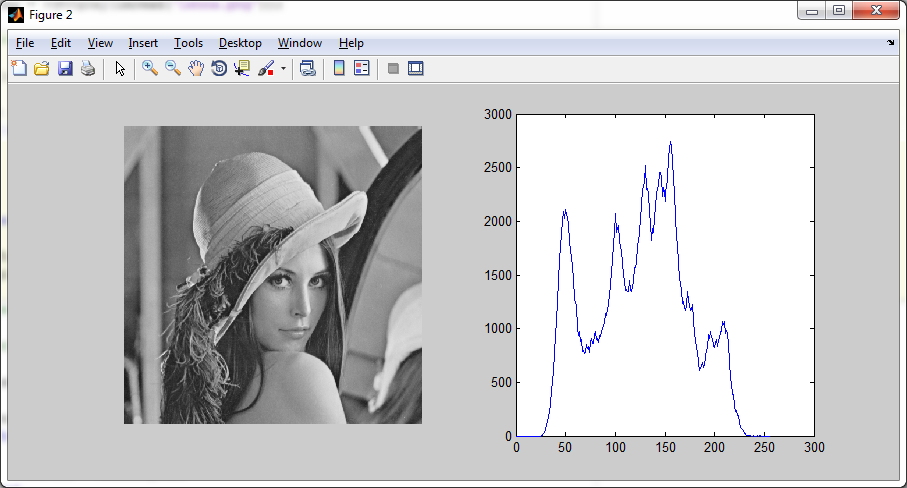
**Image Processing:**

**Tutorial 3: Image Histograms**

**Date: 22/10/18**

**Task 1: Compute Histogram**

Histograms can be rather important when it comes to adjusting the brightness of an image. Your first task is to compute a histogram of an image of your choice and display this as a plot. **[Do not use Matlab’s imhist function]**



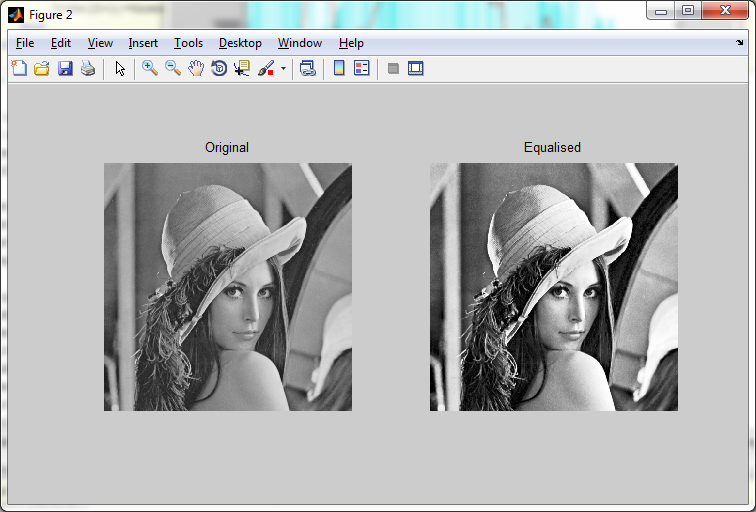
**Task 2: Histogram Equalisation**

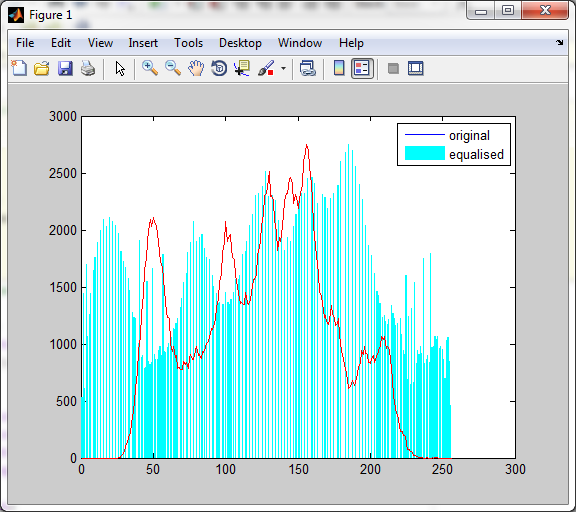
Histogram Equalisation is often used to adjust the global contrast of an image. It is most useful when an image’s background is either too dark or too light and can bring out more detail in under exposed images.

We can calculate this by:

Where *v* is the current value to be adjusted, (M x N) is the size of the image and L is the number of intensity levels i.e. 256. The *cdf* is the cumulative distribution of the histogram.

