Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ЛАБОРАТОРНАЯ РАБОТА №7

«Метод главных компонент»

Студент А. Ю. Омельчук

Преподаватель М. В. Стержанов

Минск 2019

ХОД РАБОТЫ

Задание.

Набор данных ex7data1.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит две переменные X1 и X2 - координаты точек, для которых необходимо выделить главные компоненты.

Набор данных ex7faces.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит 5000 изображений 32x32 в оттенках серого. Каждый пиксель представляет собой значение яркости (вещественное число). Каждое изображение сохранено в виде вектора из 1024 элементов. В результате загрузки набора данных должна быть получена матрица 5000x1024.

1. Загрузите данные ex7data1.mat из файла.
2. Постройте график загруженного набора данных.
3. Реализуйте функцию вычисления матрицы ковариации данных.
4. Вычислите координаты собственных векторов для набора данных с помощью сингулярного разложения матрицы ковариации (разрешается использовать библиотечные реализации матричных разложений).
5. Постройте на графике из пункта 2 собственные векторы матрицы ковариации.
6. Реализуйте функцию проекции из пространства большей размерности в пространство меньшей размерности с помощью метода главных компонент.
7. Реализуйте функцию вычисления обратного преобразования.
8. Постройте график исходных точек и их проекций на пространство меньшей размерности (с линиями проекций).
9. Загрузите данные ex7faces.mat из файла.
10. Визуализируйте 100 случайных изображений из набора данных.
11. С помощью метода главных компонент вычислите собственные векторы.
12. Визуализируйте 36 главных компонент с наибольшей дисперсией.
13. Как изменилось качество выбранных изображений?
14. Визуализируйте 100 главных компонент с наибольшей дисперсией.
15. Как изменилось качество выбранных изображений?
16. Используйте изображение, сжатое в лабораторной работе №6 (Кластеризация).
17. С помощью метода главных компонент визуализируйте данное изображение в 3D и 2D.
18. Соответствует ли 2D изображение какой-либо из проекций в 3D?

Результат выполнения:

1. Код выгрузки данных из файла представлен ниже:

```
file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex7data1.mat')
dataset = sio.loadmat(file_path)
X = dataset["X"]
```

Но здесь стоит отметить, что при визуализации начальных данных я заметил, что те данные, которые даны в этой лабораторной, используются Andrew Ng для K-Means, а те данные, которые Andrew Ng использует в PCA – даны нам в K-Means. Т. е. по сути данные с двух работ перемешаны между собой. И если данные, которые Andrew Ng использует для PCA ещё можно использовать в задаче кластеризации, то данные которые были в кластеризации использовать в PCA кажется немного неразумным, поскольку там будет довольно-таки большая ошибка, и данный метод будет работать немного некорректно, и как следствие, графики будут не такими, какими мы их ожидаем увидеть. Поэтому данные, которые были даны к этим лабораторным я опять поменял местами. Т. е. я использую такие же данные, который использует Andrew Ng. Но в любом случае эти данные можно поменять местами и посмотреть на новые графики. Код менять в этом случае не придётся.

2. Код реализации:

```
# X[:, 0] - first column
plt.scatter(X[:, 0], X[:, 1], marker="o")
plt.show()
```

Результат выполнения:

