CSC589: Introduction to Computer Vision
Yan Shi
Due: 9/18

Problem Set 1

1.

Results from Terminal:

```
~ — Python
Edgeworth:~ yanshi$ python2
Python 2.7.13 (default, Jul 18 2017, 09:16:53)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> import cv2
>>> import matplotlib.pyplot as plt
>>> import scipy
>>> from scipy import misc
>>>
```

2. Basic Matrix/Vector Manipulation (20 points)

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 0 & 2 & 2 \end{bmatrix}, a = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b = \begin{bmatrix} -1 \\ 2 \\ 5 \end{bmatrix}, c = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 2 \end{bmatrix}$$

a. Define Matrix M and Vectors a, b, c in Python. Use Numpy.

```
Matrix M:
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [0 2 2]]
Vector a:
[[1]
 [1]
 [0]]
Vector b:
[[-1]
 [ 2]
 [ 5]]
Vector c:
[[0]
 [2]
 [3]
 [2]]
```

CSC589: Introduction to Computer Vision
Yan Shi
Due: 9/18

b. **Find the dot product of vectors a and b. Save this value to the variable aDotb and write its value in your report.**
aDotb = np.dot(a.T, b)

```
The dot product is:
[[1]]
```

c. **Find the element-wise product of a and b $[a_1, b_1, a_2, b_2, a_3, b_3]^T$.**
$[a_1, b_1, a_2, b_2, a_3, b_3]^T$=[-1 2 0]
print("%s") % "Element-wise product:"
print((np.multiply(a, b)).T) newA = np.matlib.repmat(a.T, 4, 1)
answer2e = np.multiply(newA, M)

```
Element-wise product:
[[-1  2  0]]
```

d. Find $\left(a^T b\right)Ma$.

In order to multiply a vector with another vector, we must transpose the first vector before proceeding, otherwise an error will be thrown due to vector/matrix multiplication rules regarding columns and rows. What we obtain with a vector multiplied by another vector is a scalar. This one value is then multiplied to every element in the dot product of matrix M and vector a.

```
(a^Tb)Ma =
[[ 3]
 [ 9]
 [15]
 [ 2]]
```

e. **Without using a loop, multiply each row of M element-wise by a. (Hint: the function repmat() may come in handy).**
newA = np.matlib.repmat(a.T, 4, 1)
answer2e = np.multiply(newA, M)

I first needed to transpose vector *a* before I could properly use repmat to create a new array that I could then properly multiply the 4 x 3 and to adhere to matrix multiplication rules in regards to columns and rows.

```
[[1 2 0]
 [4 5 0]
 [7 8 0]
 [0 2 0]]
```

f. Without using a loop, sort all of the values of the new M from (e) in increasing order.
In order to do this, I used numpy's sort method.

```
[[0 1 2]
 [0 4 5]
 [0 7 8]
 [0 0 2]]
```

3. Basic Image Manipulation:

a. Read in the images, image1.jpg and image2.jpg. there are many different ways to read in images. Matplotlib.image is a good one. Cv2.imread() is another good one. You can also use scipy.misc.imread().

image1 = misc.imread('image1.jpg', flatten = 1)

image2 = misc.imread('image2.jpg', flatten = 1)

***Images are not according to their original size, they are reduced to show which image is which for grading purposes.**

b. **Convert the images to double precision and rescale them to stretch from minimum value 0 to maximum value 1.**
image1 = np.float64(misc.imread('image1.jpg', flatten = 1, mode='F'))
image2 = np.float64(misc.imread('image2.jpg', flatten = 1, mode='F'))

This is self-explanatory, as I casted the images as I read them in in float64.

c. **Add the images together and renormalize them to have minimum, value 0 and maximum value 1.**

```
#normalization
normalizedImage1 = np.zeros((720, 652))
normalizedImage1 = cv2.normalize(image1, normalizedImage1, 0, 1,
cv2.NORM_MINMAX)
normalizedImage2 = np.zeros((720, 652))
normalizedImage2 = cv2.normalize(image2, normalizedImage2, 0, 1,
cv2.NORM_MINMAX)
```

In order to normalize, I used cv2's normalization method for this purpose. I needed a place to store the normalized results, thus the variables.

Results:

d. **Create a new image such that the left half of the image is the left half of image1 and the right half of the image is the right half of image 2.**
I spliced the images in half using my knowledge of matrices' columns and rows. I knew that since I needed half of each image, that I would be dividing my columns in half but keeping my rows the same.

croppedImage1 = normalizedImage1[0:720, 0:652/2]
croppedImage2 = normalizedImage2[:, 652/2:]

Results:

e. **Using a for loop, create a new image such that every odd numbered row is the corresponding row from image1 and that every even row is the corresponding row from image2. Remember that indices start at 0 and not 1 in Python.**

```
frankensteinImage = np.empty((720, 652))
for row in range(len(normalizedImage1)):
    if row % 2 == 1: #odd
        frankensteinImage[row] = normalizedImage1[row]

    else: #even
        frankensteinImage[row] = normalizedImage2[row]
```

I checked for the indices of the rows and then determined whether or not it was even or odd by finding its remainder by modulus 2. I would then fill in the rows in frankensteinImage based on the calculations done within the for loop to obtain my results.

