Homework 9

Abhigyan Chattopadhyay
ME19B001

November 1, 2019

Image Reader

November 1, 2019

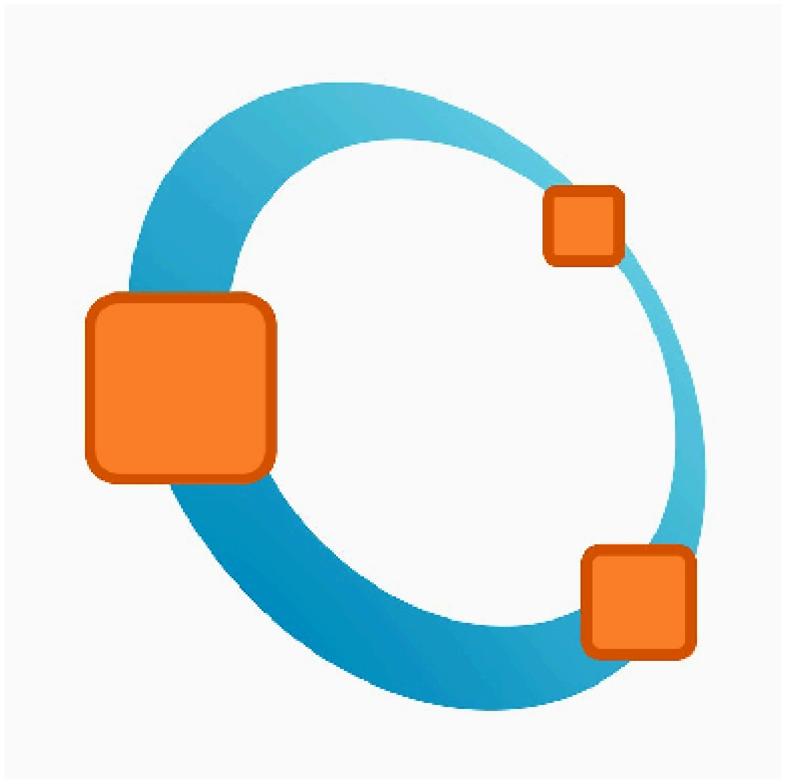
1 Homework 9 - Q1

1.1 Problem statement:

> Pick an image – say your own passport size photograph – and convert it to gray scale using a tool such as gimp. Load it in octave as a matrix. 1. Reshape the matrix to a 1 D array. Make a histogram of this image using 16 bins that span over the 256 levels of gray scale. 2. Add a constant integer such as 50 to every element of this matrix. Make the histogram again and also visualize the image using imshow. 3. Subtract 50 from every element of this matrix. Make the histogram again and view it using imshow. 4. Compare the original matrix before 1. with the matrix in case 3. and comment on what an increase or decrease of brightness control does to an image. What are the other histogram operations you could think of – eg., what is contrast enhancement or gamma correction?

First, we'll import the image into a matrix, using `imread`, and then show the image using `imshow`.

```
[1]: t = imread("../Pics/octave_logo.jpg");
     imshow(t)
```



To convert it into Grayscale, I've used the rgb2gray algorithm that is used by Octave, in a different library, available on SourceForge. There is a question on this topic on StackOverflow, which I have taken help from: Implementing rgb2gray in Octave

[2]:
```
function y = rgbToGray(image)
    y = (0.298936 * image(:,:,1) + 0.587043 * image(:,:,2) + 0.114021 * image(:
 ↪,:,3));
endfunction;
```