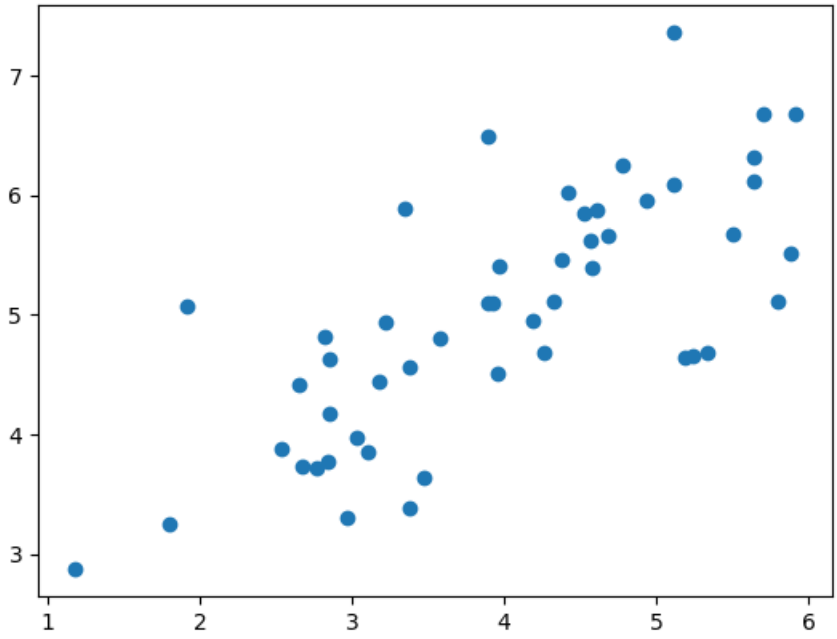

Рисунок 1 – визуализация исходных данных

3. Код реализации:

```
def compute_covariance_matrix(X):
    return X.T.dot(X) / X.shape[0]

covariance_matrix = compute_covariance_matrix(X)
print(covariance_matrix)
```

Результат выполнения:

```
[[17.26276267 20.82286988]
 [20.82286988 26.05448259]]
```

4. Код реализации:

```
def feature_normalize(X):
    means = np.mean(X, axis=0)
    X_norm = X - means
    stds = np.std(X_norm, axis=0)
    X_norm = X_norm / stds

    return means, stds, X_norm

def pca(X):
    covariance_matrix = compute_covariance_matrix(X)
    U, S, V = svd(covariance_matrix, full_matrices=True, compute_uv=True)

    return U, S

# Feature normalize
# mu, sigma
```

```
means, stds, X_norm = feature_normalize(X)
# Run SVD
U, S = pca(X_norm)
print(U, S)
```

Результат выполнения:

```
(array([[-0.70710678, -0.70710678],
       [-0.70710678,  0.70710678]]), array([1.73553038, 0.26446962]))
```

5. График приведён ниже:

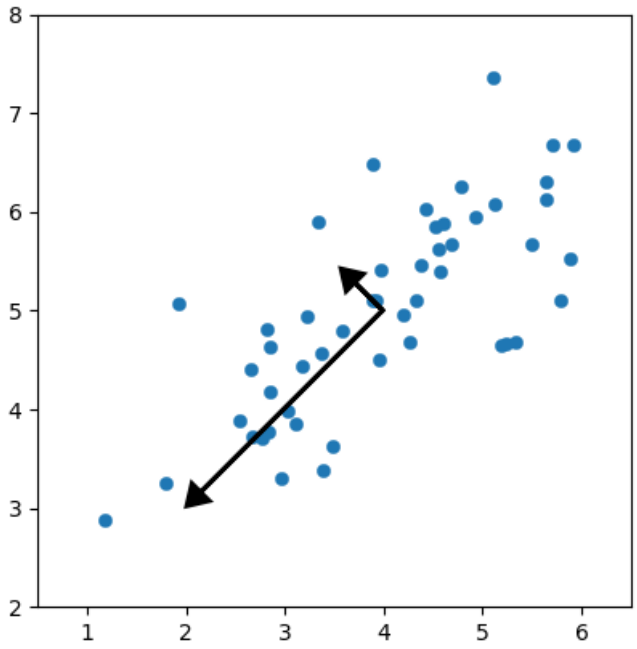

Рисунок 2 – собственные векторы матрицы ковариации

6. Код реализации:

```
# Project the data onto K = 1 dimension
K = 1
Z = project_data(X_norm, U, K)
print('Projection of the first example: {:.6f}'.format(Z[0, 0]))
print('(this value should be about    : 1.481274)')
```

Результат выполнения:

```
Projection of the first example: 1.496313
(this value should be about    : 1.481274)
```

7. Код реализации:

```
        X_rec = recover_data(Z, U, K)
        print('Approximation of the first example: [{:.6f} {:.6f}]'.format(X_rec[0, 0], X_rec[0,
1]))
        print('       (this value should be about  [-1.047419 -1.047419])')
```

Результат выполнения:

```
Approximation of the first example: [-1.058053 -1.058053]
       (this value should be about  [-1.047419 -1.047419])
```

8. График представлен ниже:

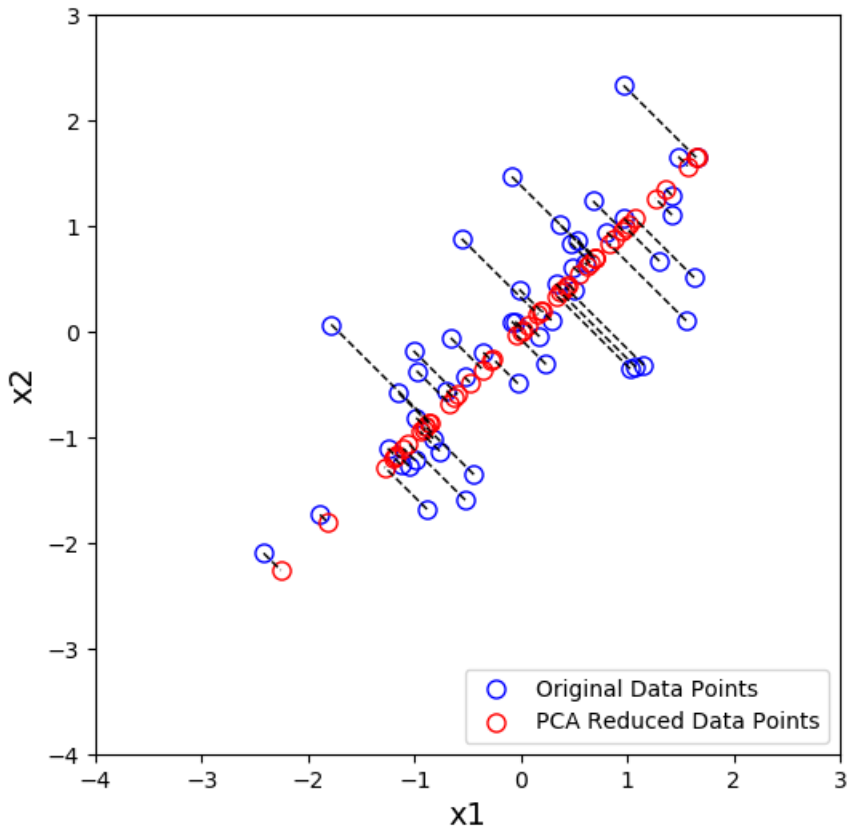

Рисунок 3 – график исходных точек и их проекции на пространство меньшей размерности

9. Код выгрузки данных:

```
file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex7faces.mat')
dataset = sio.loadmat(file_path)
X = dataset['X']
```

10. Визуализация данных:

Рисунок 4 – визуализация 100 случайных изображений из набора данных

11. Код реализации:

```
means, stds, X_norm = feature_normalize(X)
U, S = pca(X_norm)
```

12. Визуализация данных:

Рисунок 5 – 36 главных компонент с наибольшей дисперсией

13. Видно, что качество изображений ухудшилось, а именно – уменьшилась детализация картинок (компоненты с наибольшей дисперсией охватывают самые базовые черты). Можно заметить, что с каждым следующим изображением появляется всё больше и больше деталей (при уменьшении дисперсии количество деталей увеличивается).

14. Визуализация данных:

