# COMP 4107 - Assignment 1

## Due: 1st February 2020 at 11:55 pm

**Name:** Kirk Zhen
**Student ID:** 101087006

### Question 1 [5 marks]

Compute the SVD for the matrix below:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$$

**Answer:**

Calculated by Numpy library:

$$U = \begin{bmatrix} -0.3330689 & 0.73220483 & 0.37614814 & 0.45984101 \\ -0.48640367 & 0.34110504 & -0.754835 & -0.2779981 \\ -0.79307315 & -0.44109455 & 0.37868687 & -0.18184292 \\ -0.15333474 & -0.39109979 & -0.38122559 & 0.82352684 \end{bmatrix}$$

$$s = \left[1.10528306e+01,\ 9.13748280e-01,\ 1.30538231e-16\right]$$

Note that $s$ produced by Numpy is a sorted list of singular values.

$$V^T = \begin{bmatrix} -0.41903326 & -0.56492763 & -0.71082199 \\ -0.81101447 & -0.11912225 & 0.57276996 \\ -0.40824829 & 0.81649658 & -0.40824829 \end{bmatrix}$$

To compute the SVD from what we have, first of all, we have to transfer list $s$ into a diagonal matrix $S$:

$$S = \begin{bmatrix} 1.10528306e+01 & 0 & 0 \\ 0 & 9.13748280e-01 & 0 \\ 0 & 0 & 1.30538231e-16 \end{bmatrix}$$

And because the matrix $U$ produced by Numpy is $4 \times 4$, we take $U_3$, which is $4 \times 3$, as the first part of low column rank SVD, we have:

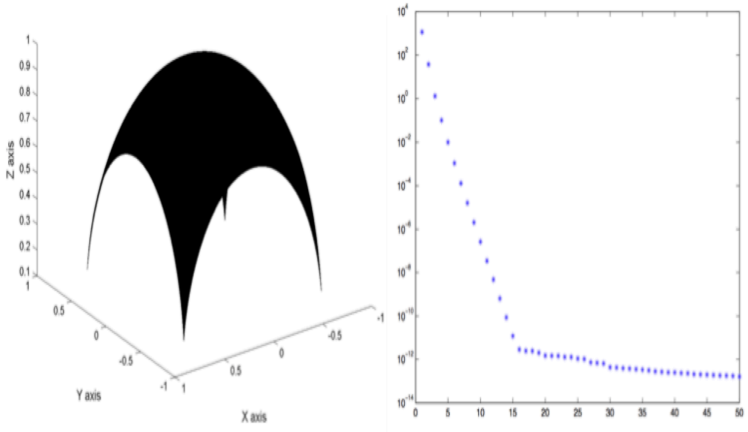
$$U_3 = \begin{bmatrix} -0.3330689 & 0.73220483 & 0.37614814 \\ -0.48640367 & 0.34110504 & -0.754835 \\ -0.79307315 & -0.44109455 & 0.37868687 \\ -0.15333474 & -0.39109979 & -0.38122559 \end{bmatrix}$$

In this way, we successfully compute the SVD for matrix $A$, where $A = U_3 S V^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$.

## Question 2 [10 marks]

Let the matrix $A = \left[a_{ij}\right] \in R^{1401\times1401}$ represent a discretization of the surface,

$$z = \sqrt{1 - x^2 - y^2},\ -0.7 \le x, y \le 0{,}7.$$




Specifically, let

$$a_{ij} = \sqrt{1 - x_i^2 - y_j^2},\ x_i = y_i = -0.7 + \Delta\,(i-1),\ \Delta = 0.001,\ 1 \le i, j \le 1401$$

The left figure above represents the surface. The right figure represents 50 of the singular values. Compute the best rank(2) matrix, $A_2$, approximation to the matrix $A$. What is $\left\| A - A_2 \right\|$ ?

**Answer:**

Compute the best rank(2) matrix, $A_2$, approximation to the matrix $A$.