[3]:
```
t_grey = rgbToGray(t);
```

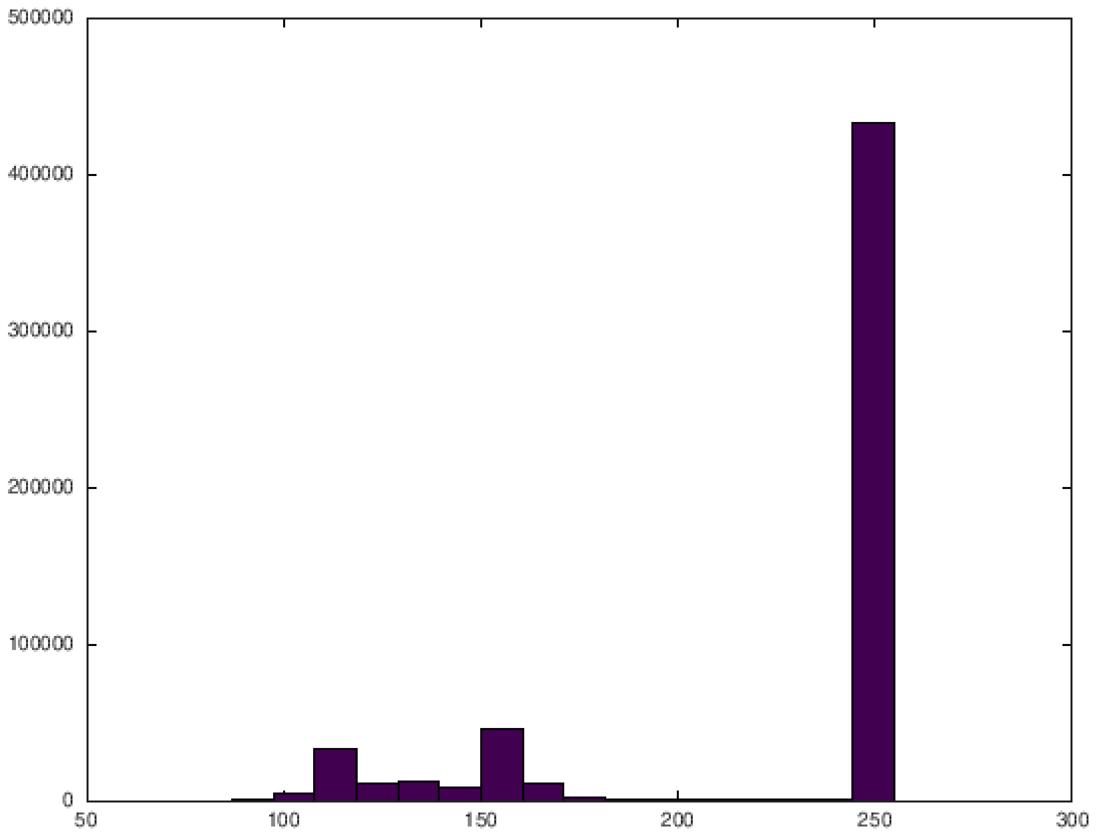
Now, we're reshaping this matrix into a single row:

[4]:
```
t_grey_line = reshape(t_grey,1,size(t_grey)(1)**2);
```

We're plotting a histogram of the numbers we have, using 16 bins, with the `hist` command.

In the histogram below, we see that the most populated number is 250, which corresponds to white, and it logically makes sense.

