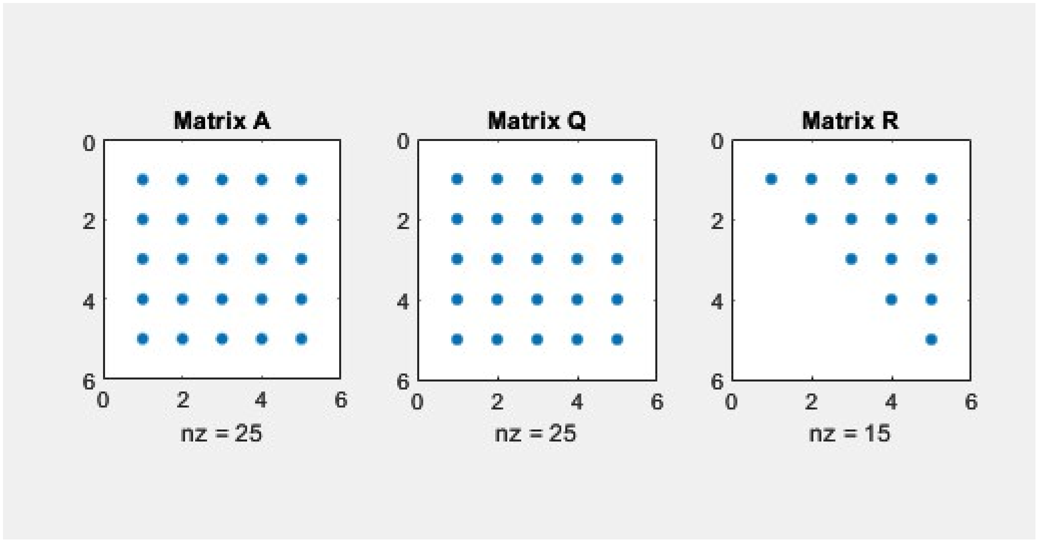
CS 3200

Assignment 5

Lianrui Geng

Q1:



Summary of my Q1 code:

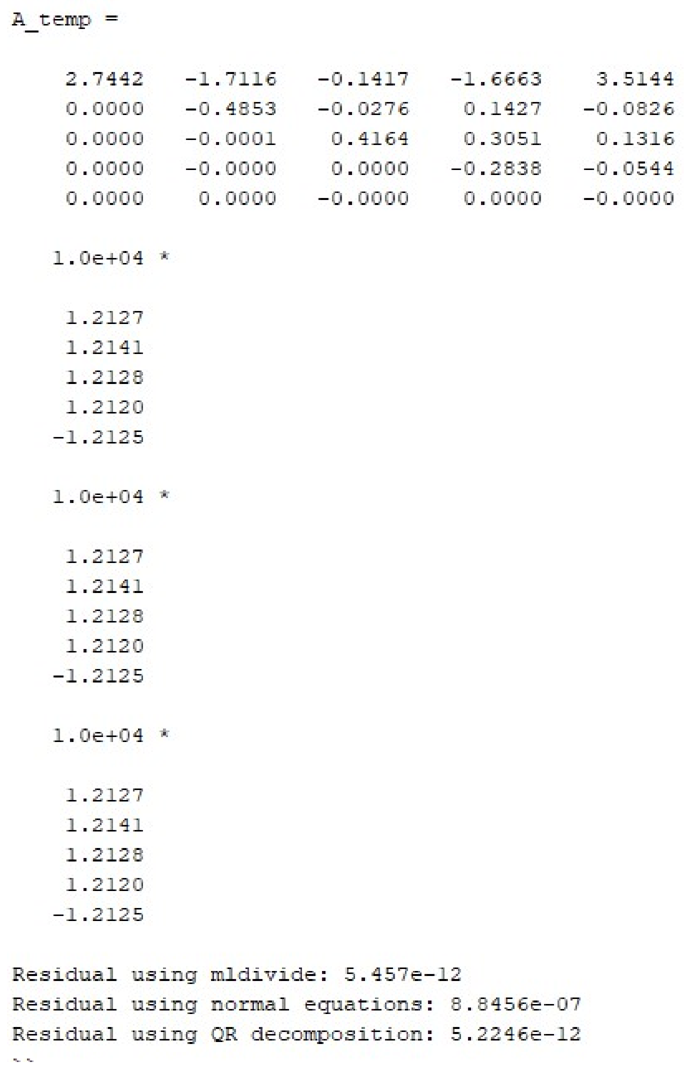
Defines a matrix A and a vector b, performs QR decomposition on A, and plots the sparsity patterns of A, Q, and R. Part b of the code iteratively computes the QR decomposition of A_temp and updates it, and then calculates its eigenvalues. Part c solves the linear system Ax=b using the backslash operator and displays the solution. Part d solves the same system using the normal equations and displays the solution. Part e solves the system using QR decomposition and displays the solution. Finally, part f computes the residuals of the three methods used to solve the system, namely backslash, normal equations, and QR decomposition, and displays them.

