CSE 6367 Assignment #1

Zahidur Talukder

February 11, 2021

Problem 1

(a) Proof

Every real symmetric $n \times n$ matrix A can be factorized as

$$A = U.D.U^T$$

- U is orthogonal
- $D = \text{diag}(\lambda_1, ..., \lambda_n)$ is diagonal, with real diagonals.
- A is diagonalizable by an orthogonal similarity transformation: $U^T AU = D$
- The column of U are an orthonormal set of n eigenvectors: $AU = UD$

$$A \begin{bmatrix} u_1 & .. & u_n \end{bmatrix} = \begin{bmatrix} u_1 & .. & u_n \end{bmatrix} \begin{bmatrix} \lambda_1 & \lambda_2 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ 0 & 0 & ... & \lambda_n \end{bmatrix} = \begin{bmatrix} \lambda_1 u_1 & \lambda_2 u_2 & .. & \lambda_n u_n \end{bmatrix}$$

We can proof by induction

- the decomposition obviously exists if $n = 1$
- suppose it exists if $n = m$ and A is an $(m + 1) \times (m + 1)$ matrix
- A has atleast one eigenvalue.
- let λ_1 be any eigenvalue and u_1 a corresponding eigenvectors with $||u_1||$=1
- let V be an $(m + 1) \times m$ matrix that makes the matrix $\begin{bmatrix} u_1 & V \end{bmatrix}$ orthogonal:

$$\begin{bmatrix} u_1^T \\ V^T \end{bmatrix} A \begin{bmatrix} u_1 & V \end{bmatrix} = \begin{bmatrix} u_1^T Au_1 & u_1^T AV \\ V^T Au_1 & V^T AV \end{bmatrix} = \begin{bmatrix} \lambda_1 u_1^T u_1 & \lambda_1 u_1^T V \\ \lambda_1 V^T u_1 & V^T AV \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & V^T AV \end{bmatrix}$$

- $V^T AV$ is a symmetric $m \times m$ matrix, so by the induction hypothesis, $V^T AV = \tilde{U}\tilde{D}\tilde{U}^T$ for some orthogonal $\tilde{U}$ and diagonal $\tilde{D}$

- matrix $U = \begin{bmatrix} u_1 & V\tilde{U} \end{bmatrix}$ is orthogonal and defines a similarity that diagonalized A:

$$U^T AU = \begin{bmatrix} u_1^T \\ \tilde{U}^T V^T \end{bmatrix} A \begin{bmatrix} u_1 & V\tilde{U} \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \tilde{U}^T V^T AV\tilde{U} \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \tilde{D} \end{bmatrix}$$

This proves that for every real symmetric $n \times n$ matrix A can be factorized as

$$A = U.D.U^T$$

(b)

Given

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

We need to plot

$$Ax | x \in \mathbb{R}^2 and ||x||_2 = 1$$

and

$$Ax | x \in \mathbb{R}^2 and ||x||_2 \leq 1$$

So x is a vector with 2 values and whose eucledean norm is equal or less than equal to 1. The region is given below-

