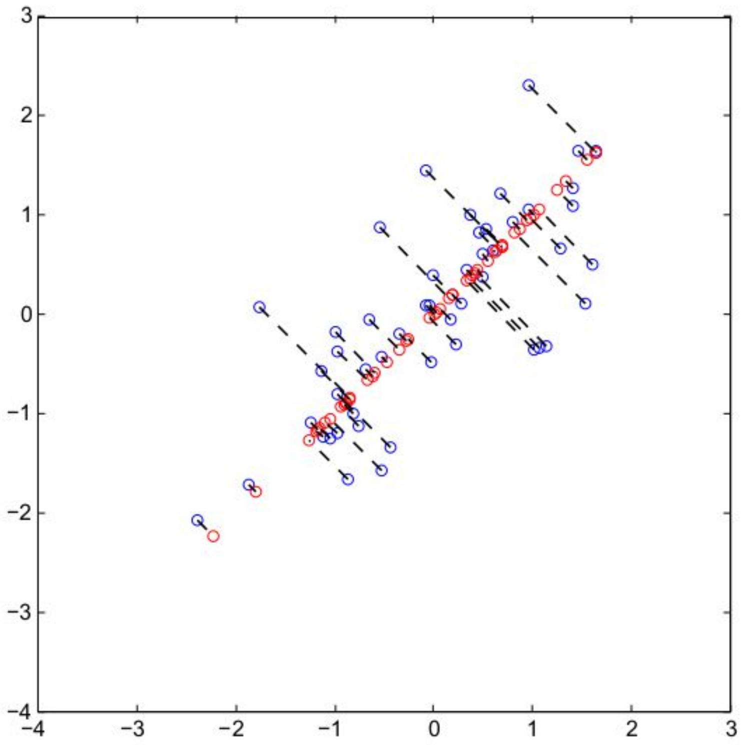
After projecting the data onto the lower dimensional space, you can approximately recover the data by projecting them back onto the original high dimensional space. Your task is to complete recoverData.m to project each example in Z back onto the original space and return the recovered approximation in X rec.

Implementation:



```
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.
%

% Return the following variables correctly.
X_rec = zeros(size(Z, 1), size(U, 1));

X_rec = Z*U(:,1:K)';

end

```

2.3.3 Visualizing the projections



The original data points are indicated with the blue circles, while the projected data points are indicated with the red circles. The projection effectively only retains the information in the direction given by U_1 .

2.4 Face Image Dataset

Instruction:
In this part of the exercise, you will run PCA on face images to see how it can be used in practice for dimension reduction. The dataset ex7faces.mat contains a dataset 3 X of face images, each 32 × 32 in grayscale. Each row of X corresponds to one face image (a row vector of length 1024).