The screenshot below showed the accuracy:

Residual using multicide: 5.457e-12

Residual using normal equations: 8.8456e-07

Residual using QR decomposition: 5.2246e-12

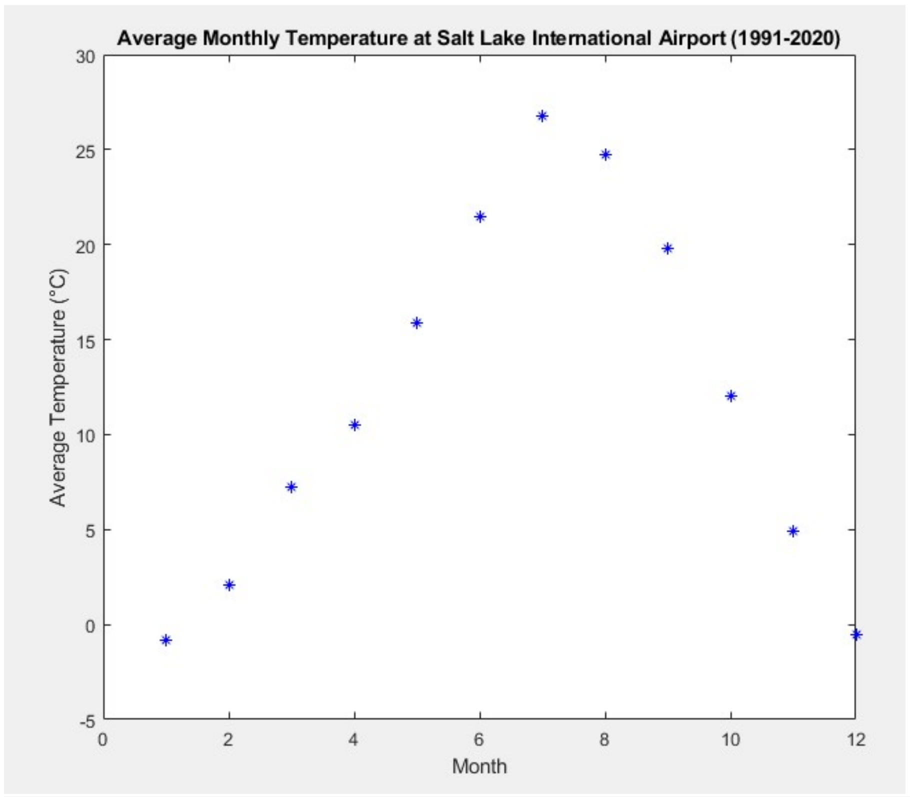


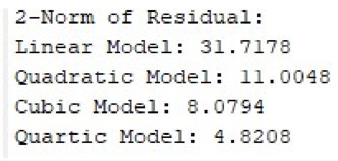
```
A_temp =

    2.7442   -1.7116   -0.1417   -1.6663    3.5144
    0.0000   -0.4853   -0.0276    0.1427   -0.0826
    0.0000   -0.0001    0.4164    0.3051    0.1316
    0.0000   -0.0000    0.0000   -0.2838   -0.0544
    0.0000    0.0000   -0.0000    0.0000   -0.0000

   1.0e+04 *

    1.2127
    1.2141
    1.2128
    1.2120
   -1.2125

   1.0e+04 *

    1.2127
    1.2141
    1.2128
    1.2120
   -1.2125

   1.0e+04 *

    1.2127
    1.2141
    1.2128
    1.2120
   -1.2125

Residual using mldivide: 5.457e-12
Residual using normal equations: 8.8456e-07
Residual using QR decomposition: 5.2246e-12
```

Q2:

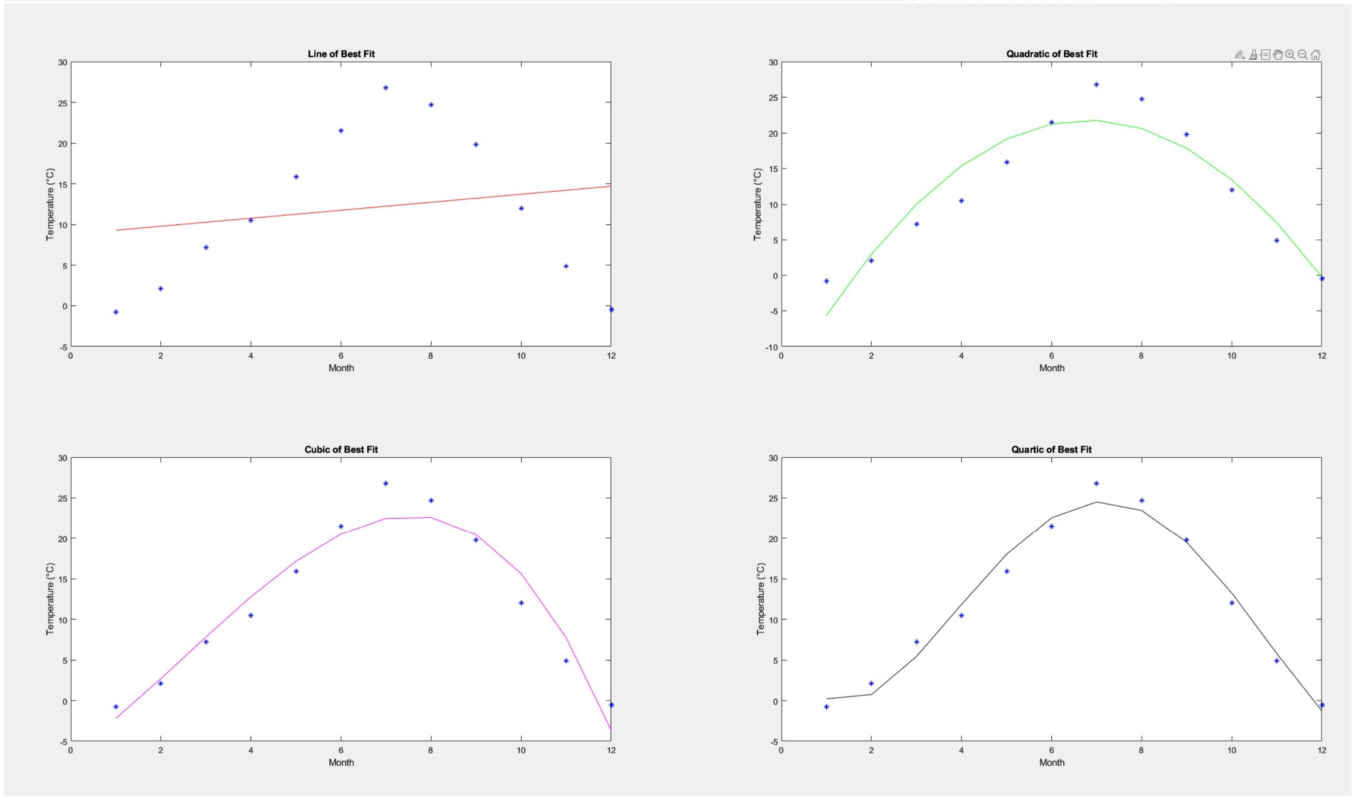
Summary of the code:

