Lab Instructions - session 4

1- Measuring the execution time

To measure the execution time of an operation or a piece of code put it inside a function and pass it to **`timeit.timeit`**:

`measure_time.py`

```
import numpy as np
import timeit

n = 1000

A = np.random.rand(n,n)

def f():
    np.linalg.inv(A)

t1 = timeit.timeit(f, number=1)
t2 = timeit.timeit(f, number=100)/100

print(t1)
print(t2)
```

- The execution time of what operation is measured?
- Which measurement is more reliable? **`t1`** or **`t2`**?

This can be done in a more compact way using the **`lambda`** functions:

```
timeit.timeit(lambda : np.linalg.inv(A), number=100)/100
```

2- Diagonal matrices

Execute the following and see the result.

`diagonal.py`

```
import numpy as np

D1 = np.diag([2,3,4])
D2 = np.diag([10,20,30,40])

print('D1=\n', D1)
```

```
print('D2=\n', D2)

A = np.array([[1,1,1,1],
              [1,2,2,2],
              [1,2,3,4]])

print('A=\n', A)
print('D1@A=\n', D1 @ A)

print('A@D2=\n', A @ D2)
```

- What is the effect of multiplying a diagonal matrix to the left and right?

3- Scaling rows and columns using broadcasting

This is an alternative to scaling the rows of a matrix using the concept of Broadcasting you learned in Lab 1.

scale_rows.py

```
import numpy as np

d1 = np.array([2,3,4]).reshape((3,1))

A = np.array([[1,1,1,1],
              [1,2,2,2],
              [1,2,3,4]])

print('d1=\n', d1)
print('A=\n', A)

print('d1.shape=\n', d1.shape)
print('A.shape=\n', A.shape)

print('d1 * A=\n', d1 * A)
```

- Write an equivalent code to scale columns of a matrix with numbers [10,20,30,40]. Is reshaping `np.array([10,20,30,40])` to shape **(1,4)** necessary for scaling columns? Why? (refer to the Broadcasting rules)
- Measure the execution time of `d1*A` and `D1@A` using `timeit` Which one is faster? Why?

Task 1

Compare the execution time of **`d1*A`** with **`D1@A`** for random matrices **`d1`** and **`A`** and **`D1=diag(d1.ravel())`**, where **`A`** is 100 by 200. Which one is faster? (Do not count the time of creating **`D1`** when computing **`D1@A`**.)

Task 2- Practice vectorized code

Consider the following:

`task2.py`

```
import numpy as np

m,n,p = 100,50, 2000

A = np.random.rand(m,n,p)
s = np.random.rand(p)

for i in range(p):
    A[:,:,i] *= s[i]
```

- In the above, replace the **`for`** loop with a single command.
- Compare the execution time of your code with the for loop using **`timeit`**.

Working with images

A grayscale (black-and-white) image can be represented as a 2D array of pixels. The value of each pixel represents the intensity of light in that pixel. Usually, 0 indicates *black*, 1.0 (float) or 255 (unsigned int) shows *white*, and other values represent the shades of gray in between. The next code reads and displays an image

`image.py`

```
import imageio
import matplotlib.pyplot as plt

I = imageio.imread('nasir-al-mulk-gray.jpg')

print('I=\n', I)
print('I.dtype=\n', I.dtype)
print('I.shape=\n', I.shape)

plt.imshow(I, cmap='gray')
plt.show()
```

```
plt.imshow(I.T, cmap='gray')
plt.show()

plt.imshow(I[100:400, 300:600], cmap='gray')
plt.show()
```

- Zoom in to see each individual pixel.
- What is the shape of the image? How many pixels does it have?
- Why is each pixel represented as an unsigned integer? What values represent the black color and the white color?

Task 3

Use what you learned in Lab 1 to create the image below **in a single line of code**.

Color Images

Each pixel can be represented in color images by the intensity value of the red, green, and blue colors (RGB). Thus, an RGB color image may be stored as a 3D array (m by n by 3).

color.py

```
import imageio
import matplotlib.pyplot as plt

I = imageio.imread('nasir-al-mulk.jpg')

print('I.shape=', I.shape)

plt.imshow(I)
plt.show()
```

- What is the shape of `I`? why?

Task 4

Change the code below to display an image changing gradually from grayscale to color. Notice that `G` is of shape (m,n) while `I` has shape (m,n,3). Hint: A color image with the same R,G and B channels looks like a grayscale image.

task4.py

```
import imageio
import matplotlib.pyplot as plt
import numpy as np

G = imageio.imread('nasir-al-mulk-gray.jpg')
I = imageio.imread('nasir-al-mulk.jpg')

for alpha in np.linspace(0,1,20):
    J = I # combine G and I

    plt.imshow(J)
    plt.draw()
    plt.pause(.1)

plt.show()
```

Task 5

Use what you learned in **Task 2** to multiply the R,G, and B channels of the above image by $|sin(alpha)|$, $|sin(alpha + \pi/4)|$and $|sin(alpha + \pi/2)|$ respectively using a *single* command. Animate by varying alpha from 0 to π. Notice that after multiplication you need to convert the image back to **`uint8`**. (**`np.uint8(I)`** or **`I.astype(np.uint8)`**).