Project Two Template

MAT-350: Applied Linear Algebra

Student Name: Edward Garcia

Date: 2/24/2024

Problem 1

Use the svd() function in MATLAB to compute A_1, the **rank-1 approximation of A.** Clearly state what A_1 is, rounded to 4 decimal places. Also, **compute** the root-mean square error (RMSE) between A and A_1.

Solution:

```
%code

% Define and create matrix A.
A = [1 2 3; 3 3 4; 5 6 7];

% Display the matrix A to the console.
fprintf('A =\n'); disp(A);
```

```
A =
     1     2     3
     3     3     4
     5     6     7
```

```
% Compute the Singular Value Decomposition (SVD) of A.
% U contains the left singular vectors, S is a diagonal matrix with singular
values
% and V contains the right singular vectors.
[U, S, V] = svd(A);

% Display the matrix U.
fprintf('U =\n'); disp(U);
```

```
U =
   -0.2904    0.9504   -0.1114
   -0.4644   -0.2418   -0.8520
   -0.8367   -0.1957    0.5115
```

```
% Display the matrix S.
fprintf('S =\n'); disp(S);
```

```
S =
   12.5318         0         0
         0    0.9122         0
         0         0    0.3499
```

```
% Display the matrix V.
fprintf('V =\n'); disp(V);
```

```
V =
   -0.4682   -0.8261   -0.3136
   -0.5581    0.0012    0.8298
   -0.6851    0.5635   -0.4616
```

```
% Construct the rank-1 approximation A1 of the original matrix A.
% This involves using only the first singular value and corresponding
vectors.
% A1 is the approximation of A with the highest significant singular value.
A1 = U(:,1:1) * S(1:1,1:1) * V(:,1:1)'
```

```
A1 = 3×3
    1.7039    2.0313    2.4935
    2.7243    3.2477    3.9867
    4.9087    5.8517    7.1832
```

```
% Display the Rank-1 Approximation.
fprintf('Rank-1 Approximation A1 =\n'); disp(A1);
```

```
Rank-1 Approximation A1 =
    1.7039    2.0313    2.4935
    2.7243    3.2477    3.9867
    4.9087    5.8517    7.1832
```

```
% The rank function calculates the number of linearly independent rows or
columns.
% Verify and display rank = 1.
fprintf('Rank of A1 = %d\n', rank(A1));
```

```
Rank of A1 = 1
```

```
%Calculate root-mean square error (RMSE) between A and A1.
% The Frobenius norm ('fro') computes the square root of the sum of the
absolute squares of its elements.
% RMSE_A1 quantifies the difference between the original matrix A and its
rank-1 approximation A1.
RMSE_A1 = norm(A - A1, 'fro') / (3 * 3)
```

```
RMSE_A1 = 0.1086
```

Problem 2

Use the svd() function in MATLAB to compute A_2, the **rank-2 approximation of** A. Clearly state what A_2 is, rounded to 4 decimal places. Also, **compute** the root-mean square error (RMSE) between A and A_2. Which approximation is better, A_1 or A_2? Explain.

Solution:

```
%code

% Construct the rank-2 approximation A2.
% This step involves using the first two singular values and the
corresponding singular vectors.
```

```
A2 = U(:,1:2) * S(1:2, 1:2) * V(:,1:2)'
```

```
A2 = 3×3
    0.9878    2.0324    2.9820
    2.9065    3.2474    3.8624
    5.0561    5.8515    7.0826
```

```
% Display A2 and RMSE.
% The resulting A2 matrix is expected to capture more of the original
matrix's structure compared to A1.
% Verify rank = 2 checks that A2 has a rank of 2 confirming that the
approximation uses two dimensions.
rank(A2)
```

```
ans = 2
```

```
% Compute the RMSE for A2.
% The Frobenius norm is used again to calculate the RMSE, providing a
measure of
% how closely A2 approximates the original matrix A.
% The RMSE is normalized by the total number of elements in A to provide an
average error per element.
RMSE_A2 = norm(A - A2, 'fro') / (3 * 3)
```

```
RMSE_A2 = 0.0389
```

```
% Determine out of A1 and A2 which is a better approximation.
% This decision is based on the RMSE values; the approximation with the
lower RMSE is considered better.
% Lower RMSE indicates a closer fit to the original matrix. The output will
be
% 'A2 is a better approximation than A1'.
if RMSE_A1 < RMSE_A2
    disp('A1 is a better approximation than A2.');
else
    disp('A2 is a better approximation than A1.');
end
```

```
A2 is a better approximation than A1.
```