The code first plots a scatter plot of average monthly temperature data for Salt Lake International Airport from 1991-2020. Part b of the code fits four different polynomial models (linear, quadratic, cubic, and quartic) to the temperature data using QR decomposition and plots the lines of best fit for each model. The code also calculates the 2-norm of the residual for each model and prints the results.





```
2-Norm of Residual:
Linear Model: 31.7178
Quadratic Model: 11.0048
Cubic Model: 8.0794
Quartic Model: 4.8208
```



2-Norm of Residual:

Linear Model: 31.7178

Quadratic Model: 11.0048

Cubic Model: 8.0794

Quartic Model: 4.8208

As we can see, the quartic model is the best fit.

Q3:

Summary of code:

This code performs different operations on a Hilbert matrix H of size 15x15. In part a, it solves a linear system of equations using backslash operator and calculates the difference between the actual and the computed solution. In part b, it performs singular value decomposition (SVD) of H, plots the sparsity patterns of the resulting matrices H, S, U, and V, and displays them. In part c, it calculates the pseudo-inverse of H using SVD, solves the linear system using the pseudo-inverse, calculates the error, and displays its norm. In part d, it tests the effect of truncating the SVD on the accuracy of the solution and displays the accuracy for different tolerance levels.

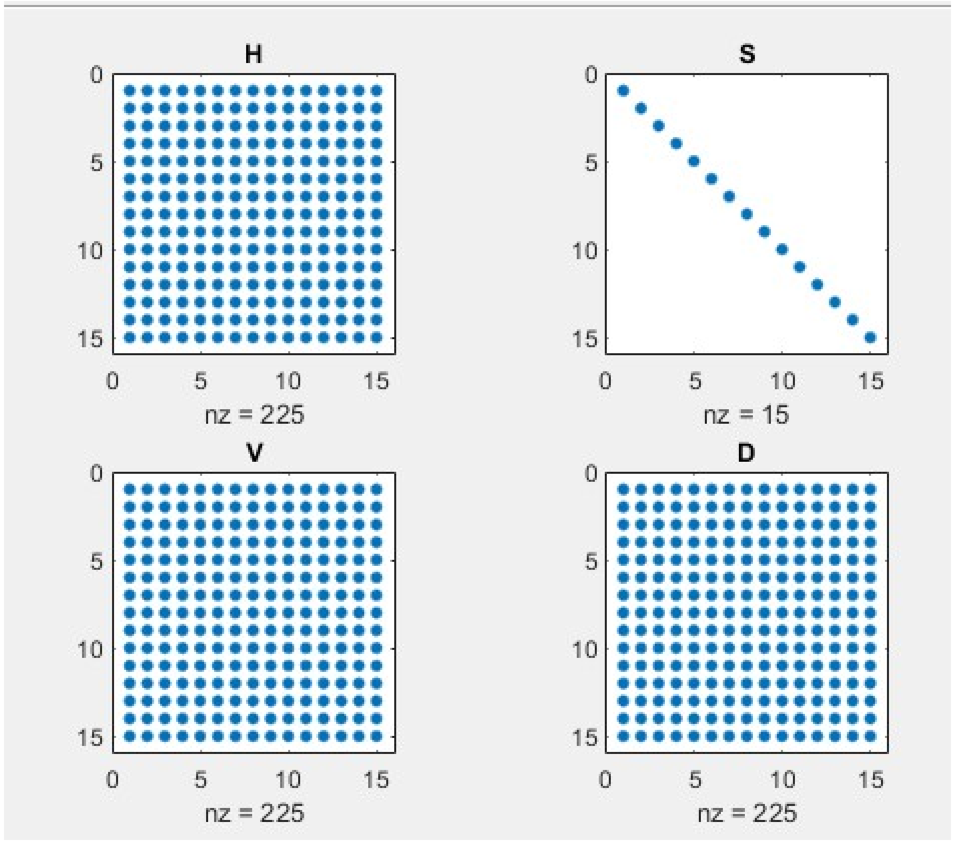
```
>> Q3

Warning: Matrix is close to singular or badly scaled. The results may be inaccurate. RCOND =
8.890153e-19.

> In Q3 (line 10)


Part A result:

  11.2989
```

