ex7 K均值聚类与主成分分析

介绍：

本练习中需要实现K均值聚类算法并应用于图片压缩，在第二部分的练习中我们将要通过主成分分析来找到面部图像的低维度表示，

第一部分 K均值聚类

实现K均值聚类算法并利用其进行图片压缩，首先从一个二维数据集开始，对K均值聚类算法的运行有一个直观印象。接着使用K均值算法进行图像压缩，通过较少图像中出现的颜色的数量，只保留那些图像中常见的颜色。使用Ex7,m文件完成该部分的练习。

1.1 实现K均值算法

K均值算法是一个能够自动将相似的样本数据进行归类的算法，具体来说，给你一个n维m个样本的训练数据集 {x (1) ,...,x (m) }，想要将这些数据归并到几个紧密相关的族中。从猜测初始的几何中心，然后通过重复将样本分配到最近的几何中心来改善自己的猜测，接着再基于前面的样本分配重新计算几何中心，K均值算法就是这样的一个迭代的过程，具体算法如下图所示：

```
% 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
```

算法的内部循环重复以下两个步骤：（1）将每一个训练样本分配到离它最近的几何中心，（2）通过分配的点再次计算每个集合中心的均值。K均值算法必定会收敛于几何中心的最后一组均值。注意这种收敛方法并不总是理想的，与初始几何中心的设定有很大的关系。因此在实践过程中常会使用几组不同的随机初始化几何中心进行运行，最终选择损失函数最低的方案。在下一节中，分别实现K均值算法的两个阶段。

1.1.1 寻找最近的几何中心

在K均值算法的族分配部分，算法将每一个训练样本 x (i) 分配给离它最近的几何中心（当前几何中心的位置已知）。特别地，我们对于每一个样本，设置：

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

c (i)即最接近x (i)的中心点的索引， μ j即第j个中心点的位置（值），注意在初始化代码部分，c (i)与idx(i)相对应。

我们需要完成在findCloestCentroids.m函数，该函数输入数据矩阵为X，在centroids中的所有中心点的位置信息，需要输出一个一维数组idx，对应于每个样本的最近的中心点的索引（均属于1-K之间的数字）。在每个样本和每个中心点上使用循环。实现该函数后，运行ex7.m文件输出[1 3 2]，表示最前面三个样本的中心点分配。

代码：

```
for i=1:size(X,1)
  for j=1:K
    distance(j)=sum((X(i,:)-centroids(j,:)).^2);
  endfor
  [closest_distance,index]=min(distance);
  idx(i)=index;
endfor
```

运行结果：

```
Closest centroids for the first 3 examples:
 1 3 2
(the closest centroids should be 1, 3, 2 respectively)
```

首次提交：

```
                        Part Name |     Score | Feedback
                        --------- |     ----- | --------
  Find Closest Centroids (k-Means) |  30 /  30 | Nice work!
  Compute Centroid Means (k-Means) |   0 /  30 |
                               PCA |   0 /  20 |
             Project Data (PCA) |   0 /  10 |
             Recover Data (PCA) |   0 /  10 |
                                  ---------------------------------
                                  |  30 / 100 |
```

1.1.2 计算中心点均值

将每个样本点分配给一个中心点之后，算法的第二部分开始计算，对每个中心点，计算分配给它的样本点的均值，特别地，对于每一个中心点k，我们设

置：

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

C k是分配给中心点k的一组样本，具体地说，如果取两个样本 x (3) 和 x (5)，分配给中心点k=2，你应该更新：

$$\mu_2 = \frac{1}{2}(x^{(3)} + x^{(5)})$$

现在完成computeCentroids.m文件，你可以对中心点和每个样本点使用循环，但是如果进行向量化的运，代码会运行地更快。此时运行ex7.m文件输出新的中心点。

代码：

```
for i=1:K
  index=find(idx==i);
  centroids(i,:)=mean(X(index,:));
endfor
```

