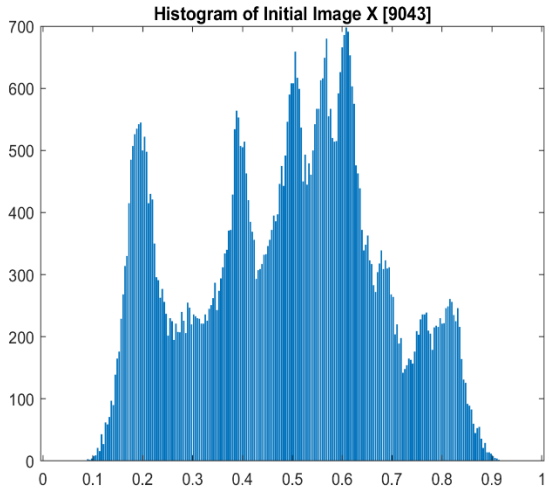
DIGITAL IMAGE PROCESSING REPORT

MATLAB ASSIGNMENT

NIKOLAOS GIAKOUMOGLOU
AEM 9043

0 INITIAL IMAGE



Initial Image and its histogram

1 POINT TRANSOFRMATION

The transformation of the output is

$$y = a \cdot x + b$$

where

x is the input X(n1, n2),

y is the output Y(n1, n2),

$a, b \in \mathbb{R}$

In order to find the a, b, we solve the linear system of 2 equations, replacing (x, y) each time with 1 point of the plane (X,Y).

- When $x > 0$ and $x < x1$, we have the points (0, 0) and (x1, y1). So

$$y1 = a \cdot x1 + b \text{ and } 0 = 0 + b \Rightarrow b = 0 \text{ and } a = y1/x1$$

- When $x>x1$ and $x<x2$, we have the points (x1, y1) and (x2, y2). So

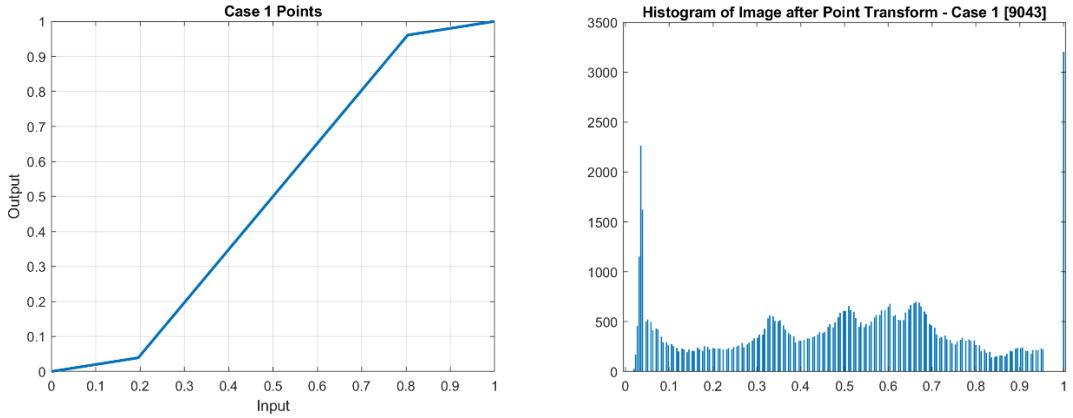
$$y1 = a \cdot x1 + b \text{ and } y2 = a \cdot x2 + b \Rightarrow a = (y2-y1)/(x2-x1) \text{ and } b = y1 - a \cdot x1$$

- When $x>x2$ and $x<1$, we have the points (x2, y2) and (1,1). So

$$y2 = a \cdot x2 + b \text{ and } 1 = a + b \Rightarrow a = (y2-1)/(x2-1) \text{ and } b = 1 - a$$

We can add the equation "=" arbitrarily since the input-output function is continuous (in the code *pointtransform.m*, the spaces are [x, y)).

In the *main.m*, we run the transformations for the given points (x1, y1) = (0.1961, 0.0392) and (x2, y2) = (0.8039, 0.9608) which represent Case 1. The results are the following:



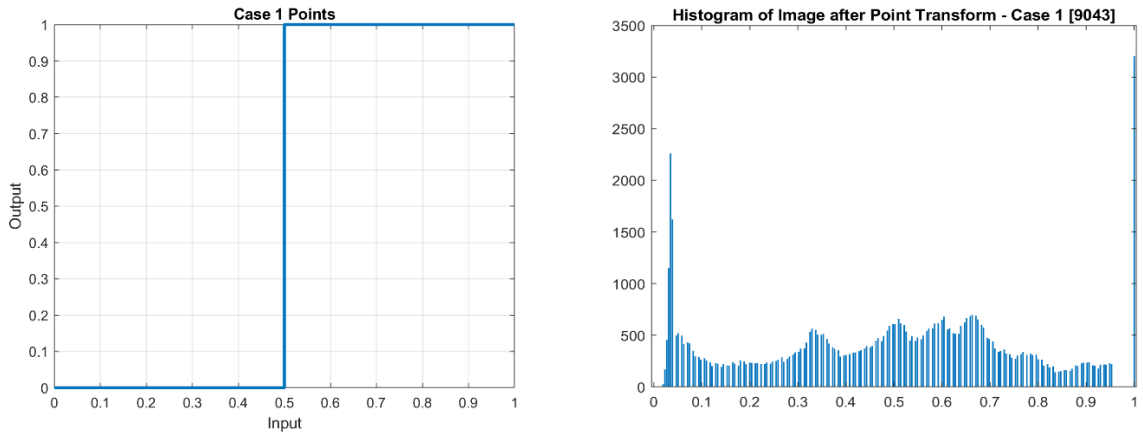
Diagrams: Case 1 Input vs Output function and image's histogram

Image: Lena after Case 1

Then we are asked to perform clipping in 0.5, which means all the values of the image less than 0.5 become 0 and the rest 1. We will never find values at 0.5 since 255/2=127.5 and the initial image consists of integers between 0 and 255, so 127.5 is not included (we can confirm that by typing *length(find(X==0.5))* in *MATLAB*'s *command window*). In order to achieve that, we set the

(x1, y1) = (0.5, 0) and (x2, y2) = (0.5, 1)

and we get the following input-output function. The results are the following:



Diagrams: Case 2 Input vs Output function and image's histogram

Image: Lena after Case 2 (clipping)

2 HISTOGRAM TRANSFORMATION

In this section, we transform the image, so that its histogram has specific properties.

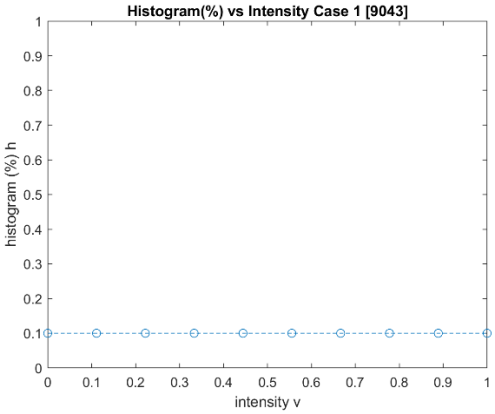
2.1 TRANSFORMATION BASED ON THE HISTOGRAM

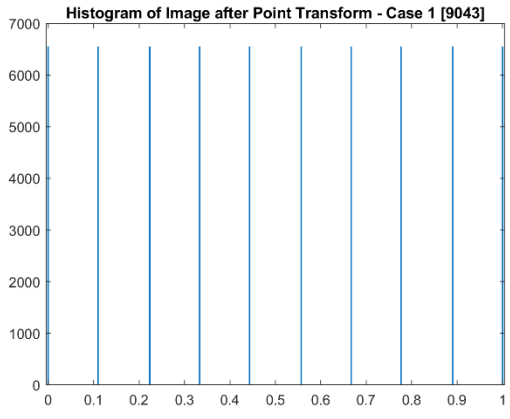
For the implementation of *histtransform* function in *MATLAB*, first we check if the sum of all h is different of 1 and if any value of v is negative. In this case, we set arbitrarily Y = X. But this is not necessary since the inputs should be checked by the user and that is the reason this section is in comments. We define by *N* the total number of pixels in the image and by *cumsumh* the cumulative sum of h (e.g. if h = [0.4 0.4 0.2], then cumsum(h) = [0.4 0.8 1]). We initialize Y by NaN values (Not A Number). We will apply the greedy algorithm for each pixel. First, we find the index [*row*, *col*] of the minimum index or indexes of the initial image X. Note that *row* and *col* could be vectors of more than 1 elements. But we only need the first index,