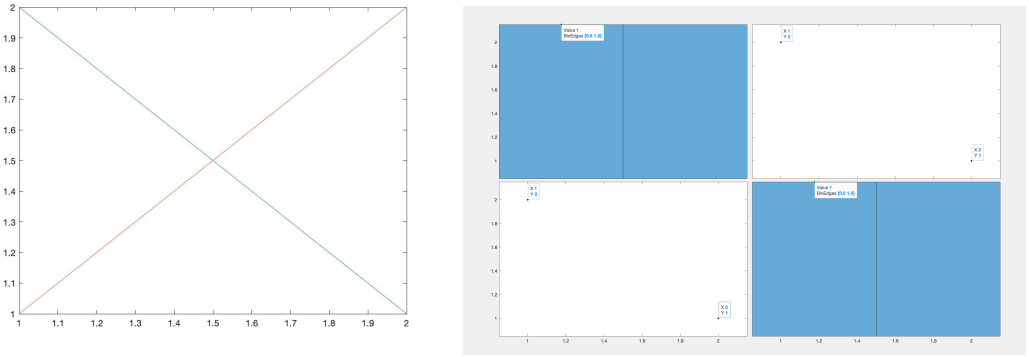


Figure 1: Plot of the region

Problem 2

The determinant of a rotation matrix is ± 1. For a two or three diamensional case, it is easy to prove that rotation matrix has a determinat of ± 1 with examples. But for higher diamensional case- In purely phisical terms. the determinant of a matrix tells the how the volume of a region is

changed when transformed by the matrix. As rotations do not change the volume, it must follow the determinat ± 1.

In mathematics, it is noted that a rotation $R \in \mathbb{R}^{n \times n}$ must preserve the dot product of two vectors - in other word,

$$x.y = (Rx).(Ry)$$

for any $x, y \in \mathbb{R}^n$.

But the entails $R^T R = I$, and consequently $det(R)^2 = 1$

All matrices that preserve inner product must have determinat ± 1. However any rotation can be continously deformed into the identity marix - i.e. ther exist a family of rotaion matrix R_λ, $\lambda \in \begin{bmatrix} 0 & 1 \end{bmatrix}$ such that $R_0 = I$ and $R_1 = R$. Since the determinant is a continous fuction, it follows that $\lambda \mapsto det(R_\lambda)$ is continuous, and so we must have $det(R) = 1$ otherwise we need a discontinuity in $det(R_\lambda)$ for some value of λ.

We can proof by Example If A_θ is an roation matrix, then it can be writen as

$$A_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Since it is a 2×2 matrix, its determinant is-

$$detA_\theta = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} = \cos^2(\theta) - (-\sin(\theta))\sin(\theta) = \cos^2(\theta) + \sin^2(\theta) = 1$$

Problem 3

(a) Proof

2D rotation matrix are the excptional case where the matrix multiplication is commutative i.e.

$$R1R2 = R2R1$$

where R1 and R2 be two rotation matrices on the plane.

Matrices commute if they preserve each others' eigenspaces: there is a set of eigenvectors that, taken together, describe all the eigenspaces of both matrices, in possibly varying partitions.

This makes intuitive sense: this constraint means that a vector in one matrix's eigenspace won't leave that eigenspace when the other is applied, and so the original matrix's transformation still works fine on it.

In two dimensions, no matter what, the eigenvectors of a rotation matrix are $\begin{bmatrix} i & 1 \end{bmatrix}$ and $\begin{bmatrix} -i & 1 \end{bmatrix}$. So since all such matrices have the same eigenvectors, they will commute.

Rotation matrices are commutative provided that we are talking about rotations on the xy-plane, and that's exactly because all those rotations share a common axis (the z-axis).

(b) Proof

3D rotation matrix multiplication is not commutative i.e.

$$R1R2 \neq R2R1$$

where R1 and R2 be two rotation matrices in 3D space.

But in three dimensions, there's always one real eigenvalue for a real matrix such as a rotation matrix, so that eigenvalue has a real eigenvector associated with it: the axis of rotation. But this eigenvector doesn't share values with the rest of the eigenvectors for the rotation matrix (because the other two are necessarily complex)! So the axis is an eigenspace of dimension 1, so rotations with different axes can't possibly share eigenvectors, so they cannot commute.

Rotations in 3d are non commutative because rotation changes direction of every potential other axis except itself (unlike in 2d where it is nothing to change because it is only one "axis" of rotation - it can be reduced in 3D to rotation about Z axis).

An geometric example can make it understand better- Imagine yourself walking a narrow bridge across a deep canyon. You stop and rotate face down onto the bridge, then rotate on your side to watch the beautiful sunset at the far end of the valley. By that time, however, someone who would have done the very same rotations, only in the opposite order, would be lying face down at the bottom of the canyon.