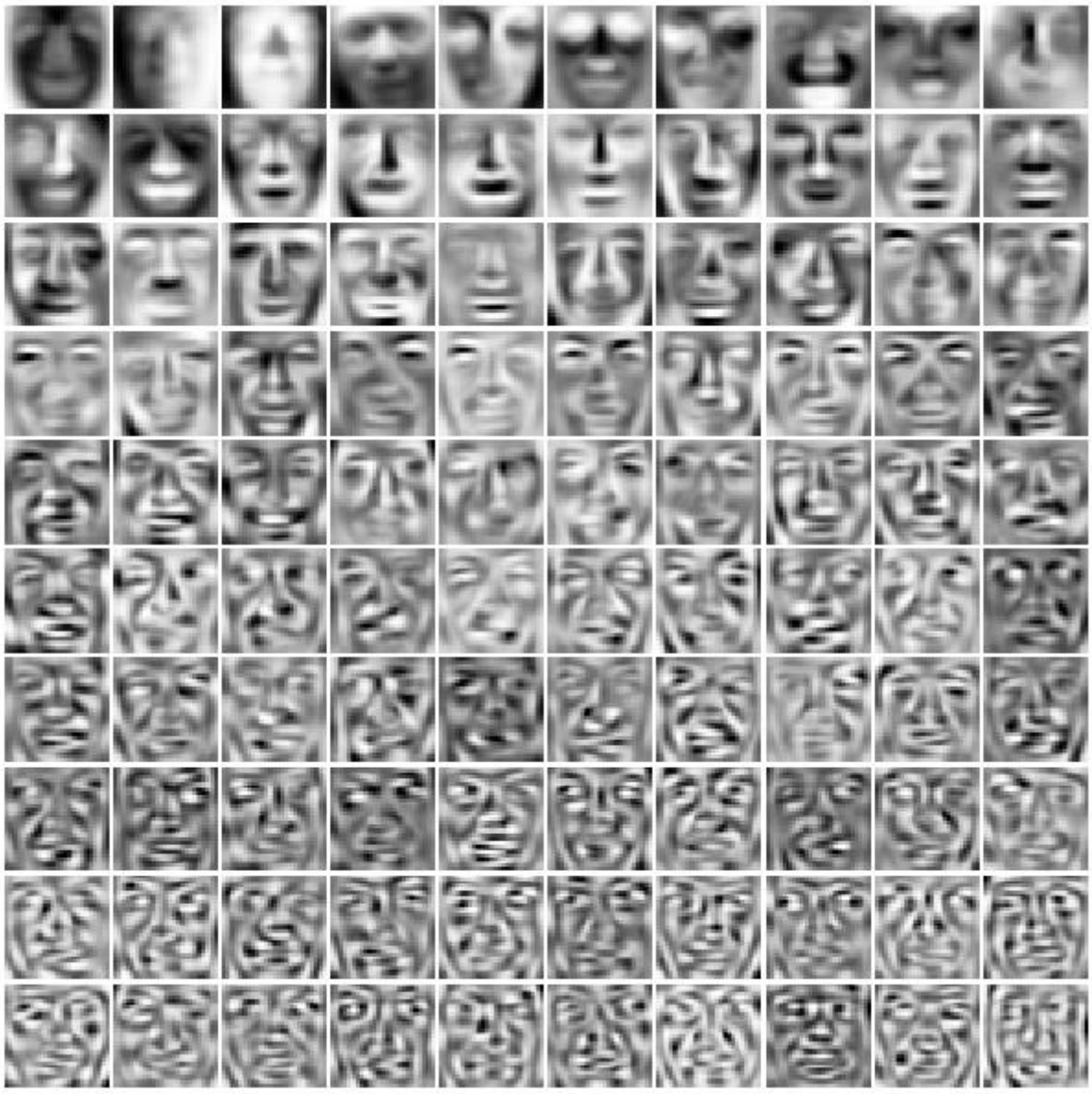

Рисунок 6 – 100 главных компонент с наибольшей дисперсией

15. Когда мы выводим больше изображений, то разница становится более заметной. Если в первых картинках изображения были похожи на пятна, то в последние имеют уже намного больше деталей.

16. Код реализации:

```
A = img.imread(os.path.join('data', 'output.jpg'))
X = A.reshape(-1, 3)
```

17. Визуализация данных:

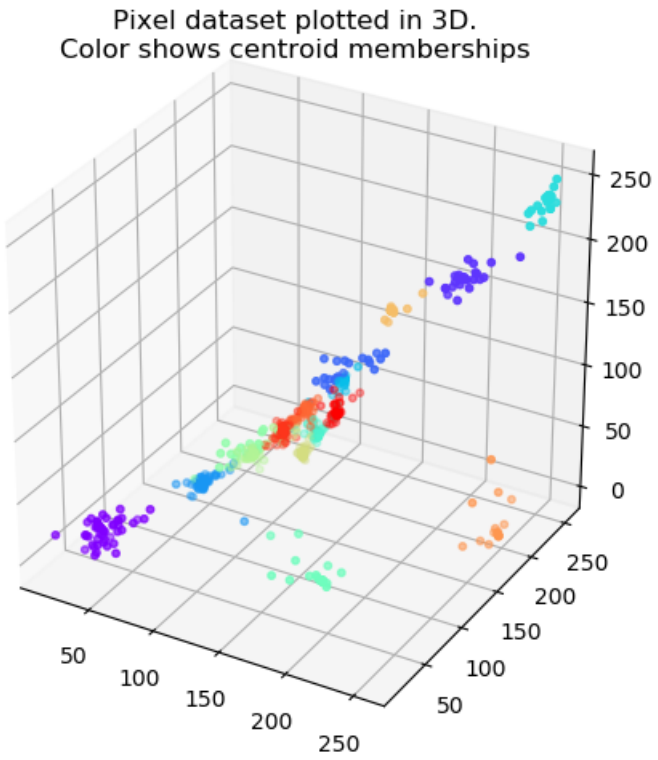

Рисунок 7 – визуализация пикселей изображения и их кластеров в 3D

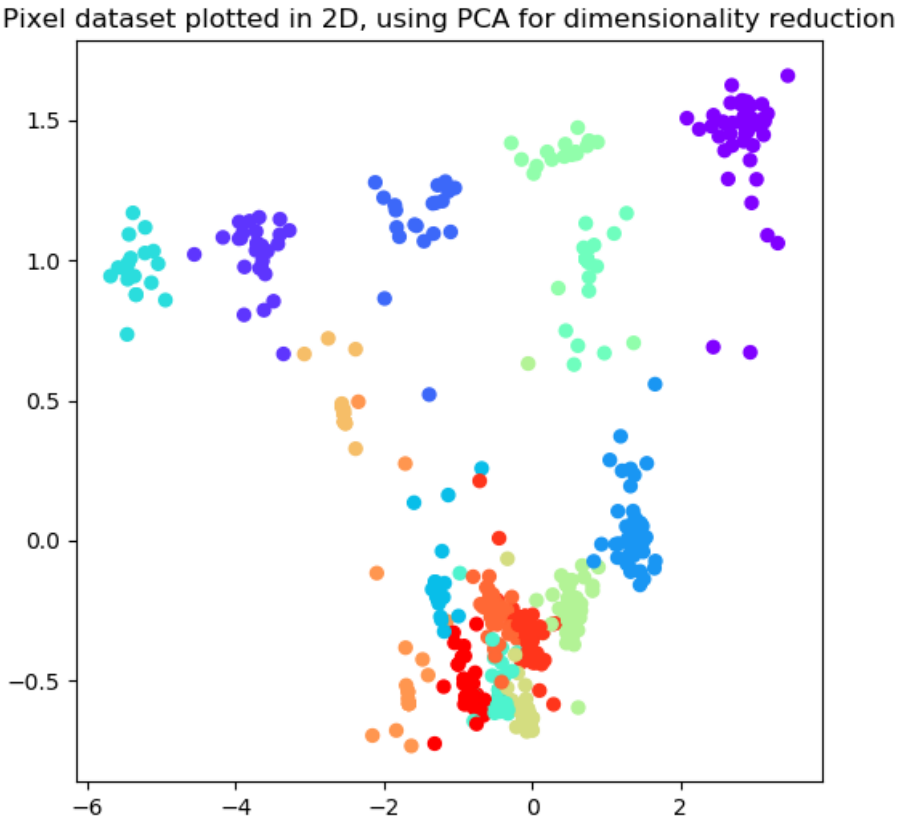

Рисунок 8 – визуализация пикселей изображения и их кластеров в 2D, с помощью PCA

18. Да, 2D изображение соответствует “лучшей” проекции 3D изображения на двумерную плоскость.

