Project Two

MATAT-350: Applied Linear Algebra

Ryan Hatch

October 22 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:

```
% Define the matrix A
A = [1 2 3;
     3 3 4;
     5 6 7];

% Compute the SVD of A
[U, S, V] = svd(A);

% Extract the first singular value and corresponding singular vectors
sigma1 = S(1,1);
u1 = U(:,1);
v1 = V(:,1);

% Construct the rank-1 approximation A1
A1 = sigma1 * u1 * v1';

% Round A1 to 4 decimal places
A1_rounded = round(A1, 4);
disp('A1 (rounded to 4 decimal places):');
```

```
A1 (rounded to 4 decimal places):
```

```
disp(A1_rounded);
```

```
    1.7039    2.0313    2.4935
    2.7243    3.2477    3.9867
    4.9087    5.8517    7.1832
```

```

% Compute the RMSE between A and A1
error = A - A1;
RMSE = sqrt(mean(error(:).^2));
disp('RMSE between A and A1:');
```

```
RMSE between A and A1:
```

```
disp(RMSE);
```

```
    0.3257
```

Explain:

I Used the svd() function to compute A1, the rank-1 approximation of A. I then displayed what A1 is, rounded to 4 decimal places and also computed the RMSE between A and A1.

Rank-1 approximation reduces data size by keeping the most significant singular value, which captures the main structure but omits detailed information. This technique significantly lowers data storage and transmission needs. However, it may not be ideal for high-quality reconstructions as it loses crucial information, shown by an RMSE of 0.3257, indicating a fairly notable difference from the original matrix.

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:

```
% Extract the first two singular values and corresponding singular vectors
sigma1 = S(1,1);
sigma2 = S(2,2);
u1 = U(:,1);
u2 = U(:,2);
v1 = V(:,1);
v2 = V(:,2);

% Construct the rank-2 approximation A2
A2 = sigma1 * u1 * v1' + sigma2 * u2 * v2';

% Round A2 to 4 decimal places
A2_rounded = round(A2, 4);
disp('A2 (rounded to 4 decimal places):');
```

```
A2 (rounded to 4 decimal places):
```

```
disp(A2_rounded);
```

```
    0.9878    2.0324    2.9820
    2.9065    3.2474    3.8624
    5.0561    5.8515    7.0826
```

```

% Compute the RMSE between A and A2
error2 = A - A2;
RMSE2 = sqrt(mean(error2(:).^2));
```

```
disp('RMSE between A and A2:');
```

```
RMSE between A and A2:
```

```
disp(RMSE2);
```

```
    0.1166
```

```

% Determine which approximation is better
if RMSE2 < RMSE
    disp('A2 is a better approximation than A1 because it has a lower RMSE.');
else
    disp('A1 is a better approximation than A2.');
end
```

```
A2 is a better approximation than A1 because it has a lower RMSE.
```

Explain:

The rank-2 approximation maintains two singular values, which enhances the reconstruction quality with a lower RMSE of 0.1166. This approach helped to capture more detail than rank-1, which ended up having a better balance between compression and quality. Nonetheless, this method is perfect for applications that require higher fidelity but not full rank. In business terms, it helps reduce storage costs while still delivering satisfactory results to users or customers.

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:

```
% Extract the singular vectors
u1 = U(:,1);
u2 = U(:,2);
u3 = U(:,3);

% Compute the dot product d1 = dot(u1, u2)
d1 = dot(u1, u2);
disp('Dot product d1 = u1 • u2:');
```

```
Dot product d1 = u1 • u2:
```

```
disp(d1);
```

```
   1.6653e-16
```

```
% Compute the cross product c = cross(u1, u2)
c = cross(u1, u2);
disp('Cross product c = u1 × u2:');
```

```
Cross product c = u1 × u2:
```

```
disp(c);
```

```
   -0.1114
   -0.8520
    0.5115
```

```

% Compute the dot product d2 = dot(c, u3)
d2 = dot(c, u3);
disp('Dot product d2 = c • u3:');
```

```
Dot product d2 = c • u3:
```

```
disp(d2);
```

```
    1.0000
```

Explain:

The dot product between vectors u1 and u2 is practically zero (1.6653e-16), showing they're almost at right angles to each other. This is crucial for singular value decomposition, as it means these vectors are independent, capturing different bits of information when breaking down the matrix. Also, the cross product of u1 and u2 creates a new vector that's orthogonal to them both. None the less, in the end, when this result's dot product with u3 turns out to be 1, it confirms u3 is also orthogonal which makes sense and ends up lining up just right in a 3D space. This kind of orthogonality is very important for keeping everything stable and efficient, especially in applications like compression where any overlap would be detrimental.

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:

```
%Compute the determinant of U:
determinant = det(U);
disp('Determinant of U:');
```

```
Determinant of U:
```

```
disp(determinant);
```

```
    1.0000
```

Explain:

The determinant of U is 1, which means U is invertible. This is important because it helps to show that the columns of U are linearly independent and span R^3. For SVD, this is essential because it makes sure that no information gets lost due to overlapping basis vectors, and the decomposition stays accurate for reconstructing or approximating the matrix. If the columns didn't span the space, I could consiquently lose important data during compression.

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:

```
% Load the image
load('MAT 350 Project Two MATLAB Image.mat'); % The image matrix is assumed to be
stored in variable A

% Display the original image
imshow(A, []);

title('Original Image');
```

Original Image

```
% Get the size of the image:
```

```
[m, n] = size(A);

%%
%% Compute the value of k for a compression ratio CR ≈ 2
%%

% Calculate k:
CR = 2; % Desired compression ratio
k = round((m * n) / (CR * (m + n + 1)));
disp(['Value of k for CR ≈ 2: ', num2str(k)]);
```

```
Value of k for CR ≈ 2: 801
```

```
%%
% Construct the r-rank k approximation:
%%

% Compute the SVD of the image
[U, S, V] = svd(double(A));

% Keep the first k singular values/vectors
U_k = U(:, 1:k);
S_k = S(1:k, 1:k);
V_k = V(:, 1:k);

% Reconstruct the approximated image
A_k = U_k * S_k * V_k';

% Display the approximated image
figure;
imshow(uint8(A_k), []);

title(['Rank-', num2str(k), ' Approximation (CR ≈ 2)']);
```

Rank-801 Approximation (CR ≈ 2)

Explain:

By calculating k using the given formula and the image dimensions, I was able to find the value of k that gave a compression ratio of about 2. I then reconstructed the image using only the first k singular values and vectors, and compressing the image data effectively.