Part b's result are figures as shown.

```
Part C result:

   6.9412


Warning: Matrix is close to singular or badly scaled. The results may be inaccurate. RCOND =
2.202024e-18.

> In Q3 (line 58)
```

Part D result:

```
Tolerance      Accuracy
    0.0000    0.0000
    0.0000    0.0000
    0.0000    0.0110
    0.0000    1.3198
```



The observation of the result as the figure as shown.

As the result showed, it's clear that the result is more accurate than part a.

The part d 's answer in above the figure.

Q4:

Summary of the code:

Part A:

Computes a matrix H using a nested loop.

Solves for x using the matrix H and its inverse.

Calculates the difference between the original x and the solved x.

Part B:

Imports an image and converts it to grayscale.

Calculates the SVD of the grayscale image.

Creates a random grayscale image of the same size as the original.

Calculates the SVD of the random image.

Plots the singular values of the random image.

Part C:

Plots the singular values of the original and random images.

Calculates and plots the cumulative sum of the singular values for both images.

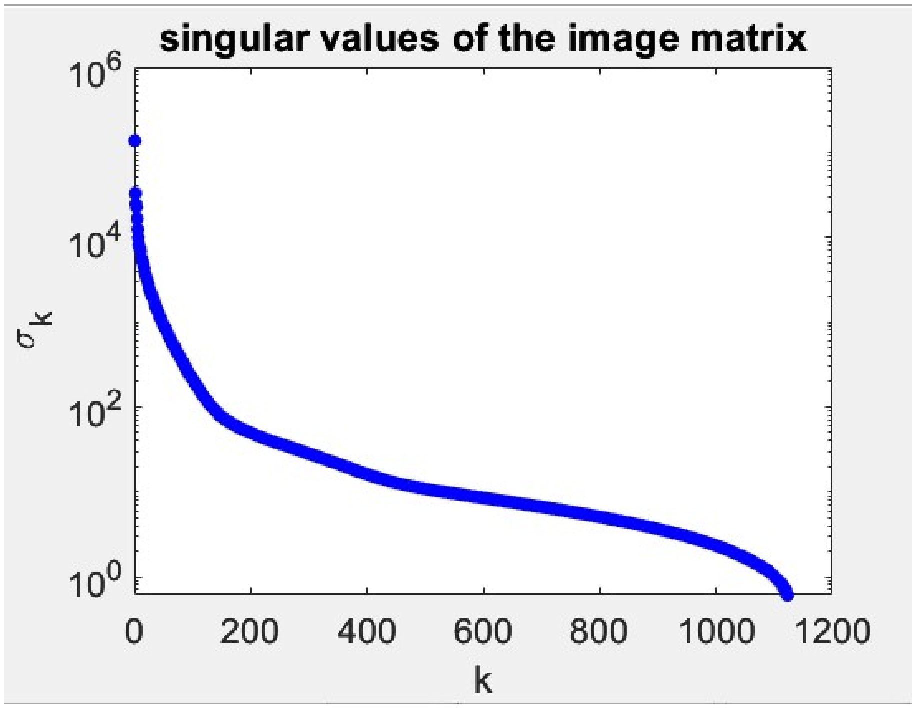
Part D:

Displays the original grayscale image.

Calculates and displays three approximations of the image using different rank values.

Calculates the accuracy of the approximations by comparing them to the original image.

Displays the accuracy results for different tolerance values.


singular values of the image matrix
10^6
10^4
10^2
10^0
σ_k
0
200
400
600
800
1000
1200
k

