

Computer Homework 1

Convex optimization

Radmehr Karimian

Dr.yassaee

April 14, 2023

1. Eigenvalues and eigenvectors

The algorithm is like this:

Let $A_0 = A$. For each $k \geq 0$:

$$A_k = Q_k R_k$$

$$A_{k+1} = R_k Q_k$$

Note that:

$$A_{k+1} = R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T A_k Q_k = Q_k^{-1} A_k Q_k$$

So all of the A_k 's are similar and therefore have the same eigenvalues

As k increases, the A_k 's converge to an upper triangular matrix, and the eigenvalues are the diagonal entries

So For the second part we have:

```
import numpy as np
n = 6
A = np.random.rand(n, n)
print("A=")
print(A)

def eigen_qr_simple(A, iterations=500000):
    Ak = np.copy(A)
    n = A.shape[0]
    QQ = np.eye(n)
    for k in range(iterations):
        Q, R = np.linalg.qr(Ak)
        Ak = R @ Q
        QQ = QQ @ Q
        if k == 500000 - 1:
            print("A", k, "=")
            print(Ak)
            print("\n")
    return Ak, QQ
eigen_qr_simple(A)
print(np.linalg.eigvals(A))
```

The overall complexity (number of floating points) of the algorithm is $O(n^3)$, which we will see is not entirely trivial to obtain. The major limitation of the QR algorithm is that already the first stage usually generates complete fill-in in general sparse matrices. The “naive QR algorithm” works flawlessly in theory, but in practice, not so well. So people implementing linear algebra algorithms found a few tricks. One of those tricks is called “shifts”.


We subtract the value:

$$A_k - s_k * I = Q_k * R_k$$

Then we put the value back in:

$$A_{k+1} = R_k * Q_k + s_k * I$$

So Having said this, our “smarter QR algorithm with shifts” looks like:



```
import numpy as np
n = 6
A = np.random.rand(n, n)
print("A=")
print(A)

def eigen_qr_simple(A, iterations=500000):
    Ak = np.copy(A)
    n = A.shape[0]
    QQ = np.eye(n)
    for k in range(iterations):
        s = Ak.item(n-1, n-1)
        smult = s * np.eye(n)
        Q, R = np.linalg.qr(np.subtract(Ak, smult))
        Ak = np.add(R @ Q, smult)
        QQ = QQ @ Q
        if k == 500000 - 1:
            print("A", k, "=")
            print(Ak)
            print("\n")
    return Ak, QQ
eigen_qr_simple(A)
print(np.linalg.eigvals(A))
```

2.SVD and Computer vision:

This part has a lot of images and plots, please check python notebook for full description.

3.PCA:

A.

$$\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)$$

$$\Sigma = \frac{1}{n-1} ((X - \bar{x})^T (X - \bar{x}))$$

B.

$$C_X = \frac{1}{n-1} X^T X = C_X^T$$

$$Y = XA$$

$$C_Y = \frac{1}{n-1} Y^T Y = \frac{1}{n-1} (XA)^T (XA) = \frac{1}{n-1} A^T (X^T X) A$$

Assuming $A = V$ each column is an eigenvector of $X^T X$

$$C_Y = \frac{1}{n-1} A^T (X^T X) A = \frac{1}{n-1} V^T (X^T X) V = \frac{1}{n-1} V^T (VDV^T) V = \frac{1}{n-1} V^{-1} V D V^{-1} V = \frac{1}{n-1} D$$

In SVD method, we can prove that the eigenvectors of X are same as C .

$$X^T X = (V \Sigma U^T)(U \Sigma V^T) = V \Sigma^2 V^T$$

$$C = V \frac{\Sigma^2}{N-1} V^T$$

$$Y = X V$$

C. We prove Everything in the previous part.

D.Linear Discriminant Analysis, or LDA, is a machine learning algorithm that is used to find the Linear Discriminant function that best classifies or discriminates or separates two classes of data points. LDA is a supervised learning algorithm, which

means that it requires a labelled training set of data points in order to learn the Linear Discriminant function. Once the Linear Discriminant function has been learned, it can then be used to predict the class label of new data points. LDA is similar to PCA (principal component analysis) in the sense that LDA reduces the dimensions. However, the main purpose of LDA is to find the line (or plane) that best separates data points belonging to different classes. The key idea behind LDA is that the decision boundary should be chosen such that it maximizes the distance between the means of the two classes while simultaneously minimizing the variance within each classes data or within-class scatter. This criterion is known as the Fisher criterion and can be expressed as the following formula for two classes

E. Full description in Python notebook.