CENG 371 - Scientific Computing
Fall 2023
Homework 3

Aksoy, Aybüke
e2448090@ceng.metu.edu.tr

December 26, 2023

---

1. a) Given Ax=b where A's dimension is mxn and m¿n with a unique solution x, rank(A) is n. Since rank of A is equal to the number of columns in A, A is full rank. The vector b is a linear combination of the columns of A such that $a_1 * x_1 + a_2 * x_2 + ... + a_n * x_n = b$ as the system has a solution. The set of vectors b+b' that satisfies the system Ax=b+b' (again with a unique solution x) must also be a linear combination of the columns of A which results in b' also being a linear combination of the columns of A. To find the dimensionality of the set of vectors b' that satisfies this condition, we need to consider the dimension of the column space of A. As A is full rank with n columns, each column has a pivot in row reduced echelon form. Thus, each column of the original A is in the basis of the column space and the dimension of the basis is n. Therefore, the dimensionality of the set B is n excluding the 0 vector.

   b) A and b can be found as:

$$A = \begin{bmatrix} \sqrt{2} & \sqrt{5} & \sqrt{10} \\ \sqrt{5} & 2\sqrt{2} & \sqrt{13} \\ \sqrt{10} & \sqrt{13} & 3\sqrt{2} \\ \sqrt{17} & 2\sqrt{5} & 5 \\ \sqrt{26} & \sqrt{29} & \sqrt{34} \end{bmatrix}$$

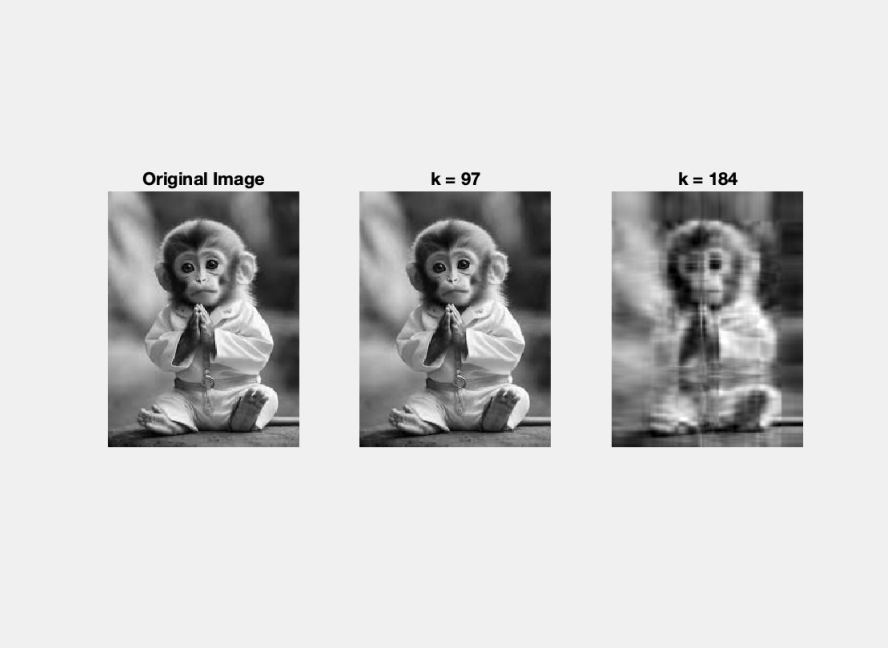
$$b = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

   To find the basis for space B, which is the column space as we have found from part a, we need to compute the row reduced echelon form of matrix A and figure out the columns with pivots in rref. The corresponding columns in A will form the basis. In ceng371hw3q1.m file, I have implemented a algorithm that uses gaussian elimination after partial pivoting in myRREF function. The rows without the pivot are set to zero as they are linearly dependent to other rows with pivots. My method only works for matrices which have more rows then columns as I am only checking the limits of columns. In is $find\_basis\_of\_column\_space$ function , I find the column space basis by traversing through the columns of $rref\_A$ and checking if they have the value 1 in the required pivot position and append the corresponding columns in A that satisfies the condition to the basis.

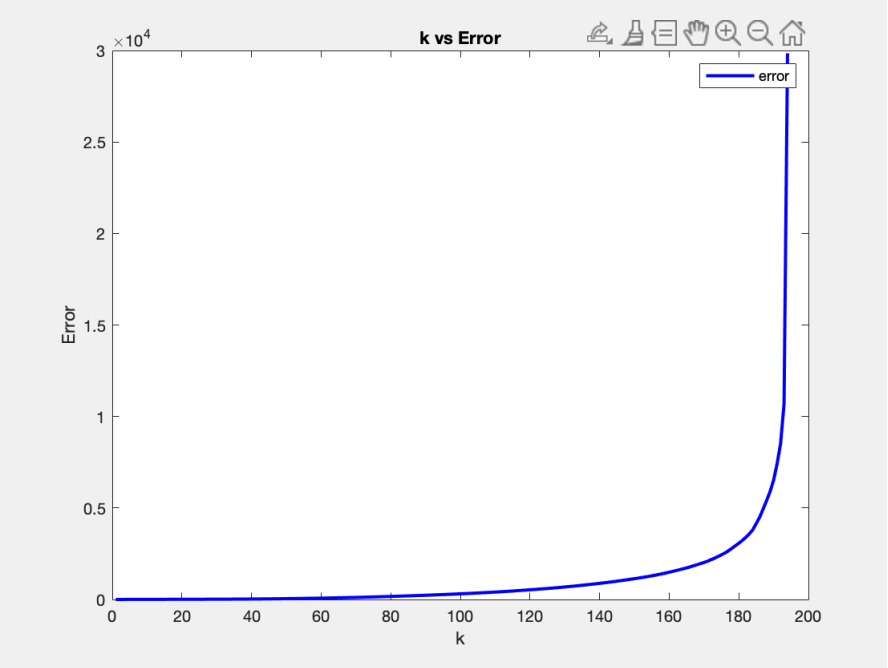
   The basis vectors for B is:

$$\begin{bmatrix} \sqrt{2} \\ \sqrt{5} \\ \sqrt{10} \\ \sqrt{17} \\ \sqrt{26} \end{bmatrix}, \begin{bmatrix} \sqrt{5} \\ 2\sqrt{2} \\ \sqrt{13} \\ 2\sqrt{5} \\ \sqrt{29} \end{bmatrix}, \begin{bmatrix} \sqrt{10} \\ \sqrt{13} \\ 3\sqrt{2} \\ 5 \\ \sqrt{34} \end{bmatrix}$$

2. a) I have used matlab's svd function and computed the low-rank approximations for given k values. For my image, r=194; so the approximations for k=97 and k=184 are as:

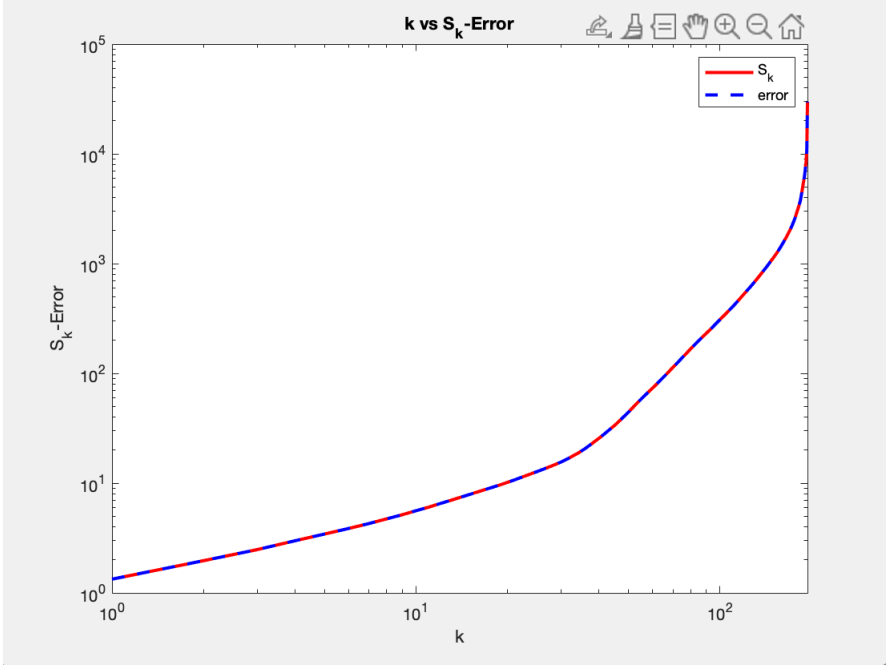


To compute I_k, we are using low-rank approximations by discarding the smallest k values. Matlab's svd function returns the S matrix such that it holds the singular values in decreasing order and the first few singular values capture the most significant information about the matrix.