Calculated by Numpy, the best rank(2) matrix approximation to the matrix $A$ is:

$$A_2 = \begin{bmatrix} 0.17447807 & 0.17754332 & 0.18056607 & \dots & 0.18056607 & 0.17754332 & 0.17447807 \\ 0.17754332 & 0.18059153 & 0.18359756 & \dots & 0.18359756 & 0.18059153 & 0.17754332 \\ 0.18056607 & 0.18359756 & 0.18658718 & \dots & 0.18658718 & 0.18359756 & 0.18056607 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0.18056607 & 0.18359756 & 0.18658718 & \dots & 0.18658718 & 0.18359756 & 0.18056607 \\ 0.17754332 & 0.18059153 & 0.18359756 & \dots & 0.18359756 & 0.18059153 & 0.17754332 \\ 0.17447807 & 0.17754332 & 0.18056607 & \dots & 0.18056607 & 0.17754332 & 0.17447807 \end{bmatrix}$$

What is $\left\| A - A_2 \right\|$ ?

$$\left\| A - A_2 \right\|_1 = 5.234429564502095$$

$$\left\| A - A_2 \right\|_2 = 1.327263475418059$$

## Question 3 [15 marks]

Using the matrix $A$ from Question 1, and let $b = [1,1,1,1,]^T$, using the gradient descent method, determine the least squares solution of $\min_{x \in R^3} \| Ax - b \|$

Using a value of $\delta = 0.01$, Provide a table giving the values of $x$ and the number of iterations taken to converge for values of $\varepsilon = 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5$. If there are values for which a solution cannot be found document why this is the case.

**Answer:**

The result of my python program is as shown in the following table:

| $\varepsilon$ | $x$ | Convergence/Divergence | Total Iterations |
|---|---|---|---|
| 0.01 | [−0.1373585, 0.06024832, 0.25785513] | Convergence | 369 |
| 0.05 | [nan, nan, nan] | Divergence | 436 |
| 0.1 | [nan, nan, nan] | Divergence | 295 |
| 0.15 | [-inf, -inf, nan] | Divergence | 250 |
| 0.2 | [nan, nan, nan] | Divergence | 227 |
| 0.25 | [nan, nan, nan] | Divergence | 211 |
| 0.5 | [nan, nan, nan] | Divergence | 175 |

As shown in the table, when $\varepsilon = 0.05, 0.1, 0.15, 0.2, 0.25, 0.5$. the solution cannot be found.

This is because the step size ($\varepsilon$ in the question) is too large. When we tried to get close to the minimum with the given $\varepsilon$, because $\varepsilon$ is too large, we would overshoot the minimum, and reach the alternative side to the minimum. In this way, the cost function could be witnessed a trend with fluctuation, and finally go divergent.

## Question 4 [10 marks]

Find two linearly independent vectors belonging to the null space of the matrix

$$A = \begin{bmatrix} 3 & 2 & -1 & 4 \\ 1 & 0 & 2 & 3 \\ -2 & -2 & 3 & -1 \end{bmatrix}$$

Are the columns of $A$ (above) linearly independent in $R^3$? Why? Are the rows of $A$ linearly independent in $R^4$? Why?
Calculate the inverse of matrix $A$.

**Answer:**

Find two linearly independent vectors belonging to the null space of the matrix

Calculated by Numpy and Scipy, the null space of matrix $A$ is:

$$Nul\,(A) = \begin{bmatrix} -0.75878571 & -0.33687873 \\ 0.59374776 & -0.65142615 \\ -0.02104018 & -0.50845005 \\ 0.26695535 & 0.45125961 \end{bmatrix}$$

We have: $Nul\,(A) = [a_1, a_2]$, where $a_1 = \begin{bmatrix} -0.75878571 \\ 0.59374776 \\ -0.02104018 \\ 0.26695535 \end{bmatrix}$, $a_2 = \begin{bmatrix} -0.33687873 \\ -0.65142615 \\ -0.50845005 \\ 0.45125961 \end{bmatrix}$.

Hence, $a_1$ and $a_2$ are two linearly independent vectors belonging to the null space of $A$.

Are the columns of $A$ (above) linearly independent in $R^3$? Why?

According to my Python program: Number of columns in matrix $A$ is 4, and Rank($A$) = 2.
Because the number of columns in $A$ is larger than Rank($A$), there are two columns in $A$ can be written as a linear combination of the other two columns.
According to the definition of Linear Independence, columns of $A$ are not linearly independent.

Are the rows of $A$ linearly independent in $R^4$? Why?

According to my Python program: Number of rows in matrix $A$ is 3, and Rank($A^T$) = 2. Because the number of rows in $A$ is larger than Rank($A^T$), there are one row in $A$ can be written as a linear combination of the other two rows.
According to the definition of Linear Independence, rows of $A$ are not linearly independent.

Calculate the inverse of matrix $A$.