Explain: In Problem 2 I took matrix A and used Singular Value Decomposition (SVD) to break it down into simpler forms of rank-1 and a rank-2 approximation. The objective is to see how much of the original matrix's structure and information could be retained with these lower-rank versions. After performing the SVD, I used the first singular value and vectors for the rank-1 approximation and the first two for the rank-2 approximation. With a decrease in "k" there is an increase in the RMSE, indicating a rise in the approximation error. In our case, moving from a rank-1 to a rank-2 approximation reduced the error, showcasing that including more singular values can lead to a more accurate representation of the original matrix. To determine how close these approximations were to the original matrix, I calculated the Root Mean Square Error (RMSE) for each. By comparing these RMSE values I concluded that between A1 and A2, A2 is the better approximation due to the less amount of error and it being closer to the original matrix.

Problem 3

For the 3×3 matrix A, the singular value decomposition is $A = USV'$ where $U = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3]$. Use MATLAB to **compute** the dot product $d_1 = dot(\mathbf{u}_1, \mathbf{u}_2)$.

Also, use MATLAB to **compute** the cross product $\mathbf{c} = cross(\mathbf{u}_1, \mathbf{u}_2)$ and dot product $d_2 = dot(\mathbf{c}, \mathbf{u}_3)$. Clearly state the values for each of these computations. Do these values make sense? **Explain**.

Solution:

```
%code

% Separate U into three separate vectors to determine the cross/dot product.
% U1, U2, and U3 are the columns of the U matrix from the SVD, representing
different singular vectors.
U1 = U(:,1)
```

```
U1 = 3×1
   -0.2904
   -0.4644
   -0.8367
```

```
U2 = U(:,2)
```

```
U2 = 3×1
    0.9504
   -0.2418
   -0.1957
```

```
U3 = U(:,3)
```

```
U3 = 3×1
   -0.1114
   -0.8520
    0.5115
```

```
% Compute the dot product d1 between U1 and U2
% The dot product measures the cosine of the angle between the two vectors,
indicating how parallel they are.
% For orthogonal vectors, expected in SVD, this should be close to 0.
d1 = dot(U1,U2)
```

```
d1 = 1.6653e-16
```

```
% Compute the cross product c between U1 and U2
% The cross product results in a vector that is perpendicular to both U1 and
U2 in a 3-dimensional space.
% This utilizes SVD where U1 and U2 are orthogonal, implying c should be
orthogonal to both.
c = cross(U1, U2)
```

```
c = 3×1
   -0.1114
   -0.8520
    0.5115
```

```
% Calculate dot product d2 between vector c and U3
% This dot product assesses the orthogonality of c from cross of U1 and U2,
with U3. This should
% also be close to 0, indicating orthogonality.
d2 = dot(c,U3)
```

```
d2 = 1.0000
```

Explain: To solve this I first separated the matrix *U* into its first three column vectors, labeled *U*1, *U*2, and *U*3 in order to analyze them individually. Next I calculated the dot product of *U*1 and *U*2, to verify their orthogonality. A result near zero indicates that the vectors are orthogonal. Then I found the cross product of *U*1 and *U*2, resulting in a new vector *c*. This vector is orthogonal to both *U*1 and *U*2, lying in the plane they span. Then I calculated the dot product of vector *c* with *U*3, to analyze the geometric relations and test the orthogonality between *c* and U3. The dot product *d*1 between the first two columns of *U*, *u*1 and *u*2, should ideally be zero if *U* is an orthogonal matrix, as SVD guarantees orthogonal column vectors in *U*. The answers seem reasonable to me due to the cross product *c* of these vectors *u*1 and *u*2 and how it should give a vector orthogonal to both, and the dot product *d*2 of this new vector with the third column *u*3 should also be zero, confirming that *u*3 is orthogonal to the plane formed by *u*1 and *u*2. The code computes these products and displays their values, which should be close to zero, due to the orthogonality property of the vectors in *U*.

