Assignment - 1

RTSP

Pratyush Jaiswal
18EE35014

Singular Value Decomposition

SVD of an m×n matrix M is a factorization of the form U·S·V^H, where U is an m×n unitary matrix, S represents an m×n rectangular diagonal matrix with non-negative real numbers on the diagonal, and V represents an n×n unitary matrix. U and V are orthogonal matrices if M is a real matrix.

factorizing; U·S·V^H = G or U S V^T = G. Here once we complete factorization, we must reconstruct the image.

a. **One largest Eigen Value:**

Im1 = U(:,1) . S(1,1) . V(:,1)' ← 1st Column of V

U(:,1) ← 1st Column of U

S(1,1) ← 1st largest eigen value

b. **2 largest Eigen Values:**

Im2 = U(:,1:2) S(1:2, 1:2) V(:,1:2)'

c. **3 largest eigen values:**

Im3 = U(:,1:3) S(1:3, 1:3). V(:,1:3)'

d. 4 largest eigen values:

$$Im4 = U(:,1:4)\ S(1:4,1:4)\ V(:,1:4)'$$

e. 5 largest eigen values:

$$Im5 = U(:,1:5)\ S(1:5,1:5)\ V(:,1:5)'$$

To determine which of the reconstructed images is closest to the original one, we must calculate the distance between two images, which is the ~~norm~~ difference of images' norm. The one which is closer is a more appropriate image

$dist(h, Im1) = \|h - Im1\|_2 = 334.098$

$dist(h, Im2) = \|h - Im2\|_2 = 267.8401$

$dist(h, Im3) = 186.5281$

$dist(h, Im4) = 109.008$

$dist(h, Im5) = 0 \Rightarrow$ Exact reconstruction

Why SVD is used:

SVD is a durable and effective orthogonal matrix decomposition method for obtaining optimum subrank approximations by decomposing a matrix into orthogonal components. The biggest object component is an SVD picture that corresponds to the Eigen image with the least singular values. The SVD algorithm is used to approximate the data matrix decomposition into an optimum approximation of the signal & noise components. This is one of the most ~~of~~ essential aspects of SVD in noise filtering, compression, and forensics, and it may also be thought of as adding noise in a proper detectable manner.

$$X = \sum_{i=1}^{k} B_i A_i C_i^T \simeq B_1 A_1 C_1^T + B_2 A_2 C_2^T + \dots B_K A_K C_K^T$$

SVD Applications:

→ Truncated SVD transforms save a lot of space while preserving a lot of information. This feature is used for picture compression.

→ The SVD algorithm can adapt to changes in an image's local statistics. SVD-based ~~b~~ watermarking techniques can make use of singular stability (which determines the ~~image~~ energy of image layer). As a result, Image forensic makes advantage of it.

→ SVD is used in noise filtering, image denoising & watermarking.

→ Among the various transformations, SVD has the highest energy packing. With multi-resolution SVD, the following picture qualities may be assessed at each level of resolution: isotropy, specity of primary components, & self-similarity under scaling.

SVD Algorithm:

Let A be a real $m \times n$ matrix. SVD of A is given by:
$$A = USV^T$$

U: $m \times m$ orthogonal, V: $n \times n$ orthogonal

S: $m \times n$ diagonal matrix with non-negative entries.

~~$\sigma_1 \leq \sigma_2$~~

$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p$; $p = \min(m, n)$ are known as singular values of A.

Let U & V have column partitions

$U = [u_1, \dots u_m]$, $V = [v_1, \dots v_n]$

$u_i = m \times 1$, $i = 1 \dots m$ $v_j = n \times 1$, $j = 1 \dots n$

From the relations known,

$A v_j = \sigma_j u_j$; $A^T u_j = \sigma_j v_j$; $j = 1, 2 \dots p$

Combining both we get $A^T A v_j = \sigma_j^2 v_j$ i.e the squares of the singular values are eigen values of $A^T A$ which is a symmetric matrix.

Pseudo Code for the Algorithm

To find SVD we will be using power iteration method, a popular method to find eigen value of a square matrix iteratively.

In this approach we try to find the dominant singular value first (the highest by magnitude) ⇒ after that subtract ~~that~~ corresponding eigen ~~value~~ vector component. We repetatively do this.