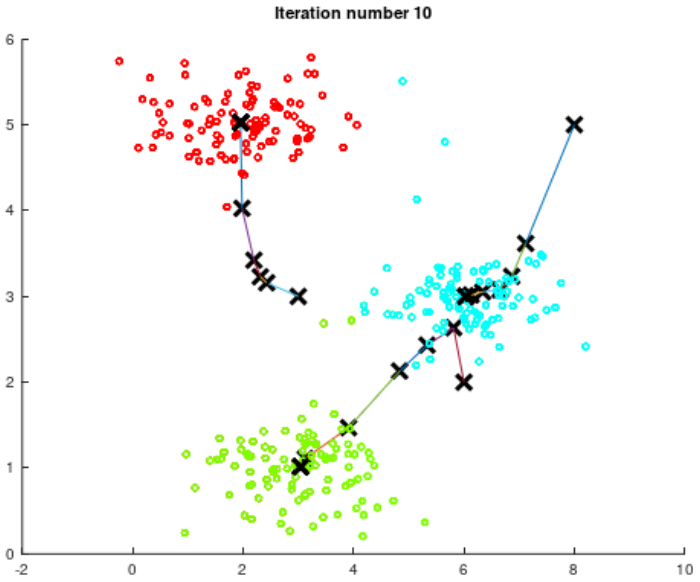
运行结果：

```
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 ]
```

第二次提交：

```
                        Part Name |     Score | Feedback
                        --------- |     ----- | --------
  Find Closest Centroids (k-Means) |  30 /  30 | Nice work!
  Compute Centroid Means (k-Means) |  30 /  30 | Nice work!
                               PCA |   0 /  20 |
                 Project Data (PCA) |   0 /  10 |
                 Recover Data (PCA) |   0 /  10 |
                                    ----------------------------------
                                    |  60 / 100 |
```

1.2 将K均值算法运用于样本数据



运行ex7.m文件使用K均值算法对样本数据集进行10次迭代运算，结果如上图所示。这里调用了runKmeans.m的脚本，查阅该脚本的代码了解其运行方式。

1.3 随机初始化

前面练习中用于进行初始分配的中心是给定的，这样才能看到Figure1中的结果，在实际操作过程中，从训练集中随机选择样本点作为初始化中心点是一个很棒的策略。在本次练习中，将以下代码填入 kMeansInitCentroids.m函数：

```
% 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), :);
```

上述代码使用randperm变更样本数据的额索引，然后取前k个样本，这保证了样本的随机性且无重复。

1.4 使用K均值算法进行图像压缩

在一张用简单的24-bits颜色表示的图像中，每个像素均用3个指定红色、绿色和蓝色的强度值的8-bits无符号整数（0-255）表示，常称为RGB编码，我们的图像包含数千种颜色，在本次练习中，我们需要将颜色数量降到16种。

通过这种简化可以有效地对图像进行压缩，你只需要存储选定的16中颜色的RGB的值，对于图像中的每个像素，现在只需要在该位置存储颜色的索引(在该位置上只需要4bits来表示16种可能性)。

在本次练习中，你将会通过K均值的算法选择用来表示压缩图片的16种颜色，具体来说，我们需要将原始图像中的每一个像素看成是一个数据样本，通过K均值算法找到能够将

像素点在RGB三维空间中最优分组的16种颜色。一旦计算出图像上的聚类中心，便可以使用16种颜色替换原始图像上的像素了。

1.4.1 对像素点使用K均值算法

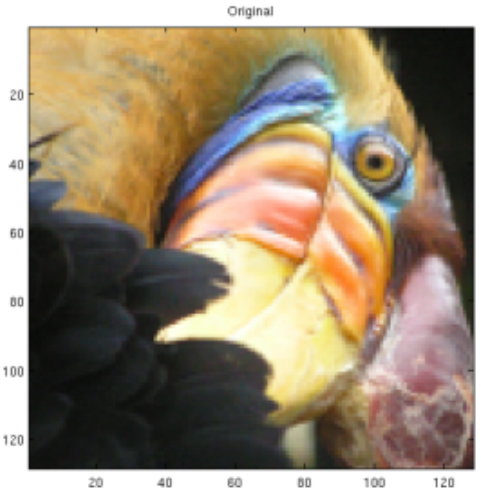
用如下代码读取图像信息：

```
% Load 128x128 color image (bird_small.png)
A = imread('bird_small.png');

% You will need to have installed the image package to used
% imread. If you do not have the image package installed, you
% should instead change the following line to
%
%   load('bird_small.mat'); % Loads the image into the variable A
```

