CMSC 25300 / 35300
Homework 4

1. **Gram-Schmidt**. In this problem, we want to perform the Gram-Schmidt algorithm and evaluate its performance on the Hilbert matrix with order 7 given as below.

$$X = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} \\ \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} \end{bmatrix}$$

a. Revisiting the Gram-Schmidt algorithm in the lecture 6, write your own code to perform Gram-Schmidt orthogonalization.
Here is some code to get you started:

```
import numpy as np

def gram_schmidt(X):
    # X is an n-by-p matrix.
    # Returns U an orthonormal matrix.
    # eps is a threshold value to identify if a vector
    # is nearly a zero vector.
    eps = 1e-12

    n, p = X.shape
    U = np.zeros((n, 0))
    for j in range(p):
        # Get the j-th column of matrix X
        v = X[:, j]
        # Write your own code here: Perform the
        # orthogonalization by subtracting the projections on
        # all columns of U. And then check whether the vector
        # you get is nearly a zero vector.

    return U
```

b. Use the function you obtained and perform Gram-Schmidt orthogonalization on the matrix X. Evaluate the accuracy by checking the orthogonality of the resulting basis. Report the $L1$ matrix norm of the error matrix you get.
Instead of manually typing the values of X, here is some code that you could utilize for generating the order 7 Hilbert matrix.

```python
def hilbert_matrix(n):
    X = np.array([[1.0 / (i + j - 1) for i in \\
    range(1, n + 1)] for j in range(1, n + 1)])
    return X
```

c. **Modified Gram-Schmidt.** Use the provided code of the modified Gram-Schmidt to perform the orthogonalization on the matrix X. Assess the accuracy of the resulting orthogonal basis and compare it with the results obtained in part (b). Discuss the distinctions between the two implementations.

```python
def modified_gram_schmidt(X):
    # Define a threshold value to identify if a vector
    # is nearly a zero vector.
    eps = 1e-12

    n, p = X.shape
    U = np.zeros((n, 0))

    for j in range(p):
        # Get the j-th column of matrix X
        v = X[:, j]
        for i in range(j):
            # Compute and subtract the projection of
            # vector v onto the i-th column of U
            v = v - np.dot(U[:, i], v) * U[:, i]
        v = np.reshape(v, (-1, 1))
        # Check whether the vector we get is nearly
        # a zero vector
        if np.linalg.norm(v) > eps:
            # Normalize vector v and append it to U
            U = np.hstack((U, v / np.linalg.norm(v)))

    return U
```

2. **The SVD**. Imagine there are three points in $\mathbb{R}^2$, where

$$\boldsymbol{x}_1 = \begin{bmatrix} 3 \\ 0 \end{bmatrix}, \qquad \boldsymbol{x}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \qquad \boldsymbol{x}_3 = \begin{bmatrix} 0 \\ \sqrt{6} \end{bmatrix}.$$

Let data matrix be $\boldsymbol{X} = \left[\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\right] \in \mathbb{R}^{2\times 3}$.

a. Find an **orthonormal basis** for the span of the three points.

b. For any vector $\boldsymbol{v}$ in $\mathbb{R}^2$, we can project all data points onto the subspace $\boldsymbol{V}$ spanned by $\boldsymbol{v}$.

i. Write the expression for the projection matrix $\boldsymbol{P}$ onto $\boldsymbol{V}$ in terms of $\boldsymbol{v}$.

ii. What is the squared distance of each data point $\boldsymbol{x}_i$ from subspace $\boldsymbol{V}$, in terms of $\boldsymbol{x}_i$ and $\boldsymbol{P}$? (HINT: what is the distance between $\boldsymbol{x}_i$ and $\boldsymbol{P}\boldsymbol{x}_i$? Try to simplify your answer using properties of $\boldsymbol{P}$)

iii. Let $\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, what is the sum of the squared distance for all 3 data points. Write the expression in terms of v_1 and v_2.