1. Dominant singular values (σ_i's) are determined using power iteration of A^TA. Since,

$$A^TA v_j = \sigma_j^2 v_j \quad (j = 1 \dots p)$$

2. Now we need to find the next dominant singular value of A / eigen value of A^TA). So we subtract the component of previous eigen vector. for i^{th} ~~th~~ step where ($i \neq 1$)

$$A = A - \sigma_{i-1} * u_i * v_i^T$$

$$(\text{where } u_i = A v_i / \sigma_i)$$

3. Repeat Step 1 for p iterations where

$$p = \min(m, n)$$

$$A \in R^{m \times n}.$$

Observation & Discussions

→ It can be seen from the ~~oo~~ difference output value (tending to zero) & plots that this method converges.

→ The norm of difference being printed is tending to zero proving that this iterative algorithm converges & matches to the built in function.

→ Also the norm of difference (dist variables) for the original & the image rotated comes out to be same.

```
clc;
clear;
```

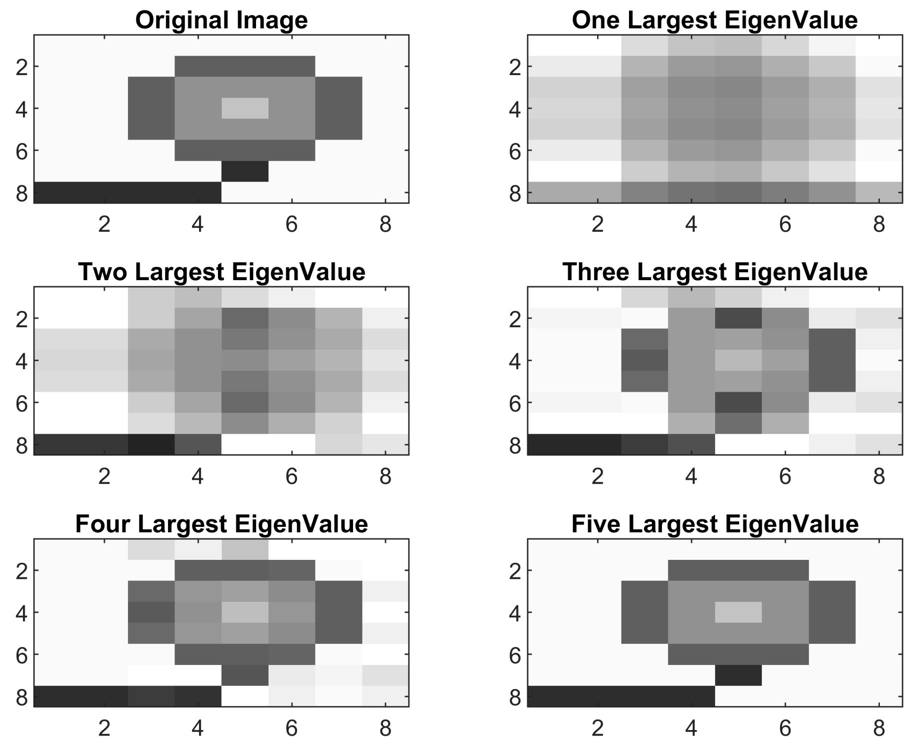
```
% Original Matirx
G = [255 255 255 255 255 255 255 255
255 255 255 100 100 100 255 255;
255 255 100 150 150 150 100 255;
255 255 100 150 200 150 100 255;
255 255 100 150 150 150 100 255;
255 255 255 100 100 100 255 255
255 255 255 255 50 255 255 255;
50 50 50 50 255 255 255 255];
```

```
% Using Matlab's inbuilt SVD Function
[U,S,V] = svd(G);
[U1,S1,V1]=svd(G.'); % After rotating G
```

```
% Reconstruction of approximate images using largest eigenvalues
img1 = U(:,1)*S(1,1)*V(:,1)';
img2 = U(:,1:2)*S(1:2,1:2)*V(:,1:2)';
img3 = U(:,1:3)*S(1:3,1:3)*V(:,1:3)';
img4 = U(:,1:4)*S(1:4,1:4)*V(:,1:4)';
img5 = U(:,1:5)*S(1:5,1:5)*V(:,1:5)';
```

```
% Plotting the approximate reconstructed images
figure;
subplot(3,2,1);
image(G);
colormap("gray");
title("Original Image");

subplot(3,2,2);
image(img1);
colormap("gray");
title("One Largest EigenValue");

subplot(3,2,3);
image(img2);
colormap("gray");
title("Two Largest EigenValue");

subplot(3,2,4);
image(img3);
colormap("gray");
title("Three Largest EigenValue");

subplot(3,2,5);
image(img4);
colormap("gray");
title("Four Largest EigenValue");
```

```
subplot(3,2,6);
image(img5);
colormap("gray");
title("Five Largest EigenValue");
```




```
% Getting norm of the difference the original image and reconstructed
% images
dist1=norm(G-img1)
```

```
dist1 = 334.0981
```

```
dist2=norm(G-img2)
```

```
dist2 = 267.8401
```

```
dist3=norm(G-img3)
```

```
dist3 = 186.5231
```

```
dist4=norm(G-img4)
```

```
dist4 = 109.0080
```

```
dist5=norm(G-img5)
```

```
dist5 = 1.5960e-12
```

```
diffOnEigenRotated=norm(S-S1)
```

```
diffOnEigenRotated = 9.0949e-13
```

```
clear all;
```

```
% Original Matirx
G = [255 255 255 255 255 255 255 255
255 255 255 100 100 100 255 255;
255 255 100 150 150 150 100 255;
255 255 100 150 200 150 100 255;
255 255 100 150 150 150 100 255;
255 255 255 100 100 100 255 255
255 255 255 255 50 255 255 255;
50 50 50 50 255 255 255 255];
```

