Task0: MATLAB & Digital Image Processing Tutorial

Name: Kareem Mostafa **Degree:** PhD **ID:** 20767080

Problem 1

general definitions and libraries (needed for all exercises)

```
import numpy as np
```

Exercise 1

```
# creating M3
M3d=[]
for i in range(10):
    x=[]
    for j in range(10):
      x.append(1)
    M3d.append(x)
M3 =np.asmatrix(M3d)
print('M3')
print(M3)

#Creating M5
M5d=[]
for i in range(5):
    x=[]
    for j in range(5):
      if i==j:
        x.append(1)
      else: x.append(0)
    M5d.append(x)
M5 =np.asmatrix(M5d)
print('M5')
print(M5)
```

Exercise 2

```
Md=[]
for i in range(6):
    x=[]
    for j in range(6):
      if i==j-1:
        x.append(1)
      else: x.append(0)
    Md.append(x)
M =np.asmatrix(Md)
print('M')
print(M)
```

Excercise 3

```
*using FOR Loops*
```

```
#creating M2
M2d=[]
c=1
for i in range(8):
    x=[]
    for j in range(8):
      x.append(c)
      c=c+1
    M2d.append(x)
M2 =np.asmatrix(M2d)
print('M2')
print(M2)

#Transposing M2 to create M1
M1=M2.transpose()
print('M1')
print(M1)


#creatng M3
M3d=[]
c=1
for i in range(8):
    x=[]
    for j in range(8):
        if j<4 and i<4:
            x.append(1)
        else: x.append(c)
        c=c+1
    M3d.append(x)
M3 =np.asmatrix(M3d)
M3=M3.transpose()
print('M3')
print(M3)

#creating M4
```

```
M4d=[]
c=1
for i in range(8):
    x=[]
    for j in range(8):
        if 3<=j<5 and 3<=i<5:
            x.append(0)
        else: x.append(c)
        c=c+1
    M4d.append(x)
M4 =np.asmatrix(M4d)
M4=M4.transpose()
print('M4')
print(M4)

#creating M5
M5d=[]
c=1
for i in range(8):
    x=[]
    for j in range(8):
        if i%2 ==0 and i<=j<=i+1:
                x.append(1)
        elif i%2==1 and i-1<=j<=i:
                x.append(1)
        else: x.append(c)
        c=c+1
    M5d.append(x)
M5 =np.asmatrix(M5d)
M5=M5.transpose()
print('M5')
print(M5)

#creating M6
M6d=[]
c=1
for i in range(8):
    x=[]
    for j in range(8):
        if i%2 ==0 and 7-i-1<=j<=7-i:
                x.append(1)
        elif i%2==1 and 7-i<=j<=7-i+1:
                x.append(1)
        else: x.append(c)
        c=c+1
    M6d.append(x)
M6 =np.asmatrix(M6d)
M6=M6.transpose()
print('M6')
print(M6)

#creating M7
M7d=[]
```

```
c=1
for i in range(8):
    x=[]
    for j in range(8):
        if i%2 ==0 and i<=j<=i+1:
                x.append(0)
        elif i%2==1 and i-1<=j<=i:
                x.append(0)
        else: x.append(c)
        c=c+1
    M7d.append(x)
M7 =np.asmatrix(M7d)
M7=M7.transpose()
print('M7')
print(M7)

#creating M8
M8d=[]
c=1
for i in range(8):
    x=[]
    for j in range(8):
        if i%2 ==0 and i<=j<=i+1:
                x.append(c)
        elif i%2==1 and i-1<=j<=i:
                x.append(c)
        else: x.append(100)
        c=c+1
    M8d.append(x)
M8 =np.asmatrix(M8d)
M8=M8.transpose()
print('M8')
print(M8)

#creating M9
M9d=[]
c=1
for i in range(8):
    x=[]
    for j in range(8):
        if i%2 ==0 and i<=j<=i+1:
                x.append(c)
        elif i%2==1 and i-1<=j<=i:
                x.append(c)
        else: x.append(0)
        c=c+1
    M9d.append(x)
M9 =np.asmatrix(M9d)
M9=M9.transpose()
print('M9')
print(M9)
```

```
*No FOR loops (using numpy functions)*
```

```
#creating M2
M2=np.reshape(list(range(1,65)),(8,8))
M2=np.asmatrix(M2)
print('M2')
print(M2)

#creating M1 by transposing M2
M1=np.transpose(M2)
print('M1')
print(M1)


#Creating M3 by replacing part of M1 with a matrix of ones
ones=np.ones((4,4))
M3=M1.copy()
M3[0:4,0:4]=ones
print('M3')
print(M3)

#Creating M4 by replacing the central part of M1 with zeroes
zeroes=np.zeros((2,2))
M4=M1.copy()
M4[3:5,3:5]=zeroes
print('M4')
print(M4)

#Creating M5 by first creating a kroneker matrix of a 4x4 identity matrix and a 2x2
matrix of ones, then using logical indexing to index (and consequently replace) the
values of M1 whose locations correspond to the ones in the kronecker matrix just produced

M5=M1.copy()
oneskron=np.kron(np.eye(4),np.ones((2,2)))
d=(oneskron==1)
M5[d]=1
print('M5')
print(M5)

#Creating M6 the same way M5 was created, but the kronecker matrix needs to be flipped
first
M6=M1.copy()
oneskronflip=np.fliplr(oneskron.copy())
d=(oneskronflip==1)
M6[d]=oneskronflip[d]
print('M6')
print(M6)

#Creating M7 the same way M5 was created, but the logically indexed values will be
changed to zeroes
M7=M1.copy()
d=(oneskron==1)
```

```
M7[d]=0
print('M7')
print(M7)

#Creating M8 in a similar fashion, but indexing the zeroes in the kronecker matrix
M8=M1.copy()
d=(oneskron==0)
M8[d]=100
print('M8')
print(M8)

#creating M9 similar to M8 but replacing the indexed values with zeroes
M9=M1.copy()
d=(oneskron==0)
M9[d]=0
print('M9')
print(M9)
```

Problem 2

```
import cv2
import numpy as np

img=cv2.imread('coloredChips.png',1)
#opencv reads images in BGR by default, this has to be converted using the command below
img2=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

UL=np.array([255,100,100]) #upper limit for pixel values
LL=np.array([150,0,0]) #lower limit for pixel values

#create a mask of the image showing only the values within the two limits above and
create its invers
mask=cv2.inRange(img2,LL,UL)
mask_inv=cv2.bitwise_not(mask)

#blackout the red chips
res = cv2.bitwise_and(img,img, mask= mask_inv)


cv2.imshow('chips',res)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Problem 3

```
import numpy as np
from matplotlib import pyplot as plt

#create a kronecker roduct matrix to represent the first 4 cells in the top-left corner
block=np.kron(np.eye(2),np.ones((100,100)))

#replicating the created matrix to vreate a fullsized board
fullsize=np.tile(block,(4,4))

#plotting
plt.imshow(fullsize,cmap=plt.cm.gray)
plt.show()
```

Problem 4

```
import cv2

img=cv2.imread('color_patch.png',1)
imgp=img.copy() #image to be procesed
h,w=img.shape[:-1]

for i in range(4):
    patch=img[int(h/4*(i)):int(h/4*(i+1)),0:int(w/4)]
    cv2.imshow('patch'+str(i+1),patch)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

imgp[101:151,101:151]=[255,0,0]  #cv2 reads images in BGR format

cv2.imshow('before',img)
cv2.imshow('after',imgp)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Problem 5

```
import cv2

img1=cv2.imread('cameraman.tif',1)
img2=cv2.imread('llama.jpg',1)

for i in range(2): #perform all required operations on both images in a loop
    temp=eval('img'+str(i+1))
    h,w = temp.shape[:-1]
    cv2.imshow('img'+str(i+1)+'original',temp)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    for j in range (3):  #rotations loop
        r=90*(j+1)    #degree of rotation
        M = cv2.getRotationMatrix2D((w/2,h/2),r,1)
        dst = cv2.warpAffine(temp,M,(w,h))
        cv2.imshow('img'+str(i+1)+'rot'+str(r),dst)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    for k in range (2): #mirroring loop
        dst=cv2.flip(temp,k)
        cv2.imshow('img'+str(i+1)+'flip'+str(k+1),dst)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Problem 6

Q003: The number of channels has changed because the original image was in grayscale (1 channel) but adding the colored circles to it has changed it to a colored image (3 channels).

Q004: the first method (plot) adds the circles as an overlay on top the image, keeping its contents and properties unharmed. any further processing for the image will be on its original form as if the circles never existed.
The second method edits the image and makes the circles a "permanent" part of it. This changes the image contents and properties (e.g. no. of channels as addressed in Q003) and any further processing of the image will include the circles just drawn

Q006: As seen from the visual representation and the 'uint8' statement, the data are encoded as arrays of 8-bit integers. Thus the maximum value an element can take is 255 (2^8 -1) below is a representation to the creation of each element in each array

Object	Values	<255	Action/Decision
Grey0	1*0=0	Yes	OK
Grey100	1*100=100	Yes	OK
Grey200	1*200=200	Yes	OK
Grey300	1*300=300	NO	store as 255

Q007: A bit is a binary value (0 or 1), the smallest unit of data in a computer and also the most fundamental as it is simply stored as electricity (either low or high) in a single capacitor in a memory device.a byte is an array of bits (typically 8). and they are the ones refered to for storage and operations since every byte represents 1 letter,number,etc.

Q010: Resolution is more of an abstract value (no. of pixels in an image) while quality is subjective (how much information does the image provide). While typically high quality images have high resolution, the same can't be said the owther way around. An example would be shaking your Iphone too much while taking a photo. The photo taken will have a high resolution by default but the it will probably be too blurry to be meaningful (low quality)

Q011: The example shows how computers store colored images as shades of three different colors (red,green,blue) under the RGB format and the final image is simply an overlay of all three images/matrices. Each one of the three images are stroed as a matrix with values from 0 to 255 with higher values indicating a brighter color

Q012: Losless compression is a way to store data that allows it to be perfectly reconstructed (e.g. zip files)

Q013: Assuming it is a colored image, the size would be the product of its dimensions and no. of channels (3)
size = 8688 x 5792x3=150,962,688 bytes (about 150MB)
for a grayscale image the size will be one third of that value since it only has one channel (about 50MB)

Q016: double simply changes the values type into double, while imdouble() also normalises them into an interval of [0,1]
Example:
double([255,51]) >> [255,51] as double
imsdouble([255,51]) >> [1,0.2] as double