MTH 420/520 Homework 4. **Yesh Godse**
Date: May 10, 2019
Prof. M. Peszynska

MTH 420-520 students turn in 1-2.
Extra credit is marked.
Show enough work and enough of `MATLAB code` to demonstrate this is your own hard work, but be concise.
Sloppy incomplete work, or lengthy algebraic calculations, or core dump of MATLAB output with errors will not receive much credit.

Problem 1. Illustrate how SVD works for the matrices
$A = [2, -1, -1, 2]; A = [7, 5; 1, 5]; A = [1, 1; 1, 1], A = [1, 0; 0, -1]$.
(i) Find the SVD (by hand or in MATLAB). Discuss what U, Σ, V' reveal about the properties of A and what they do not reveal.
(ii) Draw an illustration how $A = U\Sigma V^T$ transforms vectors step by step using SVD.
Hint: You can use as a template the code for symmetric matrices shown below. (You have to modify the code, since it does not use SVD).

Solution 1. The following MATLAB code finds the singular value decomposition of a rectangular matrix A, such that

$$A = U\Sigma V^T$$

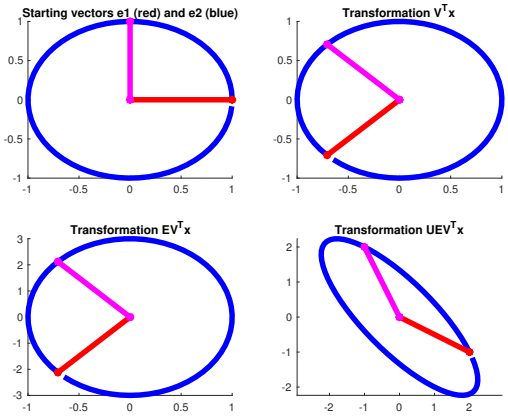
```
K = A'*A;
[V,Sigma]=eig(K);
Sigma = sqrt(Sigma);
U = ones(2,2);

U(:,1) = 1/Sigma(1,1) * A*V(:,1);
U(:,2) = 1/Sigma(2,2) * A*V(:,2);
```

For A = [2, -1; -1, 2];

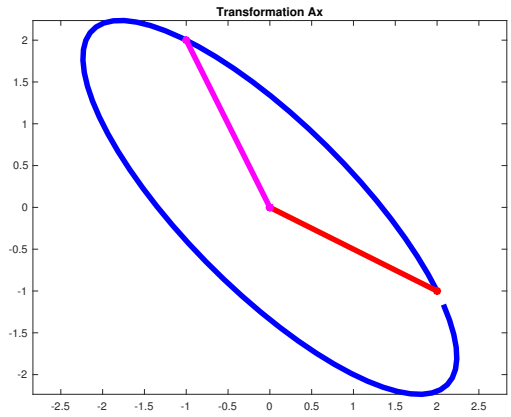
```
U =                       E =             V' =
-0.7071   -0.7071         1     0         -0.7071   -0.7071
-0.7071    0.7071         0     3         -0.7071    0.7071
```

Here, V^T rotates vectors by $\frac{3\pi}{4}$ radians and reflects them. The singular values in Σ stretch those vectors, and U rotates and reflects them again.

The following plots show the process of this transformation on the vectors $[1\ 0]^T$ and $[0\ 1]^T$



(A) Transformation through SVD



(B) Transformation Ax

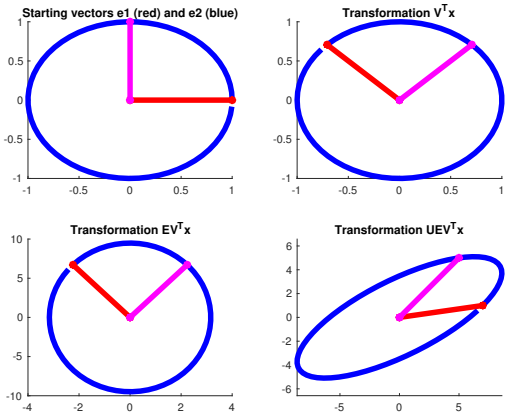
FIGURE 1. Comparing step by step SVD transformation to directly using A

For $A = [7\ 5;\ 1\ 5]$;

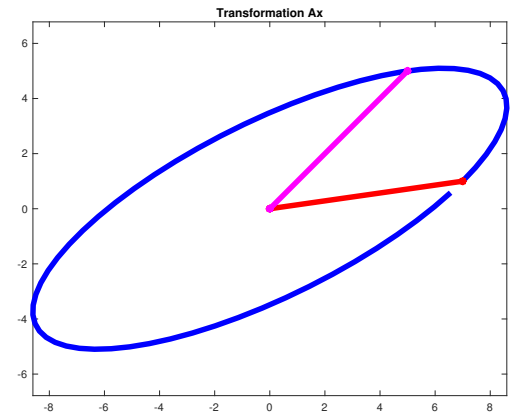
```
U =                                E =                              V' =
-0.4472    0.8944                  3.1623         0                 -0.7071    0.7071
 0.8944    0.4472                  0         9.4868                 0.7071    0.7071
```

Here, V^T rotates vectors by $\pi/4$ radians and reflects them. The singular values in Σ stretch those vectors, and U rotates and reflects them again.