Your task would be to calculate the new mappings, apply this to the image and plot the new equalised histogram. As shown below:





**Task 3: Histogram Matching**

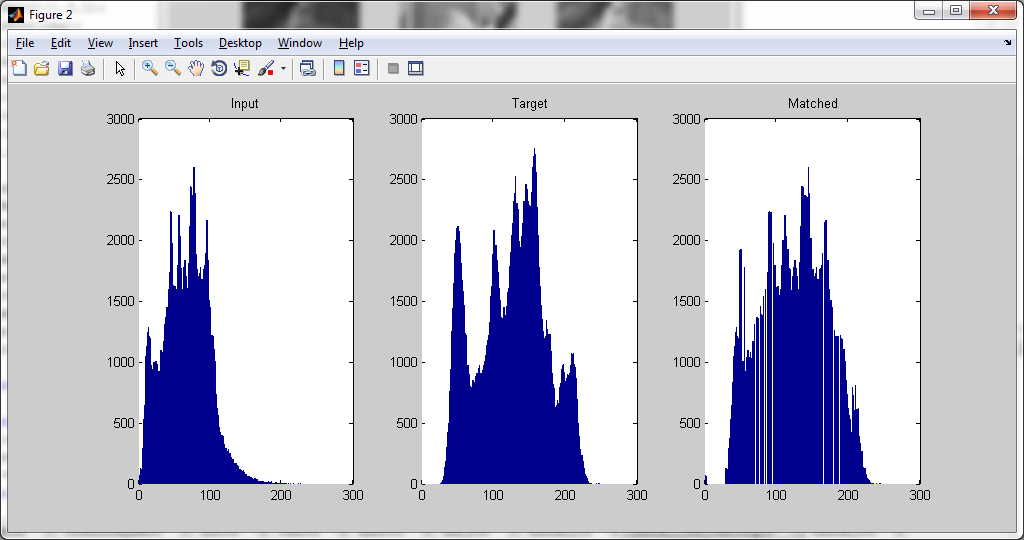
Your next task is to be able to adjust an image based on the histogram of another. This is done using histogram equalisation and can be helpful if you would like to adjust a series of images with different global illuminations.

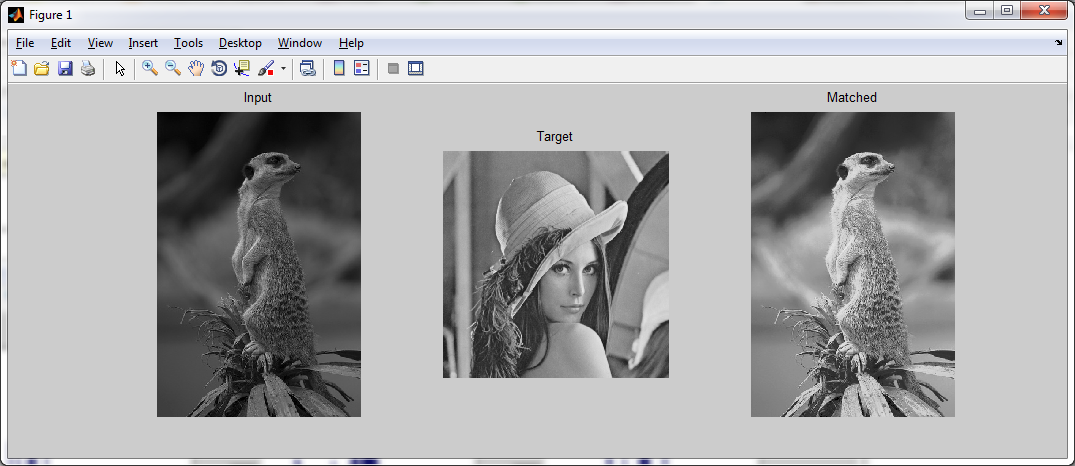
One example of how this could be used is if you captured a series of images of a friend standing on the Tower of London. When looking at the pictures the only one where your friend had their eyes open was when a cloud was covering the sun. Using histogram matching you could adjust this image’s brightness (and contrast) to match the other better illuminated images.

To perform histogram matching we need to first calculate the **normalised** **CDF** of the source image and the target image, *cdfSource* and *cdfTarget****.***

The normalised CDF for each image is obtained by dividing the value in each bin of the original CDF with the maximum value of the original CDF (last bin of the original CDF), so we get a value between 0 and 1 in each bin of the normalised CDF.

We then need to find a mapping function *F* that maps *cdfSource* to *cdfTarget*. This mapping function can then be applied to the input image. The mapping function needed is quite simple and can be obtained by comparing the values between cdfSource and cdfTarget. We would like to map *cdfSource* (*valueS*) to *cdfTarget* (*valueT*) for each *valueS* in [0,255] to obtain *valueOut* in [0,255]. One way to do this, is to find the first *cdfSource* (*valueS*) that is greater than *cdfTarget*(*valueT*), and that *valueT* will be the mapping F(*valueS*) 🡪 *valueOut*.





**Task 4: Image Transformations**

Your last task is rather simple and just requires you to run an example. The idea is just to show you a beneficial use of applying affine or perspective transformations to images and how this could be used in practice.

% Read image to be registered  
A = rgb2gray(imread('plate\_side\_3.jpg'));   
% Display  
figure, imshow(A);  
% Interactively deﬁne coords of input quadrilateral  
[x,y] = ginput(4); input\_points = [x y];

% Read base reference image  
figure, imshow(imread('reference.jpg'));  
% Interactively deﬁne coords of basequadrilateral  
[x,y] = ginput(4); base\_points = [x y];

% Create projective transformation structure  
t = cp2tform(input\_points, base\_points, 'projective');

% Apply projective transform  
registered = imtransform(A,t);

% Interactively crop result  
B = imcrop(registered);

figure, imshow(B)