Problem 4

Using the matrix $U = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3]$, **determine whether or not the columns of U span $\mathbb{R}^3$. Explain your approach.**

Solution:

```
%code

% My first step is to put the column vectors back into U.
% This will reconstruct the matrix U using its column vectors U1, U2, and U3.
% These columns are the singular vectors obtained from the SVD of some
matrix.
U = [U1 U2 U3]
```

```
U = 3×3
   -0.2904    0.9504   -0.1114
   -0.4644   -0.2418   -0.8520
   -0.8367   -0.1957    0.5115
```

```
% I then used the rref function to compute the Reduced Row Echelon Form of U.
% If the columns of U span R3, the reduced matrix will have 3 pivot columns.
reducedU = rref(U)
```

```
reducedU = 3×3
     1     0     0
     0     1     0
     0     0     1
```

```
% Next I checked the rank of the reduced U matrix.
```

```
% The rank of a matrix indicates the maximum number of linearly independent
column vectors.
% For the columns of U to span R3, the rank of U must be 3, indicating it
has full rank.
rank(reducedU)
```

```
ans = 3
```

```
% Then I checked if the RREF of U is the identity matrix.
% This is done by comparing reducedU to the identity matrix of size 3,
eye(3).
% If they are the same, it means that each column of U is a pivot column,
% which indicates that the columns are linearly independent. The program
% will output "The columns of U span R^3.
if isequal(reducedU, eye(3))
    % If reducedU is the identity matrix, then the columns of U span R^3.
    disp('The columns of U span R^3.');
else
    % If reducedU is not the identity matrix, then the columns of U do not
span R^3.
    disp('The columns of U do not span R^3.');
end
```

```
The columns of U span R^3.
```

Explain: There are 3 pivot columns in the matrix after rref, so the reduced form has no zero rows and the U columns go to R^3. I first compared the reduced form `reducedU` with the identity matrix of the same size using `isequal`. If they match, then the `reducedU` must have full rank, meaning all its columns are pivot columns and they span R^3. If not, then it suggests that *U* does not have full rank and its columns do not span the space. If the reduced echelon form does not have a zero row, then it spans R^3. This is verified by looking at the reduced matrix which does not have a zero row, which means it spans R^3.

Problem 5

Use the MATLAB imshow() function to load and display the image A stored in the image.mat file, available in the Project Two Supported Materials area in Brightspace. For the loaded image, **derive the value of** k that will result in a compression ratio of $CR \approx 2$. For this value of k, **construct the rank-*k* approximation of the image**.