Problem 6

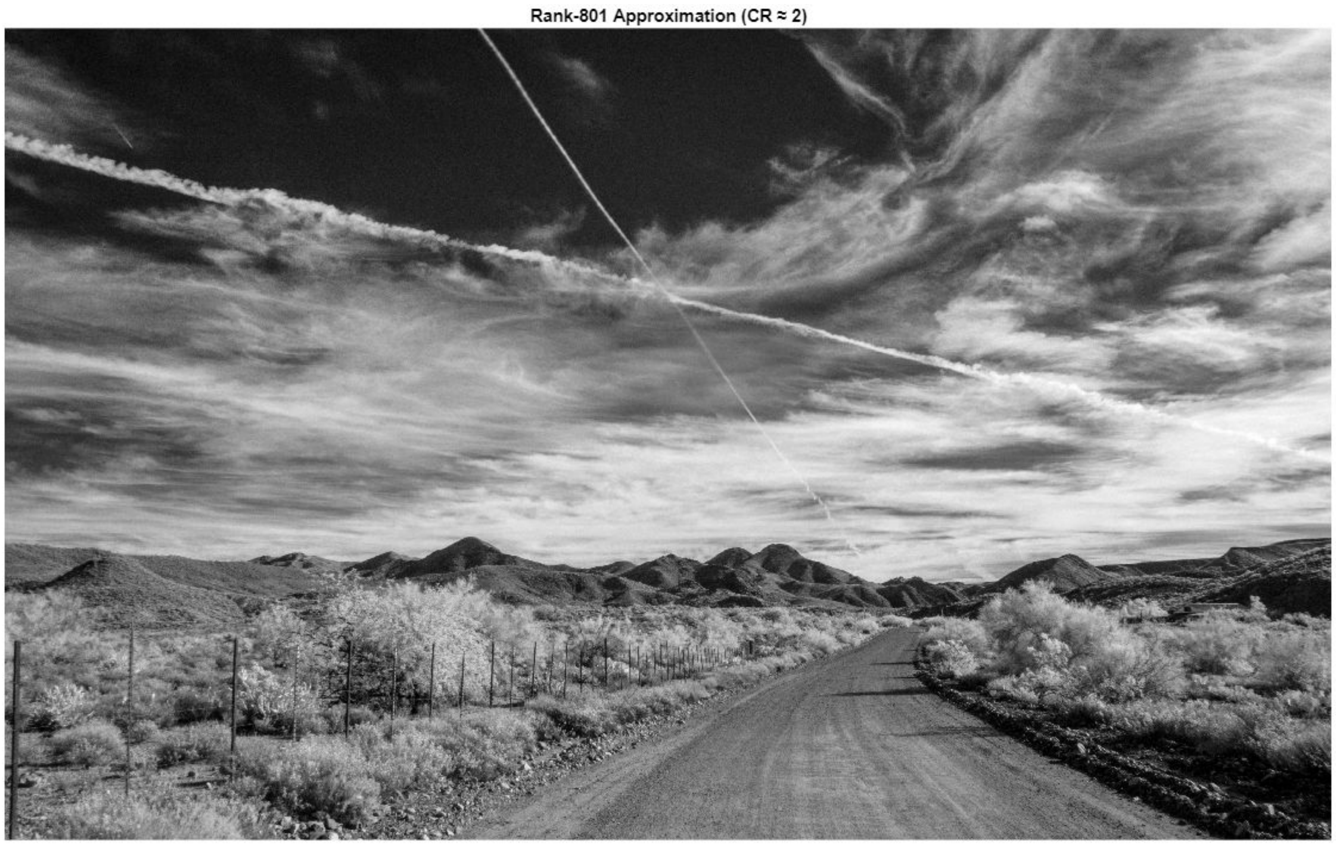
Display the image and compute the root mean square error (RMSE) between the approximation and the original image. Make sure to include a copy of the approximate image in your report.

Solution:

```
% Compute the RMSE between the original imagage and the approximated image.

% Compute the error matrix
error_image = double(A) - A_k;

% Compute RMSE
RMSE = sqrt(mean(error_image(:).^2));
disp(['RMSE between original and rank-', num2str(k), ' approximation: ',
num2str(RMSE)]);
```

```
RMSE between original and rank-801 approximation: 3.1539
```

```
% Display the approximate image again
figure;
imshow(uint8(A_k), []);
title(['Rank-', num2str(k), ' Approximation (CR ≈ 2)']);
```