iv. Now find $\boldsymbol{v}$ with $\|\boldsymbol{v}\|_2 = 1$ that minimizes sum of the squared distance for all data points. Is $\boldsymbol{v}$ unique? Is $\boldsymbol{P}$ unique? (HINT: what values of v_1 and v_2 gives the minimum value for your answer in iii.? For this problem you should be able to solve it analytically without a computer.)

c. Suppose now we are doing full SVD of $\boldsymbol{X}$, i.e., $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. Provide possible matrices $\boldsymbol{U}$ and $\boldsymbol{\Sigma}$. **Hint: use the results of part (b).**

3. Let $\boldsymbol{X}$ be an $n \times d$ matrix with right singular vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_r$, left singular vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_r$, and corresponding singular values $\sigma_1, \sigma_2, \cdots, \sigma_r$.

   a. Decompose $\boldsymbol{X}$ into a sum of rank one matrices in terms of the singular vectors and singular values above.

   b. Recall rank-k approximations $\boldsymbol{X}_k$, where $k < r$. Express $\boldsymbol{X}_k$ with the sum of rank one matrices.

4. Suppose we have a square matrix $\boldsymbol{X} = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$. What is its SVD?

5. Suppose we have a square matrix $\boldsymbol{X} = \begin{bmatrix} -3 & 0 \\ 0 & -1 \end{bmatrix}$. What is its SVD?

6. Consider the smiling face classification problem from HW 2. Design and compare the performances of the classifiers proposed in **a** and **b**, below. In each case, divide the dataset into 8 equal sized subsets (e.g., examples $1 - 16$, $17 - 32$, etc). Use 6 sets of the data to estimate $\boldsymbol{w}$ for each choice of the *regularization parameter*, select the best value for the regularization parameter by estimating the error on one of the two remaining sets of data, and finally use the $\boldsymbol{w}$ corresponding to the best value of the regularization parameter to predict the labels of the remaining "hold-out" set. Compute the number of mistakes made on this hold-out set and divide that number by 16 (the size of the set) to estimate the error rate. Repeat this process 56 times (for the 8×7 different choices of the sets used to select the regularization parameter and estimate the error rate) and average the error rates to obtain a final estimate.

   a. Truncated SVD solution. Use the pseudo-inverse $\boldsymbol{V}\boldsymbol{\Sigma}_k^+\boldsymbol{U}^T$, where $\boldsymbol{\Sigma}_k^+$ is computed by inverting the k largest singular values and setting others to zero. Here, k is the regularization parameter and it takes values $k = 1, 2, \ldots, 9$; i.e., compute 9 different solutions, $\widehat{\boldsymbol{w}}_k$.

b. Regularized LS. Let $\widehat{\boldsymbol{w}}_\lambda = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$, for the following values of the regularization parameter $\lambda = 0, 2^{-1}, 2^0, 2^1, 2^2, 2^3$, and 2^4. Show that $\widehat{\boldsymbol{w}}_\lambda$ can be computed using the SVD and use this fact in your code.

Here is some code to get you started:

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
import sys

d = sio.loadmat('face_emotion_data.mat')
X = d['X']
y = d['y']

n, p = np.shape(X)

# error rate for regularized least squares
error_RLS = np.zeros((8, 7))
# error rate for truncated SVD
error_SVD = np.zeros((8, 7))

# SVD parameters to test
k_vals = np.arange(9) + 1
param_err_SVD = np.zeros(len(k_vals))

# RLS parameters to test
lambda_vals = np.array([0, 0.5, 1, 2, 4, 8, 16])
param_err_RLS = np.zeros(len(lambda_vals))
```

c. Use the original dataset to generate 3 new features for each face, as follows. Take the 3 new features to be a random linear combination of the original 9 features. This can be done for instance with the Matlab command `X @ np.random.rand(9,3)` and augmenting the original matrix $\boldsymbol{X}$ with the resulting 3 columns. Will these new features be helpful for classification? Why or why not? Repeat the experiments in (a) and (b) above using the 12 features.