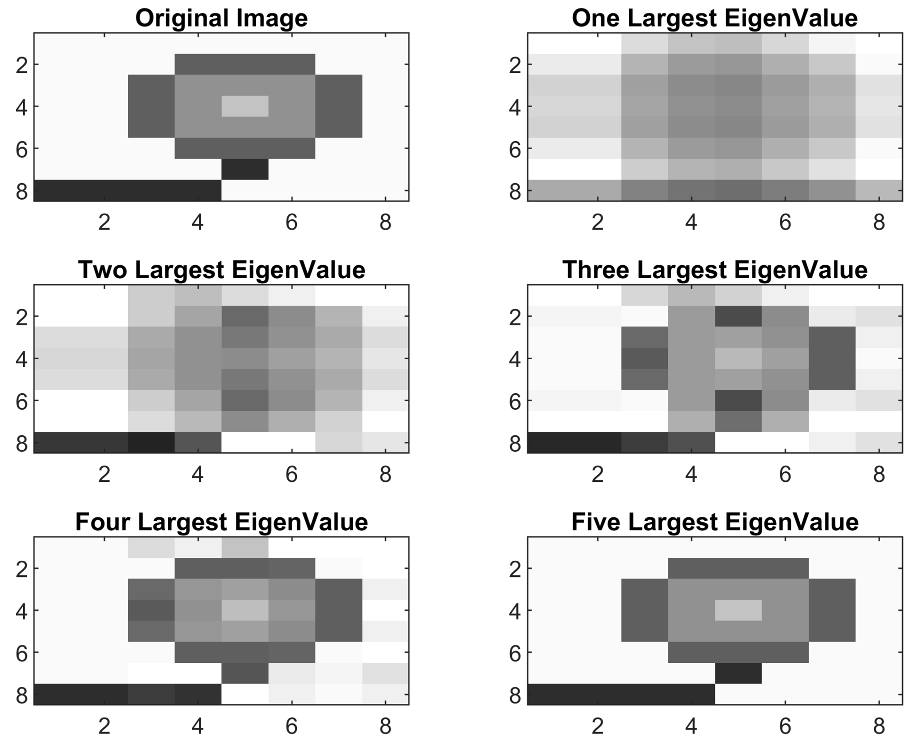
```
[U,S,V]=svd(G); % Using Matlab function

% Implementing svd from scratch
n = size(G,1);
G_copy = G;
[curr_s, curr_v] = power_iter(G);
curr_v = curr_v/norm(curr_v);
curr_u = G_copy*curr_v/curr_s;
ans_v = curr_v;
ans_u = curr_u;
ans_s = curr_s;
for i=1:n-1
    update_G = G_copy - curr_s*curr_u*(curr_v)';
    [update_eigen,update_v] = power_iter(update_G);
    update_u = update_G*update_v/update_eigen;
    update_v = update_v/norm(update_v);
    ans_u = [ans_u update_u];
    ans_v = [ans_v update_v];
    ans_s = [ans_s update_eigen];
    curr_u = update_u;
    curr_v = update_v;
    curr_s = update_eigen;
    G_copy = update_G;
end
ans_s = diag(ans_s);
diff = norm(ans_s-S)
```

```
diff = 1.6259e-07
```

```
% Reconstruction of approximate images using largest eigenvalues
img1 = ans_u(:,1)*ans_s(1,1)*ans_v(:,1)';
img2 = ans_u(:,1:2)*ans_s(1:2,1:2)*ans_v(:,1:2)';
img3 = ans_u(:,1:3)*ans_s(1:3,1:3)*ans_v(:,1:3)';
img4 = ans_u(:,1:4)*ans_s(1:4,1:4)*ans_v(:,1:4)';
img5 = ans_u(:,1:5)*ans_s(1:5,1:5)*ans_v(:,1:5)';
```

```
% Plotting the approximate reconstructed images
figure;
```

```
subplot(3,2,1);
image(G);
colormap("gray");
title("Original Image");

subplot(3,2,2);
image(img1);
colormap("gray");
title("One Largest EigenValue");

subplot(3,2,3);
image(img2);
colormap("gray");
title("Two Largest EigenValue");

subplot(3,2,4);
image(img3);
colormap("gray");
title("Three Largest EigenValue");

subplot(3,2,5);
image(img4);
colormap("gray");
title("Four Largest EigenValue");

subplot(3,2,6);
image(img5);
colormap("gray");
title("Five Largest EigenValue");
```




```
% Getting norm of the difference the original image and reconstructed
% images
dist1=norm(G-img1)
```