As it can be seen from the results on k=97 and k=184, the image approximated from k=184 is much worse in terms of image quality and there is not much difference between the original image and the approximation on k=97 as the error is exponential. In our implementation, since we are discarding the smallest k singular values, as we increase the k value, we will take less singular values into account and will retain less information from the overall image which results in a less accurate approximation. However, increasing the k value, reduces the computational cost by shrinking the matrices U,S and V.

b) I have used Matlab svd function in "vector" form to get the singular values in a vector with descending order. Then, I changed the order such that they are ascending and computed the S(k) values accordingly. Lastly, I have computed the errors $||I - I_k||_F$ using the approximations I have found in the previous part. The log plots of the S(k) and the error is as:



S(k) provides a measure of the information loss during the low-rank approximation as it is directly related to the sum of the k smallest singular values, which are intentionally discarded in the low-rank approximation. A higher value of S(k) indicates that more information from the original matrix is lost in the low-rank approximation. The formula for error is:

$$||I - I_k||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |I_{ij} - I_{k_{ij}}|^2}$$

Frobenius norm measures the overall magnitude of differences across all elements, providing a comprehensive assessment of the accuracy of the approximation. S(k) and the error are calculated in a similar way. As S(k) increases, capturing less information, the error increases, resulting in a less accurate representation of the original matrix. As the information loss occurring during the approximation is the main reason for the difference and error between the original and approximated image, their graphs exhibit similar trend and for this specific implementation they are the same.

c) The low-rank approximation mainly can be used in compression of images, videos and texts etc. In our implementation, we have seen the results on images. As images often exhibit strong correlations between pixels, a significant amount of image information can be captured by a relatively small number of dominant singular values. By discarding the smallest k singular values in the SVD decomposition where k is not very large, we can still retain enough information from the image and obtain a representation of a matrix close to the original that requires less storage than what is needed to store all the entries of original matrix, which is the basic idea behind the lossy image compression. Even though the approximated image shows degradation from the first image, the differences are not too pronounced when k is choosed carefully. In fact, the approximated image may be good enough for some practical purposes where we have limited storage resources.

3. a) I have constructed the matrix A and $A\_obs \in R^{nx4}$ such that:

$$\mathrm{A} = \begin{bmatrix} (1/(n+1))^0 & (1/(n+1))^1 & (1/(n+1))^2 & (1/(n+1))^3 \\ (2/(n+1))^0 & (2/(n+1))^1 & (2/(n+1))^2 & (2/(n+1))^3 \\ (3/(n+1))^0 & (3/(n+1))^1 & (3/(n+1))^2 & (3/(n+1))^3 \\ . & . & . & . \\ . & . & . & . \\ (n/(n+1))^0 & (n/(n+1))^1 & (n/(n+1))^2 & (n/(n+1))^3 \end{bmatrix}$$

$$A\_obs = \begin{bmatrix} (1/(n+1)+\epsilon)^0 & (1/(n+1)+\epsilon)^1 & (1/(n+1)+\epsilon)^2 & (1/(n+1)+\epsilon)^3 \\ (2/(n+1)+\epsilon)^0 & (2/(n+1)+\epsilon)^1 & (2/(n+1)+\epsilon)^2 & (2/(n+1)+\epsilon)^3 \\ (3/(n+1)+\epsilon)^0 & (3/(n+1)+\epsilon)^1 & (3/(n+1)+\epsilon)^2 & (3/(n+1)+\epsilon)^3 \\ . & . & . & . \\ . & . & . & . \\ (n/(n+1)+\epsilon)^0 & (n/(n+1)+\epsilon)^1 & (n/(n+1)+\epsilon)^2 & (n/(n+1)+\epsilon)^3 \end{bmatrix}$$

t values are calculated for the set given and small error generated by randn*0.01 is added in A_obs matrix. Each column of A and A_obs is for the corresponding k value. $A_nisconstructedsuchthatitappliestheN$ which is Nobs(t) only.