Problem 7

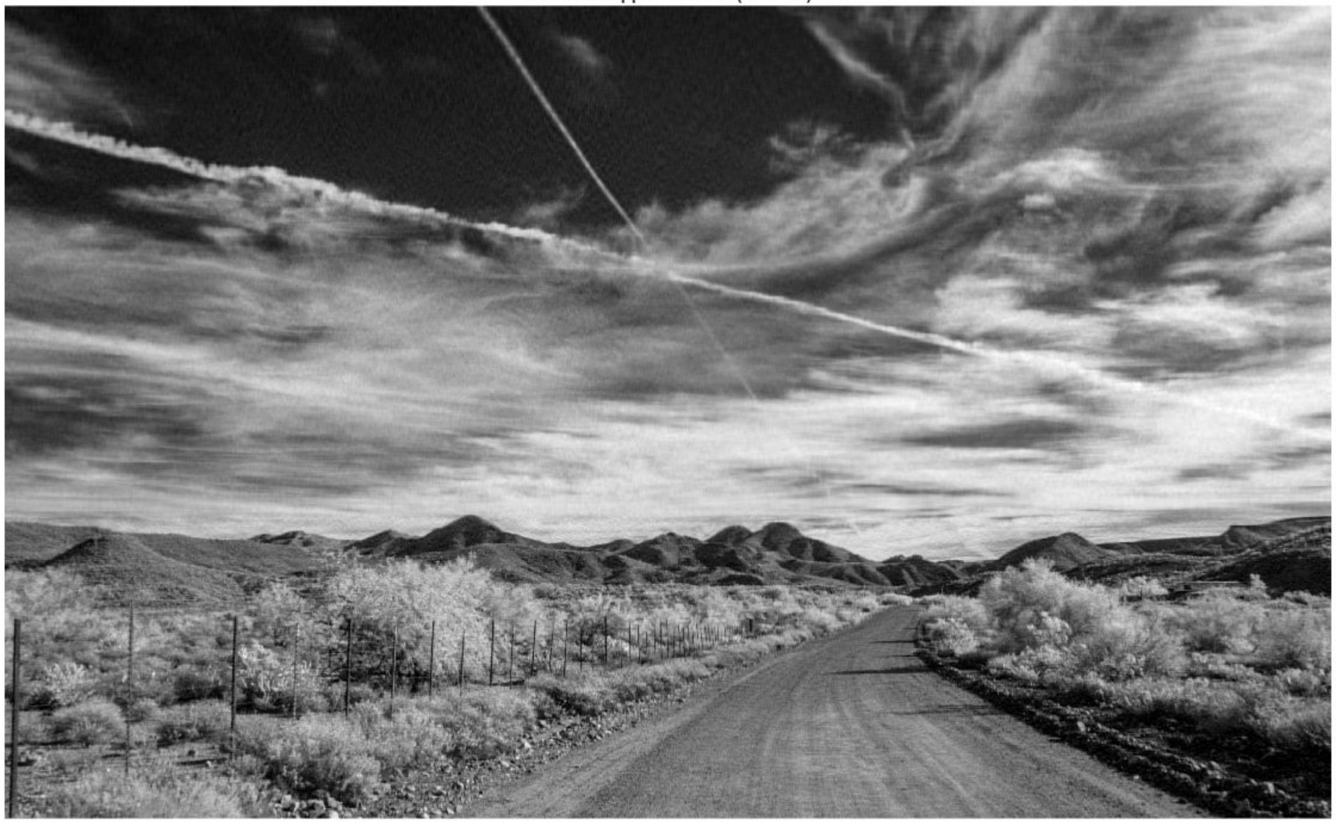
Repeat Problems 5 and 6 for $CR \approx 10$, $CR \approx 25$, and $CR \approx 75$. **Explain** what trends you observe in the image approximation as CR increases and provide your recommendation for the best CR based on your observations. Make sure to include a copy of the approximate images in your report.

Solution:

```
%Repeat the process for different compression ratios.

compression_ratios = [10, 25, 75];

for CR = compression_ratios
    % Calculate k for the given CR
    k = round((m * n) / (CR * (m + n + 1)));
    disp(['Value of k for CR ≈ ', num2str(CR), ': ', num2str(k)]);

    % Reconstruct the image
    U_k = U(:, 1:k);
    S_k = S(1:k, 1:k);
    V_k = V(:, 1:k);
    A_k = U_k * S_k * V_k';

    % Compute RMSE
    error_image = double(A) - A_k;
    RMSE = sqrt(mean(error_image(:).^2));
    disp(['RMSE for CR ≈ ', num2str(CR), ': ', num2str(RMSE)]);

    % Display the approximated image
    figure;
    imshow(uint8(A_k), []);
    title(['Rank-', num2str(k), ' Approximation (CR ≈ ', num2str(CR), ')']);
end
```