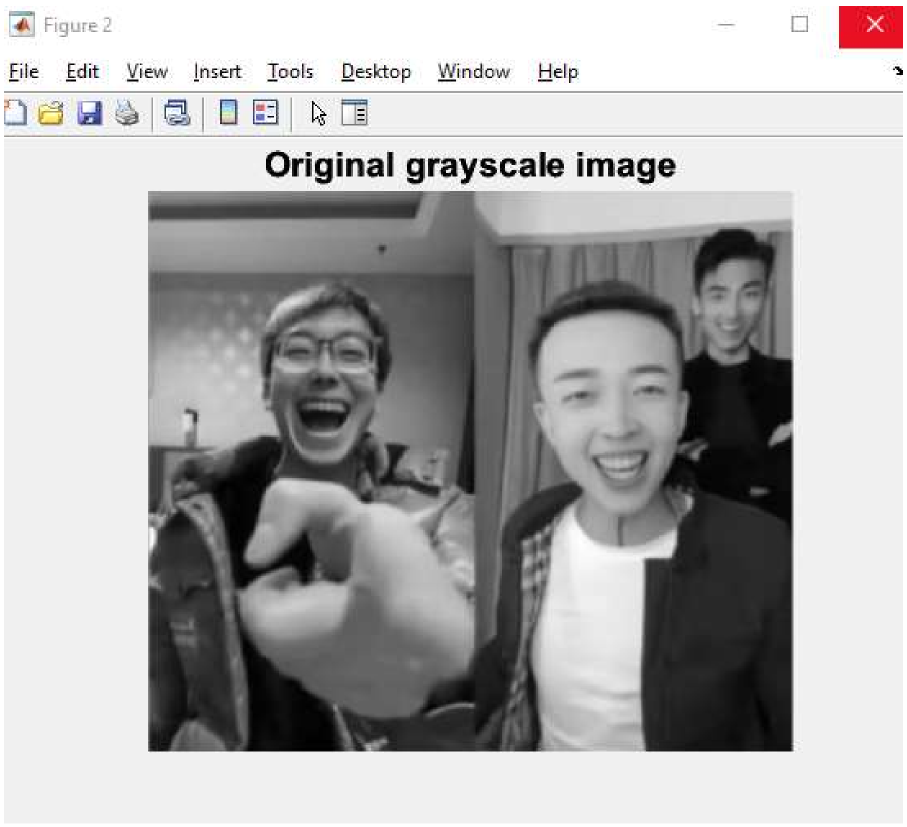


Figure 2
File
Edit
View
Insert
Tools
Desktop
Window
Help
Original grayscale image

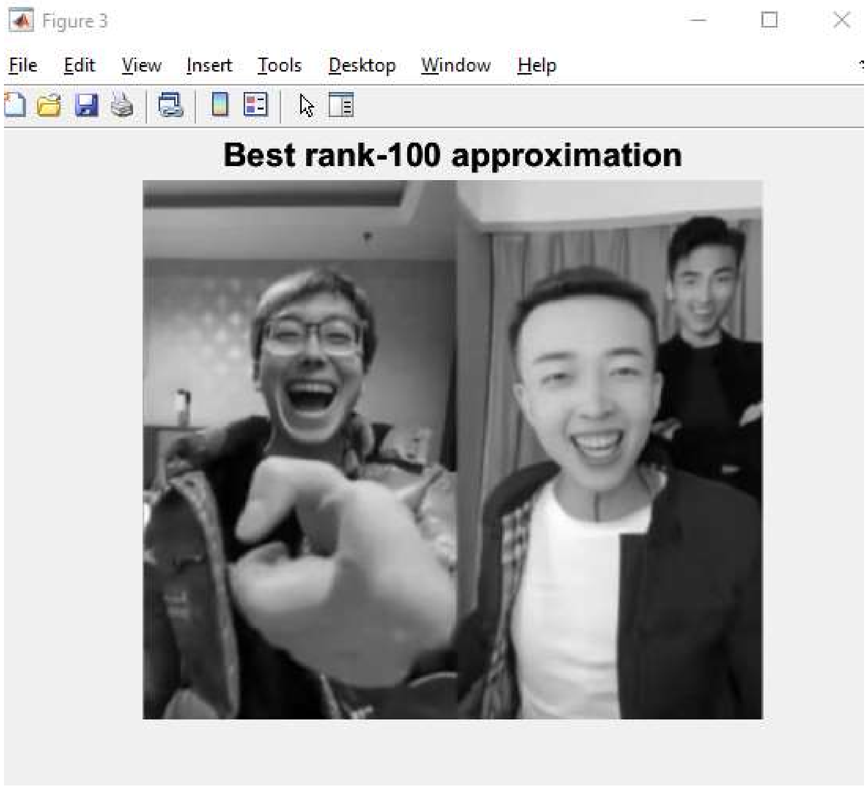


Figure 3
File
Edit
View
Insert
Tools
Desktop
Window
Help
Best rank-100 approximation