This makes that the 3D ratation matrices are not commute.

Problem 4

The proper matrix representation of 3D rotational matrix can be written as

$$R(\kappa,\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where the axis of rotation points in the z-direction (i.e., along the unit vector κ).

The equation for determining the eigen value can be expressed as -

$$(1-\lambda)[(\cos\theta-\lambda)^2+\sin^2\theta]=0$$

Which can be simplified to:

$$(1-\lambda)(\lambda^2-2\lambda\cos\theta+1)=0$$

after using $\sin^2\theta+\cos^2\theta=1$

After solving the equation, the eigen values that we get are $\lambda_1=1$, $\lambda_2=\exp^{i\theta}$ and $\lambda_3=\exp^{-i\theta}$. Depending on the value of θ the are three distict cases. So 1 is an eignenvalue of 3D matrix R.

The eigen vector of a rotation matrix can be expressed as -

$$R(\hat{n},\theta)\hat{n}=\hat{n}$$

where $\hat{n}$ is an eigenvector of $R(\hat{n},\theta)$corresponding to the eigenvalue 1.

In particular, the eigenvalue 1 is nondegenerate for any $\theta\neq 0$, in which case $\hat{n}$ can be determined up to an overall sign by computing the eigenvalues and the normalized eigenvectors of $R(\hat{n},\theta)$.

Problem 5

Here, $\sigma_i(B)$ is the ith singular value of B in sorted order. So, if A_1 and A_2 are $m \times n$ matrices, then we need to proof-

$$\sigma_{i+j-1}(A_1 + A_2) \leq \sigma_i(A_1) + \sigma_j(A_2)$$

We know if $1 \leq j \leq i \leq n$, then we can write from Fan(1951) theorem-

$$\sigma_i(A_1 + A_2) \leq \sigma_{i-j-1}(A_1) + \sigma_j(A_2)$$

In particular, we can say-

$$\sigma_i(A_1 + A_2) \leq \sigma_i(A_1) + \sigma_1(A_2)$$

And further, we can say-

$$\sigma_1(A_1 + A_2) \leq \sigma_1(A_1) + \sigma_1(A_2)$$

So, if $i + j \leq n + 1$ or $i + j - 1 \leq n$, We can say that $\sigma_{i+j-1}(A_1 + A_2) \leq \sigma_i(A_1) + \sigma_j(A_2)$

Problem 6

(a)

In this problem we have loaded the given image "board.tif" in matlab and then extracted the rectangular block between (200,90) and (300,180). The figure is attached here. The runtime for this work is 0.1404 second.

