Project Two Template

MAT-350: Applied Linear Algebra

Student Name: Justin Paul Guida

Date: 10/16/2025

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:

```
% 3 x 3 Matrix A
A = [1 2 3;
     3 3 4;
     5 6 7]
[U,S,V] = svd(A)
% finding the rank-1 approximation of A
A1 = U(:,1:1) * S(1:1,1:1) * V(:,1:1)'
A1 = round(A1,4)  % round A1 values to 4 decimals
RankA1 = rank(A1) % check the rank
% getting RMSE between A and A1
rmse_1 = sqrt(mean((A(:)-A1(:)).^2))
```

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:

```
% Computing rank approximation, by adding the first two singular components
A2 = U(:,1)*S(1,1)*V(:,1)' + U(:,2)*S(2,2)*V(:,2)'
% Showing rank of A2
RankA2 = rank(A2)
% getting RMSE between A and A2
rmse_2 = sqrt(mean((A(:)-A2(:)).^2))
```

Explain:

RMSE: accounts for how different my approximation is from the original matrix. It measures the average error between A1 and A2. A smaller RMSE means the approximation is closer to the real matrix and fits it better.

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: $

```
% Extracting three column vectors from U
U1 = U(:,1), U2 = U(:,2), U3 = U(:,3)
% compute the cross product c = cross(u1,u1); compute the dot product d2 =
dot(c, u3)
c = cross(U1, U2), d2 = dot(c, U3)
```

Explain:

The dot product d2 = dot(c, U3) is 1.0000, meaning C and U3 point the same way and both have a length of 1. This shows that U1, U2, and U3 are orthogonal unit vectors. That just means they're all perpendicular to each other and each one has a magnitude of 1. Together they make a right-handed orthonormal basis.

Problem 4

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

```
% Matrix u = [u1, u2 u3]
U = [U1, U2, U3]

% double check and test the Rank
r = rank(U)
spans = (r ==3) % check span is = R^3; spans = logical should be 1.
% Use RREF to show RREF View
RREF_U = rref(U)
```

Explain:

rank(U) = 3, so they span R^3

Additionally, you can see in echelon form that the matrix reduces to the identity. This means each column is independent, and together they cover the whole 3-D space.

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:

```
% use imshow() to load and display image A
load("/Users/jguida941/Downloads/MAT 350 Project Two MATLAB Image.mat")
```

```
imshow(A)
% get the size of the image
[m,n] = size(A)
% 1st I set the target compression ratio (CR) and solve for k
% The formula is CR = (m*n) / (k*(m + n + 1))
% It is rearranged to find k that gives the CR
CR = 2;
k = round((m*n)/(CR*(m+n+1)));
disp(['Calculated k for CR≈', num2str(CR), ' is ', num2str(k)])
% make the rank-k version of the image
[U,S,V] = svd(double(A),'econ')
A_compressed = U(:,1:k)*S(1:k,1:k)*V(:,1:k)'
% show both images side by side to double check
figure
subplot(1,2,1), imshow(A,[]), title('Original Image is:')
subplot(1,2,2), imshow(A_compressed,[]), title(['Rank:', num2str(k), '
Approximation:'])
% final check

% same image size, and rank should equal k
size(A), size(A_compressed), rank(A_compressed), k
```

Explain:

SVD was used to rebuild the image using the top k singular values that give a compression ratio of around CR = 2. This allows the resulting image to keep most of its detail while also reducing data size. The rank-k version proves the image can be rebuilt with far fewer values without losing key structure.

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:

```
close all
% Displayed with imshow(A, []) using double precision
% Converting to uint8 is unnecessary in modern MATLAB and slightly alters
% the RMSE due to quantization. The display works correctly without
conversion
imshow(A_compressed, [])
RMSEk = norm(double(A)-A_compressed,'fro')/sqrt(m*n)
title(sprintf('Rank: %d Approximation, RMSE = %.4f', k, RMSEk))
set(gcf,'Position',[100 100 700 600])  % fit window
```

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 problems 5, 6
% with CR = 10, 25, and 75

% precompute once
[m,n] = size(A);
[U,S,V] = svd(double(A),'econ');

% CR = 10
CR = 10;                            % set target compression ratio to 10
k = round((m*n)/(CR*(m+n+1)));     % solve for k using CR formula

% build rank-k approximation using the first k singular values
A10 = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';

% calculate RMSE to measure the differance in this image compared to the
original
RMSE10 = norm(double(A)-A10,'fro')/sqrt(m*n);

% display the new compressed image and display CR  and RMSE for the title
figure, imshow(A10,[])
title(['CR ≈ 10, RMSE = ', num2str(RMSE10)]);set(gcf,'Position',[100 100
700 600]) % fit in title % CR = 25
CR = 25; k = round((m*n)/(CR*(m+n+1)));
A25 = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
RMSE25 = norm(double(A)-A25,'fro')/sqrt(m*n);
figure, imshow(A25,[]), title(['CR ≈ 25, RMSE = ', num2str(RMSE25)]);
set(gcf,'Position',[100 100 700 600]) % fit in title % CR = 25
% CR = 75
CR = 75; k = round((m*n)/(CR*(m+n+1)));
A75 = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
RMSE75 = norm(double(A)-A75,'fro')/sqrt(m*n);
figure, imshow(A75,[]), title(['CR ≈ 75, RMSE = ', num2str(RMSE75)]);
set(gcf,'Position',[100 100 700 600]) % fit in title % CR = 25
% Display RMSE values for rank-1 and rank-2 approximations
disp(['RMSE for rank-1 approximation: ', num2str(rmse_1)]);, disp(['RMSE
for rank-2 approximation: ', num2str(rmse_2)]);
disp('My suggestion would be CR ≈ 15:')
CR = 15;
k15 = round((m*n)/(CR*(m+n+1)));
A15 = U(:,1:k15)*S(1:k15,1:k15)*V(:,1:k15)';
RMSE15 = norm(double(A)-A15,'fro')/sqrt(m*n);
figure; imshow(A15,[]);
set(gcf,'Position',[100 100 700 600]);
title(['My suggestion would be: CR ≈ 15, RMSE = ', num2str(RMSE15)]);
disp(['RMSE for CR ≈ 15: ', num2str(RMSE15)]);
```

Explain:

At a low compression rate (like CR=10), the image remains sharper and more detailed but the trade off is more storage.

A medium compression rate (CR=25) reduces the file size, but the trade off is itt introduces some blurring and texture loss.

At a high rate (CR=75), the image becomes heavily blurred showing the trade-off between size and quality.

To find a better balance, I calculated CR≈15 using the same formula. This value keeps more singular components than CR=25, so edges and fine details stay clearer, while the file size is still much smaller than at CR=10.

Based on the results, CR≈15 provides the best compromise between visual quality and compression efficiency.

But if I had to choose only between CR=10, CR=25, and CR=75, I would pick CR=25. It keeps most of the image structure while still reducing the file size more effectively than CR=10.