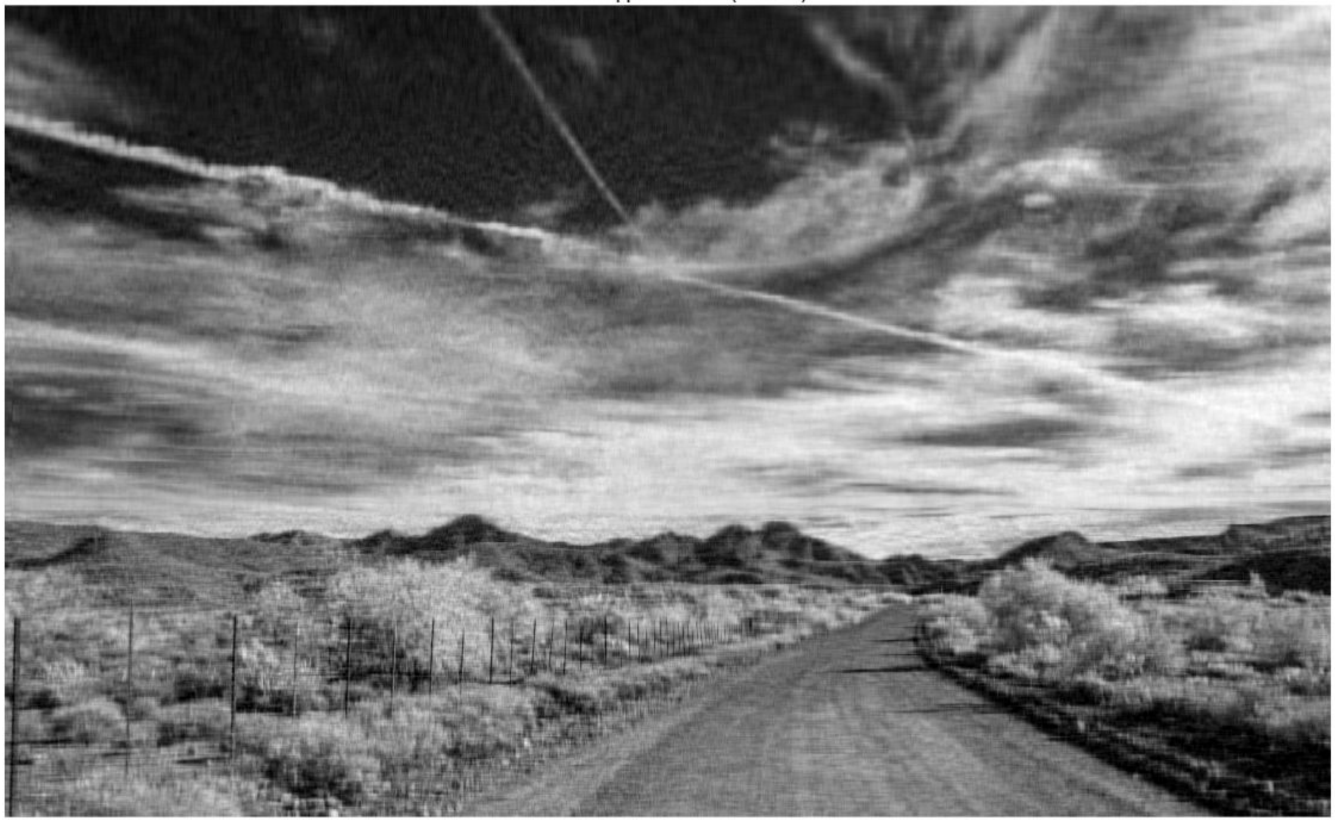
```
Value of k for CR ≈ 10: 160
RMSE for CR ≈ 10: 8.2118
```

Rank-160 Approximation (CR ≈ 10)



