Scientific Computing Assignment

Erdem
Şamlıoğlu
2448843

Question 1

Part A

$Ax = b + b^{'}$ shows that b' is a vector and when added to b, it stays within the range of A. So for any $b^{'}$ in B, $b + b^{'}$ is a vector in the range of A. We know that B is consisted of vectors $b^{'}$, so the vectors in B form a subspace of R^m. Its dimensionality is the same with dimensionality of range of A. As a conclusion, since B's dimensionality is equal to A's rank, and A's maximum rank is n (A is m x n matrix), we can say that B's maximum dimensionality can be at most n. For the exact dimensionality, it depends on the matrix A. In case of A having full rank, then the dimensionality of B is n, in case A is rank deficient, then dimensionality of B is less than n.

Part B

The matrix A and vector b is already given to us, so constructing the matrix and vector in the code, and using Singular Value Decomposition algorithm we have seen in the course to find the column space of A and lastly the basis of column A since it will be the basis of B.

Here is the python code for it and from the files I have uploaded, question1.py:

```
import numpy as np


A=np.array([[np.sqrt(i**2+j**2) for j in range(1,4)] for i in range(1,6)])
b=np.array([i for i in range(1,6)])

U, S, Vt=np.linalg.svd(A, full_matrices=False)

non_zero_elements= S>1e-10
basis_B= U[:,non_zero_elements]
```

```
print("Matrix A:")
print(A)
print("\nVector b:")
print(b)
print("\nBasis for B:")
print(basis_B)
```

Question 2

Part A

For this part, I used Matlab, and my code is in question2ab.m. In the code, I have first read the image I named as ”catgrayscale”, and converted it to a double precision matrix for Singular Value Decomposition. Then I computed the Singular Value Decomposition of the image matrix I. I set the r as the minimum dimensions of I, computed the low rank approximations of I, which are at r/2 and r-10.

Here are the images I get:

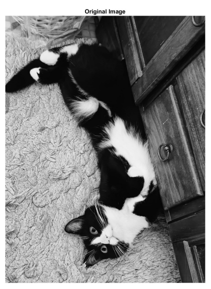

Figure 1: Original

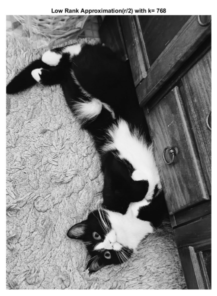

Figure 2: r/2

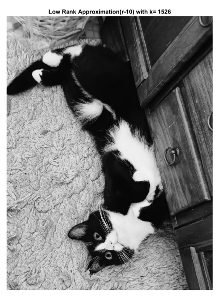

Figure 3: r-10

Part B

For this part, I used Matlab as well, and my code is in question2ab.m. In the code, I have done the same first steps as part a. After computing Singular Value Decomposition, I plotted the singular values on log-log scale to be able to see how quickly the singular values decay. I calculated the Frobenius norm of the error for each low rank approximation I_k and plotted it to be able to see how the approximation error decreases while k increases. Lastly I calculated S(k) and plotted it to be able to see the energy captured by the first k singular values.

Here are the plots:

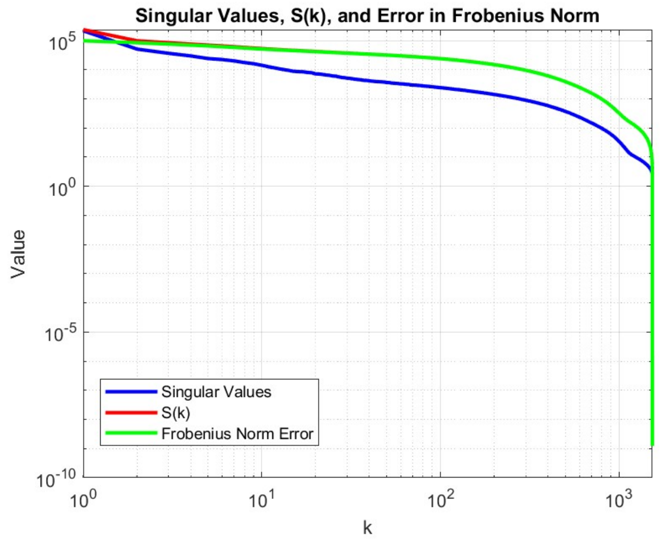

Figure 4: Plot

Part C

For part a and part b, I have tried my code for other grayscale pictures such as taking a screenshot of the question 2 directly, and ran my code with it (I will upload the screenshot I took as question2.jpg), and for that one, r/2 was visibly different from the original one. But for the cat picture there wasn't any visible difference. Although there was difference, it wasn't a huge difference to effect the all of the image, and for the cat picture there wasn't any visible changes I have noticed. In conclusion, low rank approximation can be used in real time image or video streaming services, since the image won't change much and the efficiency will be higher. This would mean that we can provide a data compression such that it wouldn't effect the quality much but the speed and space for bandwidth will be better.

Question 3

Part A

For this part, I used Matlab, and my code is in question3ab.m. In the code, I have defined my function N(t) as it is given in the assignment, and then I defined another function to calculate the noisy observations by adding Gauissian noise with a standard deviation of the N(t) values for 't' points. Then as it is stated in the question, for each value n=5, n=10 and n=100, I constructed matrix A_n as $[1, t, t^2, t^3]$ (representing columns), and for the vector b_n for the noisy observations, and then printed them in my code.

Part B

For this part, I used Matlab again, same code as in question3ab.m since I have calculated both parts in the same file. I have used least squares method from the course to be able to find c_k, from the equation $A_n c = b_n$. Then I stored the best fitting coefficients based on the noisy observations. Here are the example outputs from one run I get for n=5, n=10 and n=100 in order.

```
Coefficients for n=5:
    1.7847
    1.4079
   -1.0540
    0.4518
Coefficients for n=10:
    1.8631
    0.8581
    0.0357
   -0.1731

Coefficients for n=100:
    1.7960
    1.2530
   -0.6805
    0.2301
```

Part C

I ran my code multiple times to make observations on c_k values and how n affects its quality. The average errors as n increases don't show anything consistent. This is normally unexpected since n increases, the estimation should have been more accurate. In general the values for n=10 and n=100 were very close and weren't significantly different than n=5. Another observation I made is that c_1, c_2, c_3 had higher errors compared to c_0, which shows that it is harder to estimate it accurately for higher order terms. The reasons for this could be due to the noise level and the range and distribution of 't'.
As a conclusion from the observations I made, the n values don't show anything about the estimation value. I can only see that the higher order coefficients are more sensitive to the noise.