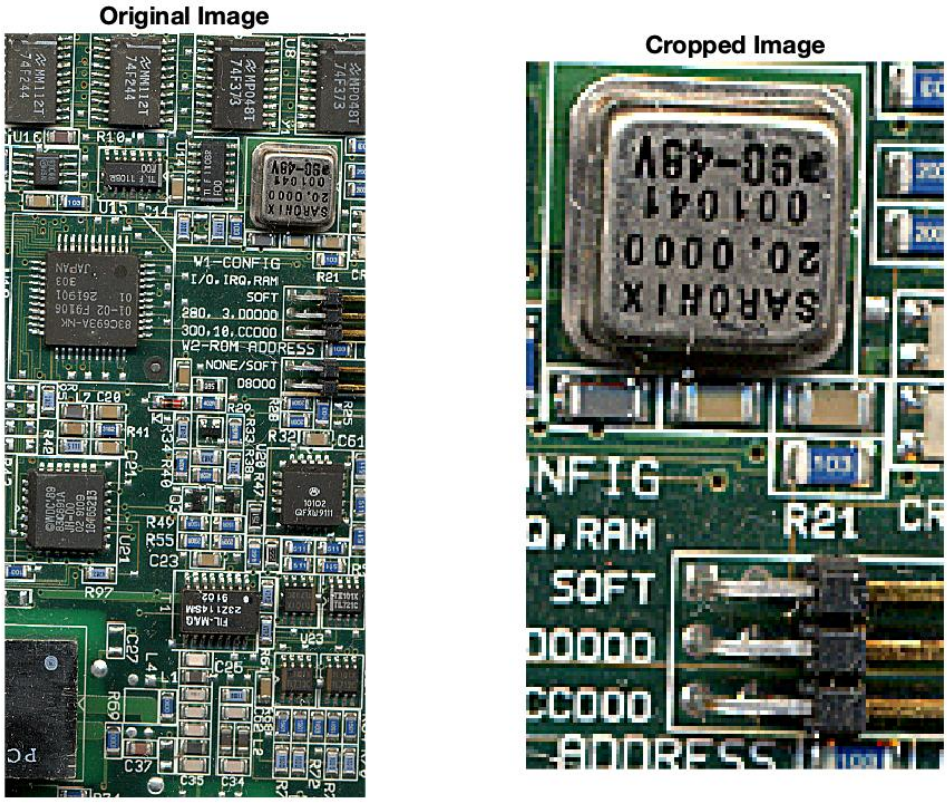


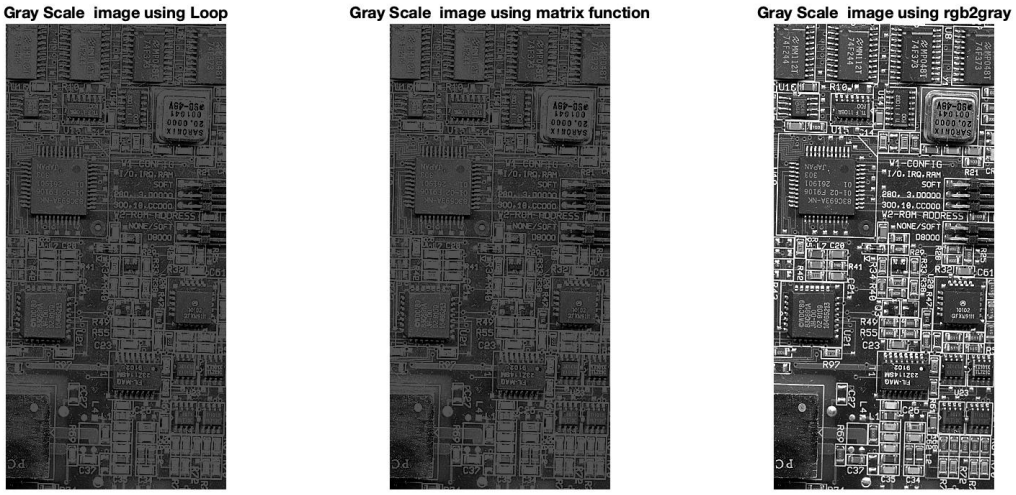
Figure 2: Main FIgure and the Cropped Rectangular Block.

(b)

In this particular problem, We have to convert the RGB image to grayscale images. We have used three techniques- (i) using nested for loops and accessing each pixel individually, (ii) using MATLAB matrix operations, (iii) using built-in image processing functions rgb2gray.

For the first two case, the grayscale image is average of the R, G, and B values. And for the build in rgb2gray it does not use average method, rather weigted method. The time taken for executing three cases are - 0.2200, 0.2096 and 0.1986 seconds respectively. So, the time taken for nested loop is higher and for build in function is lower in this case.