dist1 = 334.0981

```
dist2=norm(G-img2)
```

dist2 = 267.8401

```
dist3=norm(G-img3)
```

dist3 = 186.5231

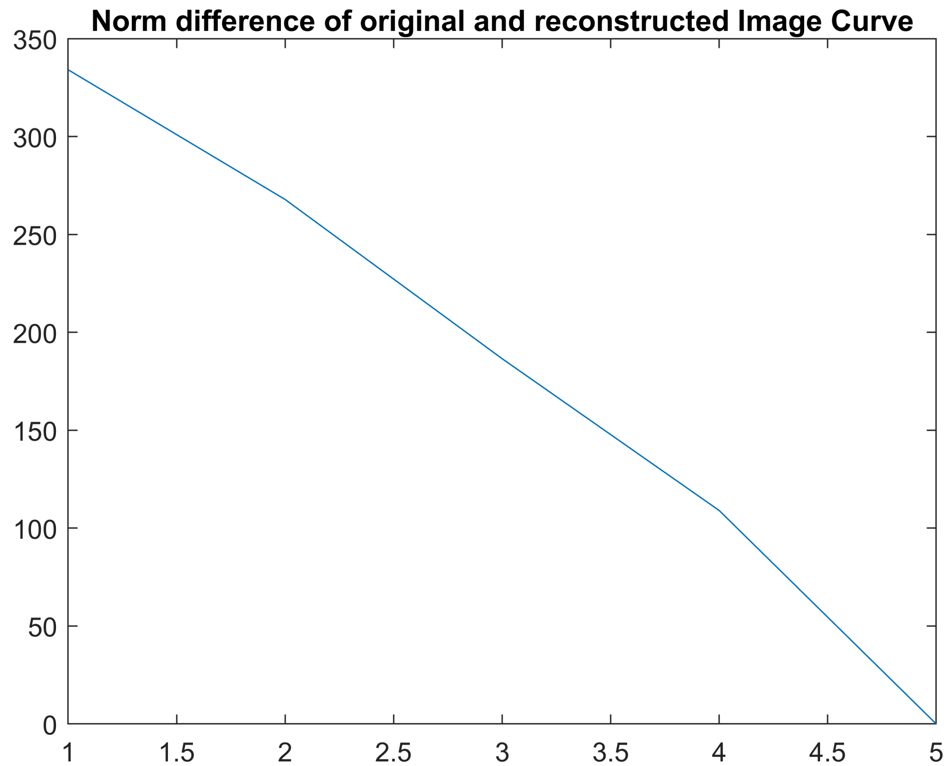
```
dist4=norm(G-img4)
```

dist4 = 109.0080

```
dist5=norm(G-img5)
```

dist5 = 2.5886e-13

```
dist=[dist1 dist2 dist3 dist4 dist5];
figure;
plot(dist);
title("Norm difference of original and reconstructed Image Curve");
```




```
function[eigen_value,eigen_vector] = power_iter(G)
    n = size(G,1);
    v = ones(n,1);
    G = (G)'*G;
    val = v\G*v;
    while true
        v_update = G*v/norm(G*v);
        val_update = v_update\G*v_update;
        if(abs(val-val_update)<0.0001)
            break
        end
        v = v_update;
        val = val_update;
    end
    eigen_value = sqrt(val);
    eigen_vector = v;
end
```