[5]:
```
hist(t_grey_line,16)
```

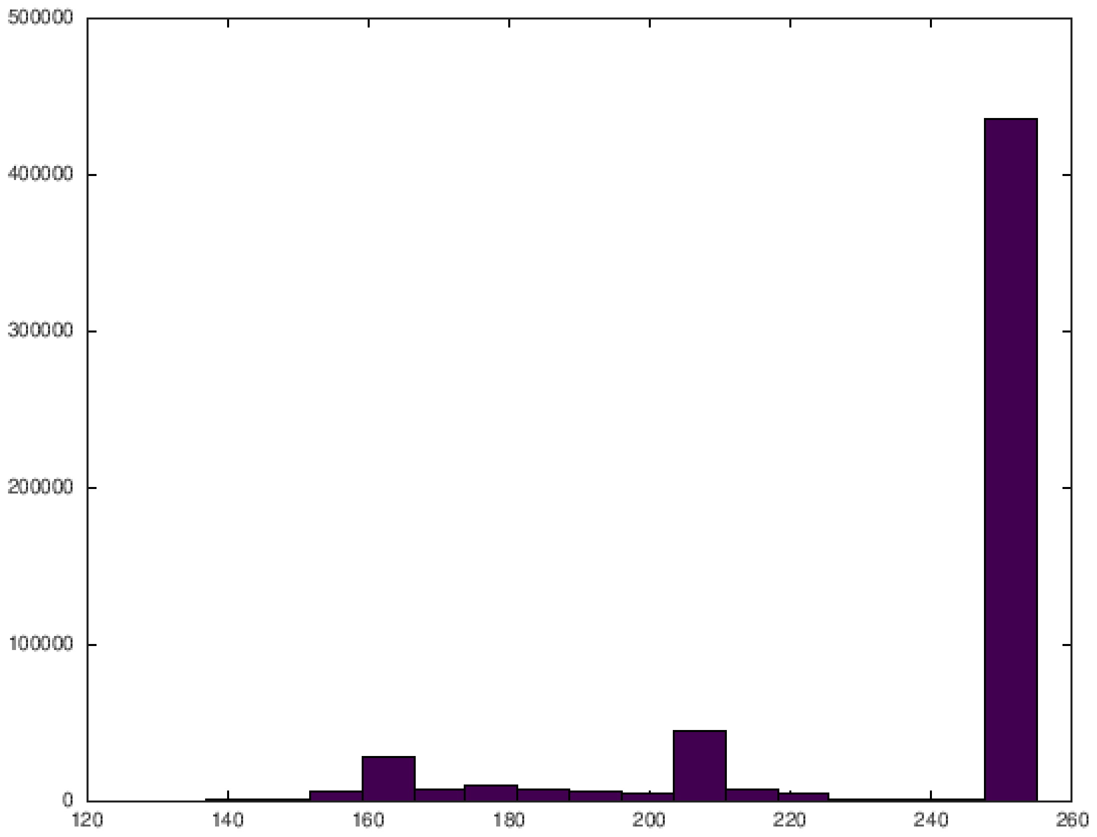


Showing the grayscale version of the image:

```
[6]: imshow(t_grey)
```

Now, adding 50 to each element, and then plotting the histogram with 16 bins:

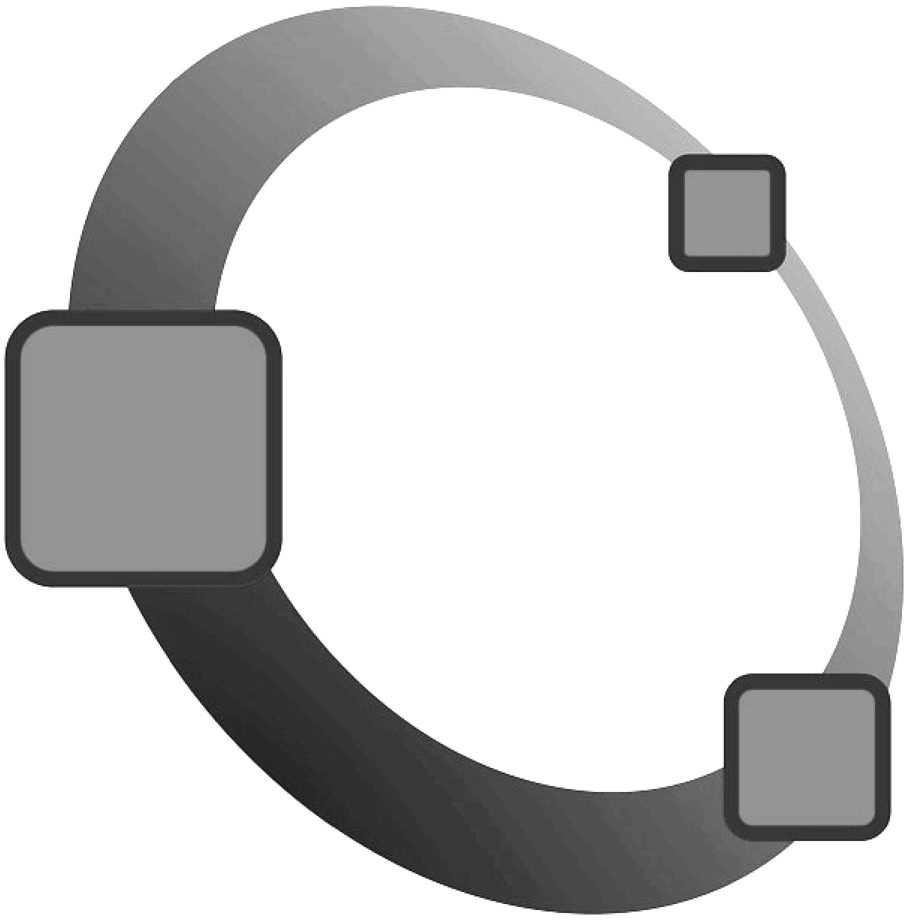
[7]:
```
t_grey_added= t_grey_line.+50;
hist(t_grey_added,16)
```



Now, we're converting the line matrix back to a 750x750 image, using the `reshape` command, and then plotting it:

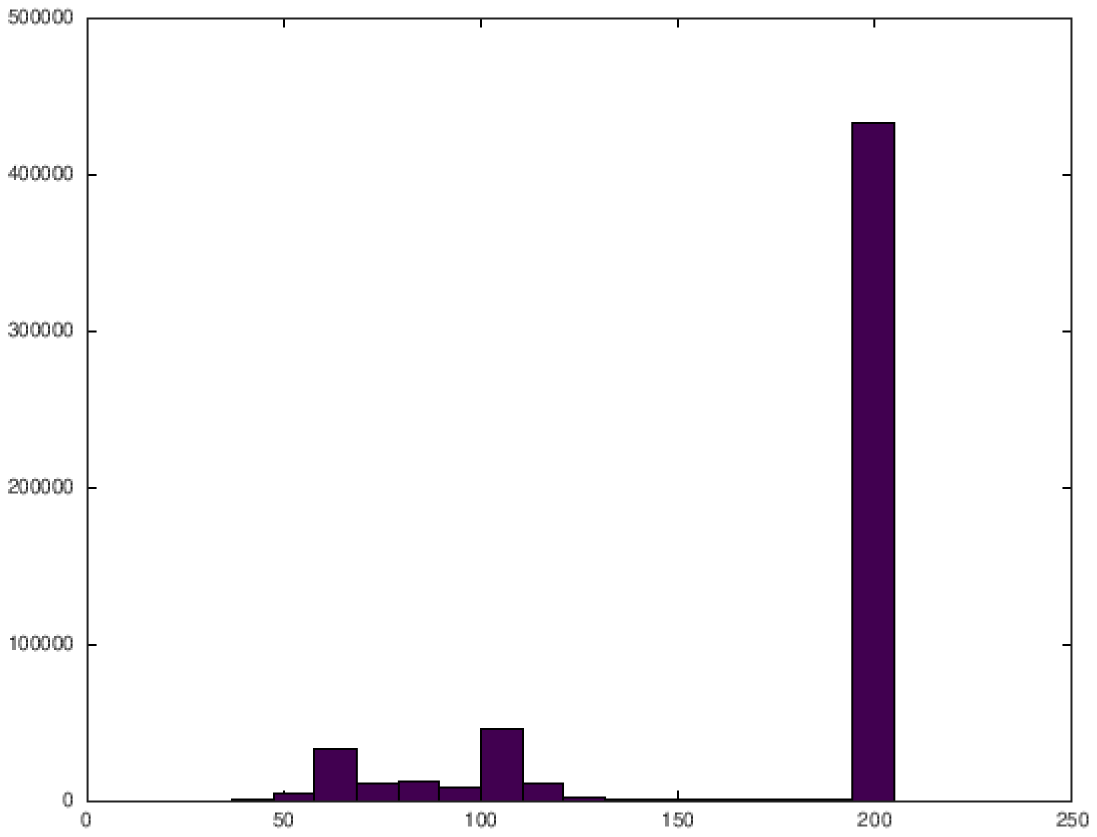
[8]:
```
t_grey_bright = reshape(t_grey_added,750,750);
```

[9]:
```
imshow(t_grey_bright)
```



Now, we're repeating the same as above, but after subtracting 50 from each element of the matrix.

[10]:
```
t_grey_subtracted = t_grey_line.-50;
hist(t_grey_subtracted,16)
```



[11]:
```
t_grey_dark = reshape(t_grey_subtracted,750,750);
imshow(t_grey_dark)
```

Now, we're finding the difference between the bright and the dark images:

```
[12]: difference = (t_grey_bright - t_grey_dark)(1);
      difference
```

```
difference = 54
```

Next, the difference between the normal and the bright images:

```
[13]: difference2 = (t_grey_bright - t_grey)(1);
      difference2
```

```
difference2 = 4
```

Finally, the difference between the normal and the dark images:

[14]:
```
difference3 = (t_grey - t_grey_dark)(1);
difference3
```

```
difference3 = 50
```

Color Normaliser

November 1, 2019

1 Homework 9 - Q2

1.1 Problem statement:

> Create a 2D matrix T of dimensions (100 x 100) with random numbers. Apply the computation represented by the following pseudo code – say 50 times – and see what does it do to the array. Visualize using pcolor plot and comment on what you observed.

```
counter = 1
Step-1:
For each i, j (skipping the ones on the boundary):
Tnew(i,j) = T(i,j) + 0.2*(T(i+1,j) + T(i-1,j) + T(i,j+1) + T(i,j-1) - 4*T(i,j))
Step-2:
T = Tnew
Step-3:
pcolor(T)
counter++;
Go back to Step-1 till the counter reaches 50.
```