Solution:

```
%code

% Load the image data from 'Image.mat' into the workspace variable 'A'
load Image.mat;

% Display the original image contained in variable 'A' and determine the
dimensions of the image
% which are (M = 2583 x N = 4220)
figure;
imshow(A);
```

```
% Compute the Singular Value Decomposition (SVD) of the image
% The svd function decomposes the image matrix 'A' into three matrices U, S,
and V
% 'U' and 'V' contain the left and right singular vectors and 'S' contains
the singular values
[U, S, V] = svd(double(A))
```

```
U = 2583×2583
   -0.0106   -0.0360   -0.0006    0.0032   -0.0032    0.0041   -0.0066    0.0022 ···
   -0.0105   -0.0361   -0.0006    0.0030   -0.0035    0.0049   -0.0062    0.0020
   -0.0105   -0.0362   -0.0006    0.0034   -0.0037    0.0042   -0.0064    0.0025
   -0.0105   -0.0362   -0.0009    0.0029   -0.0035    0.0052   -0.0056    0.0028
   -0.0106   -0.0361   -0.0011    0.0034   -0.0035    0.0046   -0.0061    0.0022
   -0.0106   -0.0363   -0.0011    0.0031   -0.0030    0.0049   -0.0061    0.0031
   -0.0106   -0.0364   -0.0008    0.0032   -0.0032    0.0043   -0.0057    0.0033
   -0.0106   -0.0365   -0.0006    0.0029   -0.0033    0.0050   -0.0052    0.0031
   -0.0106   -0.0366   -0.0007    0.0033   -0.0031    0.0040   -0.0053    0.0033
   -0.0106   -0.0368   -0.0009    0.0030   -0.0034    0.0044   -0.0052    0.0032
      ⋮

S = 2583×4220
10^5 ×
    4.0600         0         0         0         0         0         0         0 ···
         0    0.8702         0         0         0         0         0         0
         0         0    0.4169         0         0         0         0         0
         0         0         0    0.4104         0         0         0         0
         0         0         0         0    0.3405         0         0         0
         0         0         0         0         0    0.2992         0         0
         0         0         0         0         0         0    0.2550         0
         0         0         0         0         0         0         0    0.2268
```

```
        0         0         0         0         0         0         0         0
        0         0         0         0         0         0         0         0
     ⋮
```

```
V = 4220×4220
   -0.0130    0.0044   -0.0358    0.0028   -0.0085    0.0177    0.0128   -0.0163 ···
   -0.0130    0.0045   -0.0357    0.0024   -0.0079    0.0184    0.0134   -0.0162
   -0.0130    0.0045   -0.0359    0.0025   -0.0078    0.0181    0.0124   -0.0168
   -0.0130    0.0046   -0.0361    0.0030   -0.0087    0.0185    0.0125   -0.0158
   -0.0130    0.0045   -0.0366    0.0032   -0.0095    0.0182    0.0116   -0.0149
   -0.0129    0.0046   -0.0369    0.0034   -0.0095    0.0185    0.0112   -0.0151
   -0.0130    0.0047   -0.0372    0.0046   -0.0104    0.0178    0.0103   -0.0141
   -0.0130    0.0048   -0.0377    0.0045   -0.0097    0.0179    0.0108   -0.0145
   -0.0130    0.0046   -0.0375    0.0049   -0.0094    0.0170    0.0102   -0.0147
   -0.0130    0.0045   -0.0379    0.0043   -0.0090    0.0166    0.0095   -0.0142
     ⋮
```

```
% Calculate the value of k for a desired Compression Ratio (CR) of 2
% The formula for the compression ratio (CR) is given, I found solving for k
gives k = 801 to achieve CR = 2
CR2 = (2583 * 4220) / (801 *(2583 + 4220 + 1))
```

```
CR2 = 2.0000
```

```
% Construct the rank-801 approximation of the image 'A'
% This approximation uses the first 801 singular values and vectors
A801 = U(:,1:801) * S(1:801, 1:801) * V(:,1:801)'
```

```
A801 = 2583×4220
   26.4896   27.2541   30.5810   28.9530   23.3828   25.7705   35.1037   29.3968 ···
   32.6831   34.0733   28.4258   30.5682   27.6882   28.4547   35.6629   31.2002
   35.5230   30.8250   18.7994   19.9743   19.8952   17.0400   24.6764   26.0911
   33.6440   29.7702   26.1173   29.8858   26.5095   15.9739   24.7017   25.8886
   27.7165   26.0012   30.5018   36.7547   35.3434   29.8915   34.5078   25.1416
   27.3996   25.5183   26.6092   29.4463   25.5795   28.8615   33.9147   23.0624
   32.0484   32.4889   27.5089   22.7250   20.3684   25.0803   33.3388   26.7700
   26.0954   32.2044   27.4183   18.1894   21.2836   28.1417   31.7244   26.2813
   23.2187   25.5412   22.1689   25.1362   29.2165   30.3222   34.5845   30.5812
   21.3048   20.9797   19.1568   25.5860   26.9030   24.8220   30.0068   30.1700
     ⋮
```

```
% Verify the rank of the approximation is 801
% The rank function confirms that the constructed approximation holds the
desired rank of 801
rank(A801)
```

```
ans = 801
```

```
% Convert the approximation back to uint8 and display the image
% This conversion is needed due to the original image likely being in uint8
format
A801 = uint8(round(A801))
```

```
A801 = 2583x4220 uint8 matrix
   26   27   31   29   23   26   35   29   33   36   30   26   32   27   28   34 ···
   33   34   28   31   28   28   36   31   30   27   22   28   40   34   34   34
   36   31   19   20   20   17   25   26   30   23   22   27   35   35   37   34
```

```
34    30    26    30    27    16    25    26    28    25    22    24    33    35    32    30
28    26    31    37    35    30    35    25    25    25    23    27    33    33    27    27
27    26    27    29    26    29    34    23    22    27    29    34    33    33    28    26
32    32    28    23    20    25    33    27    27    29    35    33    27    28    31    29
26    32    27    18    21    28    32    26    29    32    36    31    28    31    34    34
23    26    22    25    29    30    35    31    30    35    34    27    20    24    34    34
21    21    19    26    27    25    30    30    28    34    29    20    16    22    29    32
 ⋮
```

