Assignment 1

COMP 478 Image Processing

Etienne Pham Do

40130483

Etienne Pham Do 40130483

COMP 478 Assignment 1

1.

a) The discrete histogram equalization uses a finite set of numbers, which are integers as well to approximate a PDF through a discrete sum as opposed to its continuous counterpart that uses an integral instead. Though an input intensity of a pixel would be mapped to a new output intensity, the probability of the new intensity would be roughly equal to the probability of the input intensity. In other words, the equalized histogram would not be flat, but would have its pixels spread uniformly across all intensity levels.

b) A histogram-equalized image would have its pixels spread uniformly across all intensity levels. Doing a second pass would only yield a linear transformation as the input image is already histogram-equalized. In other words, there would be no change in intensity.

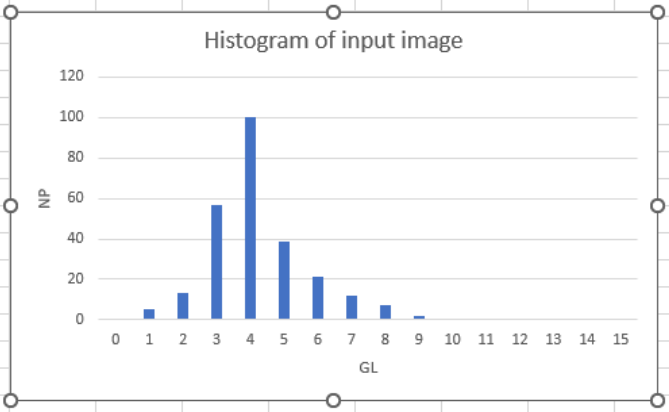
2.

Given 2 sets of numbers, A = {1,-1,9} and B = {4,5,6}, and a and b both equal to 1, we define H to be the median operator. Using H[af(x,y) + bg(x,y)] = H[A + B] = H[{5,4,15}], the operator yields 5. Now, H[f(x,y)] = H[A] = 1 and H[B] = 5, given a sum of 6, which is not equal to H[A + B], violating the property: H[af(x,y) + bg(x,y)] = aH[f(x,y)] + bH[g(x,y)]. Therefore, the median operator is non-linear.

3.

a)

GL	NP
0	0
1	5
2	13
3	57
4	100
5	39
6	21
7	12
8	7
9	2
10	0
11	0
12	0
13	0
14	0
15	0



b)

i)

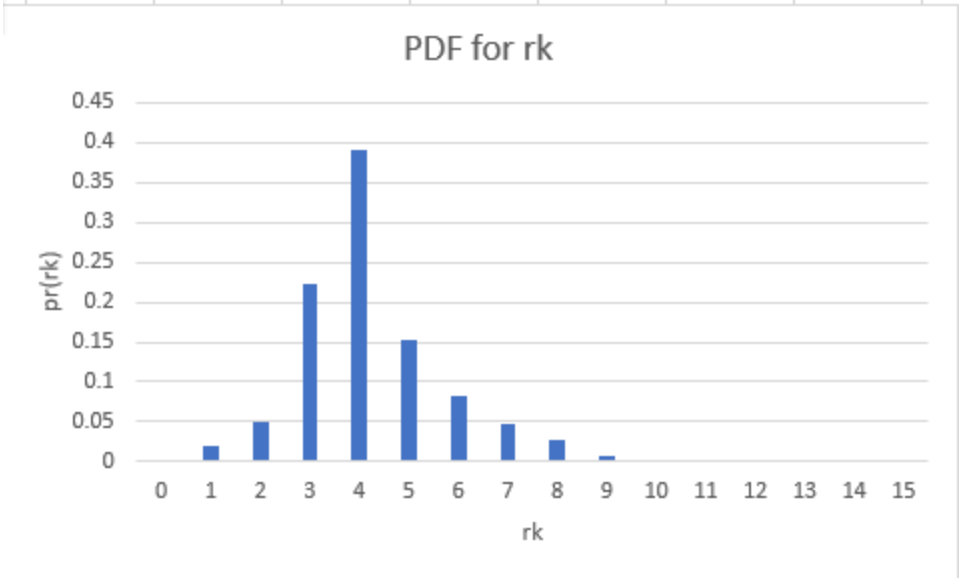
We can find s_k using this formula: $s_k = (L-1)\sum_{j=0}^{k}(\frac{n_j}{n})$ with given values:

rk	nk	pr(rk) = nk/	Column4	MN
0	0	0		256
1	5	0.01953125		
2	13	0.05078125		
3	57	0.22265625		
4	100	0.390625		
5	39	0.15234375		
6	21	0.08203125		
7	12	0.046875		
8	7	0.02734375		
9	2	0.0078125		
10	0	0		
11	0	0		
12	0	0		
13	0	0		
14	0	0		
15	0	0		