c_k is given as:

$$c\_k = \begin{bmatrix} 0.3 \\ 2 \\ -1.2 \\ 0.5 \end{bmatrix}$$

The problem to be solved is constructed such that $A\_n * c\_k = b\_n$ where b_n represents our Nobs for each t. It is found as:

$$b_n = c_1 * A\_obs(:,1) + c_2 * A\_obs(:,2) + c_3 * A\_obs(:,3) + c_4 * A\_obs(:,4)$$

I have generated O_n list for each n with dimension R^{nx2} that includes the t values as the first column and the b_n ($N\_obs(t)$) values as the second column. I append the observation list for each n run to the O_list.
For n=10, A_n, A_obs, b_n and O_n are:

```
A_n =

    1.0000    0.0909    0.0083    0.0008
    1.0000    0.1818    0.0331    0.0060
    1.0000    0.2727    0.0744    0.0203
    1.0000    0.3636    0.1322    0.0481
    1.0000    0.4545    0.2066    0.0939
    1.0000    0.5455    0.2975    0.1623
    1.0000    0.6364    0.4050    0.2577
    1.0000    0.7273    0.5289    0.3847
    1.0000    0.8182    0.6694    0.5477
    1.0000    0.9091    0.8264    0.7513
```

```
A_n_obs =

    1.0000    0.0822    0.0056    0.0008
    1.0000    0.1869    0.0305    0.0049
    1.0000    0.2486    0.0753    0.0194
    1.0000    0.3800    0.1313    0.0495
    1.0000    0.4574    0.2060    0.0928
    1.0000    0.5538    0.3180    0.1588
    1.0000    0.6420    0.3999    0.2233
    1.0000    0.7231    0.5440    0.3967
    1.0000    0.8008    0.6759    0.5400
    1.0000    0.9162    0.8451    0.7644
```

```
b_n =

    0.4581
    0.6398
    0.7165
    0.9271
    1.0140
    1.1053
    1.2158
    1.2917
    1.3604
    1.5005
```

```
O_n =

    0.0909    0.4581
    0.1818    0.6398
    0.2727    0.7165
    0.3636    0.9271
    0.4545    1.0140
    0.5455    1.1053
    0.6364    1.2158
    0.7273    1.2917
    0.8182    1.3604
    0.9091    1.5005
```

b) (For this part, I have used Linear Least Squares method with normal equations as it is advised to check the related example in part 8 in lecture notes which uses this method. Where A^TA is singular, it is not the best practice to use this method. Hence, my implementation can be improved by changing to alternatives using SVD or QR approximation.)

Now assuming, we do not know what the c_k are, we need to solve the system $A\_n * c\_k = b\_n$. We have constructed the required matrix A and Nobs in the previous part. Hence, we can use the Linear Least Squares method for an overdetermined system since our system is also overdetermined with n being 5,10,100 and k having 4 values. We need to approximate c_k find one that minimizes the error $||b\_n - A\_n * c\_k||_2^2$. Using normal equations, we can derive c_k as:

$$c\_k = (A\_n^T * A\_n)^{-1} * A\_n^T * b\_n$$

where $(A\_n^T * A\_n)^{-1} * A\_n^T$ is the pseudo-inverse of A_n. Matlab's pinv function can be used to find the c_k's from this formula. I am applying 500 runs and approximate 500 different c_k's for each n. Some of the approximated c_k's are found as:

```
approximated_c_k =

    0.2993
    1.9931
   -1.3030
    0.6864
```

```
approximated_c_k =

    0.2915
    1.9986
   -1.1434
    0.4552
```

c) Over 500 runs for each n, the average errors are calculated as:

```
c_k_error_list =

    0.8429
    0.2131
    0.0727
```

$$n = 5,\ ||c\_k - approximated\_c\_k||_2 = 0.8429$$
$$n = 10,\ ||c\_k - approximated\_c\_k||_2 = 0.2131$$
$$n = 100,\ ||c\_k - approximated\_c\_k||_2 = 0.0727$$

Comparing the errors, we can observe that the error is getting smaller as n increases. So, as we increase the number of data points and the observations, our approximation becomes more accurate. When we have less number of observations in our hand, which are constructed by sampling random variables, any noise between those observations would affect our approximation extremely and they would not be stable as our system is very sensitive to noise and outliers. With larger number of observations, the system can capture the underlying trend more effectively.