Explain: The goal of this problem is to compress an image using the Singular Value Decomposition (SVD) method, working with an image I uploaded from a file named 'Image.mat'. The process first starts by me displaying the original, uncompressed image file and determining its dimensions which are 2583 x 4220 pixels. I then used the SVD method on the image to find the value of k, which I calculated to be 801 that makes the Compression Ratio equal to 2. Next I used the rank k approximation to verify the rank of 801 which was the correct approximation. I then converted the image to go from an 8-bit unsigned integer format to a double precision format to enable the SVD function, so the image is able to display. After analyzing the data I concluded that the k value of A801 was able to achieve a compression ratio of 2.

Problem 6

Display the image and compute the root mean square error (RMSE) between the approximation and the original image. Make sure to include a copy of the approximate image in your report.

Solution:

```
%code

% Display the Rank 801 approximation image.
% This utilizes the imshow function to display the rank-801 approximation of
the original image.
figure;
imshow(A801);
```

```
% Determine the Root Mean Square Error (RMSE) between the original image A
and the approximation A801
% The RMSE quantifies the error between the original image and its
approximation.
% The result is normalized by the square root of the total number of pixels
in the image to get the average error per pixel.
RMSE801 = norm(double(A) - double(A801), 'fro') / sqrt (2583 * 4220)
```

```
RMSE801 = 3.1664
```

Problem 7

Repeat Problems 5 and 6 for $CR \approx 10$, $CR \approx 25$, and $CR \approx 75$. **Explain** what trends you observe in the image approximation as CR increases and provide your recommendation for the best CR based on your observations. Make sure to include a copy of the approximate images in your report.

Solution:

```
%code

% Calculate the value of k for CR = 10
% This step involves determining the value of k that results in a
compression ratio of approximately 10.
% The compression ratio formula is applied to find the value of k.
CR10 = (2583 * 4220) / (160 * (2583 + 4220 + 1))
```

```
CR10 = 10.0127
```

```
% Compute the rank-160 approximation of the image
% This approximation uses the first 160 singular values and vectors to
construct a compressed version of the original image.
A160 = U(:,1:160) * S(1:160, 1:160) * V(:,1:160)'
```

A160 = 2583×4220

29.0415	29.0508	30.3481	28.6840	28.4125	25.8146	26.5310	25.2709	...
29.4990	29.1068	29.9563	28.2658	28.2759	26.2124	27.5634	26.0996	
26.0110	25.7091	26.5596	24.7097	24.6572	23.0067	24.2242	22.2803	
28.3636	28.5688	29.5533	27.3636	27.6176	25.7857	27.0061	24.7262	
28.5023	28.4627	29.3196	27.9087	28.6184	26.5532	27.9189	25.7023	
26.6380	26.7335	27.2446	25.9346	26.6994	25.3844	27.4323	25.3185	
27.7105	26.9342	27.3695	25.8611	26.2616	24.8003	26.8496	24.7796	
26.6047	26.2936	26.6370	25.8935	26.9915	25.8782	27.7421	25.4926	
25.4761	25.3280	25.3720	24.6828	25.3964	24.6848	26.1862	23.9901	
24.0066	23.4106	23.1904	22.8845	23.3535	22.7318	23.9445	22.0238	
⋮								

```
% Verify the rank of the approximation.
rank(A160)
```

```
ans = 160
```

```
% Convert the approximation to uint8 to display image.
A160 = uint8(round(A160))
```