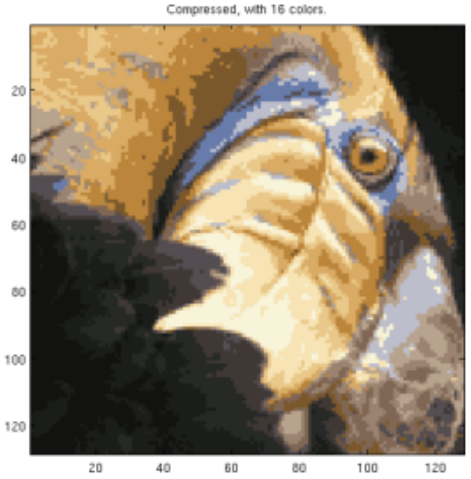
上述代码创建了一个三维矩阵A，前两个索引表示像素点的位置信息，最后一个索引表示红绿蓝信息。比如 A(50, 33, 3)表示在50行33列的像素点的蓝色的强度。

运行ex7.m文件会首先加载图像信息，然后将其转换成16384*3（128*128=1638即像素点的个数，三列分别指的是RGB值）的像素颜色矩阵，接着对其调用K均值算法函数，在找到表征图像信息的16种颜色之后，可以使用findClosestCentroids.m函数将每个像素点位置分配给其最近的中心点，这使得我们能够使用分配给每个像素点的质心来表示原始图像。注意此时我们已经大大减少了描述原始图像所需的比特数。原始图像中128*128像素点位置中的每一个都需要24bits（每个像素点均看成是一种颜色），这就导致共需要128*128*24=393216bits，新的表示方法只需要以16种颜色的字典的形式进行一些开销存储，每种颜色需要24bits，但图像的每一个像素点位置仅需要4bits，最终总计需要的数量为16 * 24 + 128 * 128 * 4 = 65920 bits。相当于将原始图像压缩为1/6。

原始图像：



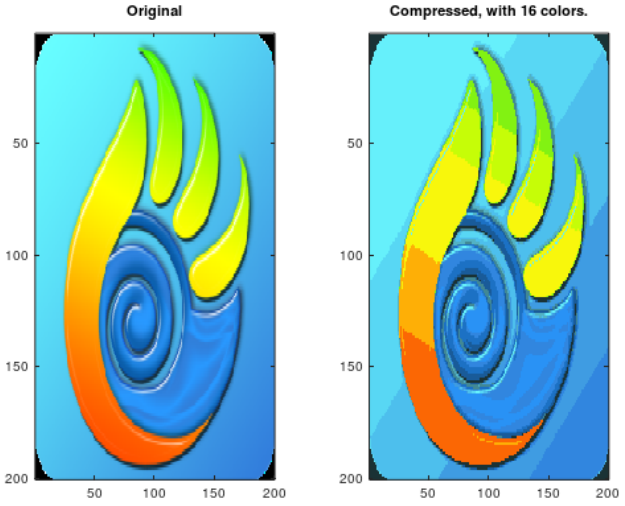
重构图像：



最终我们可以看到仅仅基于中心点分配的图像压缩的效果，特别地，我们可以使用分配给每个像素点的质心的均值替换像素点的位置，尽管重建的图像保留了原始图像的大部分特征，我们还是能看到一些人工压缩的痕迹。