The images that we got by three cases are given below:



(a) Using Nested Loop (b) Using Matrix Operation (c) Using rgb2gray Function

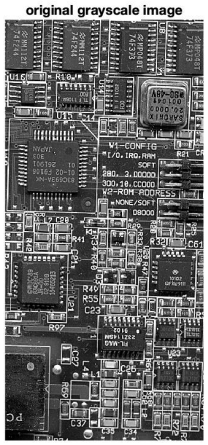
Figure 3: RGB to gray scale Image

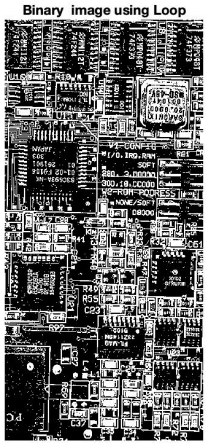
(c)

In this particular problem, We have to convert the grayscale image to a binary image using the mean grayscale value as the threshold. We have used three techniques- (i) using nested for loops and accessing each pixel individually, (ii) using MATLAB matrix operations, (iii) using built-in image processing functions im2bw.

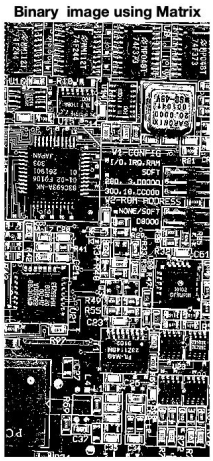
For doing the binary image, we have taken the gray scale image that we got from the build in function call of the rgb2gray function. The time taken for executing three cases are - 0.1730, 0.2082 and 0.2500 seconds respectively. So, the time taken for nested loop is lower and for build in function im2bw is higher in this case.

The images that we got by three cases are given below:

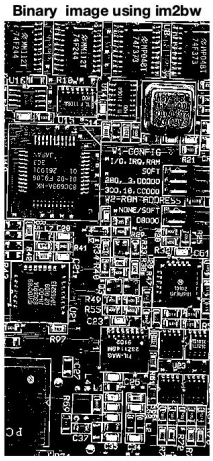




(a) Using Nested Loop



(b) Using Matrix Operation



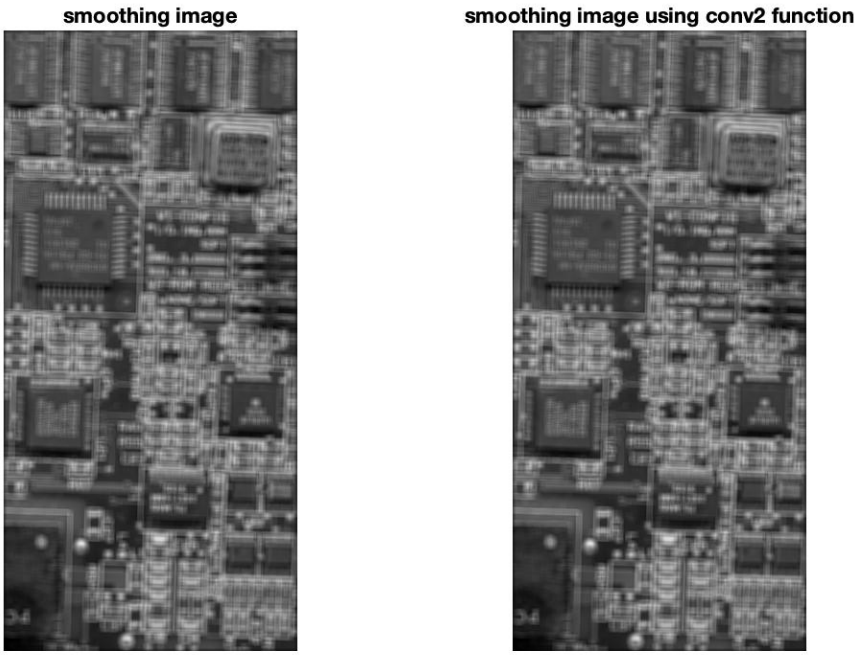
(c) Using rgb2gray Function

Figure 4: Gray scale to Binary Image

(d)

In this particular problem we need to smooth the grayscale image created above using a 7×7 averaging filter. we have used two different ways: (i) using for loops, (ii) using the MATLAB function conv2. The concept behind this problem is to handle the edge of the images. To solve this issue, i have use zero padding around the image with 3 extra indexes all around. Thus all the points in the main grayscale image is covered and we get a smoothed filtered image.