A160 = 2583x4220 uint8 matrix

29	29	30	29	28	26	27	25	28	27	26	25	28	27	27	28	...
29	29	30	28	28	26	28	26	28	27	26	25	28	27	28	29	
26	26	27	25	25	23	24	22	24	24	24	23	26	25	27	28	
28	29	30	27	28	26	27	25	26	26	26	24	28	27	27	28	
29	28	29	28	29	27	28	26	27	26	27	25	28	27	27	28	
27	27	27	26	27	25	27	25	27	26	26	24	26	26	27	27	
28	27	27	26	26	25	27	25	26	25	26	24	26	26	27	27	
27	26	27	26	27	26	28	25	27	25	25	23	25	24	25	25	
25	25	25	25	25	25	26	24	25	24	24	21	23	23	23	23	
24	23	23	23	23	23	24	22	23	21	21	19	21	21	22	23	
⋮																

```
% Display the rank-160 approximation image.
figure;
imshow(A160);
```

```
% Determine the Root Mean Square Error (RMSE) between A and 160
RMSE160 = norm(double(A) - double(A160), 'fro') / (2583 * 4220)
```

```
RMSE160 = 0.0025
```

```
% Calculate the value of k so CR = 25. I found that if k = 64 then the
% Compression Ratio will equal approximately 25
CR25 = (2583 * 4220) / (64 * (2583 + 4220 + 1))
```

```
CR25 = 25.0318
```

```
% Now that the value of k has been found, the rank A64 will need to be
calculated.
A64 = U(:,1:64) * S(1:64, 1:64) * V(:,1:64)'
```

```
A64 = 2583×4220
   24.9993   24.3447   22.0100   22.2062   21.9511   21.9226   23.0593   23.7963 ···
   25.2032   24.4691   22.2505   22.4141   22.3977   22.5232   23.5883   24.2551
   23.7727   23.0131   20.8102   21.1738   20.9707   21.3994   22.4306   23.1625
   24.3150   23.8209   21.6649   21.8529   21.8114   22.1064   23.1593   23.8830
   24.8134   24.0232   21.8824   22.0873   22.2178   22.3894   23.4309   24.2761
   24.1187   23.4903   21.2471   21.5617   21.6885   21.8974   22.7524   23.5456
   24.0810   23.2159   21.0505   21.3372   21.4513   21.7045   22.5713   23.3211
   25.0993   24.3390   22.2436   22.6802   23.0230   23.0038   23.7316   24.4781
   23.7390   22.9025   20.8467   21.1917   21.5129   21.5312   22.1834   23.0650
   22.2805   21.5541   19.4265   20.0297   20.3143   20.3821   20.8926   21.9840
      ⋮
```

```
% Use the rank function to verify if the rank = 64
```

```
rank(A64)
```

```
ans = 64
```

```
% Change back to uint8 and display the image (A64)
A64 = uint8(round(A64))
```

```
A64 = 2583x4220 uint8 matrix
   25   24   22   22   22   22   23   24   25   26   27   27   29   29   28   30 ...
   25   24   22   22   22   23   24   24   25   26   27   27   30   29   29   31
   24   23   21   21   21   21   22   23   24   25   26   26   29   28   28   29
   24   24   22   22   22   22   23   24   25   26   27   27   29   28   28   29
   25   24   22   22   22   22   23   24   25   26   27   27   29   29   28   29
   24   23   21   22   22   22   23   24   24   25   26   26   28   28   27   28
   24   23   21   21   21   22   23   23   24   25   26   26   28   27   26   27
   25   24   22   23   23   23   24   24   25   26   27   27   29   28   27   28
   24   23   21   21   22   22   22   23   24   25   26   25   27   27   26   27
   22   22   19   20   20   20   21   22   23   24   25   25   27   26   26   26
     ⋮
```

```
% Display image (A64) with a compression rate 25 and rank 64.
figure;
imshow(A64);
```

```
% Determine the Root Mean Square Error (RMSE) between A and 64
RMSE64 = norm(double(A) - double(A64),'fro') / (2583 * 4220)
```

```
RMSE64 = 0.0037
```

```
% Next Calculate the value of k so CR = 75. I found that if k = 64 then the
```

```
% Compression Ratio will equal approximately 25
CR75 = (2583 * 4220)/(21*(2583 + 4220 + 1))
```

```
CR75 = 76.2875
```

```
% Now that the value of k has been found, the rank A21 will need to be