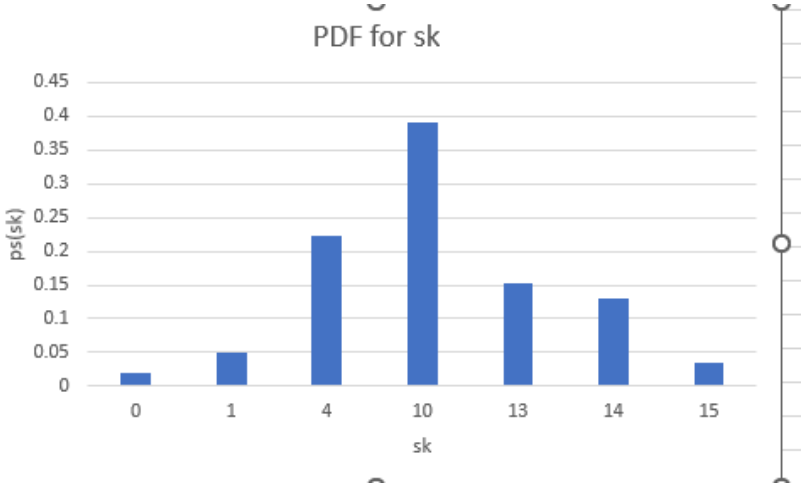
Yields:

sk
0
0
1
4
10
13
14
14
15
15
15
15
15
15
15
15

ii)

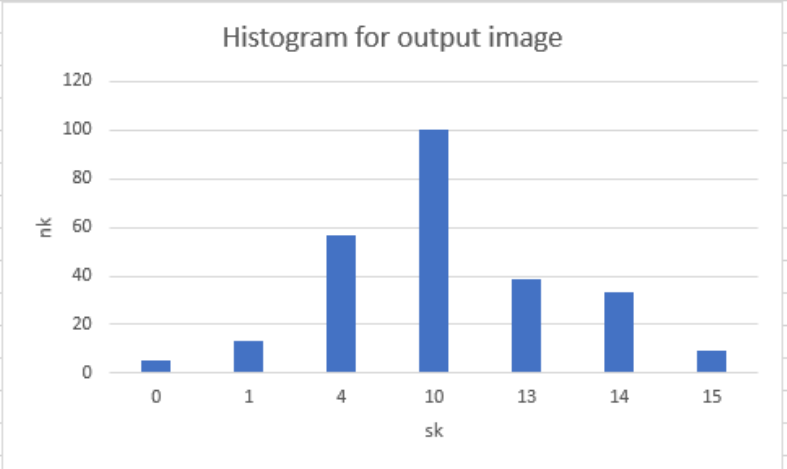


sk	ps(sk)
0	0.01953125
1	0.05078125
4	0.22265625
10	0.390625
13	0.15234375
14	0.12890625
15	0.03515625



c)

sk	ps(sk)	nk
0	0.01953125	5
1	0.05078125	13
4	0.22265625	57
10	0.390625	100
13	0.15234375	39
14	0.12890625	33
15	0.03515625	9



4. For f(x,y) + g(x,y), let h(x,y) be the sum of the 2 images. Given that pixels of f(x,y) have an intensity level of r_k and that the pixels of g(x,y) have an intensity of constant value c, the output image h(x,y) would have the resulting intensities of its pixels shifted by c (as in $r_k + c$), but the spacing in the resulting histogram remains the same. Similarly for the case of f(x,y) * g(x,y), let j(x,y) be the product of the 2 images. This output image would have all the intensities of f(x,y) multiplied by constant c, which doesn't affect the ratio of the number of pixels per intensity, meaning that the spacing in the resulting histogram, also, remains the same.