The images that we got by three cases are given below:

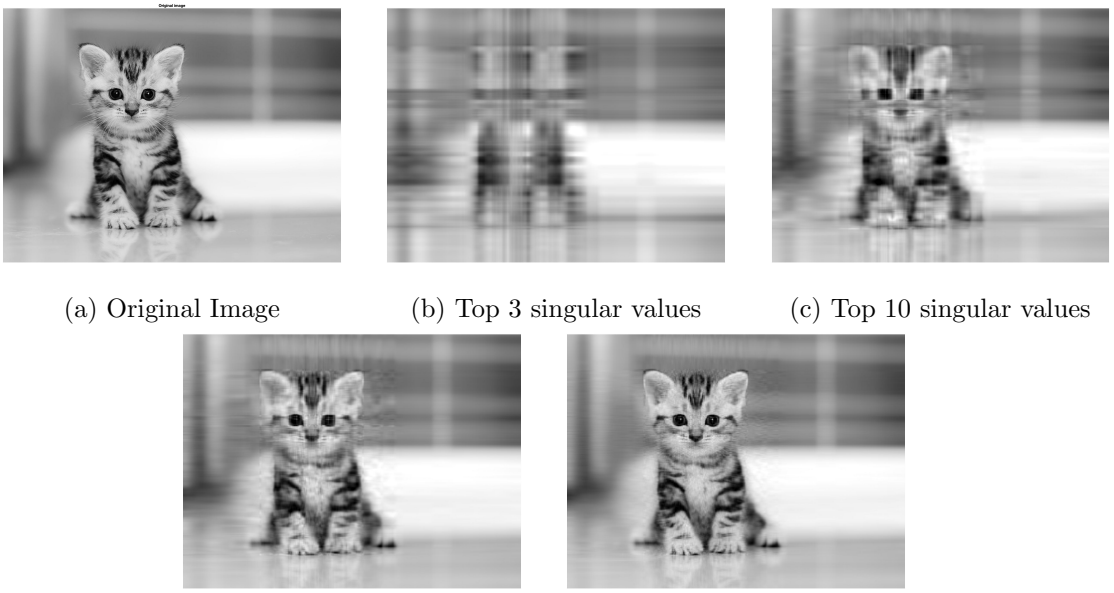


(a) Smothing Using For loops (b) Smoothing using conv2 Function

Figure 5: Smoothing Gray scale image by 7x7 averaging filter

Extra Credit

In this particular problem, I have taken an image called "cat.jgp" and the size of it is 1442x1920. I have attache this image in the report. Then I have used SVD compression to the images with top 3, 10, 20, 40 singular values and vectors. The image that we got by compression, We see if we increase the number of singular values, the image become more visible and error is reduced. The images that we got are given below:



* (a) Original Image (b) Top 3 singular values (c) Top 10 singular values

(d) Top 20 singular values (e) Top 40 singular values

Figure 6: Image with compressed SVD values

Then we have plotted the error Vs singular values. We have seen that the relative error decresses with the increase in number of singular values.

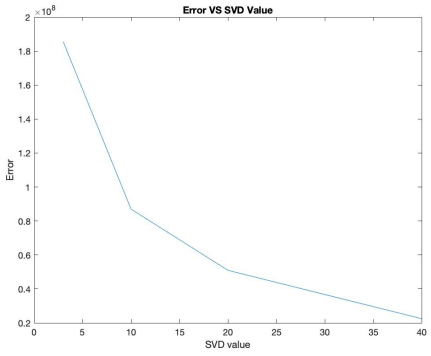


Figure 7: Error Vs Singular Values

And the compression ratio is tabulated below-

Table 1: Compression ratio

Singular Values	Compression raio
3	$7.24 \times 10^{-}3$
10	0.0241
20	0.0483
40	0.0966