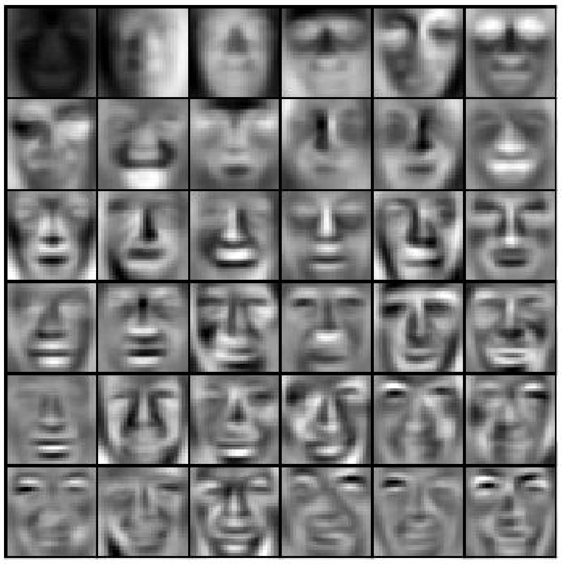
2.4.1 PCA on Faces

To run PCA on the face dataset,

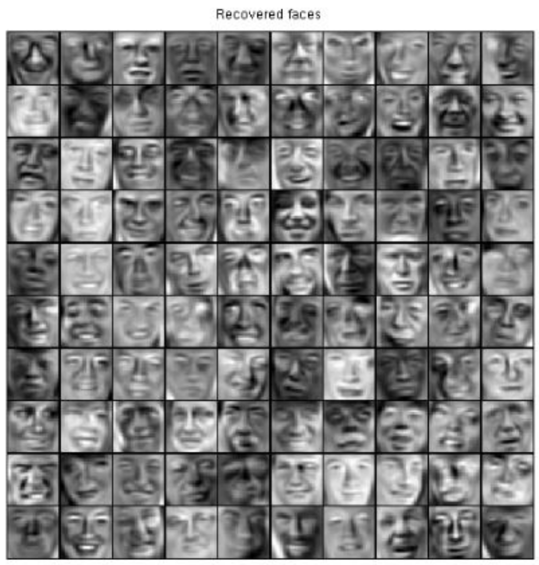
1. normalize the dataset by subtracting the mean of each feature from the data matrix X
2. After running PCA, you will obtain the principal components of the dataset.
   Notice that each principal component in U (each row) is a vector of length n (where for the face dataset, n = 1024).
3. It turns out that we can visualize these principal components by reshaping each of them into a 32 × 32 matrix that corresponds to the pixels in the original dataset.

2.4.2 Dimensionality Reduction

Instruction:
Now that you have computed the principal components for the face dataset, you can use it to reduce the dimension of the face dataset. This allows you to use your learning algorithm with a smaller input size (e.g., 100 dimensions) instead of the original 1024 dimensions. This can help speed up your learning algorithm.

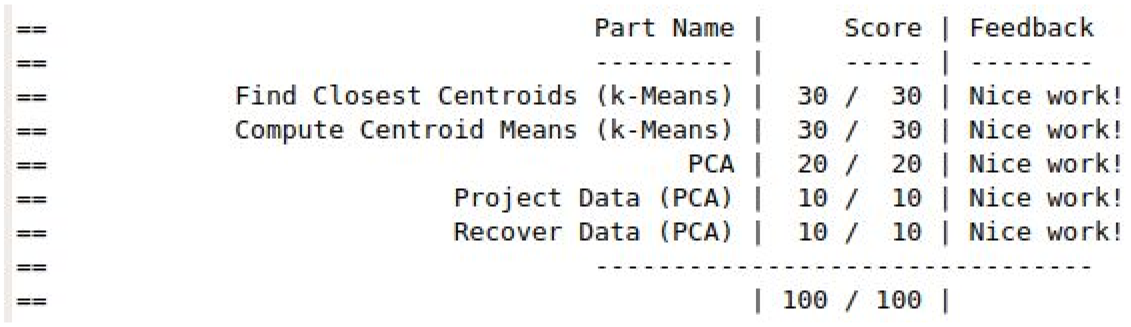


Principal Components



From the reconstruction, you can observe that the general structure and appearance of the face are kept while the fine details are lost. This is a remarkable reduction (more than 10×) in the dataset size that can help speed up your learning algorithm significantly. For example, if you were training a neural network to perform person recognition (given a face image, predict the identity of the person), you can use the dimension reduced input of only a 100 dimensions instead of the original pixels.

3 Scores



```
==                                   Part Name |     Score | Feedback
==                                   --------- |     ----- | --------
==            Find Closest Centroids (k-Means) |  30 /  30 | Nice work!
==            Compute Centroid Means (k-Means) |  30 /  30 | Nice work!
==                                         PCA |  20 /  20 | Nice work!
==                          Project Data (PCA) |  10 /  10 | Nice work!
==                          Recover Data (PCA) |  10 /  10 | Nice work!
==                                   --------------------------------
==                                             | 100 / 100 |
```