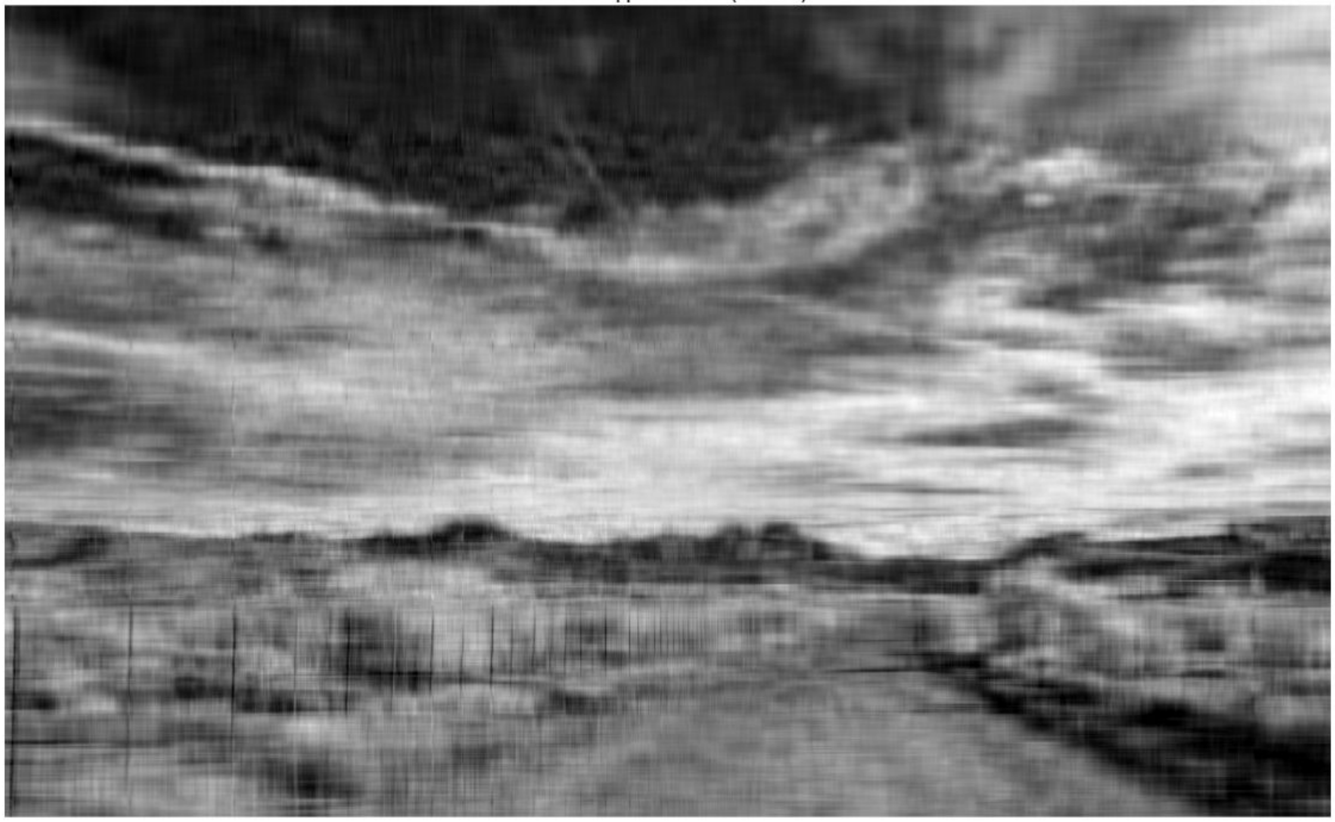
```
Value of k for CR ≈ 25: 64
RMSE for CR ≈ 25: 12.3039
```

Rank-64 Approximation (CR ≈ 25)



```
Value of k for CR ≈ 75: 21
RMSE for CR ≈ 75: 18.2656
```

Rank-21 Approximation (CR ≈ 75)



Trends Observed:

As the compression ratio increases, meaning fewer singular values are used, the quality of the reconstructed image worsens. This appears as more blurring and loss of detail. The RMSE between the original and compressed images rises, indicating a larger difference from the original.

Recommendation for balance:

It's best to find a balance between compression and image quality. A CR of 10 strikes a good compromise, reducing file size while keeping acceptable quality. Higher ratios like 25 or 75 result in significant quality loss, making them unsuitable for applications needing clear images.