1.5 可选练习:使用自己的图片

调整代码运行自己的图片进行图像压缩，如果图像很大，K均值算法将会运行较长的时间，推荐使用较小尺寸的图片，同时可以调整K的值来观察不同的压缩效果。



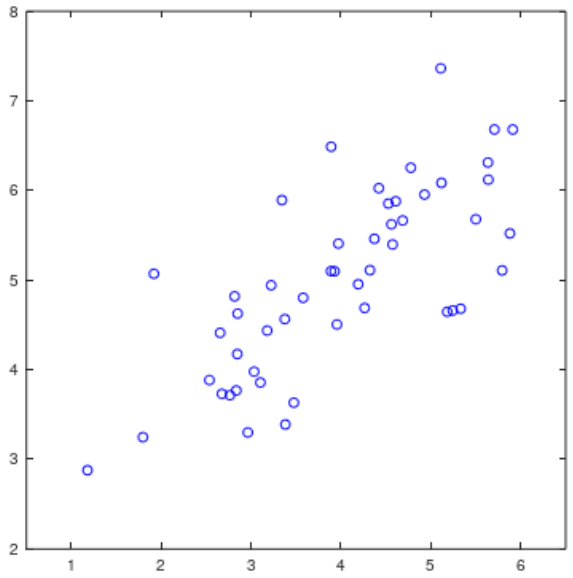
第二部分 主成分分析

在本次练习中，你将使用主成分分析（PCA）来进行降维，我们首先对2D样本数据进行试验来了解PCA的运行机制，然后将其应用于5000个面部图像数据集。使用ex7_pca.m文件。

2.1 样本数据

为了理解PCA是如何进行工作的，我们先从一个2维数据集开始，该数据集的一个方向是大的变化，一个方向是小的变化。运行ex7_pca.m将训练集数据显示出来。在本部分的练习中需要将使用PCA将数据从2维降到1维时发生的变化进行可视化。虽然在实际操作中，

可能需要将256维降到50维，但是在本例中，使用更低维度的数据可以使我们对算法进行可视化并加深理解。



2.2 实现PCA算法

在本练习中我们需要实现PCA算法，其包含了两个计算步骤：首先计算数据的协方差矩阵；接着使用Octave的SVD奇异值分解函数来求得特征向量U1、U2、U3...Un。这些相当于数据中变化的主成分。

再使用PCA之前，对数据进行标准化很重要，方法是从数据集中减去每个特征的平均值，并对每个维度进行缩放，使它们处于相同的范围内。在ex7_pca.m中，标准化已经通过featureNormalize.m文件完成了。

之后就可以运行PCA来获取主成分，我们需要完成pca.m文件中的代码来计算数据集的主成分，首先需要计算数据的协方差矩阵，通过以下公式求得：

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

X是每一行为一个样本的数据矩阵，m是样本的个数，注意∑是一个n*n的矩阵，不是求和操作符。求完协方差矩阵，对其运行SVD函数来求主成分，使用[U S V]=svd(Sigma)即可，U包含了全部的主成分，而S则包含一个对角线矩阵。

完成pca.m文件后，ex7_pca.m将会在样本数据集上运行PCA，打印出相应找到的主成分，该脚本也会输出top的主成分（特征向量），期望输出为[-0.707,-0.707]。

代码：

```
Sigma=X'*X/m;
[U S V]=svd(Sigma)
```

运行结果：

```
Running PCA on example dataset.

U =

  -0.70711  -0.70711
  -0.70711   0.70711

S =

Diagonal Matrix

   1.70082         0
         0   0.25918

V =

  -0.70711  -0.70711
  -0.70711   0.70711

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

(you should expect to see -0.707107 -0.707107)
```

第三次提交：

```
                        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) |   0 /  10 |
               Recover Data (PCA) |   0 /  10 |
                                  ---------------------------------
                                  |  80 / 100 |
```

2.3 使用PCA进行降维

求完主成分后，可以通过将每个样本投影到低维度空间上来降低你的数据集的特征维度，

$x^{(i)} \rightarrow z^{(i)}$

在本部分的练习中，你将使用PCA返回的特征向量并且将样本数据集投影到1维空间中。

在实际操作过程中，如果你使用一个线性回归或者神经网络的算法，现在就可以使用投影的数据而不是原始数据。通过使用投影数据可以加快模型训练的速度，因为输入的维度变低了。