Because matrix $A$ is not a square matrix, we apply Moore-Penrose Pseudoinverse:
Calculated by Numpy, the Moore-Penrose Pseudoinverse of matrix $A$ is:

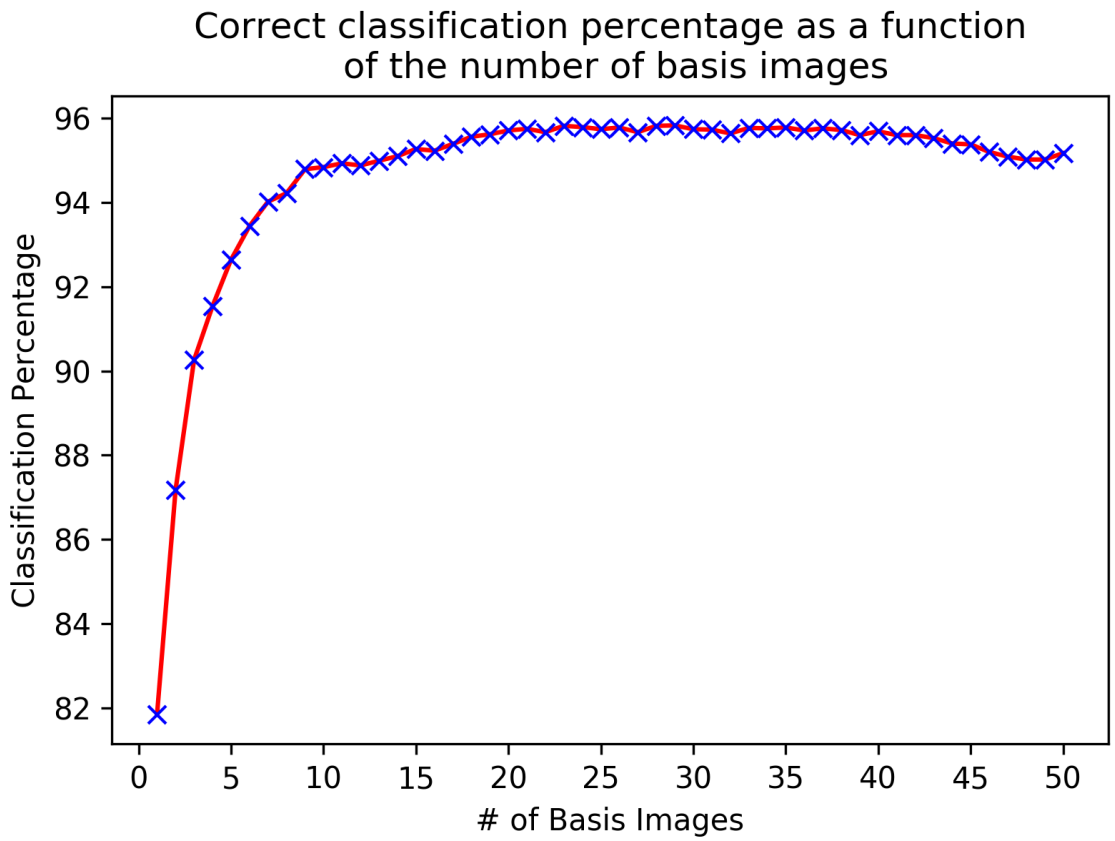
$$A^+ = \begin{bmatrix} 0.06507304 & 0.01460823 & -0.05046481 \\ 0.03984064 & -0.03187251 & -0.07171315 \\ -0.00929615 & 0.14077025 & 0.1500664 \\ 0.09561753 & 0.12350598 & 0.02788845 \end{bmatrix}$$

## Question 5 [25 marks]

Reading 4 associated with lecture 3 describes an SVD-based method for classifying digits. Section 3 describes a representation of the data that allows SVD to be used where a subset of the images is used for classification. Using the representation of images described in Section 3, generate the results that are seen in Figure 4. Specifically, plot the accuracy as a function of the basis size used in the decomposition. You should run multiple experiments, sampling the original image datasets randomly. This will generate multiple A matrices. Reference 13 of the paper indicates where the MNIST data may be found.

**Answer:**

According to my python program, here is my regeneration of the results in Figure 4.
(The times cost of the program is a little bit expensive, it take near 1 hour to terminate on Colab, and even much longer on my Macbook)



## Question 6 [25 marks]

Reading 5 associated with lecture 3 describes an SVD-based method for generating a recommendation system. Using the MovieLens dataset referred to in the paper, recreate the experimental evaluation described in Section 4. Read section 3 on "Incremental SVD Algorithms" before you start. Specifically, you should be able to reproduce Figure 3 and Figure 4, although the absolute values on your y axis for Figure 4 will be quite different because of your personal machine environments.

**Answer:**

According to my python program, here is my regeneration of the results in Figure3 and Figure 4 (As same as in Question 5, the run time of the program is a little bit long. And because I shuffled the data, the result of each time you run this program may be a little bit different.)