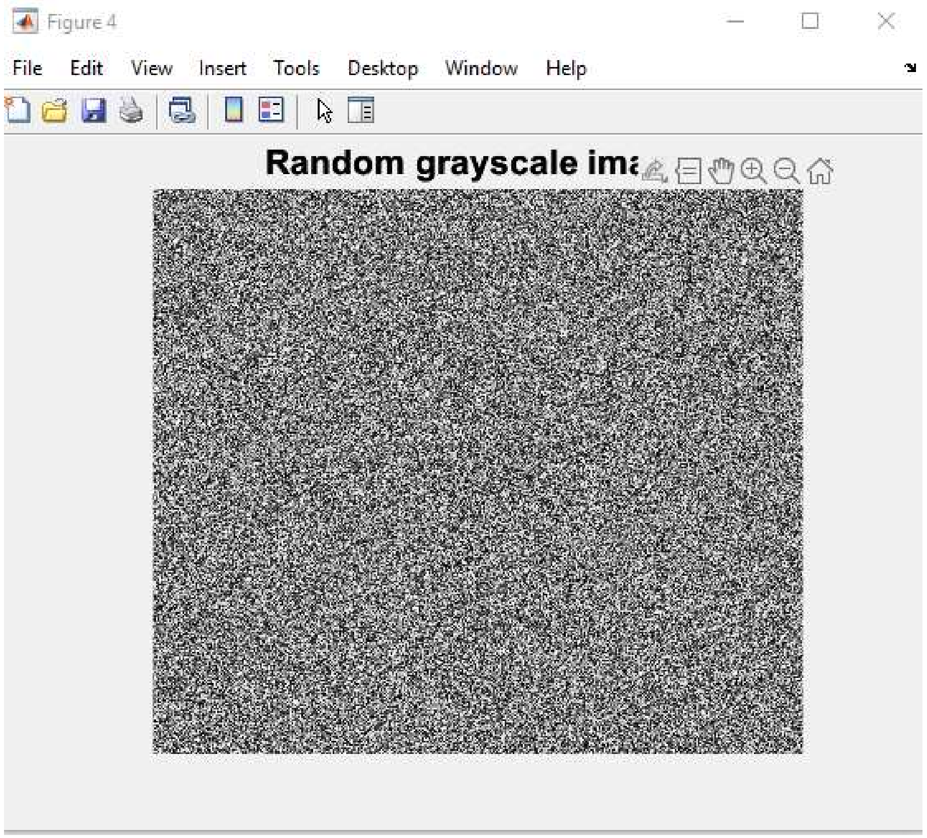
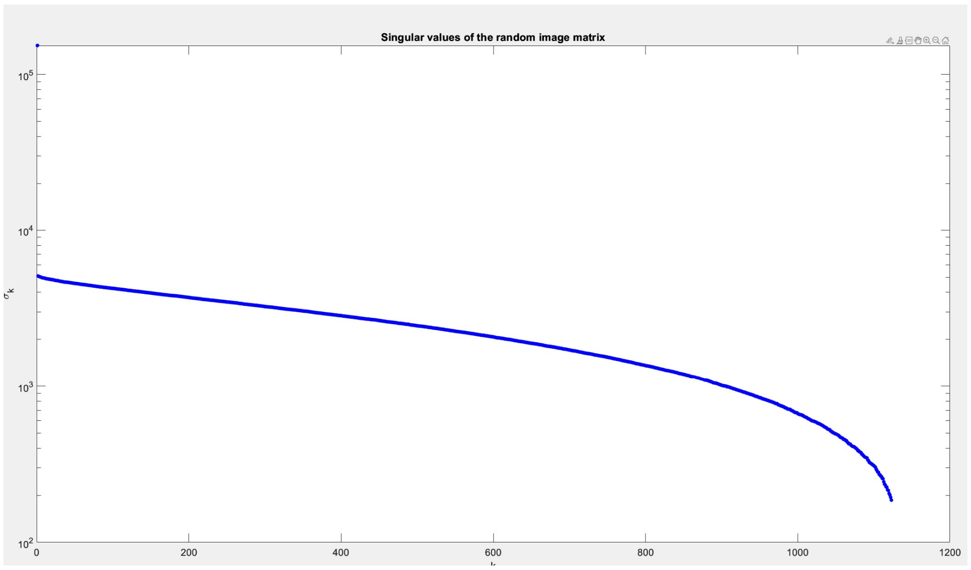