The following plots show the process of this transformation on the vectors $[1\ 0]^T$ and $[0\ 1]^T$



(A) Transformation through SVD



(B) Transformation Ax

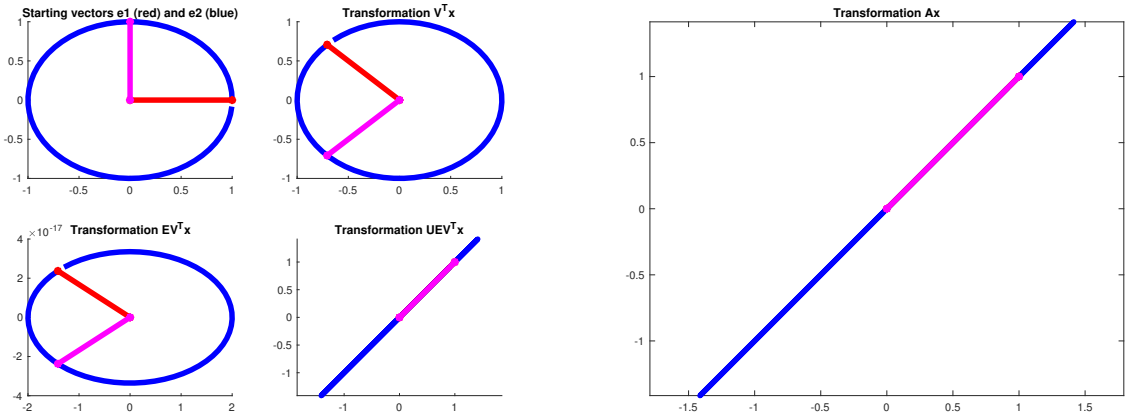
FIGURE 2. Comparing step by step SVD transformation to directly using A

For $A = [1\ 1;\ 1\ 1]$;

```
U =                          E =                          V' =
-0.7071   -0.7071            2.0000         0             -0.7071   -0.7071
-0.7071    0.7071            0         0.0000             0.7071   -0.7071
```

Here, V^T rotates vectors by $3\pi/4$ radians and does not reflect them. The singular values in Σ stretch those vectors. U drives every vector to the line $y = x$

The following plots show the process of this transformation on the vectors $[1\ 0]^T$ and $[0\ 1]^T$



(A) Transformation through SVD (B) Transformation Ax

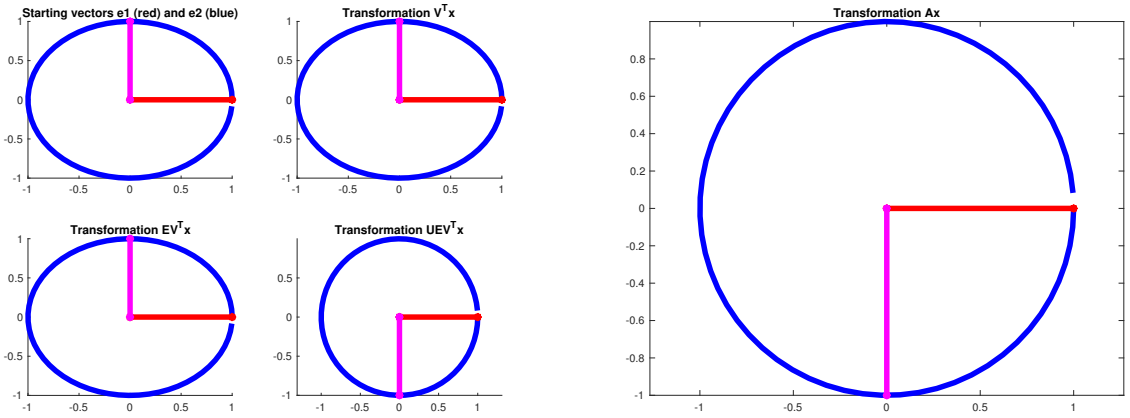
FIGURE 3. Comparing step by step SVD transformation to directly using A