Программный код:

```
from mpl_toolkits.mplot3d import Axes3D
from goto import with_goto
import scipy.io as sio
import matplotlib.image as img
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import os
import numpy as np
from numpy.linalg import svd


def feature_normalize(X):
    means = np.mean(X, axis=0)
    X_norm = X - means
    stds = np.std(X_norm, axis=0)
    X_norm = X_norm / stds

    return means, stds, X_norm


def compute_covariance_matrix(X):
    return X.T.dot(X) / X.shape[0]


def pca(X):
    covariance_matrix = compute_covariance_matrix(X)
    U, S, V = svd(covariance_matrix, full_matrices=True, compute_uv=True)

    return U, S


def project_data(X, U, K):
    return X.dot(U[:, :K])


def recover_data(Z, U, K):
    return Z.dot(U[:, :K].T)


def grid_plot(X, dim):
    fig = plt.figure(figsize=(6, 6))
    M, N = X.shape

    gs = gridspec.GridSpec(dim, dim)
    gs.update(bottom=0.01, top=0.99, left=0.01, right=0.99,
              hspace=0.05, wspace=0.05)

    k = 0
    for i in range(dim):
        for j in range(dim):
            ax = plt.subplot(gs[i, j])
            ax.axis('off')
            ax.imshow(-X[k].reshape(int(np.sqrt(N)), int(np.sqrt(N))).T,
                      cmap=plt.get_cmap('Greys'),  # vmin=-1, vmax=1,
                      interpolation='nearest')  # ,alpha = 1.0)
            k += 1
```

```
    plt.show()


@with_goto
def main():
    goto .task
    label .task
    # 1
    file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex7data1.mat')
    dataset = sio.loadmat(file_path)
    X = dataset["X"]

    # 2
    # X[:, 0] - first column
    plt.scatter(X[:, 0], X[:, 1], marker="o")
    plt.show()

    # 3
    covariance_matrix = compute_covariance_matrix(X)
    print(covariance_matrix)

    # 4
    # Feature normalize
    # mu, sigma
    means, stds, X_norm = feature_normalize(X)
    # Run SVD
    U, S = pca(X_norm)
    print(U, S)

    # 5
    #  Draw the eigenvectors centered at mean of data. These lines show the
    #  directions of maximum variations in the dataset.
    fig, ax = plt.subplots()
    ax.plot(X[:, 0], X[:, 1], 'o', mew=0.25)

    for i in range(len(S)):
        ax.arrow(means[0], means[1], 1.5 * S[i]*U[0, i], 1.5 * S[i]*U[1, i],
                 head_width=0.25, head_length=0.2, fc='k', ec='k', lw=2, zorder=1000)

    ax.axis([0.5, 6.5, 2, 8])
    ax.set_aspect('equal')
    ax.grid(False)

    plt.show()

    print('Top principal component: U[:, 0] = [{:.6f} {:.6f}]'.format(U[0, 0], U[1, 0]))
    print(' (you should expect to see [-0.707107 -0.707107])')

    # 6
    #  Project the data onto K = 1 dimension
    K = 1
    Z = project_data(X_norm, U, K)
    print('Projection of the first example: {:.6f}'.format(Z[0, 0]))
    print('(this value should be about    : 1.481274)')

    # 7
    X_rec = recover_data(Z, U, K)
```

```
        print('Approximation of the first example: [{:.6f} {:.6f}]'.format(X_rec[0, 0], X_rec[0,
1]))
        print('       (this value should be about  [-1.047419 -1.047419])')

        # 8
        plt.figure(figsize=(6, 6))
        plt.plot(X_norm.T[0], X_norm.T[1], 'bo', mfc='none', mec='b', ms=8, label='Original
Data Points')
        plt.plot(X_rec.T[0], X_rec.T[1], 'ro', mfc='none', mec='r', ms=8, label='PCA Reduced
Data Points')
        plt.xlabel('x1', fontsize=14)
        plt.ylabel('x2', fontsize=14)
        plt.legend(loc=4)
        for (x, y), (x_rec, y_rec) in zip(X_norm, X_rec):
            plt.plot([x, x_rec], [y, y_rec], 'k--', lw=1)
        plt.xlim(-4, 3)
        plt.ylim(-4, 3)
        plt.show()

        # 9
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex7faces.mat')
        dataset = sio.loadmat(file_path)
        X = dataset['X']

        # 10
        grid_plot(X, 10)

        # 11
        # mu, sigma
        means, stds, X_norm = feature_normalize(X)
        U, S = pca(X_norm)

        # 12, 13
        grid_plot(U.T, 6)

        # 14-15
        grid_plot(U.T, 10)

        # 16
        A = img.imread(os.path.join('data', 'output.jpg'))
        #A.setflags(write=1)
        #A /= 255
        X = A.reshape(-1, 3)

        # 17
        # Sample 1000 random indexes (since working with all the data is
        #  too expensive. If you have a fast computer, you may increase this.
        sel = np.random.choice(X.shape[0], size=1000)

        fig = plt.figure(figsize=(6, 6))
        ax = fig.add_subplot(111, projection='3d')
        idx = np.loadtxt(os.path.join('data', 'output.txt'))
        ax.scatter(X[sel, 0], X[sel, 1], X[sel, 2], cmap='rainbow', c=idx[sel], s=10)
        ax.set_title('Pixel dataset plotted in 3D.\nColor shows centroid memberships')

        plt.show()

        # 18
        mu, sigma, X_norm = feature_normalize(X)
```

```
    # PCA and project the data to 2D
    U, S = pca(X_norm)
    Z = project_data(X_norm, U, 2)

    fig = plt.figure(figsize=(6, 6))
    ax = fig.add_subplot(111)

    ax.scatter(Z[sel, 0], Z[sel, 1], cmap='rainbow', c=idx[sel], s=32)
    ax.set_title('Pixel dataset plotted in 2D, using PCA for dimensionality reduction')
    ax.grid(False)

    plt.show()


if __name__ == '__main__':
    main()
```