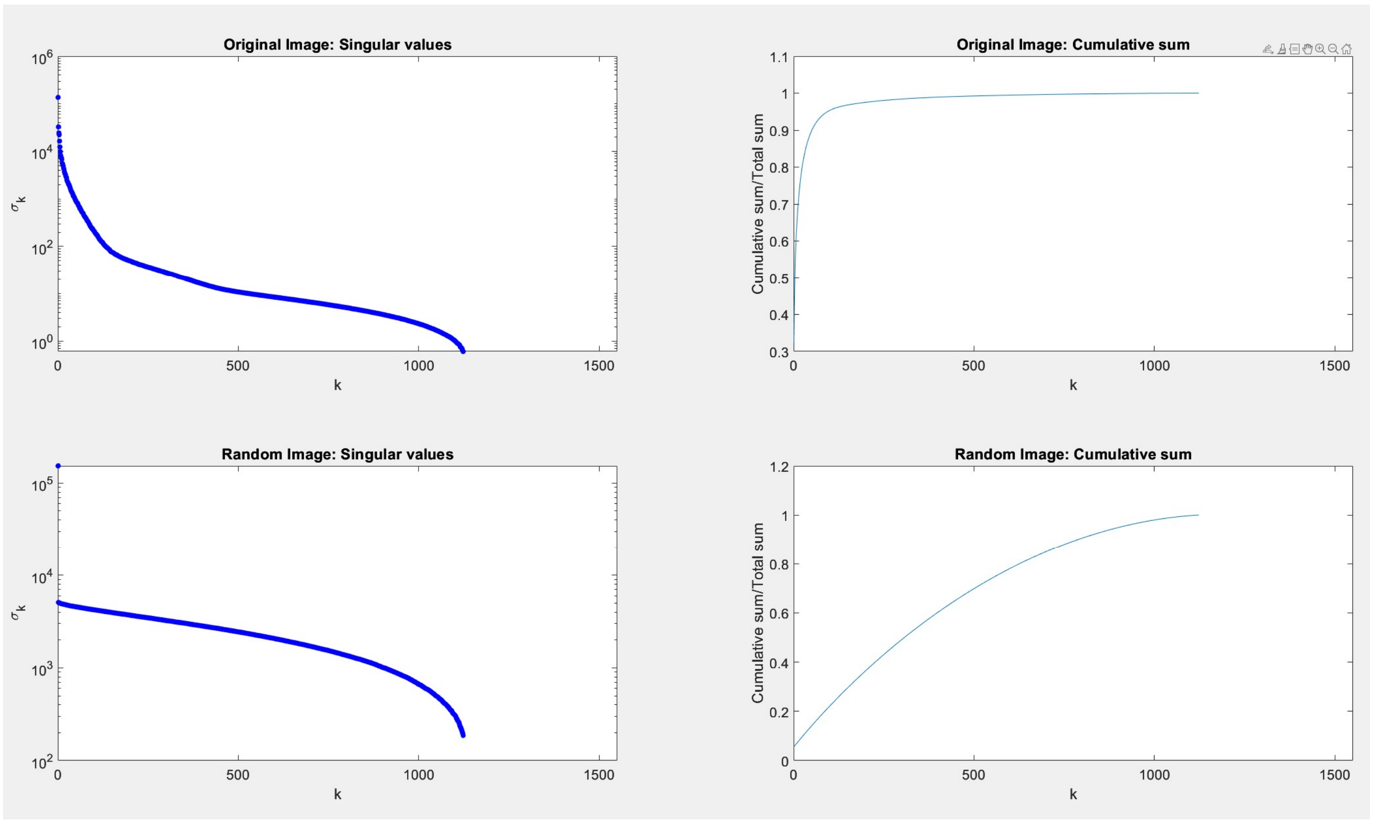
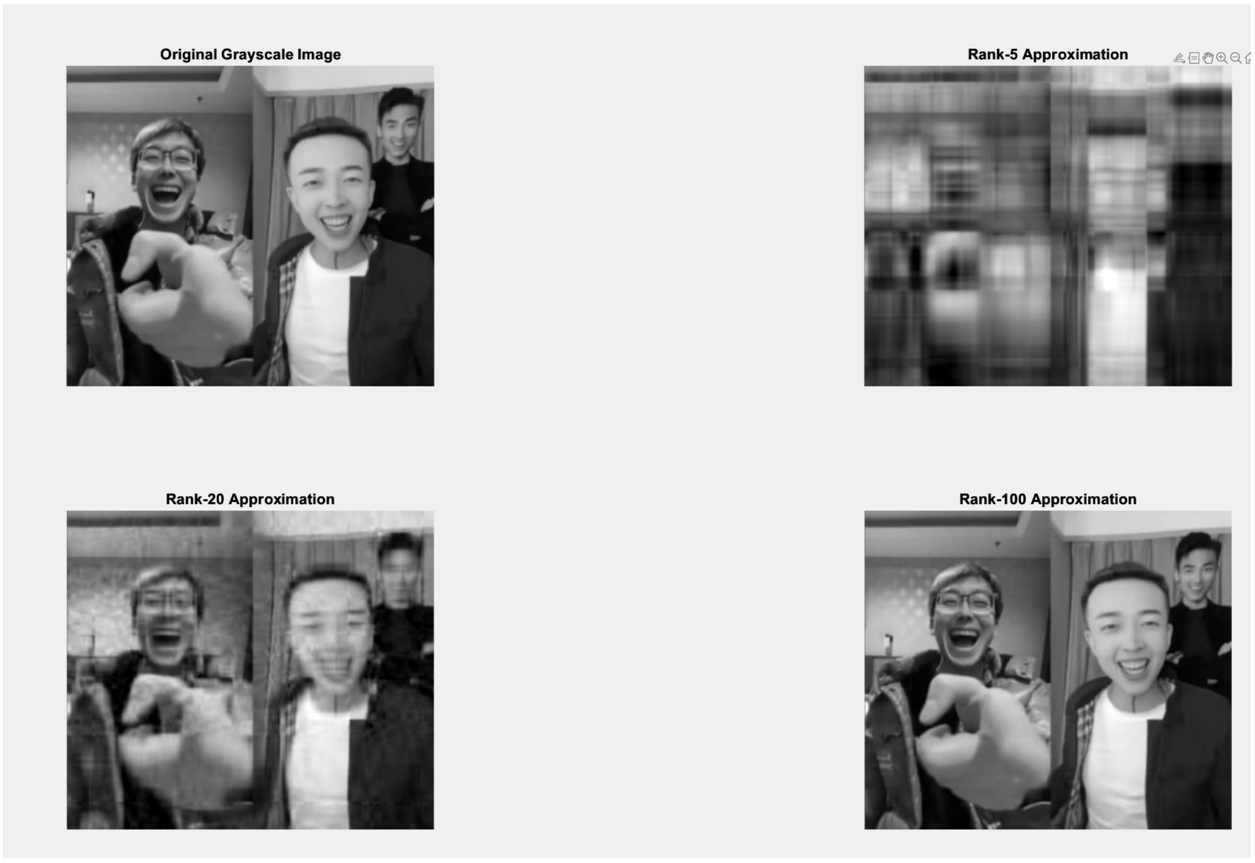