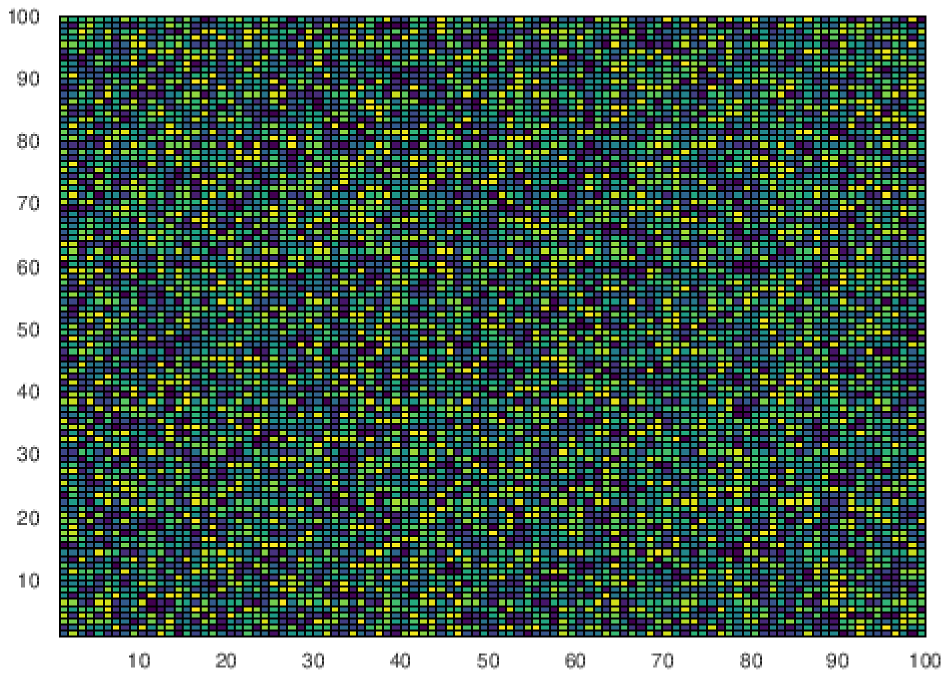
Here, we implement an algorithm to normalize the difference between random points, and then see the final image

In the cell below, we create a random 100×100 matrix, which is displayed using the `pcolor` command of Octave.

(I've suppressed the output using the `;` as it isn't very pleasing to see a 100×100 matrix :)

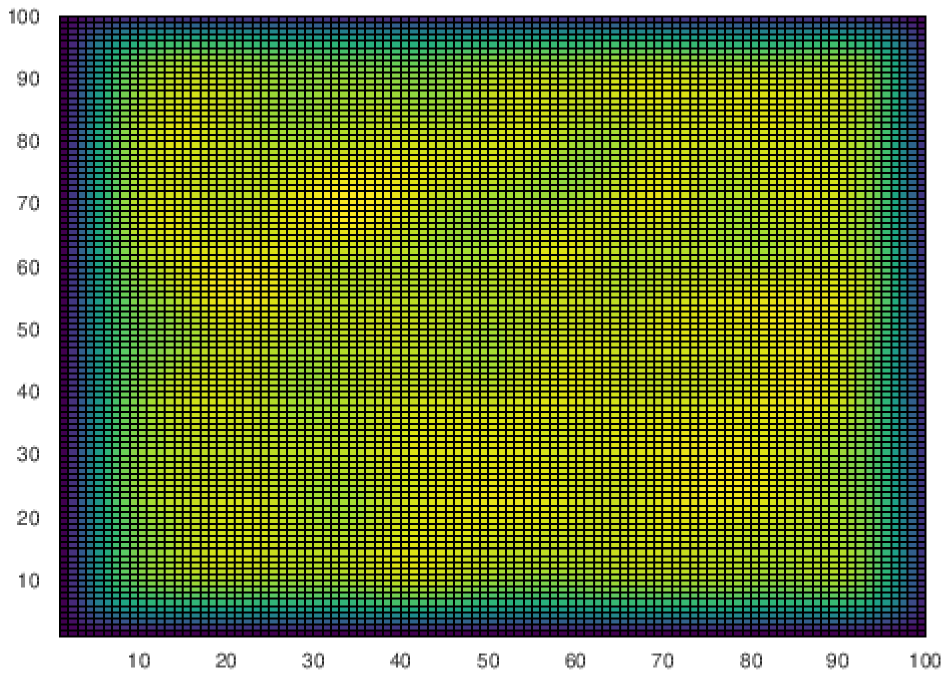
[1]:
```
A = rand(100);
B = zeros(100);

pcolor(A)
```



In the cell below, we perform the task as mentioned in the question, and obtain the following resultant matrix:

[2]:
```
for iterator = 1:50
    for i = 2:size(A,1)-1
        for j = 2:size(A,2)-1
            B(i,j) = A(i,j) + 0.2*(A(i+1,j) + A(i-1,j) + A(i,j+1) + A(i,j-1) -␣
 ↪4*A(i,j));
        end;
    end;
    A = B;
    pcolor(A)
end;
```



This code takes a little while to run, but the final output is dark at the edges, and bright at the centre.

Thus, the code was able to reduce the differences of values of the points, and we finally obtained a homogenous arrangement, distributed about the centre.

Voronoi Diagrams & Delaunay Triangulation

November 1, 2019

1 Homework 9 - Q4

1.1 Problem statement:

> Look up what is either (a) a voronoi construction or (b) Delauney triangulation using octave from the internet. Illustrate the same using an example code and output.

We're going to do both of them

1.1.1 Voronoi Diagram:

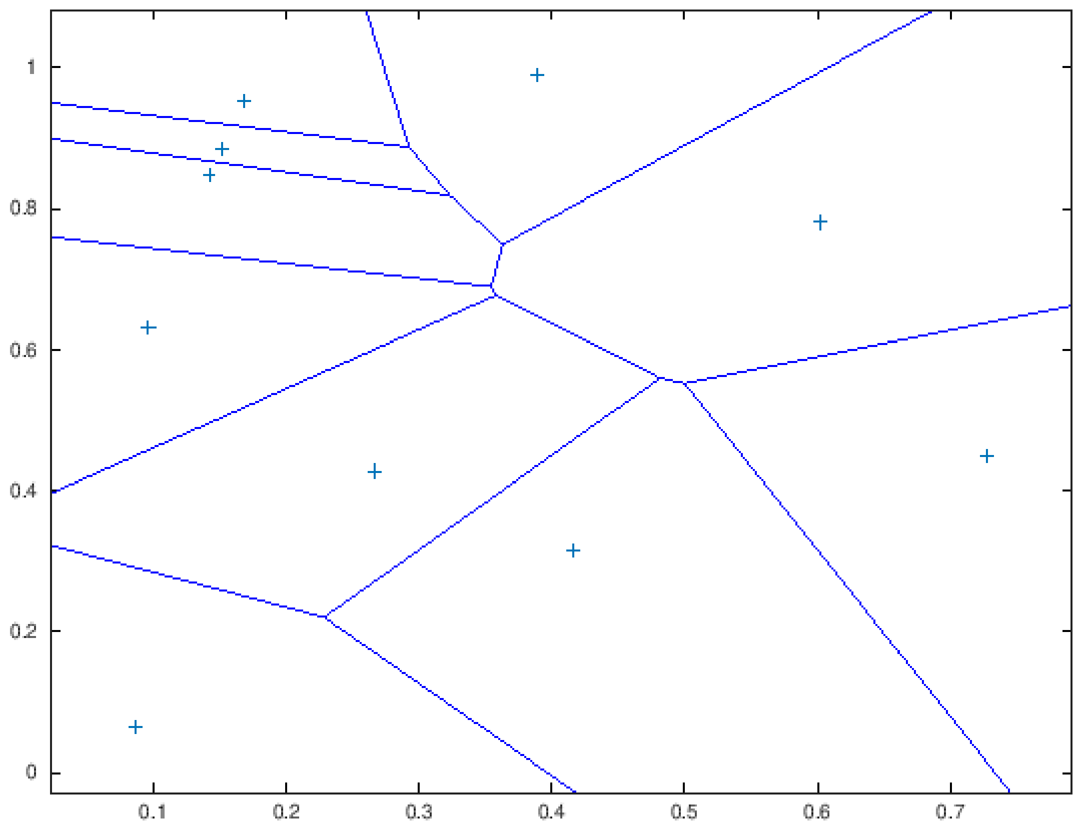
Now, the first one: A voronoi is basically the set of lines that are equidistant from any 2 points in a space filled with points. So, for this we're going to need a set of points, which we will do using the `rand` command in Octave.

[1]:
```
x = rand(1,10);
```

[2]:
```
y = rand(1,10);
```

It looks like this:

[3]:
```
v = voronoi(x,y);
```



1.1.2 Delaunay Triangulation:

This is a kind of complement to the Voronoi Diagram, hence I couldn't help but put it along with this :)

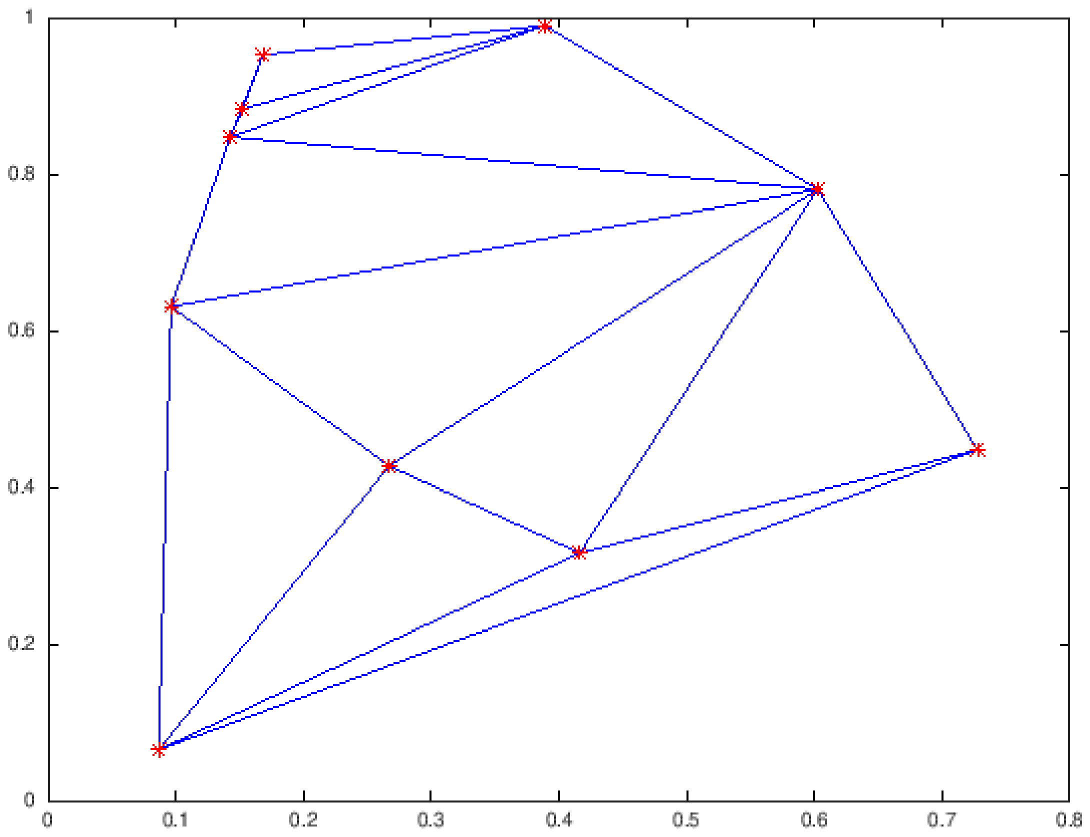
Now, Delaunay Triangulation is the set of triangles for a given set of points such that no point is inside the circumcircle of any of the triangles.

It turns out that the circumcenters of the said triangles are the vertices of the Voronoi Diagram!

More on this: Voronoi Diagram and Delaunay Triangulation

[4]:
```
T = delaunay(x,y);
```

[5]:
```
X = [ x(T(:,1)); x(T(:,2)); x(T(:,3)); x(T(:,1)) ];
Y = [ y(T(:,1)); y(T(:,2)); y(T(:,3)); y(T(:,1)) ];
p1 = plot(X,Y,"b",x,y,"r*");
```


1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

Series of Curves in Octave

November 1, 2019

1 Homework 9 - Q6

1.1 Problem Statement:

Choose x to be a 1D array with values from 0.01 to 0.99 in intervals of 0.02. Plot a family of curves of the following function for values of M ranging from 0 to 4 in steps of 0.5.

$$f(x) = x\log(x) + (1-x)\log(1-x) + Mx\log(x)$$

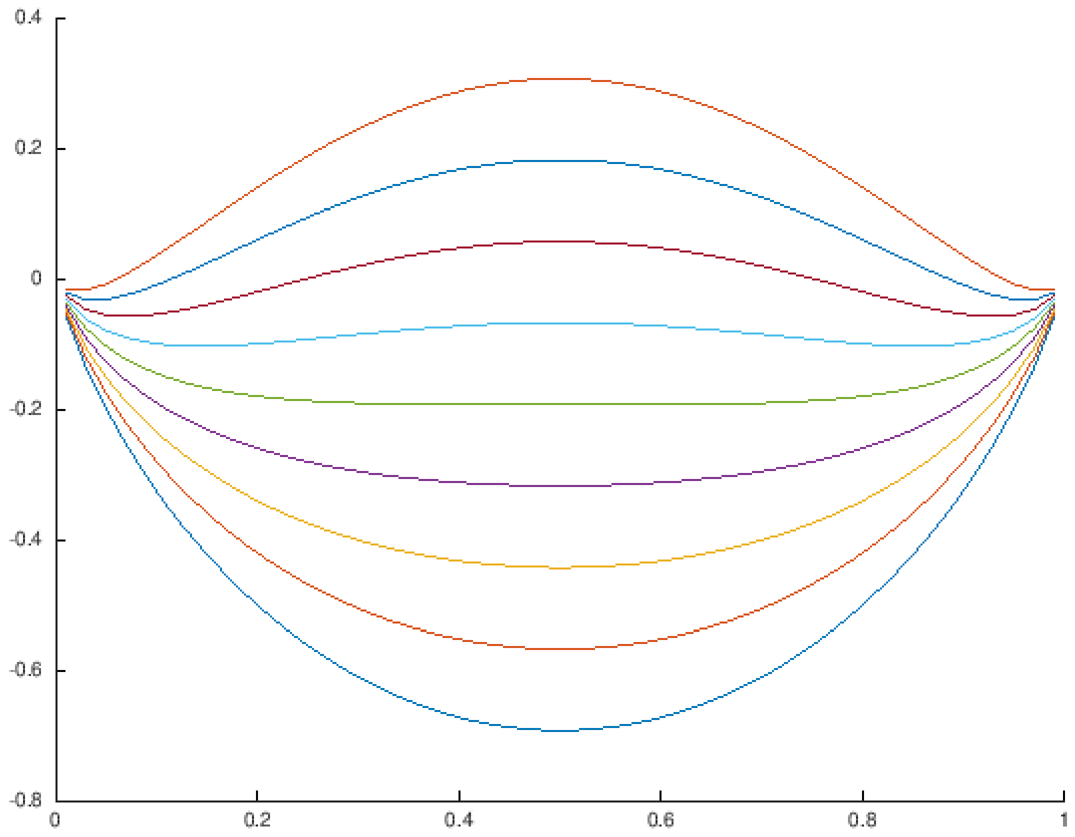
Generating the array `x`:

[1]:
```
x = [0.01:0.02:0.99];
```

`hold on` is used to prevent plotting from occuring in separate graphs.

Thus, we have the following graph for different values of M

[2]:
```
hold on
for M = [0:0.5:4]
    y = x.*log(x)+(1.-x).*log(1.-x).+M.*x.*(1.-x);
    plot(x,y)
endfor;
```


0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
0.2
0.4
0.6
0.8
1

Orthogonal Rotation Matrices

November 1, 2019

1 Homework 9 - Q7

1.1 Problem Statement:

Consider the following two matrices:

$$A = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.1 & 2.1 & 3.1 \\ 2.5 & 1.6 & 3.3 \end{bmatrix} \text{ and } T = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The new matrix B is defined as follows:

$$B = T \times A \times \text{transpose}(T)$$

Evaluate the trace of A and B for different values of θ and comment on your observation.

This is a very simple problem in GNU Octave/MATLAB, as they have very extensive linear algebra support.

[1]:
```
A = [1.0,2.0,3.0;1.1,2.1,3.1;2.5,1.6,3.3]
```

```
A =

   1.0000   2.0000   3.0000
   1.1000   2.1000   3.1000
   2.5000   1.6000   3.3000
```

[2]:
```
theta = [pi/2:pi/2:5*pi]
```

```
theta =

 Columns 1 through 8:

    1.5708    3.1416    4.7124    6.2832    7.8540    9.4248   10.9956   12.5664

 Columns 9 and 10:

   14.1372   15.7080
```

[3]:
```
for i = 1:10
    T = [cos(theta(i)), sin(theta(i)), 0; -1*sin(theta(i)), cos(theta(i)),0;␣
 ↪0,0,1];
    B = T * A * T';
    disp('trace(A) = '),disp(trace(A))
    disp('trace(B) = '),disp(trace(B))
end;
```

```
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
trace(A) =
 6.4000
trace(B) =
 6.4000
```

Thus, the value of `trace(A)` is equal to `trace(B)` whenever we apply this transformation to A.

Basically, the matrix $T = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is the inverse of the orthogonal rotation matrix in the z direction, hence, when we multiply it with A, we basically rotate each of its columns by θ in the clockwise direction. Let us call this transformed matrix as A_{rot}.

Now, when we multiply the new matrix, A_{rot} with the transpose of T, we're basically getting a new matrix with the same trace.