so we will gain that by *row*(1) and *col*(1). We set the *row(*1) x *col*(1) pixel of the transformed image equal to v(j), where j is initialized with 1, and we set the initial image *row*(1) x *col*(1) pixel equal to inf, so we will not use it in any next iteration. In order to move to the next element of v(j), we use the cumulative sum. The reason we apply the algorithm for each pixel i=1,2, 3, …, N is that when we exceed the h(j) pixels that are transformed, j is increased by 1. But since the iteration is in increasing number of pixels i=1,2, 3, …, N, we need to sum the elements of the current j from 1 to j (that is how cumulative sum is defined by default). The loop is finished when we transform every pixel i=1, 2, 3, …, N into the assigned value v(j). We can be sure that this is true. But how can we be sure that v is not out of bounds? When we reach the very final pixel i=N, N/N=1 which must be equal to sumsumh(length(j)) which is true since the sum of all h is equal to 1 and therefore the last element of sumsumh is always 1.

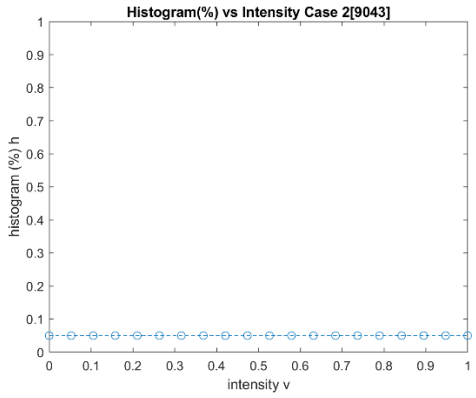
Then we were asked to run this function for 3 cases.

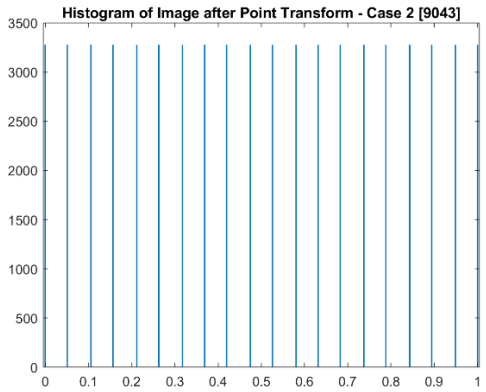
In the first case, we have 10 bins in [0,1] with the same percentage 10% to each. In the second case, we have 20 bins in [0,1] with the same percentage 5% to each. We expect to see an image with 10 and 20 distinctive values of brightness respectively and a histogram with “uniform” distribution (a uniform distribution should have (1/255) ·100% percentage in each distinctive value) with all the values concentrated on the values v. Obviously, in case 2, the image is more clear than in case 1 due to the fact that the amount of different distinctive brightness is doubled. The results are the following:





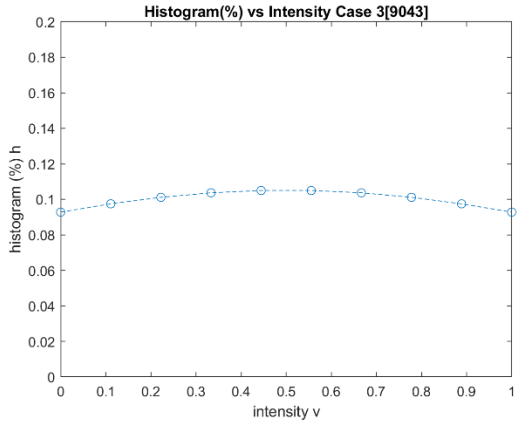
Case 1: apply hist transform

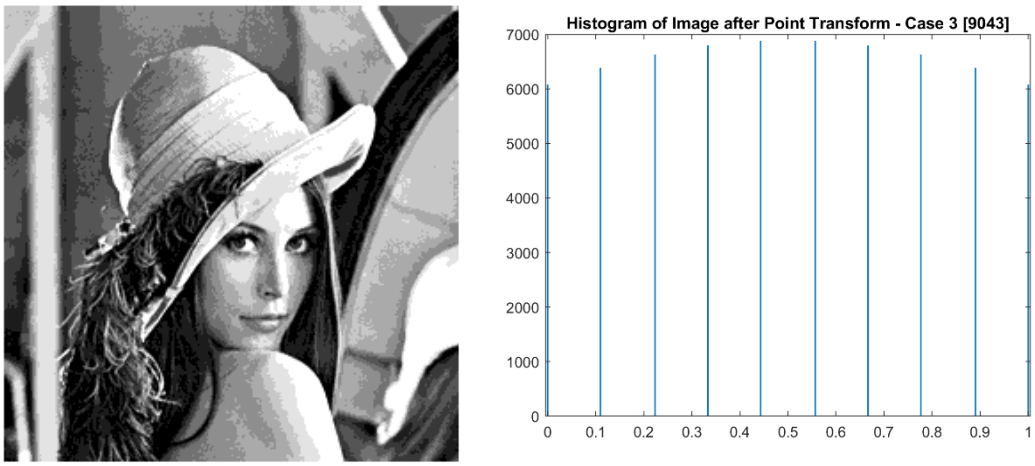




Case 2: apply hist transform

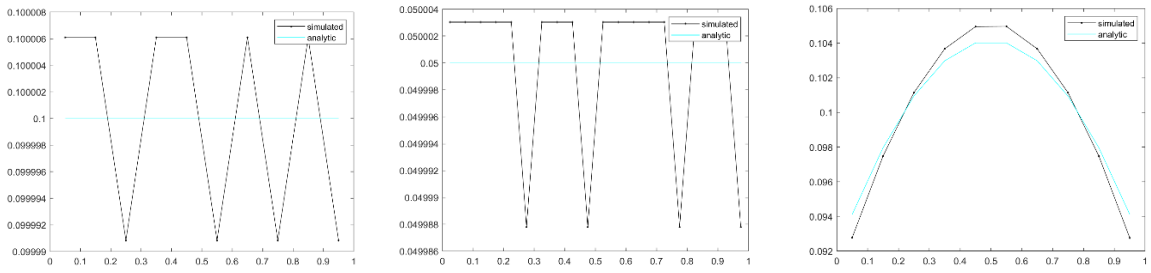
In the third case we also have 10 bins with the same transformed values, but the percentage is defined by a Normal (Gaussian) distribution with mean equal to 0.5 and variance equal to 1 (defined by default). So, we expect to see an image and a histogram with 10 distinctive values of brightness, but most of the values will be concentrated on mean 0.5 and will be less as we moving away from the mean 0.5. However, we will not see any significant difference from Case 1, since the normal distribution can be easily fitted by a uniform distribution which mean the difference of percentage of a uniform and the normal tend to be 0 (that is because variance is 1). The results are the following





Case 3: apply hist transform

In all 3 cases, the histogram of the transformed image is very well fitted in the histogram (%) vs intensity as shown here:



Histogram from transformed image vs analytic histogram of v-h (left is Case 1, middle is Case 2 and right is Case 3)

Note than in the first two cases, it might seem that it does not seem, but the units are very close at 0.000001, which means that each bin differ from each other at +/- 1. In the third case, the different are much more significant that the first two cases.

*Note: A second greedy algorithm histtransform2 was implemented as well. Define N the total number of pixels of the initial image. The algorithm loops for every v(j). Each of these loops, transform Nv(j) pixels, where Nv(j)=N*h(j) (e.g. if h(j) = 0.1 or 10%, and N=900 then the loop of v(j) should consist of 90 loops itself). We find the index row-col of the minimum index or indexes of the initial image X. Note that row and col could be vectors of more than 1 elements. But we only need the first index, so we will gain that by row(1) and col(1). We set the row(1) x col(1) pixel of the transformed image equal to v(j) and we set the initial image row(1) x col(1) pixel equal to inf, so we will not use it in any next iteration. The drawback of this algorithm is that Nv(j) is not essentially integer, so we have to use the floor function (not the ceil, because we might exceed the boundaries of the image). But this way, we might lose some pixels and not change them. Those pixels stay as they were initialized at v(length(v)). So we except an abrupt raise in the histogram to the last value v(length(v)). The advantage of this algorithm is that for v(j)=1, 2, …, length(v)-1, the histogram's distribution fits the desired histogram with 0 error.*

2.2 ESTIMATE HISTOGRAM FROM DISTRIBUTION

The *pdf2hist* function, takes intervals as an input d, where the first interval is [d(1), d(2)], the second interval is [d(2), d(3)], … etc. and f is a function pointer declared like this e.g. f = @(v) v+1. In order to calculate f(v = 1) we call the function feval: feval(f,1). Obviously, d is any vector where d(i)<d(i+1) for each i and d(i) can be outside the [0, 1]. Now that we have clarify what exactly is the input, lets explain how to code works. First, we define n as the length of d. The h will be of length n-1 since the intervals of are n-1. The loop iterates from 1 to n-1. We will estimate the

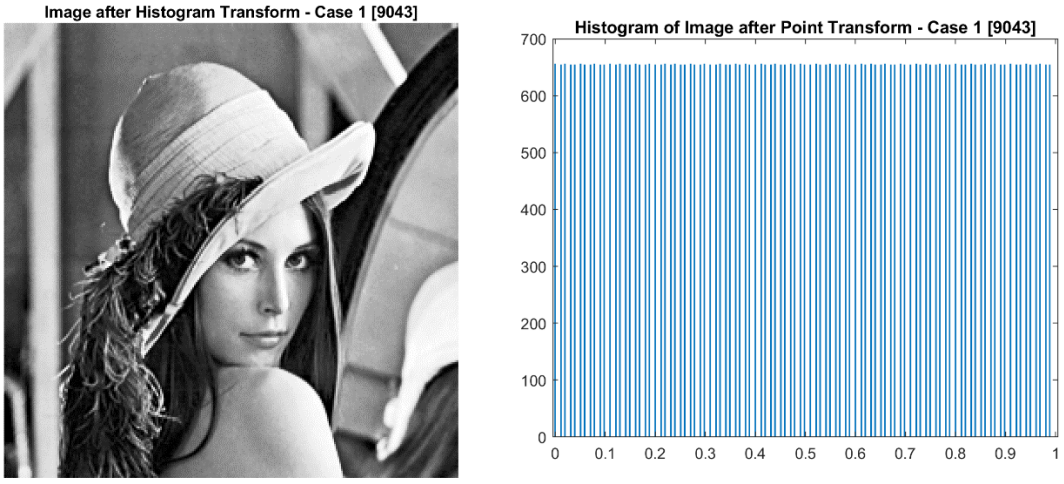
$$\Pr\{d(i) < V < d(i+1)\} = \int f(v)\, dv \text{ from } v(i) \text{ to } v(i+1),$$

with the area of the rectangle with width v(i+1) – v(i) and height f [(v(i)+v(i+1)]/2).

That is the value of h(i). Then we will divide each h(i) with the sum of h(i) for each i, so as the sum of all h is equal to 1.

