Assignment-2

J. Kalyan Raman

17EE35004.

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.

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq --- \geq \sigma_p, \quad p = \min\{m,n\}$$

known as singular values of A.

Let U and V have column partitions

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

$$u_i : m \times 1, \; i = 1, -- m \qquad v_j : n \times 1, \; j = 1, -- n.$$

From the relations known,

$$A v_j = \sigma_j u_j \;, \quad A^T u_j = \sigma_j v_j \quad ; \; j = 1, 2, -- p.$$

Combining both, we get

$$A^T A v_j = \sigma_j^2 v_j$$

i.e., squares of singular values are eigenvalues of $A^T A$, which is a symmetric matrix.

Now symmetric QR algorithm can be applied to A^TA to obtain a decomposition.

$$A^TA = VS^TSV^T$$

Then, the relations $A \cdot v_j = \sigma_j u_j$, $j = 1, \ldots, p$; can be used in conjuction with the QR factorization with column pivoting to obtain U.

Similarly : $AA^T u_i = \sigma^2 u_i$, ~~$i = 1, \ldots, p$~~

$$AA^T = USS^TU^T$$

→ Apart from this method, we can implicitly apply the symmetric QR algorithm to A^TA.

<u>Results</u> : using above algorithm, results are obtained.

$[U, S, V] = \text{SVDALGO}(G)$; $[U1, S1, V1] = \text{SVD}(G)$

Tol $= 10^{-7}$

norm (U − U1) = norm (V − V1) = 2

norm (S − S1) = 0.

So designed algorithm is giving almost same results as that of Matlab's one.

MATLAB CODES :

RTSP_Assg_2_17EE35004.m :

```
%% Initializing all Matrices

clc;
clear;

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] = SVDALGO(G,0.0000001); % SVD algo based on QR decomposition

[U1,S1,V1] = svd(G);            % Matlab's SVD algo

dist = [norm(U-U1) norm(S-S1) norm(V-V1)]; % Distance between the matrices found

%% Reconstruction using SVD algo designed
% Reconstruction of images using largest eigenvalues

im1 = U(:,1)*S(1,1)*V(:,1)';
im2 = U(:,1:2)*S(1:2,1:2)*V(:,1:2)';
im3 = U(:,1:3)*S(1:3,1:3)*V(:,1:3)';
im4 = U(:,1:4)*S(1:4,1:4)*V(:,1:4)';
im5 = U(:,1:5)*S(1:5,1:5)*V(:,1:5)';

%% Finding the distance between original and reconstructed images

dist1 = zeros(5,1);
dist1(1) = norm(G-im1);
dist1(2) = norm(G-im2);
dist1(3) = norm(G-im3);
dist1(4) = norm(G-im4);
dist1(5) = norm(G-im5);

%% ploting the reconstructed images

figure;
subplot(3,2,1);imagesc(G);
title("original image");
subplot(3,2,2);imagesc(im1);
title("one largest eigenvalue");
subplot(3,2,3);imagesc(im2);
title("two largest eigenvalues");
subplot(3,2,4);imagesc(im3);
title("three largest eigenvalues");
subplot(3,2,5);imagesc(im4);
title("four largest eigenvalues");
```