2.3.1 将数据投影到主成分上

完成projectData.m文件，确切地说，给你一个数据集X，主成分U，以及想要的维度的数量降低到K，你需要将X中的每一个样本投影到U中前K个成分上去，注意U中的前K个成分为U矩阵的前K列，U_reduce=U(:,1:K)。一旦完成projectData.m文件中的代码，ex7_pca.m将会将第一个样本投影到第一个维度上，你将看到一个大约是-1.481/1.481的数字。

代码：

U_reduce=U(:,1:K)

Z=X*U_reduce;

运行结果：

```
Projection of the first example: 1.481274

(this value should be about 1.481274)
```

第四次提交：

```
                        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) |   0 /  10 |
                                  --------------------------------
                                  |  90 / 100 |
```

2.3.2 对数据的大约值的重现

此时通过将数据投影回原来的高维度空间可以对数据进行大概的还原，我们的任务是完成recoverData.m文件，将z上的每个样本投影到原来的空间，返回大概的恢复数据X_rec，一旦完成该文件，运行ex7_pca.m便可以看到大约 [-1.047 -1.047]。

代码：

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

运行结果：

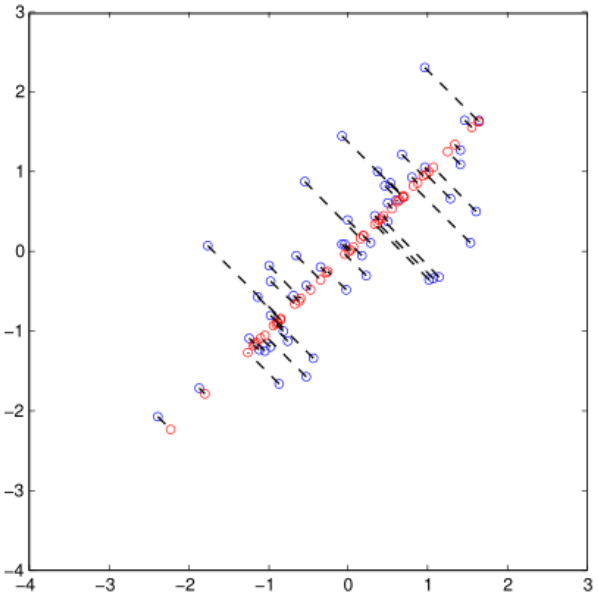
```
Approximation of the first example: -1.047419 -1.047419

(this value should be about  -1.047419 -1.047419)
```

第五次提交：

```
                        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 |
```

2.3.3 投影可视化

接下来 ex7_pca.m将会同时执行投影和近似重现来展现投影是如何影响到数据的。



2.4 面部图像数据集

本练习中将会在面部图像上运行PCA，看看在实践过程中是如何被用于降维的，数据集ex7faces.mat包含了面部图像数据集X，每张均为32*32的灰度图像，X的每一行对应一张面部图像，行向量长度1024， ex7_pca.m中的下一步将会加载并可视化前100张面部图像。