calculated.
A21 = U(:,1:21) * S(1:21, 1:21) * V(:,1:21)'
```

```
A21 = 2583×4220
   26.2163   25.6337   24.9907   25.1005   25.5472   26.3732   27.4089   27.1685 ···
   26.5129   25.9007   25.2598   25.3699   25.8707   26.7271   27.7391   27.5112
   26.3007   25.6709   25.0385   25.1923   25.6988   26.5671   27.6553   27.4353
   27.0439   26.5387   25.9096   26.0592   26.5085   27.3315   28.3511   28.1032
   26.9099   26.2511   25.6362   25.6966   26.2742   27.0451   28.0548   27.8644
   26.6065   25.9442   25.3392   25.4571   26.0535   26.8352   27.8248   27.6089
   26.8196   26.1732   25.5741   25.6539   26.1657   26.8895   27.8945   27.6443
   27.2339   26.6413   26.0681   26.0830   26.6782   27.3193   28.2429   27.9771
   26.0167   25.3796   24.8159   24.7942   25.3968   25.9820   26.8889   26.6155
   27.3611   26.7141   26.1752   26.1811   26.7707   27.3992   28.3091   28.0254
     ⋮
```

```
% Use the rank function to verify if the rank = 21
rank(A21)
```

```
ans = 21
```

```
% Change back to uint8 and display the image (A21)
A21 = uint8(round(A21))
```

```
A21 = 2583x4220 uint8 matrix
   26   26   25   25   26   26   27   27   28   28   29   29   30   31   30   32 ···
   27   26   25   25   26   27   28   28   29   29   30   29   30   31   31   32
   26   26   25   25   26   27   28   27   29   29   29   29   30   31   31   32
   27   27   26   26   27   27   28   28   29   29   30   30   31   32   31   33
   27   26   26   26   26   27   28   28   29   29   30   30   31   32   31   33
   27   26   25   25   26   27   28   28   29   29   30   29   30   31   31   32
   27   26   26   26   26   27   28   28   29   29   30   29   30   31   31   32
   27   27   26   26   27   27   28   28   29   29   30   30   31   32   32   33
   26   25   25   25   25   26   27   27   28   28   29   28   30   30   30   32
   27   27   26   26   27   27   28   28   29   29   30   30   31   32   31   33
    ⋮
```

```
% Display image (A21) with a compression rate 75 and rank 21. This image
% seems to be the most distorted out of all of them.
figure;
imshow(A21);
```

Explain: As k decreases it leads to a higher compression ratio, and the quality of the image approximation decreases. This is noticed when image details are lost, and the image becomes more distorted. The RMSE values increase with higher compression ratios, indicating a greater error between the original and approximated images. As *k* increases, the approximation of the image becomes more accurate and closer to the original image. This is due to the fact that a higher *k* value in Singular Value Decomposition (SVD) allows for more singular values to be included in the reconstruction of the image. With a lower *k* value as in the example *k* = 21 for CR = 75, the approximation results in a much more distorted and less detailed image. This is because fewer singular values are used. I noticed when *k* starts to increase in the examples of *k* = 64 for CR = 25 and *k* = 160 for CR = 10), more details in the image are preserved, leading to higher quality approximations that more closely resemble the original image. I utilized different values for k in order to determine the compression ratio using the given compression ratios CR = 10, 25 and 75. After further analyzation my recommendation would be image 10 (A160). I found that in the image where the CR = 10 and k = 160 had the best image quality. The image (A21) with the CR of 75 and a rank of 21 I found to be the most distorted image with the lowest of quality.