Programming

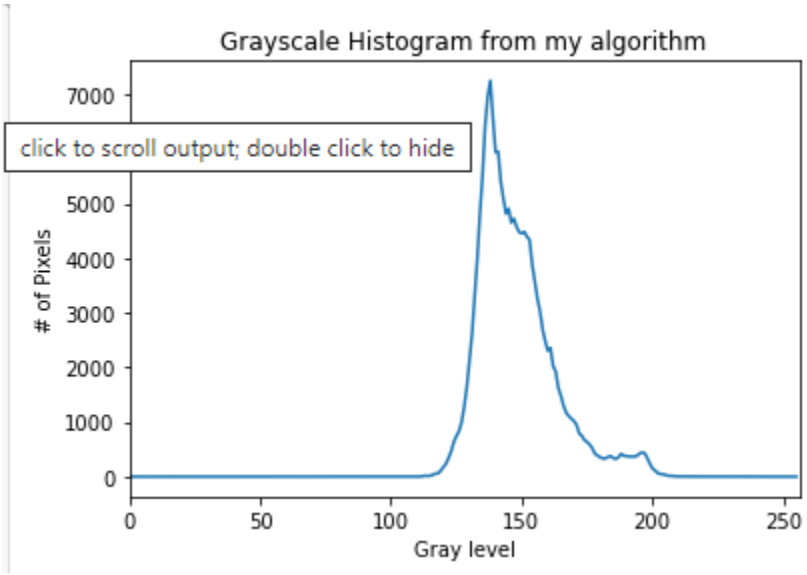
1.

```
import cv2
from matplotlib import pyplot as plt


#Greyscaling image and displaying it
current_image = cv2.imread('HawkesBay.jpeg',0)
cv2.imshow('Greyscale Image', current_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
print('Done showing image')
```

2.

```
# Calculating histogram from image array (not the most optimal algorithm, so takes a while to run)
rk = []
for i in range(256):
   rk.append(i)
pixels_count_per_intensity = []
sum = 0
for x in rk:
   for i in range(len(current_image)):
      for j in range(len(current_image[i])):
         if current_image[i][j] == x:
            sum += 1
   pixels_count_per_intensity.append(sum)
```

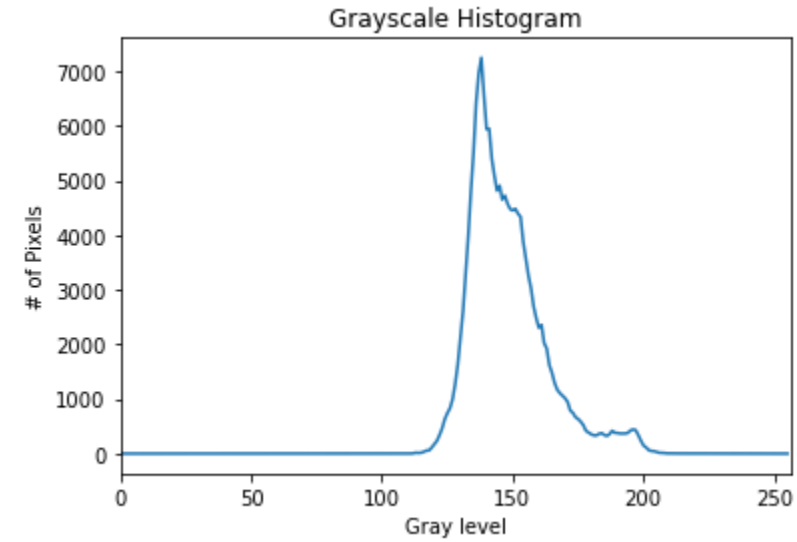
```
    sum = 0


# Displaying histogram from algorithm
plt.figure()
plt.title("Grayscale Histogram from my algorithm")
plt.xlabel("Gray level")
plt.ylabel("# of Pixels")
plt.plot(rk, pixels_count_per_intensity)
plt.xlim([0, 256])
plt.show()
```



3.

```
# Using built-in histogram calculation function and displaying histogram
hist = cv2.calcHist([current_image], [0], None, [256], [0,256])
plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("Gray level")
plt.ylabel("# of Pixels")
plt.plot(hist)
plt.xlim([0, 256])
```

plt.show()

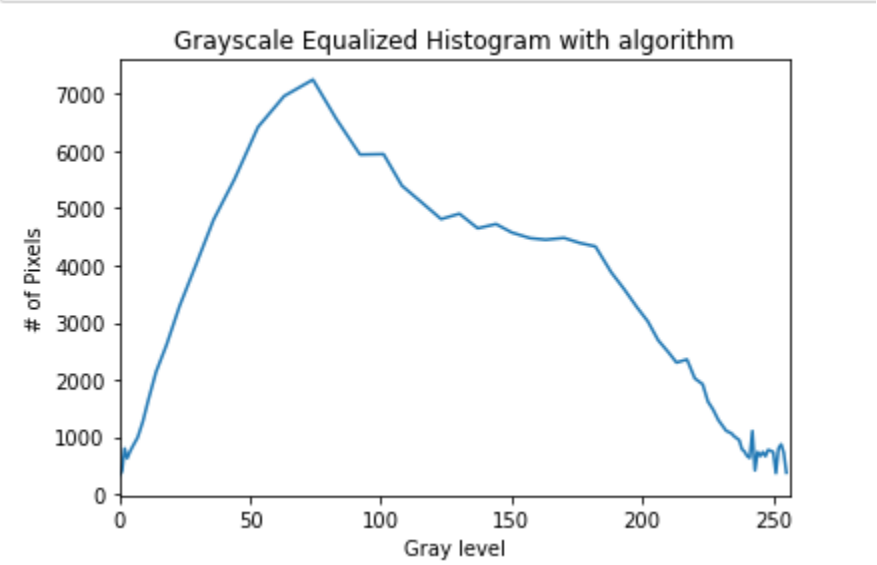


4.

```
# Histogram equalization calculation
def get_length(a):
    return int(a[0])
np = map(get_length,hist)


np = list(np)
s = []
sum_of_prob_r = 0
sum_of_all_pixels = sum(np)


max_intensity = len(np) - 1
for i in range(len(np)):
    prob_of_r = np[i]/sum_of_all_pixels
    sum_of_prob_r += prob_of_r
    s.append(max_intensity * sum_of_prob_r)
```

```
rounded_s = [int(round(num)) for num in s]

# Function to associate the rounded sk values to the number of pixels
def get_number_of_pixels_per_intensity(intensities_ptr, new_np):
    sum = 0
    intensities_ptr.append(rounded_s[0])
    for i in range(len(rounded_s)):
        if i == len(rounded_s) - 1:
            new_np.append(sum)
            break
        if rounded_s[i] in intensities_ptr:
            sum += np[i]
        else:
            intensities_ptr.append(rounded_s[i])
            new_np.append(sum)
            sum = np[i]

# Populating the arrays that will be used to plot histogram
intensities_ptr = []
new_np = []
get_number_of_pixels_per_intensity(intensities_ptr, new_np)

# Displaying equalized histogram from algorithm
plt.figure()
plt.title("Grayscale Equalized Histogram with algorithm")
plt.xlabel("Gray level")
plt.ylabel("# of Pixels")
plt.plot(intensities_ptr, new_np)
plt.xlim([0, 256])
```

plt.show()



5.

Displaying equalized histogram using built-in function

equilized_image = cv2.equalizeHist(current_image)

hist2 = cv2.calcHist([equilized_image], [0], None, [256], [0,256])

plt.figure()

plt.title("Grayscale Equalized Histogram with built-in function")

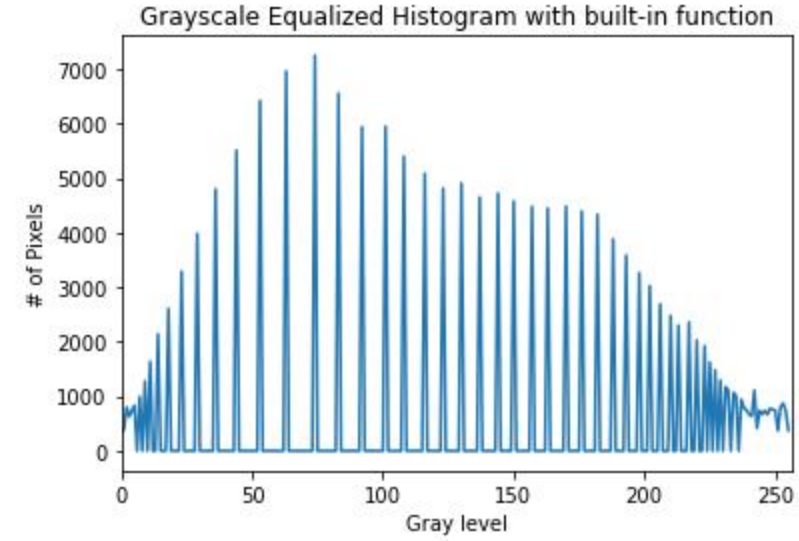
plt.xlabel("Gray level")

plt.ylabel("# of Pixels")

plt.plot(hist2)

plt.xlim([0, 256])

plt.show()


Grayscale Equalized Histogram with built-in function
# of Pixels
Gray level
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250