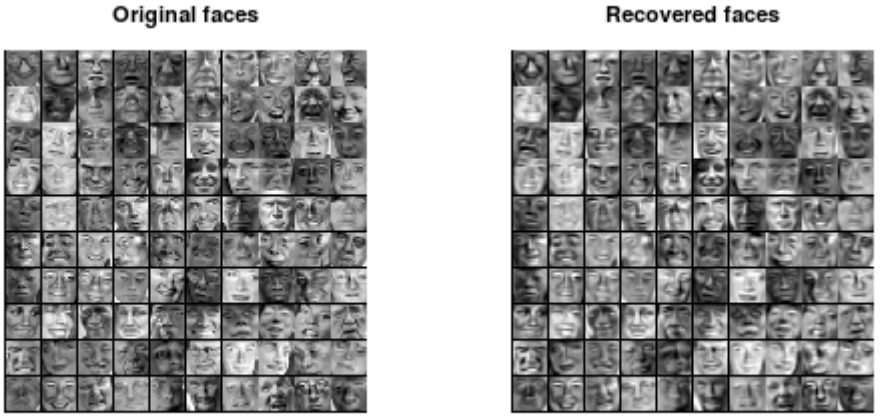
2.4.1 对面部图像使用PCA

为了在面部数据集上使用PCA，我们首先通过求出X矩阵每一个特征的均值对数据集进行标准化，ex7_pca.m会完成这一步，然后其就会运行PCA代码，之后你将获得数据集的主成分，注意U中的每一个主成分（行向量），是一个长度为n的向量（这里n为1024）。通过将它们每一个转换成对应于原始图像中像素的32*32的矩阵，我们就可以对主成分进行可视化。ex7 pca.m显示了描述最大变化的前36个主成分。通过修改代码显示更多主成分能够观察到主成分是如何抓取越来越多的细节的。

2.4.1 降维

在求出面部数据的主成分之后，便可以利用它来为面部数据进行降维了，使得学习算法的输入规模小上许多（比如100维），而不是初始的1024维度，这样能够为你的算法进行

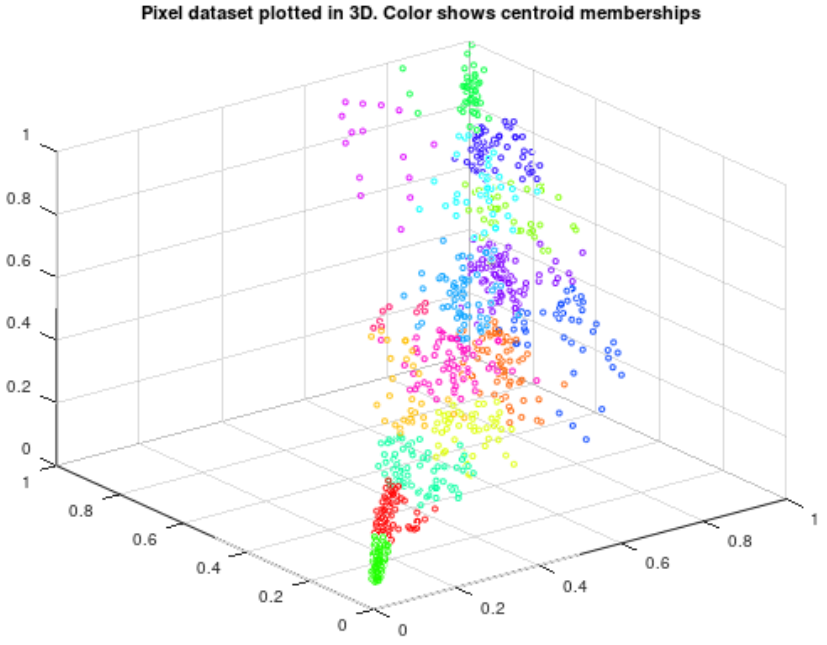
加速学习。ex7_pca.m文件将会将面部数据投影到前100个主成分上，每张图像均被描述成一个100维的z（i）向量。在降维的过程中，什么是被舍弃的？ex7_pca.m中我们使用投影后的数据集进行数据还原，观察原始图像和投影的图像比较不同可以发现，面部的整体结构和外观被保留，但是细节被舍弃了。



这种在数据集规模上进行超过10倍的降维能极大地加速你的算法，比如如果你正在训练一个用于人脸识别的神经网络算法，你就可以使用100维的输入而不是使用原始像素的图像数据。

2.5 可选练习：PCA之于可视化

在前面的K均值图像压缩练习中，我们在三维RGB空间中使用了K均值算法，而在ex7_pca.m脚本中我们使用scatter3函数对在该空间中的最终的像素分配进行可视化，每个数据点的颜色与它所分到的质心相关。在三维或者更高维度空间中对数据集进行可视化比较麻烦，通常选择以牺牲部分信息为代价而将数据只是显示在2维空间中。在实践过程中，PCA对数据进行降维常常是处于可视化的目的。



运行ex7_pca.m应用PCA将三维数据降至二维并显示成散点图，PCA投影可以理解为是为了最大化观察数据分布而进行的视角的旋转，总是呈现最佳视角。