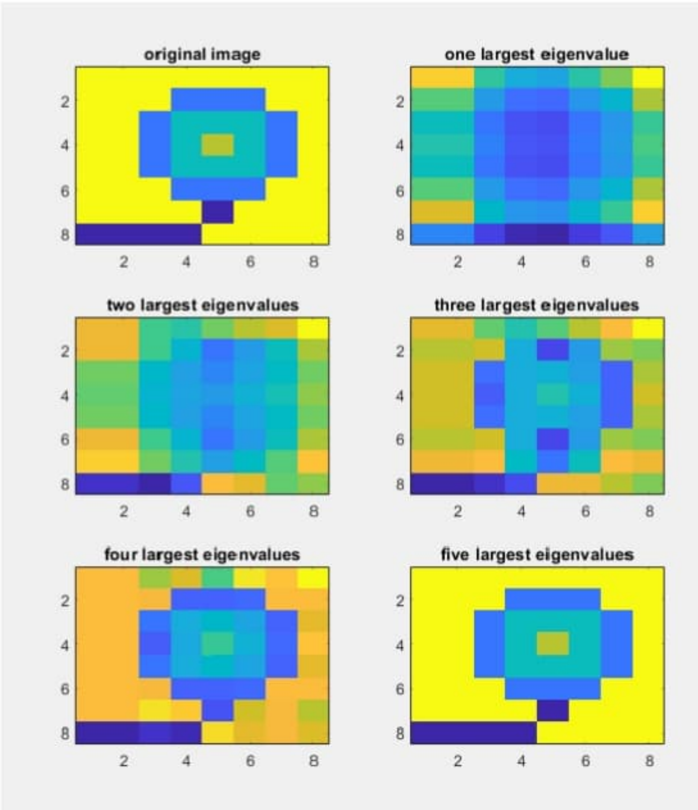
```matlab
subplot(3,2,6);imagesc(im5);
title("five largest eigenvalues");


%% Reconstruction using Matlab's SVD algo
% Reconstruction of images using largest eigenvalues

imo1 = U1(:,1)*S1(1,1)*V1(:,1)';
imo2 = U1(:,1:2)*S1(1:2,1:2)*V1(:,1:2)';
imo3 = U1(:,1:3)*S1(1:3,1:3)*V1(:,1:3)';
imo4 = U1(:,1:4)*S1(1:4,1:4)*V1(:,1:4)';
imo5 = U1(:,1:5)*S1(1:5,1:5)*V1(:,1:5)';

%% Finding the distance between original and reconstructed images

dist2 = zeros(5,1);
dist2(1) = norm(G-imo1);
dist2(2) = norm(G-imo2);
dist2(3) = norm(G-imo3);
dist2(4) = norm(G-imo4);
dist2(5) = norm(G-imo5);

%% ploting the reconstructed images

figure;
subplot(3,2,1);imagesc(G);
title("original image");
subplot(3,2,2);imagesc(imo1);
title("one largest eigenvalue");
subplot(3,2,3);imagesc(imo2);
title("two largest eigenvalues");
subplot(3,2,4);imagesc(imo3);
title("three largest eigenvalues");
subplot(3,2,5);imagesc(imo4);
title("four largest eigenvalues");
subplot(3,2,6);imagesc(imo5);
title("five largest eigenvalues");
```

SVDALGO.m :

```
function [U,S,V] = SVDALGO(A,T)
% A is the rectangular matrix and T is the tolerance accepted
%% SVD Algorithm using QR decomposition

if ~exist('tol','var')
   T = eps*1024;
end

% Reserve space in advance
sizea = size(A);
loopmax = 100*max(sizea);
loopcount = 0;

% Initializing U, S, and V
U = eye(sizea(1));
S = A';
V = eye(sizea(2));
Error = realmax;
while Error>T && loopcount<loopmax
%   log10([Er tol loopcount loopmax]); pause
    [q,S] = qr(S');
    U=U*q;
    [q,S] = qr(S');
    V=V*q;

%   exit when we get "close"
    e1 = triu(S,1);
    E = norm(e1(:));
    F = norm(diag(S));
    if F==0
      F = 1;
    end
    Error = E/F;
    loopcount = loopcount+1;
end
% [Er/T loopcount/loopmax]

% Fix the signs in S
ss = diag(S);
S = zeros(sizea);

for n=1:length(ss)
    ssn = ss(n);
    S(n,n) = abs(ssn);
    if ssn<0
       U(:,n)=-U(:,n);
    end
end

if nargout<=1
   U = diag(S);
end
return
```

Reconstructed images using

(a) Designed SVD algorithm :



(b) Matlab’s SVD algorithm