2.3 TRANSFORMATION BASED ON THE PROBABILITY DENSITY

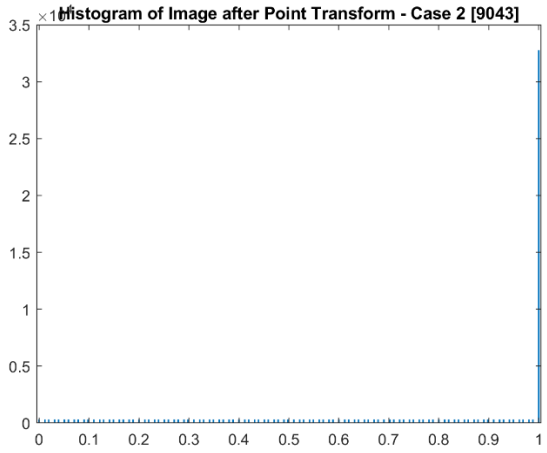
We will run *pdf2hist* with 3 different pdf (probability density function). In Case 1, we define f as a uniform distribution in [0, 1] so f(v) = 1 and d=0:1 with step 0.1. Then we take the output h and use it as an input to *histtransform* with v = d. After this we plot the histogram and the image after the transformations. We observe that the quality of the image is quite nice since we have 100 distinctive samples of brightness. The results are the following

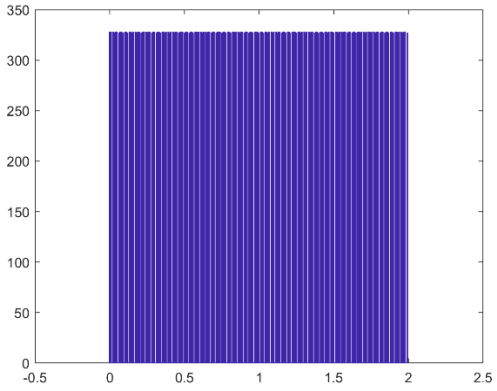


Case 1: apply pdf2hist

In Case 2, we define f as a uniform distribution in [0, 2] so f(v) = 1/2 and d=0:2 with step 0.1. This is the only case where the distribution is not defined in [0, 1] so we will analyze it further. We take the output h and use it as an input to *histtransform* with v = d. After this we plot the histogram and the image after the transformations from 0 to 1 with step 1/255. It is important to see that the histogram of the ploted image has more than 50% of its values at [1, 2]. That is because the transformation creates pixels outside the corresponding interval without taking into consideration that the image has values between 0 and 1. All the values which are more than 1, are assigned to 1, so the histogram of the ploted image looks like this.

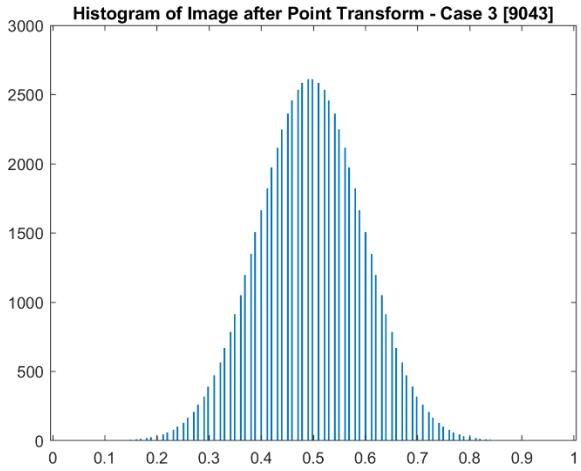
However, if we check the histogram of the values of the image, they are indeed a uniform distribution in [0, 2] as we wanted. But when we plot the image, all the values more than 1 are assigned to 1 and become white. We observe that the quality of the image is not good since all brightness samples more than 1 are clipped to 1 (white). The results are the following





Case 2: apply pdf2hist

In Case 3, we define f as a normal distribution in [0, 1] with mean 0.5 and variance 0.1 so f(v) = (1/(0.1*sqrt(2*pi)))*exp(-0.5*(((v-0.5)/0.1).^2)) and d=0:1 with step 0.1. Then we take the output h and use it as an input to *histtransform* with v = d. After this we plot the histogram and the image after the transformations. We observe that we have many "greys" since most of the samples have brightness equal to 0.5 and near that value, since the mean of the normal distribution is 0.5 and the variance is 0.1 which mean that 70% of the values are at [0.5-0.1, 0.5+0.1] = [0.4, 0.6] and 95% of the values are at [0.5-0.2, 0.5+0.2] = [0.3, 0.7] according to Normal's distribution property. The results are the following



Case 3: apply pdf2hist