Figure 4
File
Edit
View
Insert
Tools
Desktop
Window
Help
Random grayscale ima


Singular values of the random image matrix
σ_k
k
10^2
10^3
10^4
10^5
0
200
400
600
800
1000
1200


Original Image: Singular values
σ_k
k
Original Image: Cumulative sum
Cumulative sum/Total sum
k
Random Image: Singular values
σ_k
k
Random Image: Cumulative sum
Cumulative sum/Total sum
k



Answer to the question:

Comparing the two plots of each image, we can see that the singular values of the original grayscale image are significantly higher, and the half-plot decreases faster compared to the random grayscale image, indicating that the original image has more important features or information. In addition, the original image indicates that fewer singular values can capture a larger percentage of the total variation in the original image. This suggests that the original image has more structure and consistency than the random image.

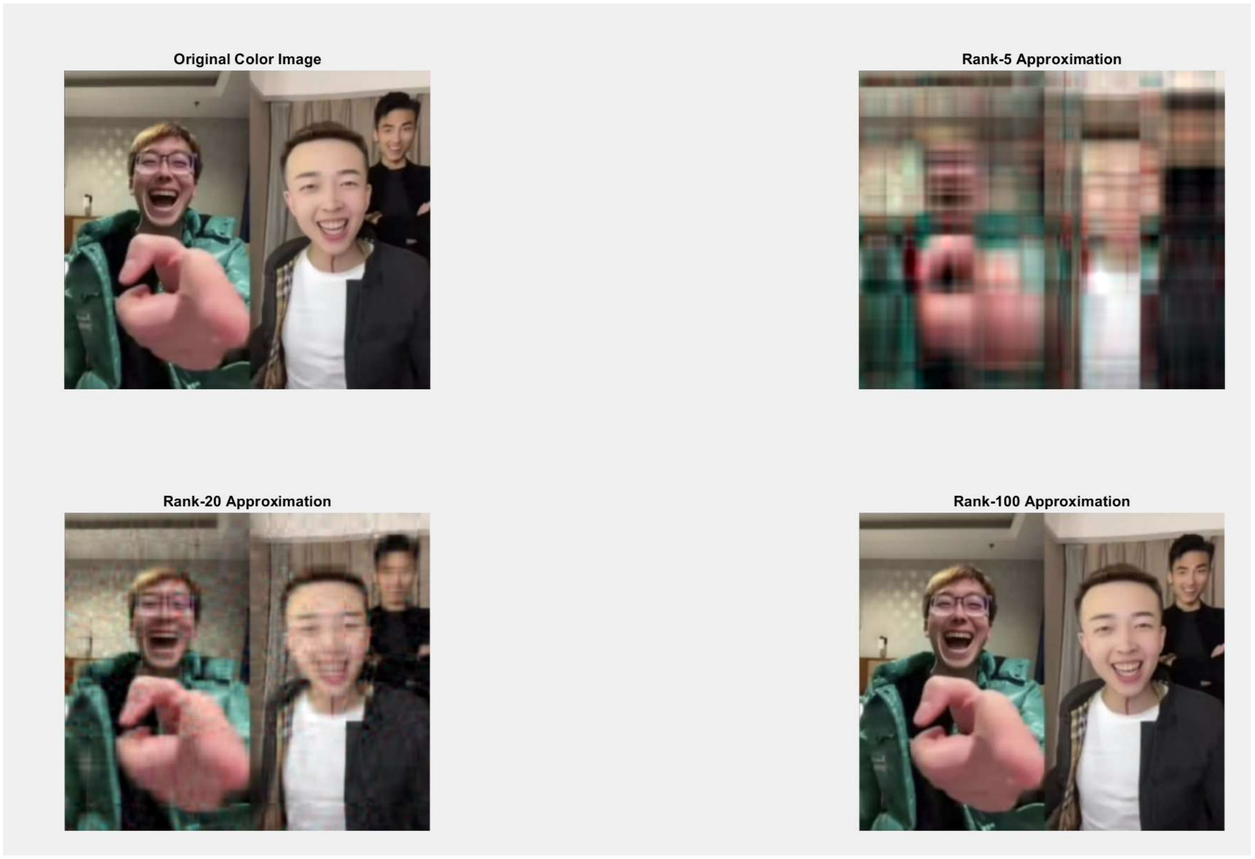
Extra Credit:

Summary of the code:

This code loads an original color image and calculates the SVD for each color channel (red, green, and blue) using the svd function. Then, it defines an array of k_values to experiment with different values of the rank-k approximation.

Next, it displays the original color image and compressed images for different values of k using the subplot function to create a 2x2 subplot. The reconstruct_color_image function is called to compute the color image using the rank-k approximations. This function takes in the SVD decomposition matrices and the chosen k value and outputs the reconstructed color image.

Finally, the code defines the reconstruct_color_image function that takes in the SVD matrices and a chosen k value and outputs the reconstructed color image by computing the matrix multiplication of U, S, and V for each color channel separately using cat to concatenate the three channels.



The extent to which the image is compressed can be seen by the difference in k values. w We can achieve a reasonable representation of the color image while retaining most of the image details by the rank-k approximation with k=20. I estimate that our space may be saved by about 50%.