For $A = [1\ 0;\ 0\ -1]$;

```
U =          E =          V' =
1    0       1    0       1    0
0   -1       0    1       0    1
```

Here, V^T and Σ are identity matrices that have no impact on the vectors. U simply reflects each vector across the x-axis

The following plots show the process of this transformation on the vectors $[1\ 0]^T$ and $[0\ 1]^T$



(A) Transformation through SVD (B) Transformation Ax

FIGURE 4. Comparing step by step SVD transformation to directly using A

MATLAB functions to create subplots of step by step transformation.

```
function f = illustrateSVD(A)
    theta =0:0.1:2*pi ;
    u = [cos(theta);sin(theta)]; %% parametrize points on a circle;
    zero =[0;0]; one1 = [1;0]; one2= [0;1];

    K = A'*A;
    [V,E]=eig(K);
    E = sqrt(E);
    U = ones(2,2);

    U(:,1) = 1/E(1,1) * A*V(:,1);
    U(:,2) = 1/E(2,2) * A*V(:,2);

    % Plot vectors at origin
    figure();
    win(1) = subplot(2, 2, 1); win(2) = subplot(2, 2, 2);
    win(3) = subplot(2, 2, 3); win(4) = subplot(2, 2, 4);
    set(win,'Nextplot','add')
    drawVectors(u, zero, one1, one2, win(1),
    'Starting vectors e1 (red) and e2 (blue)');
    drawVectors(V'*u, zero, V'*one1, V'*one2, win(2),
    'Transformation V^Tx');
    drawVectors(E*V'*u, zero, E*V'*one1, E*V'*one2, win(3),
    'Transformation EV^Tx');
    drawVectors(U*E*V'*u, zero, U*E*V'*one1, U*E*V'*one2, win(4),
    'Transformation UEV^Tx');
```

```
    drawVectors_alone(A*u, zero, A*one1, A*one2,
    'Transformation Ax');
end
```

```
function f = drawVectors(u, zero, one1, one2, plot_id, the_title)
    plot(plot_id, u(1,:),u(2,:),'b-'); axis equal;
    vector = [zero, one1];
    plot(plot_id, vector(1,:),vector(2,:),'r-*');
    vector = [zero, one2];
    plot(plot_id, vector(1,:),vector(2,:),'m-*');
    title(plot_id,the_title);
end
```

Problem 2. The code below (i) loads an image, (ii) converts it to grayscale, (iii) constructs its svd, and then (iv) displays its lower rank approximation.

```
>> beaver = imread('Oregon_State_Beavers2_thumb.jpg');
>> graybeaver = rgb2gray(beaver);imshow(graybeaver)
>> [U,S,V]=svd(double(graybeaver));
>> k=50;imshow(U(:,1:k)*S(1:k,1:k)*V(:,1:k)')
>> k=10;imshow(U(:,1:k)*S(1:k,1:k)*V(:,1:k)')
```

Use your OWN picture (photo of your face). (If it is high resolution, you may have to select just a part of it or use some image processing software to save it in lower resolution. More than 300×200 pixels might be too much for your local computer to handle.) Construct lower rank approximation with $k = 5, 10, 50$ etc. Show to your best friend/family member the approximations. Do they recognize you? For which k they say "yes" and for which k do they say "no"?

The original image is shown below in grayscale:

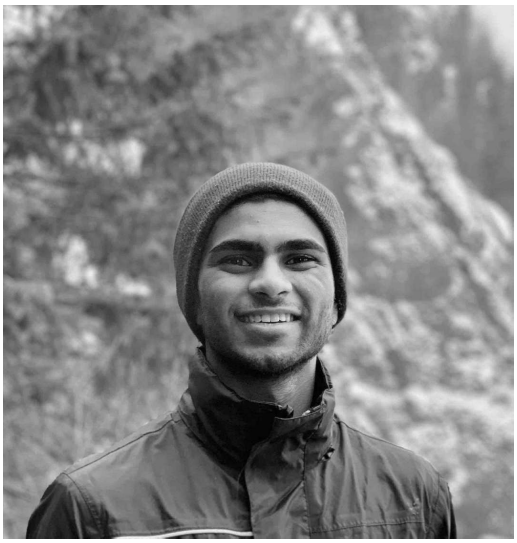


FIGURE 5. Original Image converted to grayscale

I tested SVD compression for the following numbers of singular values, k:

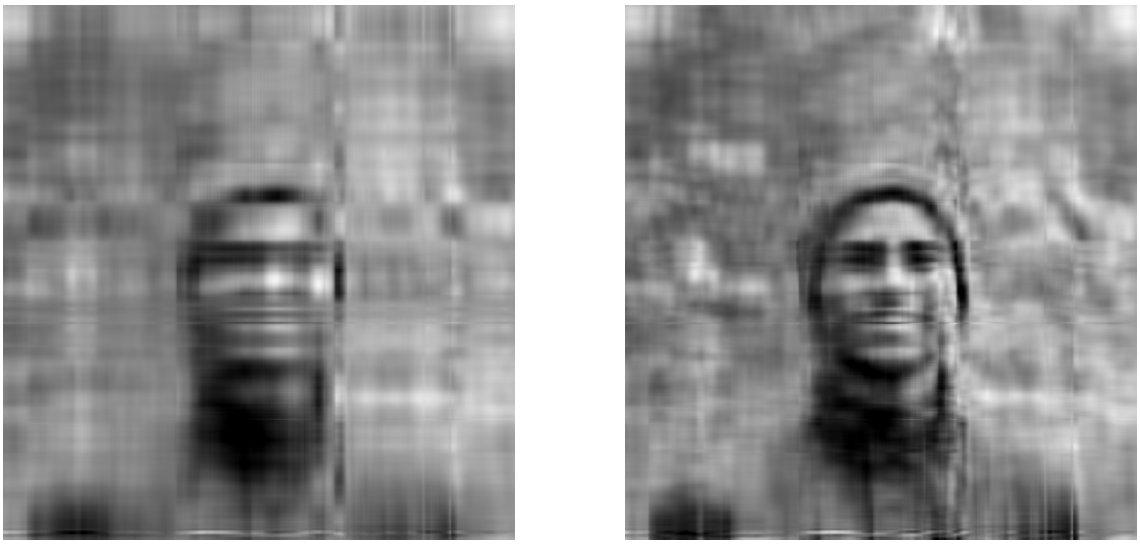
```
k = [5,10,25,50,75,100];
```

I also calculated the error between the SVD compression and the original image by taking the euclidean (L2) norm, as demonstrated in the following code:

```
% Decompose with SVD
[U,S,V]=svd(db_gray_yg);

% Arrays to record number of singular values (k) and error
dispEr = [];
numSVals = [];

for k=[5,10,25,50,75,100]
    figure();
    buffer = sprintf('Image output using %d singular values', k);

    % Construct and display image with singular values
    D = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
    imshow(uint8(D));

    % Compute error
    error=sum(sum((db_gray_yg-D).^2));

    % store vals for display
    dispEr = [dispEr; error];
    numSVals = [numSVals; k];
```

```
end

% dislay the error graph
figure;
title('Error in compression');
plot(numSVals, dispEr);
grid on
xlabel('Number of Singular Values used');
ylabel('Error between compress and original image');
```

My face is not recognizable at k=5, but is at k=10. The images below show how the SVD compression is able to retain the most important information in the image (the most interesting parts of the image). The background is roughly the same with k=10 and k=5, but the detail on my face is much improved with more singular values.



(A) Using 5 Singular Values (B) Using 10 Singular Values

FIGURE 6. Comparing lossy SVD compression of 5 singular values to 10 singular values

Moving on to an even higher value for k, the compressed image looks very close to the original. Here are the results for k=50 and k=100.

(A) Using 50 Singular Values (B) Using 100 Singular Values

FIGURE 7. Comparing lossy SVD compression of 50 singular values to 100 singular values

The error plot below reinforces the idea that most of the important details are present in the compressed image by around k=10. As k increases further, other parts of the image, such as the out of focus background, get more details filled in. Because these parts of the image have less variability, they don't contribute too much to the error and we see that each additional increase in the number of singular values used does less to minimize the difference between the original and compressed images.

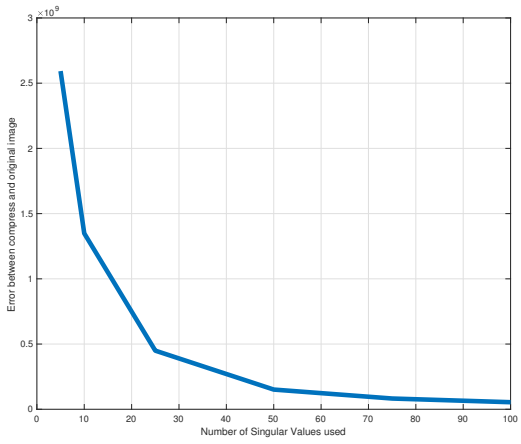


FIGURE 8. Error between compressed images that used a varying number singular values and the original image, measured by L2 norm