Results:

f. **Accomplish the same task as part e without using a for-loop (functions reshape and repmat may be helpful here).**

```
oddMatrix = normalizedImage1[1::2]
evenMatrix = normalizedImage2[::2]
frankensteinImage2 = np.empty((720, 652))
frankensteinImage2[1::2] = oddMatrix
frankensteinImage2[::2] = evenMatrix
```

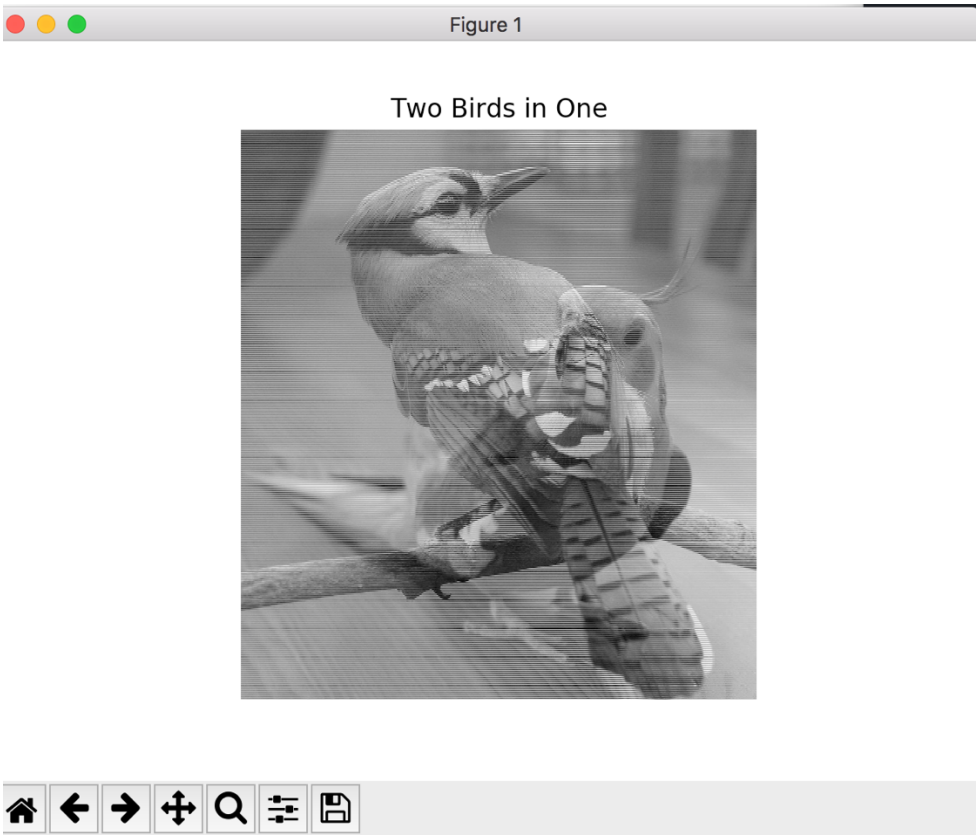
Here, I had spliced the images based on the indices given in steps. The oddMatrix contained the image's first row and every other two rows. The even row was the second row, and then every two rows. frankensteinImage2's rows were then filled in based on its own start point and step points and swapping with oddMatrix.

Results:

g. Convert the result from part f to a grayscale image. Display the grayscale image with a title in your report.

I used a color map provided by the matplotlib that turned my image into a grayscale image.

The result:



4. Compute the average face: (20 pts)

a. **Download labeled dataset. (google: LFW face dataset or click the link). Pick a face with at least 100 images.**
Dataset can be found in George_W_Bush folder.

b. **Call numpy.zeroes to create a 250 x 250 x 3 float64 tensor to hold the results.**
average_array = np.zeros((250, 250, 3), dtype=np.float64)

This is self-explanatory, I called numpy.zeroes to create an empty array that will hold my average face calculations.

c. **Read each image with skimage.io.imread, convert to float and accumulate**.
images_array requires an array that holds the multiple arrays of the images

images_array = np.float64(np.array([np.array(io.imread('George_W_Bush/' + fname)) for fname in folder]))

I require the first matrix to hold all the other matrices that will be generated by my for loop as I loop through the files in the George_W_Bush folder and save the image's matrices information.

d. Write the averaged result with skimage.io.imsave. Post your resulted image in the report.

In the code, you will find that I used numpy's mean() function in order to calculate the average result. I could have also taken the sum of all the arrays in the image arrays and divided it by the number of images (100) as well. That method would have required a for loop.

Result: