CSE 252A (Computer Vision I) · Fall 2020 · Assignment 0

Instructor: David Kriegman

Assignment published on Monday, October 5, 2020

Due on Wednesday, October 14, 2020 at 11:59 pm Pacific Time

Instructions

- Review the academic integrity and collaboration policies on Canvas. This assignment must be completed individually.
- All solutions must be written in this notebook. Programming aspects of the assignment must be completed using Python (preferably 3.6+).
- If you want to modify the skeleton code, you may do so. The existing code is merely intended to provide you with a framework for your solution.
- You may use Python packages for basic linear algebra (e.g. simple operations from NumPy or SciPy), but you may **not** use packages that directly solve the problem. If you are unsure about using a specific package or function, please ask the instructor and teaching assistants for clarification.
- You must submit, through Gradescope, both (1) this notebook exported as a PDF **and** (2) this notebook as an `.ipynb` file. You must mark every problem in the PDF on Gradescope. If you do not submit both the `.pdf` and `.ipynb`, and/or if you do not mark every problem in the PDF on Gradescope, you may receive a penalty.
- It is highly recommended that you begin working on this assignment early.
- **Late policy:** Assignments submitted late will receive a 10% grade reduction for each day late (e.g. an assignment submitted an hour after the due date will receive a 10% penalty, an assignment submitted 10 hours after the due date will receive a 10% penalty, and an assignment submitted 28 hours after the due date will

receive a 20% penalty). Assignments will not be accepted 72 hours after the due date. If you require an extension (for personal reasons only), you must request one as far in advance as possible. Extensions requested close to or after the due date will only be granted for clear emergencies or clearly unforeseeable circumstances.

Welcome to CSE 252A: Computer Vision I! This course will give you a comprehensive introduction to computer vision, covering topics such as low-level vision, inference of 3D properties from images, and object recognition. In this class, we will utilize a variety of tools which may require some initial configuration. To ensure a smooth transition into later assignments, we'll get you set up with most of those course tools in this assignment. You will also practice some basic image manipulation techniques.

When you are finished, you will need to export this Jupyter notebook as a PDF, and submit both that PDF and this `.ipynb` file to Gradescope.

Piazza, Gradescope and Python

Piazza

Go to [Piazza (https://piazza.com/ucsd/fall2020/cse252a)](https://piazza.com/ucsd/fall2020/cse252a) and sign up for the class using your `ucsd.edu` email account. You'll be able to ask the professor, the TAs, and your classmates questions on Piazza. Class announcements will also be made using Piazza, so make sure you check your email or Piazza frequently.

Gradescope

See the Piazza post on how to add CSE 252A in Gradescope. You will be required to submit each assignment to Gradescope for grading. Make sure you mark the pages in the PDF that are associated with each question.

Python

We will use the Python programming language for all assignments in this course, in tandem with a few popular libraries (e.g. NumPy and matplotlib). Assignments will be given in the form of browser-based Jupyter notebooks, such as the one you are currently viewing. We expect that many of you will already have some experience with Python and NumPy. If so, great! If not, don't worry (as long as you have *some* programming experience). You can pick it up quickly. If you lack Python knowledge but have previous experience with Matlab, you might want to check out the [NumPy for Matlab users (https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html)](https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html) page.

The section below will serve as a quick introduction to NumPy and some of the other libraries that we'll use. (Just run through the cells; you don't need to do anything for credit until you get to Problem 1.)

Getting started with NumPy

NumPy is a fundamental package for scientific computing with Python. It provides a powerful N-dimensional array object as well as functions for working with such arrays.

Arrays

In [4]:

```python
import numpy as np

array1d = np.array([1,0,0])           # a 1D array

print("1D array :")
print(array1d)
print("Shape :", array1d.shape)       # print the shape of the array

array2d = np.array([[1], [2], [3]])   # a 2D array
print("\n2D array :")
print(array2d)
print("Shape :", array2d.shape)       # print the size of the array; notice the difference

print("\nTranspose 2D :", array2d.T)  # transpose of a 2D array
print("Shape :", array2d.T.shape)

print("\nTranspose 1D :", array1d.T)  # notice how the 1D array did not change after the transpose
print("Shape :", array1d.T.shape)

allzeros = np.zeros([2, 3])           # 2x3 array of zeros
allones =  np.ones([1, 3])            # 1x3 array of ones
identity = np.eye(3)                  # identity matrix
rand3_1 = np.random.rand(3, 1)        # random matrix with values in [0, 1]
arr = np.ones(allones.shape) * 3      # create a matrix from shape

print("\n arr printed")
print(arr)
```

```
1D array :
[1 0 0]
Shape : (3,)

2D array :
[[1]
 [2]
 [3]]
Shape : (3, 1)

Transpose 2D : [[1 2 3]]
Shape : (1, 3)

Transpose 1D : [1 0 0]
Shape : (3,)

 arr printed
[[3. 3. 3.]]
```

Array Indexing

In [5]:

```
import numpy as np
array2d = np.array([[1, 2, 3], [4, 5, 6]])  # create a 2d array
with shape (2, 3)
print("Access a single element")
print(array2d[0, 2])                         # access an element
array2d[0, 2] = 252                          # a slice of an arra
y is a view into the same data;
print("\nModified a single element")
print(array2d)                               # this will modify t
he original array

print("\nAccess a subarray")
print(array2d[1, :])                         # access a row (to 1
d array)
print(array2d[1:, :])                        # access a row (to 2
d array)
print("\nTranspose a subarray")
print(array2d[1, :].T)                       # notice the differe
nce of the dimension of resulting array
print(array2d[1:, :].T)                      # this will be helpf
ul if you want to transpose it later

# Boolean array indexing
# Given a array m, create a new array with values equal to m
# if they are greater than 0, and equal to 0 if they less than o
r equal 0

array2d = np.array([[3, 5, -2], [50, -1, 0]])
arr = np.zeros(array2d.shape)
arr[array2d > 0] = array2d[array2d > 0]
print("\nBoolean array indexing")
print(arr)
```

```
Access a single element
3

Modified a single element
[[  1   2 252]
 [  4   5   6]]

Access a subarray
[4 5 6]
[[4 5 6]]

Transpose a subarray
[4 5 6]
[[4]
 [5]
 [6]]

Boolean array indexing
[[ 3.  5.  0.]
 [50.  0.  0.]]
```

Operations on Arrays

Elementwise Operations

In [7]:

```
import numpy as np

a = np.array([[1, 2, 3], [2, 3, 4]], dtype=np.float64)
print(a * 2)                                          # scalar
multiplication
print(a / 4)                                          # scalar
division
print(np.round(a / 4))
print(np.power(a, 2))
print(np.log(a))

b = np.array([[5, 6, 7], [5, 7, 8]], dtype=np.float64)
print(a + b)                                          # elemen
twise sum
print(a - b)                                          # elemen
twise difference
print(a * b)                                          # elemen
twise product
print(a / b)                                          # elemen
twise division
```

```
[[2. 4. 6.]
 [4. 6. 8.]]
[[0.25 0.5  0.75]
 [0.5  0.75 1.  ]]
[[0. 0. 1.]
 [0. 1. 1.]]
[[ 1.  4.  9.]
 [ 4.  9. 16.]]
[[0.         0.69314718 1.09861229]
 [0.69314718 1.09861229 1.38629436]]
[[ 6.  8. 10.]
 [ 7. 10. 12.]]
[[-4. -4. -4.]
 [-3. -4. -4.]]
[[ 5. 12. 21.]
 [10. 21. 32.]]
[[0.2        0.33333333 0.42857143]
 [0.4        0.42857143 0.5       ]]
```

Vector Operations

In [8]:

```
import numpy as np

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

print("sum of array")
print(np.sum(a))                # sum of all array elements
print(np.sum(a, axis=0))        # sum of each column
print(np.sum(a, axis=1))        # sum of each row

print("\nmean of array")
print(np.mean(a))               # mean of all array elements
print(np.mean(a, axis=0))       # mean of each column
print(np.mean(a, axis=1))       # mean of each row
```

```
sum of array
10
[4 6]
[3 7]

mean of array
2.5
[2. 3.]
[1.5 3.5]
```

Matrix Operations

In [9]:

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print("matrix-matrix product")
print(a.dot(b))                    # matrix product
print(a.T.dot(b.T))

x = np.array([1, 2])
print("\nmatrix-vector product")
print(a.dot(x))                    # matrix / vector product
print(a @ x)                       # Can also make use of the @ i
nstad of .dot(); requires Python 3.5+
```

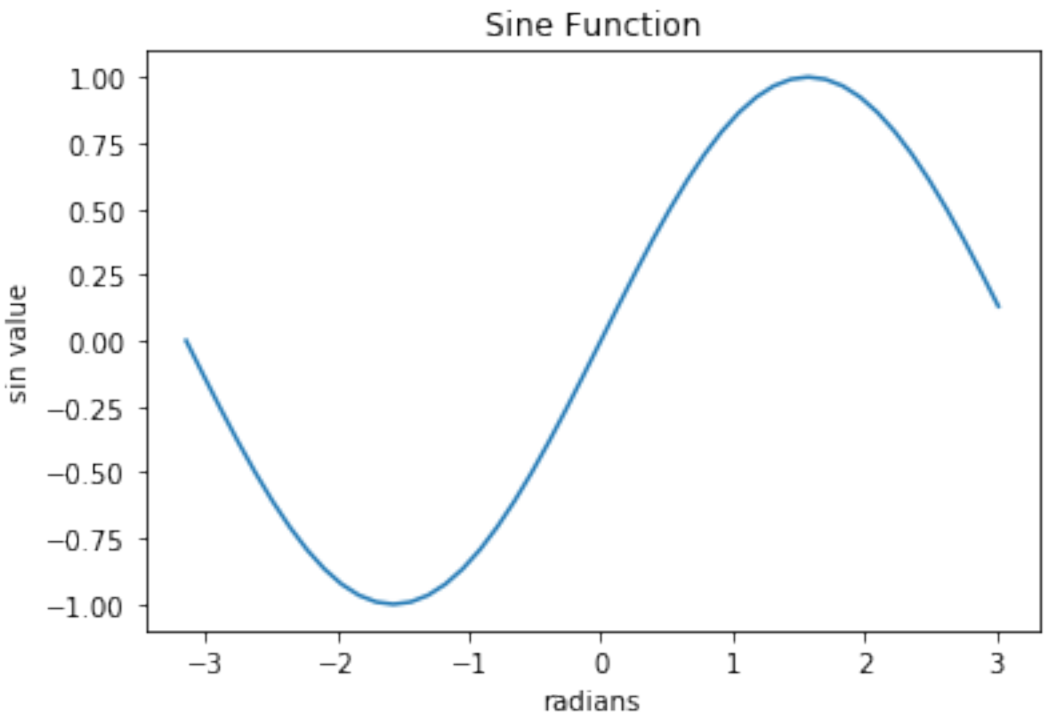
```
matrix-matrix product
[[19 22]
 [43 50]]
[[23 31]
 [34 46]]

matrix-vector product
[ 5 11]
[ 5 11]
```

Matplotlib

Matplotlib is a plotting library. We will use it to show the results in this assignment.

In [10]:

```
# this line prepares IPython for working with matplotlib
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import math

x = np.arange(-24, 24) / 24. * math.pi
plt.plot(x, np.sin(x))
plt.xlabel('radians')
plt.ylabel('sin value')
plt.title('Sine Function')

plt.show()
```



This brief overview introduces many basic functions from a few popular libraries, but it is far from complete. Check out the documentation for NumPy (https://docs.scipy.org/doc/numpy/reference/) and matplotlib (https://matplotlib.org/) to find out more.

Problem 1: Image Operations and Vectorization (1 point)

Vector operations using NumPy can offer a significant speedup over performing operations iteratively on an image. The problem below will demonstrate the time it takes for both approaches to change the color of quadrants of an image.

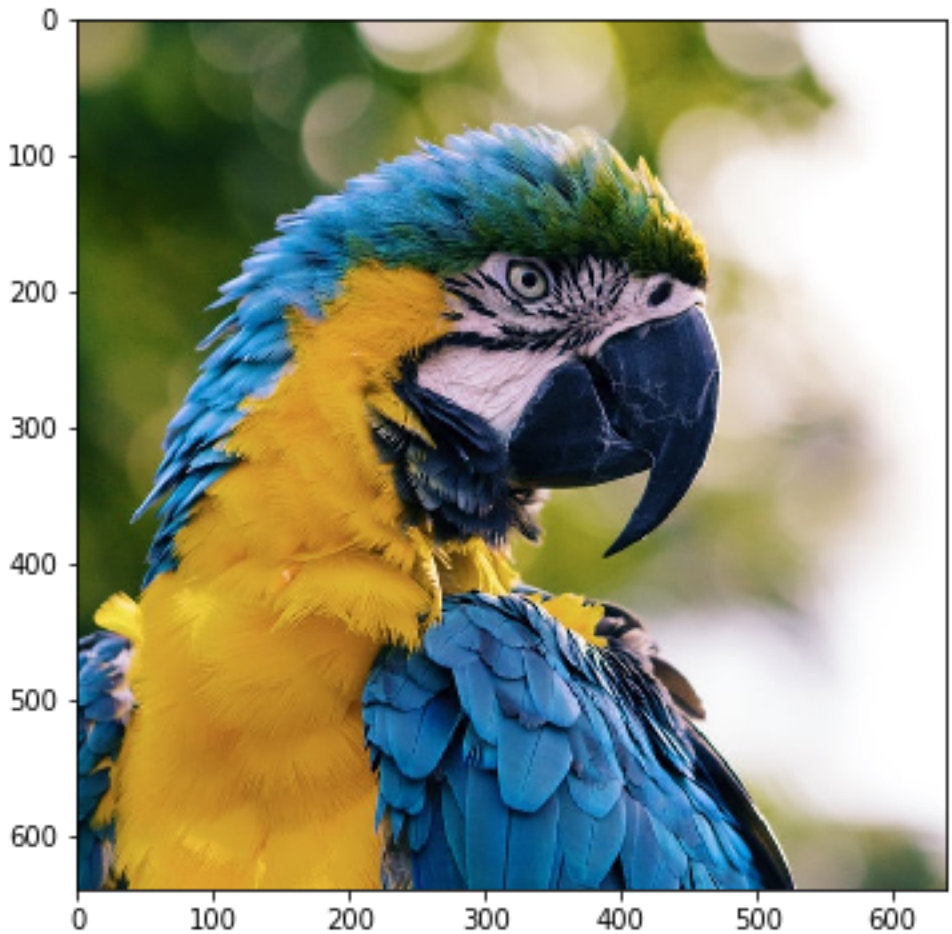
The problem reads the image `macaw.jpg` that you will find in the assignment folder. Two functions are then provided as alternatives for performing an operation on the image.

Your job is to follow the code and then fill in the `piazza` function according to instructions on Piazza.

In [15]:

```
import numpy as np
import matplotlib.pyplot as plt
import copy
import time

%matplotlib inline

img = plt.imread('macaw.jpg')                    # read a JPEG image
print('Image shape: %r' % (img.shape,))    # print image size and
color depth

plt.figure(figsize=(6, 6))
plt.imshow(img)                                  # displaying the origi
nal image
plt.show()
```

```
Image shape: (640, 640, 3)
```



In [16]:

```
def iterative(img):
    image = copy.deepcopy(img)  # create a copy of the image mat
rix
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            if x < image.shape[0]/2 and y < image.shape[1]/2:
                image[x,y] = image[x,y] * [0,1,1]  # removing th
e red channel
            elif x > image.shape[0]/2 and y < image.shape[1]/2:
                image[x,y] = image[x,y] * [1,0,1]  # removing th
e green channel
            elif x < image.shape[0]/2 and y > image.shape[1]/2:
                image[x,y] = image[x,y] * [1,1,0]  # removing th
e blue channel
            else:
                pass
    return image

def vectorized(img):
    image = copy.deepcopy(img)
    a = int(image.shape[0]/2)
    b = int(image.shape[1]/2)
    image[:a,:b] = image[:a,:b]*[0,1,1]
    image[a:,:b] = image[a:,:b]*[1,0,1]
    image[:a,b:] = image[:a,b:]*[1,1,0]

    return image
```

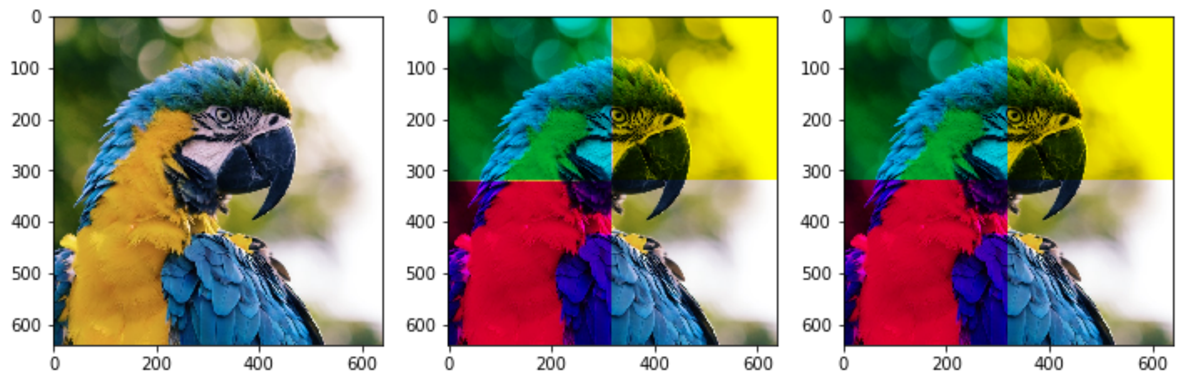
In [19]:

```
### The code for this problem is posted on Piazza. Sign up for t
he course if you have not.
### Then find the function definition included in the post "Welc
ome to CSE 252A" to complete this problem.
### This is the only cell you will need to edit for Problem 1.
def piazza():
    start = time.time()
    image_iterative = iterative(img)
    end = time.time()
    print("Iterative method took {0} seconds".format(end-start))
    start = time.time()
    image_vectorized = vectorized(img)
    end = time.time()
    print("Vectorized method took {0} seconds".format(end-start)
)
    return image_iterative, image_vectorized

# Run the function
image_iterative, image_vectorized = piazza()
```

```
Iterative method took 2.688349962234497 seconds
Vectorized method took 0.009750127792358398 seconds
```

In [20]:

```
# Plotting the results in separate subplots

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)  # create (1x3) subplots, indexing from 1
plt.imshow(img)       # original image

plt.subplot(1, 3, 2)
plt.imshow(image_iterative)

plt.subplot(1, 3, 3)
plt.imshow(image_vectorized)

plt.show()            # displays the subplots

plt.imsave('multicolor_macaw.png', image_vectorized) # saving an
image
```



Problem 2: Further Image Manipulation (7 points)

In this problem, you will solve a jigsaw puzzle given by the `jigsaw.png` image provided with the assignment. Note that the TAs have created the puzzle specifically for this class, meaning the solution will be a hopefully recognizable man (hint: a ______ -man!). There are a total of nine jigsaw pieces of size 256x256x3, which together form a 768x768x3 image. There are three types of transformations on the pieces that you will need to resolve:

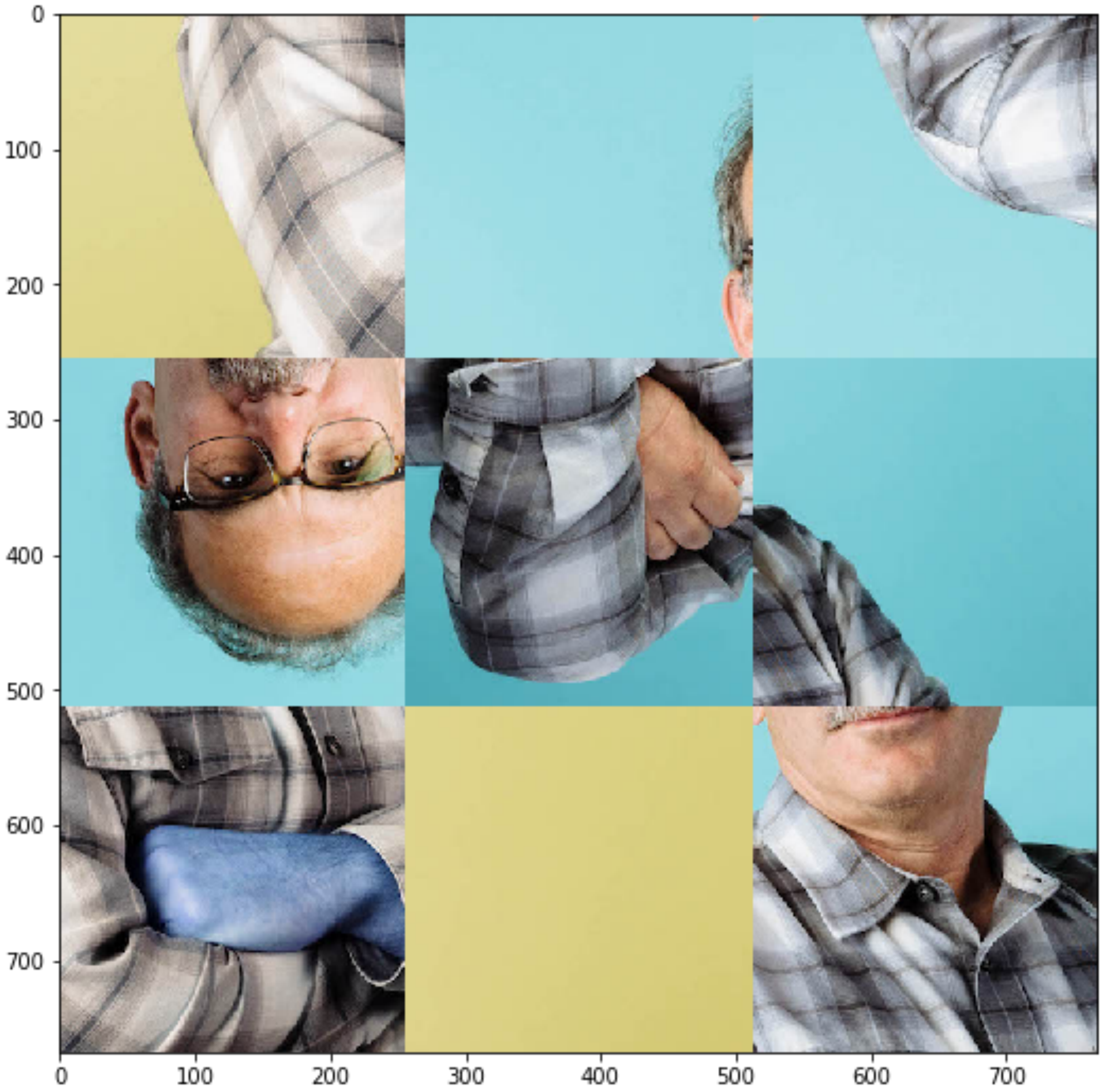
- First, the pieces are jumbled spatially.
- Second, some of the pieces are rotated (by either 90, 180, or 270 degrees).
- And third, some of the pieces have had their channels swapped from RGB to BGR.

Your task will be to correct all these transformations in order to solve this image manipulation jigsaw puzzle and recreate the original image. To do so, first implement the four helper functions below, which you will use to solve the puzzle. For this assignment, you are required to implement and use these three functions. Also, the code must be vectorized, i.e. you are not allowed to use any `for` -loops over the spatial dimensions of the image.

In [21]:

```
import numpy as np
import matplotlib.pyplot as plt
import copy
plt.rcParams['image.cmap'] = 'gray'  # necessary to override def
ault matplotlib behaviour

%matplotlib inline

# Read the image
jigsaw = plt.imread('jigsaw.png')
print('Image shape: %r' % (jigsaw.shape,))

# Display the image
plt.figure(figsize=(9, 9))
plt.imshow(jigsaw)
plt.show()
```

Image shape: (768, 768, 3)



In [85]:

```
# Implement each of the following four helper functions

def get_tile(image, row_index, col_index, tile_size):
    """This function returns a particular square tile of the ima
ge.

    (ROW_INDEX, COL_INDEX) describes the location of the tile in
the image.

    - For example, the (0, 0) tile is the top-left tile,
      the (0, 1) tile is the tile to the right of the top-left t
ile,
      and the (1, 0) tile is the tile directly below the top-lef
t tile.
```

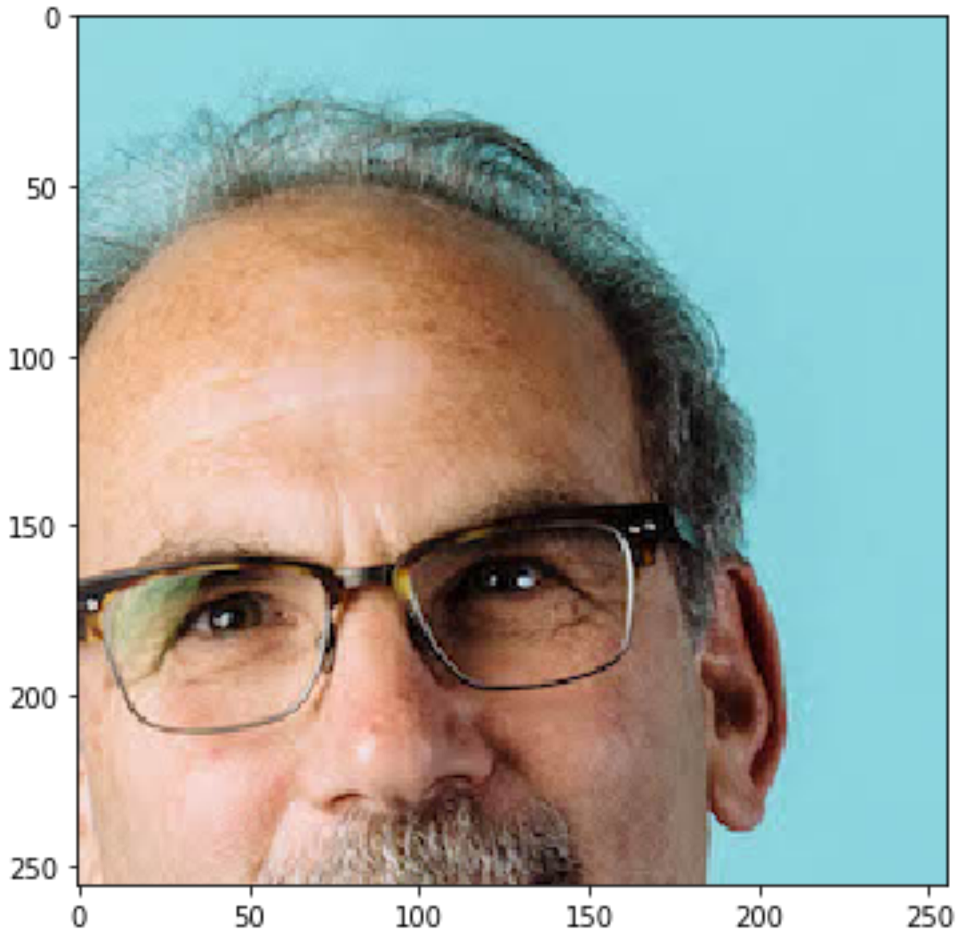
```
    TILE_SIZE is the side length of each square tile.

    - For example, if TILE_SIZE is 256, then the (0, 1) tile
      will be the tile with indices on the vertical axis from 0
to 255,
      and with indices on the horizontal axis from 256 to 511.
    """
    height, width = image.shape[:2]
    assert tile_size <= height and tile_size <= width

    """YOUR CODE HERE; you may delete and replace the code below
"""

    yourTile = np.zeros((tile_size, tile_size, 3))

    rowStart, rowStop = row_index*tile_size, (row_index+1)*tile_
size
    colStart, colStop = col_index*tile_size, (col_index+1)*tile_
size
    yourTile[0:tile_size, 0:tile_size, 0:3] = image[rowStart:row
Stop, colStart:colStop, 0:3]

    return yourTile



def RGBtoBGR(image):
    """This function swaps the first and last channels in the co
lor image.
    For example, if the image is originally formatted as an RGB
image, the result will be a BGR image.
    And vice-versa: this function is its own inverse, and will a
lso perform the BGR -> RGB conversion.

    You may assume that the image has three channels.
    If the input image is of shape (h, w, 3), then the return va
lue will also be of shape (h, w, 3).
    """
    assert len(image.shape) == 3, 'the image must have a channel
dimension'
    assert image.shape[-1] == 3, 'the image must have three chan
nels'
    image = copy.deepcopy(image)  # create a copy in order to en
```

```
sure that the original image does not change

    """YOUR CODE HERE; you may delete and replace the code below
"""

    length, width = image.shape[0], image.shape[1]
    #print(length, width)

    newImage = np.zeros((length, width, 3))
    newImage[:,:,0] = image[:,:,2]
    newImage[:,:,1] = image[:,:,1]
    newImage[:,:,2] = image[:,:,0]

    return newImage


def rotate90(image):
    """This function rotates an image 90 degrees counterclockwis
e.

    You may assume that the image is square.

    You may NOT use np.rot90; we would like you to implement rot
ation yourself using array indexing.
    Since you only need to handle a specific 90 degree rotation
case, this should be doable.

    Be sure that your function rotates the image COUNTERCLOCKWIS
E.
    """
    image = copy.deepcopy(image)  # create a copy in order to en
sure that the original image does not change
    """YOUR CODE HERE; you may delete and replace the code below
"""

    #slice each color and rotate
    tempAxisR = np.zeros((256,256))
    tempAxisG = np.zeros((256,256))
    tempAxisB = np.zeros((256,256))
    tempAxisR = image[:,:,0]
    tempAxisB = image[:,:,1]
    tempAxisG = image[:,:,2]
    tempAxisR = np.flip(tempAxisR.T,0)
    tempAxisB = np.flip(tempAxisB.T,0)
    tempAxisG = np.flip(tempAxisG.T,0)
```

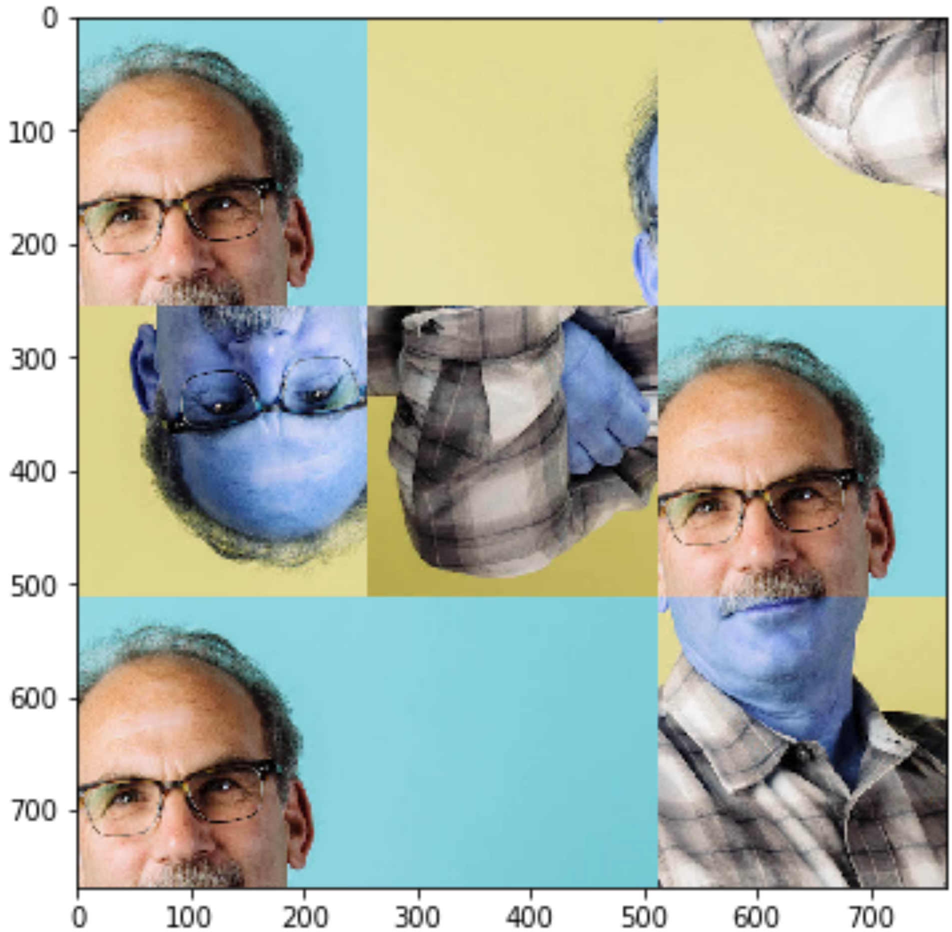
```
    #reasssemble the image matrix

    newImage = np.zeros((256,256,3))
    newImage[:,:,0] = tempAxisR
    newImage[:,:,1] = tempAxisB
    newImage[:,:,2] = tempAxisG
    #print(tempAxisR.shape)
    return newImage


def replace_tile(image, row_index, col_index, tile):
    """This function replaces the existing tile at (ROW_INDEX, C
OL_INDEX) with the provided tile.
    You will use this to put together the final puzzle solution.

    (ROW_INDEX, COL_INDEX) describes the location of the tile to
replace.
    You may assume that the tile size is given by the shape of t
he TILE argument.

    Replace whatever is already at the tile position with the pa
ssed-in TILE.
    Then return the modified full image (we are doing this in a
non-destructive way).
    """
    image = copy.deepcopy(image)  # create a copy in order to en
sure that the original image does not change

    """YOUR CODE HERE; you may delete and replace the code below
"""

    tile_size = tile.shape[0]
    image[row_index*tile_size:(row_index+1)*tile_size, col_index
*tile_size:(col_index+1)*tile_size, :] = tile
    return image
```

In [90]:

```
### TEST CELL ###

tileTest = get_tile(jigsaw, 1, 0, 256)
tileTest = RGBtoBGR(tileTest)
tileTest = RGBtoBGR(tileTest)
tileTest = rotate90(tileTest)
tileTest = rotate90(tileTest)
plt.figure(figsize=(6, 6))
plt.imshow(tileTest)
plt.show()
print(tileTest.shape)
#newImage = np.zeros((256, 256, 3))
#print(newImage)
newImage = RGBtoBGR(jigsaw)
swappedImage = newImage
swappedImage = replace_tile(swappedImage, 0, 0, tileTest)
swappedImage = replace_tile(swappedImage, 1, 2, tileTest)
swappedImage = replace_tile(swappedImage, 2, 0, tileTest)
plt.figure(figsize=(6, 6))
plt.imshow(swappedImage)
plt.show()
```



```
(256, 256, 3)
```

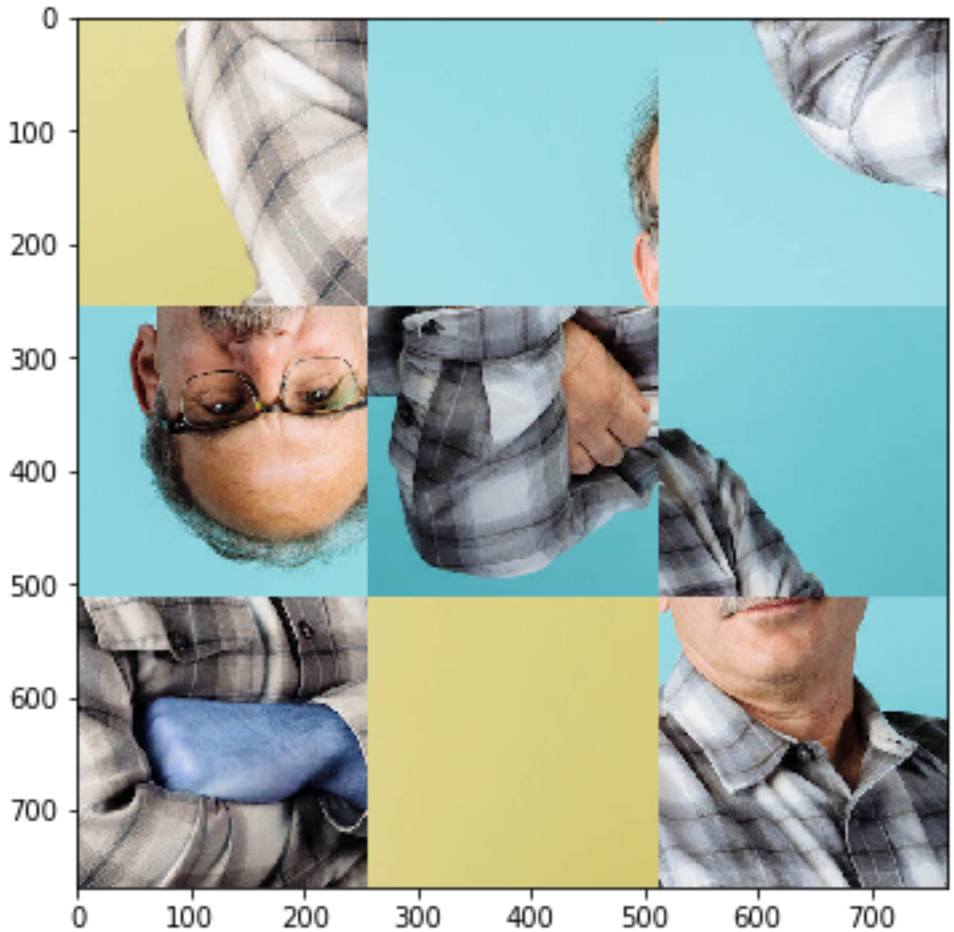


Now you must use the helper functions you just implemented in order to solve the jigsaw puzzle. When you're finished, display your result!

Note: just to be clear, you are not expected to write general code to solve arbitrary jigsaw puzzle problems. You only need to solve this one in particular (and this is just meant to be a fun exercise in basic image manipulation operations). Thus, all you need to do is hardcode the transformations for this puzzle. There's no need to (for example) search over the space of transformations and select the best ones based on consistency between different boundaries.

In [91]:

```
# For reference, here is the jigsaw puzzle again

plt.figure(figsize=(6, 6))
plt.imshow(jigsaw)
plt.show()
```

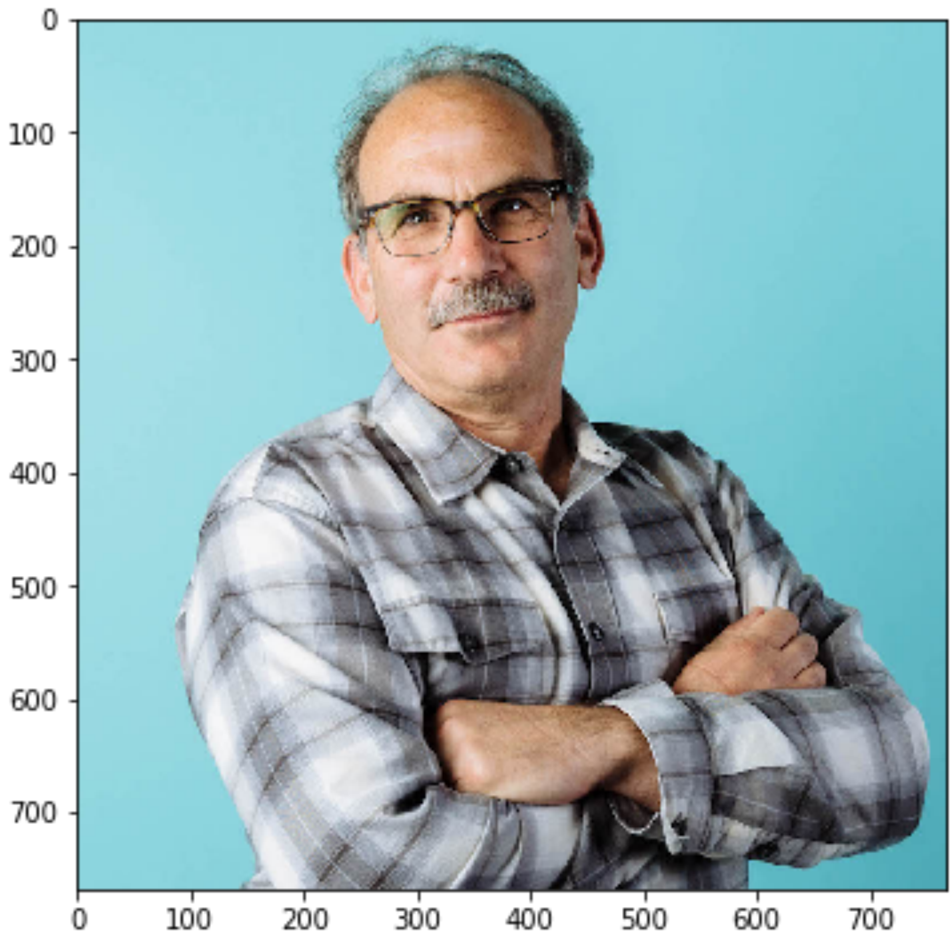


In [95]:

```
# Solve the jigsaw puzzle

tile_size = 256

# Extract tiles
# Note: the tile named "tileIJ" refers to the tile at row I and
column J
tile00 = get_tile(jigsaw, 0, 0, tile_size)
tile01 = get_tile(jigsaw, 0, 1, tile_size)
tile02 = get_tile(jigsaw, 0, 2, tile_size)

tile10 = get_tile(jigsaw, 1, 0, tile_size)
tile11 = get_tile(jigsaw, 1, 1, tile_size)
```

```
tile11 = get_tile(jigsaw, 1, 1, tile_size)
tile12 = get_tile(jigsaw, 1, 2, tile_size)

tile20 = get_tile(jigsaw, 2, 0, tile_size)
tile21 = get_tile(jigsaw, 2, 1, tile_size)
tile22 = get_tile(jigsaw, 2, 2, tile_size)

# Transform tiles as necessary
# You will want to use RGBtoBGR and rotate90.
# Hint: three tiles are rotated, and three (different) tiles hav
e been switched from RGB to BGR.
"""YOUR CODE HERE"""
tile00 = RGBtoBGR(tile00)
tile20 = RGBtoBGR(tile20)
tile21 = RGBtoBGR(tile21)

tile10 = rotate90(rotate90(tile10))
tile11 = rotate90(tile11)
tile02 = rotate90(rotate90(rotate90(tile02)))


solution = np.zeros_like(jigsaw)
# Place tiles in their proper position in the solution image
"""YOUR CODE HERE; modify the row_index and col_index arguments
to the `replace_tile` calls"""

solution = replace_tile(solution, 2, 0, tile00)
solution = replace_tile(solution, 0, 0, tile01)
solution = replace_tile(solution, 1, 0, tile02)#####

solution = replace_tile(solution, 0, 1, tile10)
solution = replace_tile(solution, 2, 2, tile11)
solution = replace_tile(solution, 1, 2, tile12)

solution = replace_tile(solution, 2, 1, tile20)
solution = replace_tile(solution, 0, 2, tile21)
solution = replace_tile(solution, 1, 1, tile22)

print('My jigsaw solution')
plt.figure(figsize=(6, 6))
plt.imshow(solution)
plt.show()
```

```
My jigsaw solution
```



Problem 3: Mathematics Background Check (3 points)

The last part of this homework is a review of the math prerequisites in order to make sure you're prepared for the rest of the course. The topics include linear algebra, probability, and calculus. You should make a solid attempt at every question, but you don't need to get every question correct to get full credit. In fact, this problem will be graded based on effort. We just want you to use this section as review, and perhaps as a diagnostic check as well.

To submit your answers to this problem, you can scan your handwritten work and include it in the PDF, or create a Markdown cell below to answer each of the problems (possibly using LaTeX wherever it is helpful; note that you can use LaTeX in a Markdown cell with `$...$` syntax for inline math, and `$$...$$` syntax for non-inline math).

You may find the following resources helpful for this part:

- [Linear algebra review (http://www.cse.ucsd.edu/classes/fa11/cse252A-a/linear_algebra_review.pdf)](http://www.cse.ucsd.edu/classes/fa11/cse252A-a/linear_algebra_review.pdf)
- [Random variables review (http://www.cse.ucsd.edu/classes/fa11/cse252A-a/random_var_review.pdf)](http://www.cse.ucsd.edu/classes/fa11/cse252A-a/random_var_review.pdf)

Part 1: Linear Algebra

1. Consider the following vectors:
   $\mathbf{v}_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}_3 = \begin{bmatrix} 2 \\ -3 \end{bmatrix}, \mathbf{v}_4 = \begin{bmatrix} 0 \\ 1 \\ -5 \end{bmatrix}$
   A. What is $\mathbf{v}_1 + \mathbf{v}_2$?
   B. What is $\mathbf{v}_1 \bullet \mathbf{v}_2$?
   C. What is $\mathbf{v}_1 \times \mathbf{v}_2$?
   D. Express the dot product from B in terms of matrix multiplication.
   E. Explain why $\mathbf{v}_3 \bullet \mathbf{v}_4$ is not defined.
2. Consider the following matrices: $A = \begin{bmatrix} 1 & 2 \\ 3 & 3 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$
   A. What is the transpose of A? Is A symmetric? Is B symmetric?
   B. What is the rank of B?
   C. What is the determinant of A?
   D. What is the trace of A?
   E. Is B invertible? How many linearly independent columns does B have?
   F. What is the nullspace of B? What is its dimension?
   G. Write the general form of a 2x2 rotation matrix. Denote this as R.
      a. What is its determinant?
      b. What is $R^T R$?
   H. Does the equation $Ax = b$ have a solution for all b? Can you say the same for the equation $Bx = b$?
3. Consider the matrix $C = \begin{bmatrix} 1 & 6 \\ 1 & 2 \end{bmatrix}$
   A. What are the eigenvalues and eigenvectors of C?

Part 2: Calculus

1. Note the following definitions and answer the questions below.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad f(\mathbf{x}) = x_1^2 x_2, \quad g(y) = y^3 + 2y^2, \quad h(x) = \sin(x)$$

A. What is the gradient of $f(\mathbf{x})$?
B. What is the Hessian of $f(\mathbf{x})$?
C. What are the local minimum and maxima of $g(y)$?
D. What is the derivative of $(h \circ g)(y)$ w.r.t y?

Part 3: Probability

1. Let A and B be two random events with $P(A) = x$, $P(B) = y$.
   A. Assume A and B are independent events. Now answer the following questions.
      a. What is $P(A|B)$?
      b. What is $P(AB)$?
      c. What is $P(A \cup B)$
   B. Now answer the same questions given that A and B are mutually exclusive.
2. Let X be a random event with $E(X) = 0.2$ and $E(X^2) = 0.5$. What is the variance of X?

In [102]:

```
WrittenOne = plt.imread('CSE 252A HW0 Written-3.jpg')
# read a JPEG image

plt.figure(figsize=(20, 20))
plt.imshow(WrittenOne)                              # displaying th
e original image
plt.show()
```

Part 1: Linear Algebra

1. Consider the following vectors: $\mathbf{v}_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}_3 = \begin{bmatrix} 2 \\ -3 \end{bmatrix}, \mathbf{v}_4 = \begin{bmatrix} 0 \\ 1 \\ -5 \end{bmatrix}$

   A. What is $\mathbf{v}_1 + \mathbf{v}_2$?
   B. What is $\mathbf{v}_1 \bullet \mathbf{v}_2$?
   C. What is $\mathbf{v}_1 \times \mathbf{v}_2$?
   D. Express the dot product from B in terms of matrix multiplication.
   E. Explain why $\mathbf{v}_3 \bullet \mathbf{v}_4$ is not defined.

1.) A) $V_1 + V_2 = \begin{bmatrix} 1+0 \\ -1+1 \\ 2-1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$

B.) $V_1 \cdot V_2 = 1 \cdot 0 + -1 \cdot 1 + 2 \cdot -1$
$= 0 + -1 - 2 = -3$

C.) Assuming $V_1 \times V_2$ means element wise multiplication?
(can't be matrix mult. b/c of dimensions)

so $V_1 \times V_2 = \begin{bmatrix} 0 \\ -1 \\ -2 \end{bmatrix}$

D.) $V_1 \cdot V_2$ is equivalent to $V_1^T \cdot V_2$

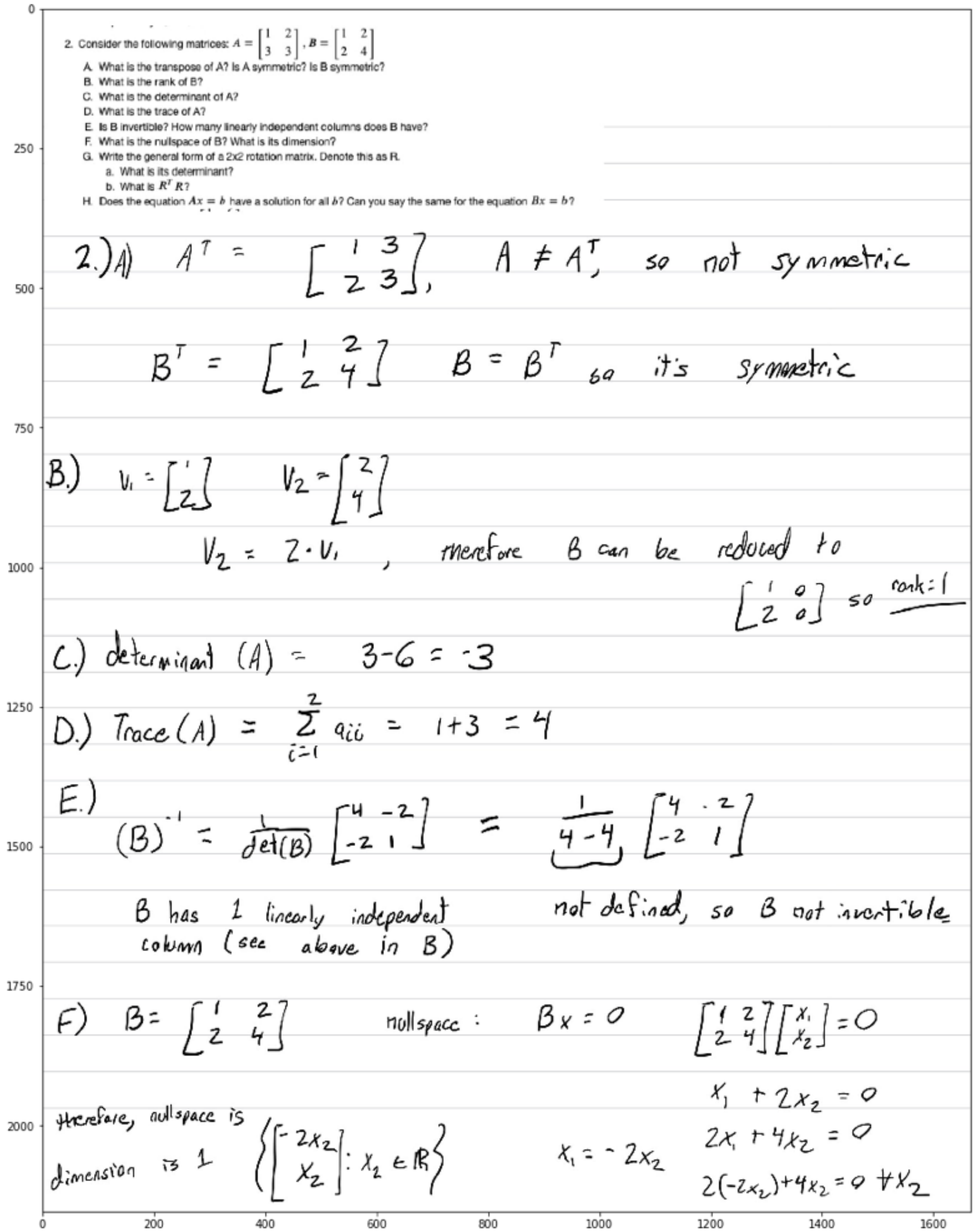
$V_1^T = [1 \quad -1 \quad 2]$

so $V_1^T \cdot V_2 = [1 \quad -1 \quad 2] \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} = -3$ ✓

E.) $V_3 \cdot V_4$ is not defined b/c the dimensions don't match. V_3 is in space $\mathbb{R}^2$ while V_4 is in space $\mathbb{R}^3$, since dot product shows the mag. of one vector in the direction of another, it cannot be applied since they're in different coordinate spaces and there's no way of knowing how they line up w/ each other. For example, V_3 in V_4's $\mathbb{R}^3$ space could be $\begin{bmatrix} 0 \\ 2 \\ -3 \end{bmatrix}$, or $\begin{bmatrix} 2 \\ -3 \\ 0 \end{bmatrix}$, or something completely different as well.

In [103]:

```python
WrittenTwo = plt.imread('CSE 252A HW0 Written-4.jpg')
# read a JPEG image

plt.figure(figsize=(20, 20))
plt.imshow(WrittenTwo)                                  # displaying th
e original image
plt.show()
```



2. Consider the following matrices: $A = \begin{bmatrix} 1 & 2 \\ 3 & 3 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$

A. What is the transpose of A? Is A symmetric? Is B symmetric?
B. What is the rank of B?
C. What is the determinant of A?
D. What is the trace of A?
E. Is B invertible? How many linearly independent columns does B have?
F. What is the nullspace of B? What is its dimension?
G. Write the general form of a 2x2 rotation matrix. Denote this as R.
  a. What is its determinant?
  b. What is $R^T R$?
H. Does the equation $Ax = b$ have a solution for all b? Can you say the same for the equation $Bx = b$?

2.) A) $A^T = \begin{bmatrix} 1 & 3 \\ 2 & 3 \end{bmatrix}$, $A \neq A^T$, so not symmetric

$B^T = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ $B = B^T$ so it's symmetric

B.) $v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $v_2 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$

$v_2 = 2 \cdot v_1$, therefore B can be reduced to $\begin{bmatrix} 1 & 0 \\ 2 & 0 \end{bmatrix}$ so rank = 1

C.) determinant $(A) = 3 - 6 = -3$

D.) $\text{Trace}(A) = \sum_{i=1}^{2} a_{ii} = 1 + 3 = 4$

E.) $(B)^{-1} = \frac{1}{\det(B)} \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix} = \underbrace{\frac{1}{4-4}}_{} \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix}$

B has 1 linearly independent column (see above in B)

not defined, so B not invertible

F) $B = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ nullspace : $Bx = 0$ $\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$

$x_1 + 2x_2 = 0$

$2x_1 + 4x_2 = 0$

$x_1 = -2x_2$

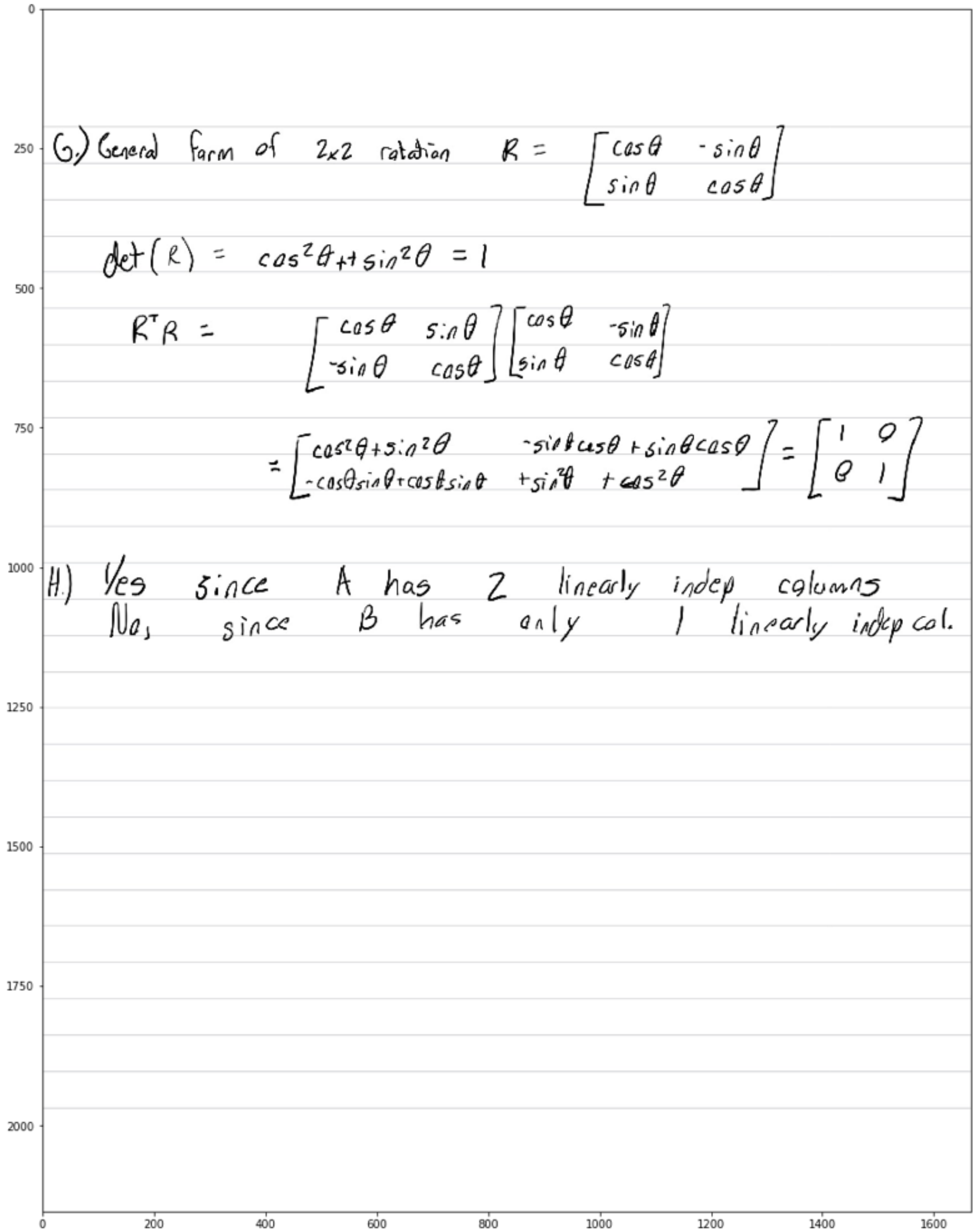
$2(-2x_2) + 4x_2 = 0 \ \forall x_2$

therefore, nullspace is $\left\{ \begin{bmatrix} -2x_2 \\ x_2 \end{bmatrix} : x_2 \in \mathbb{R} \right\}$

dimension is 1

In [104]:

```
WrittenThree = plt.imread('CSE 252A HW0 Written-5.jpg')
# read a JPEG image

plt.figure(figsize=(20, 20))
plt.imshow(WrittenThree)                                    # displaying the original image
plt.show()
```



G.) General form of 2x2 rotation $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

$\det(R) = \cos^2\theta + \sin^2\theta = 1$

$R^T R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

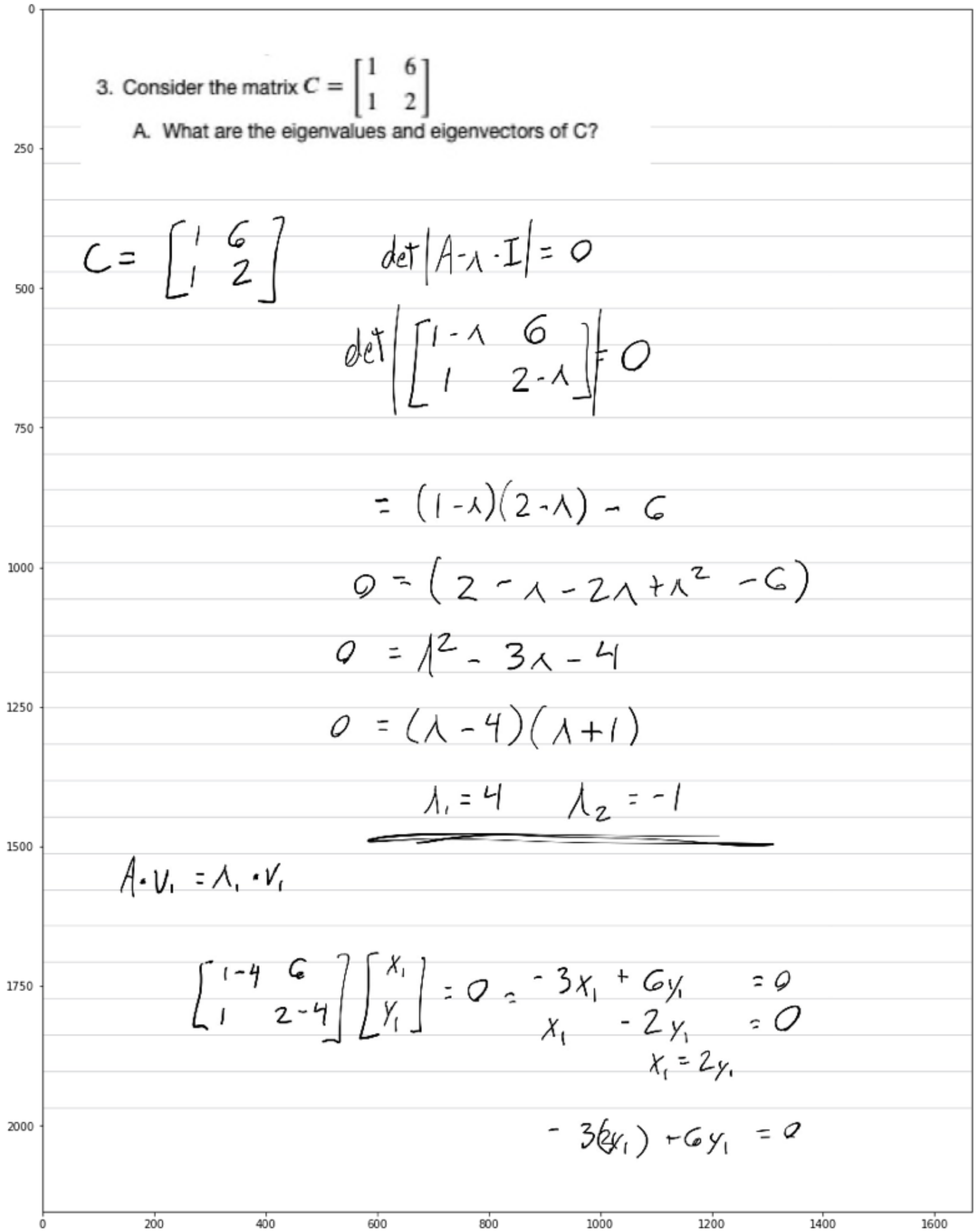
$= \begin{bmatrix} \cos^2\theta + \sin^2\theta & -\sin\theta\cos\theta + \sin\theta\cos\theta \\ -\cos\theta\sin\theta + \cos\theta\sin\theta & +\sin^2\theta + \cos^2\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

H.) Yes since A has 2 linearly indep columns

No, since B has only 1 linearly indep col.

In [105]:

```
WrittenFour = plt.imread('CSE 252A HW0 Written-6.jpg')
# read a JPEG image

plt.figure(figsize=(20, 20))
plt.imshow(WrittenFour)                                # displaying t
he original image
plt.show()
```



3. Consider the matrix $C = \begin{bmatrix} 1 & 6 \\ 1 & 2 \end{bmatrix}$

A. What are the eigenvalues and eigenvectors of C?

$C = \begin{bmatrix} 1 & 6 \\ 1 & 2 \end{bmatrix}$ $\quad \det|A - \lambda \cdot I| = 0$

$$\det\left|\begin{bmatrix} 1-\lambda & 6 \\ 1 & 2-\lambda \end{bmatrix}\right| = 0$$

$$= (1-\lambda)(2-\lambda) - 6$$

$$0 = (2 - \lambda - 2\lambda + \lambda^2 - 6)$$

$$0 = \lambda^2 - 3\lambda - 4$$

$$0 = (\lambda - 4)(\lambda + 1)$$

$$\lambda_1 = 4 \qquad \lambda_2 = -1$$

$A \cdot V_1 = \lambda_1 \cdot V_1$

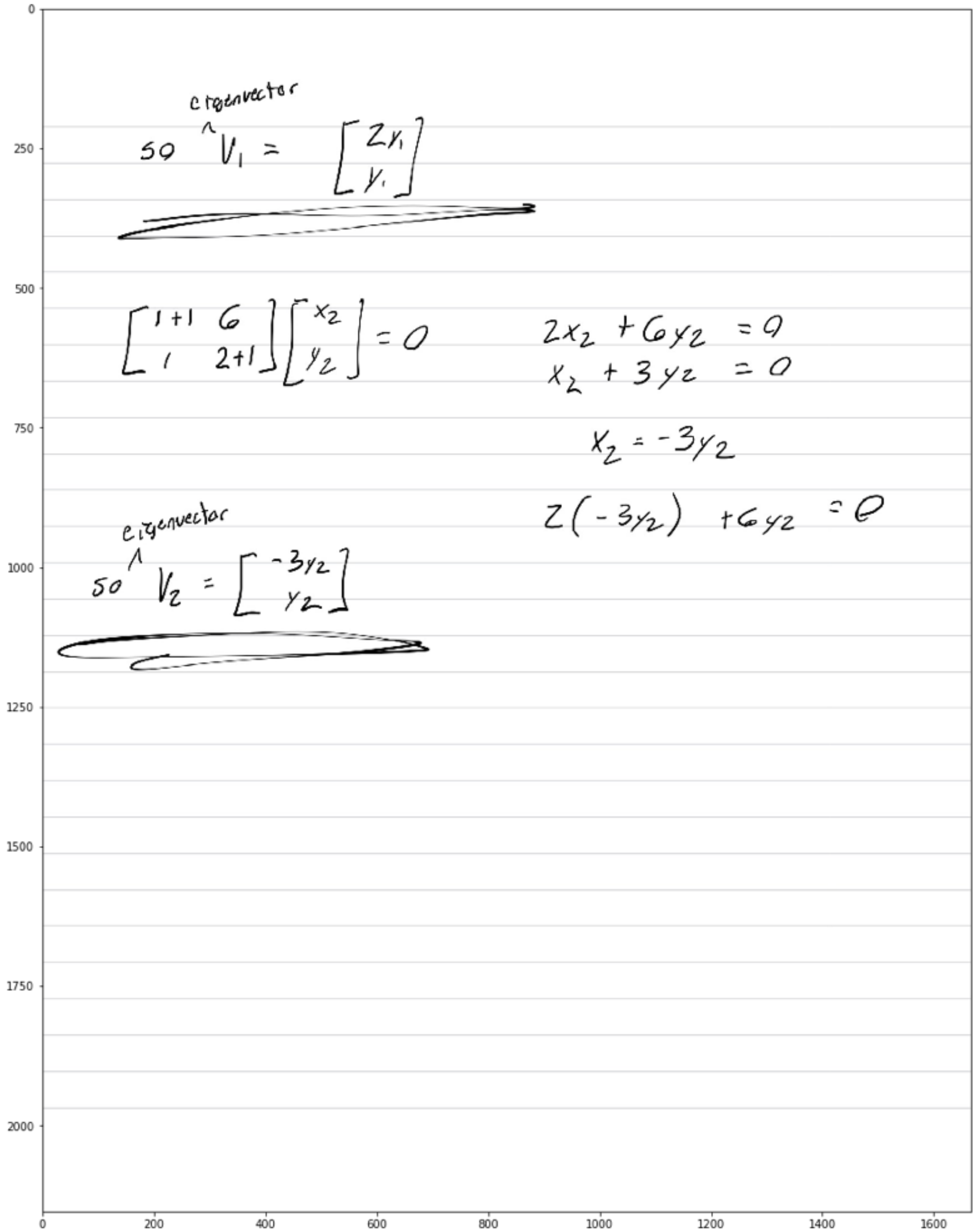
$$\begin{bmatrix} 1-4 & 6 \\ 1 & 2-4 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = 0 = \begin{array}{l} -3x_1 + 6y_1 = 0 \\ x_1 - 2y_1 = 0 \end{array}$$

$$x_1 = 2y_1$$

$$-3(2y_1) + 6y_1 = 0$$

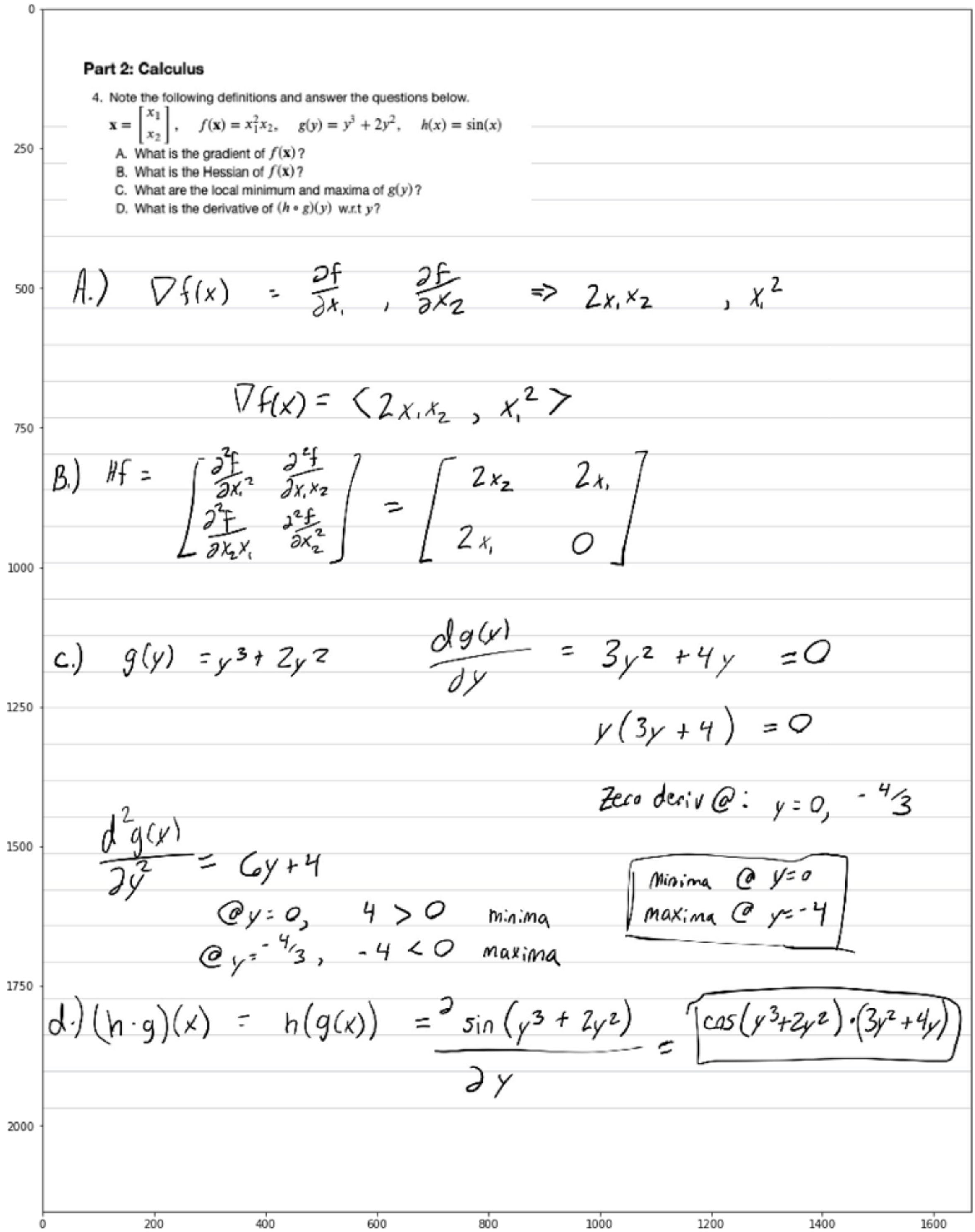
In [106]:

```
WrittenFive = plt.imread('CSE 252A HW0 Written-7.jpg')
# read a JPEG image

plt.figure(figsize=(20, 20))
plt.imshow(WrittenFive)                                    # displaying t
he original image
plt.show()
```



In [107]:

```
WrittenSix = plt.imread('CSE 252A HW0 Written-8.jpg')
# read a JPEG image

plt.figure(figsize=(20, 20))
plt.imshow(WrittenSix)                                    # displaying th
e original image
plt.show()
```



Part 2: Calculus

4. Note the following definitions and answer the questions below.

$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad f(\mathbf{x}) = x_1^2 x_2, \quad g(y) = y^3 + 2y^2, \quad h(x) = \sin(x)$

A. What is the gradient of $f(\mathbf{x})$?
B. What is the Hessian of $f(\mathbf{x})$?
C. What are the local minimum and maxima of $g(y)$?
D. What is the derivative of $(h \circ g)(y)$ w.r.t y?

A.) $\nabla f(x) = \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \Rightarrow 2x_1 x_2, \; x_1^2$

$\nabla f(x) = \langle 2x_1x_2, \; x_1^2 \rangle$

B.) $Hf = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 2x_2 & 2x_1 \\ 2x_1 & 0 \end{bmatrix}$

c.) $g(y) = y^3 + 2y^2 \qquad \frac{dg(y)}{dy} = 3y^2 + 4y = 0$

$y(3y+4) = 0$

Zero deriv @: $y = 0, \; -4/3$

$\frac{d^2 g(y)}{\partial y^2} = 6y + 4$

$@ y = 0, \quad 4 > 0$ minima

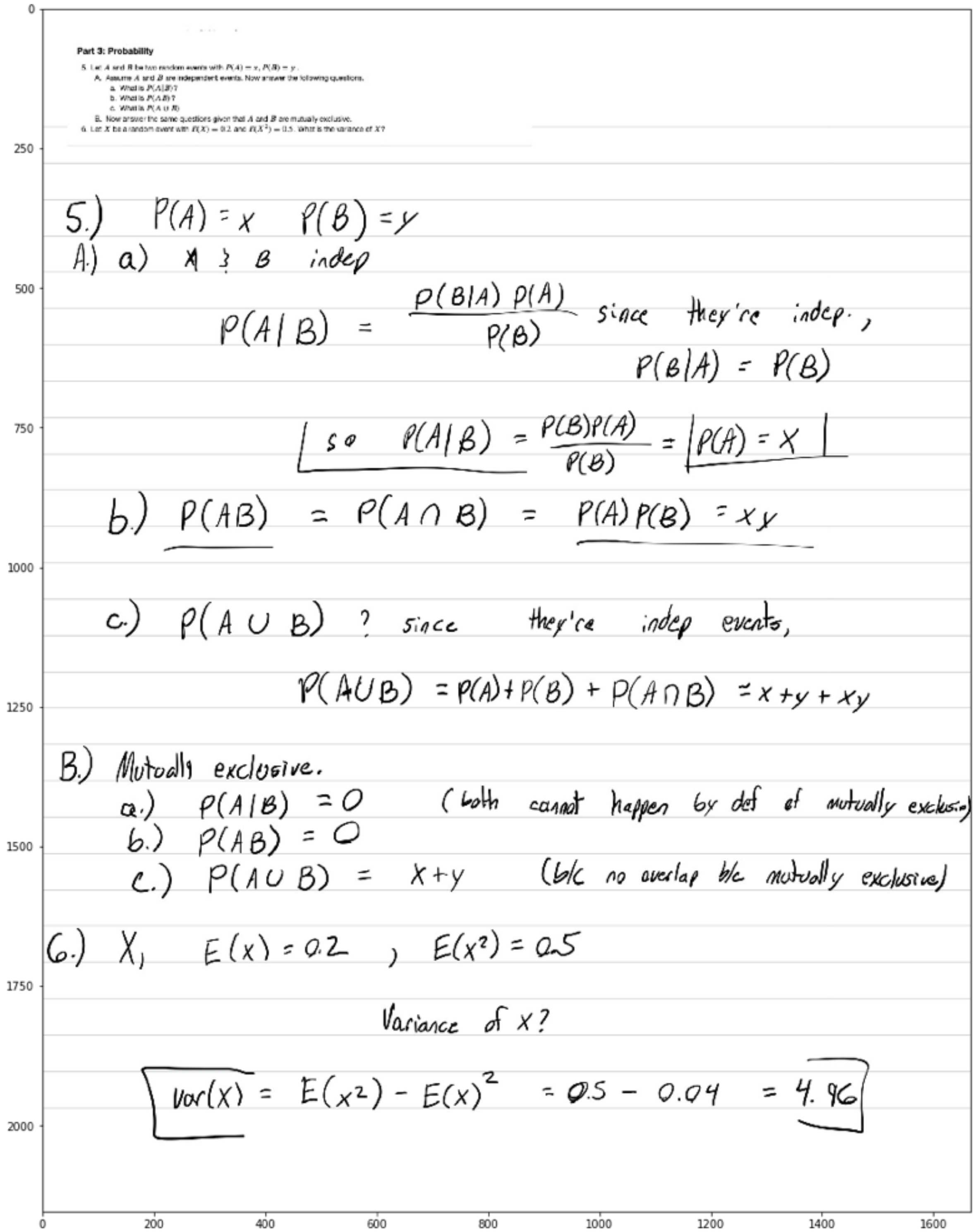
$@ y = -4/3, \quad -4 < 0$ maxima

$\boxed{\text{Minima } @ \; y = 0 \quad \text{Maxima } @ \; y = -4}$

d.) $(h \cdot g)(x) = h(g(x)) = \frac{\partial \sin(y^3 + 2y^2)}{\partial y} = \boxed{\cos(y^3 + 2y^2) \cdot (3y^2 + 4y)}$

In [108]:

```
WrittenSeven = plt.imread('CSE 252A HW0 Written-9.jpg')
# read a JPEG image

plt.figure(figsize=(20, 20))
plt.imshow(WrittenSeven)                                    # displaying
the original image
plt.show()
```



Part 3: Probability

5. Let A and B be two random events with $P(A) = x$, $P(B) = y$.
   A. Assume A and B are independent events. Now answer the following questions.
      a. What is $P(A|B)$?
      b. What is $P(AB)$?
      c. What is $P(A \cup B)$
   B. Now answer the same questions given that A and B are mutually exclusive.
6. Let X be a random event with $E(X) = 0.2$ and $E(X^2) = 0.5$. What is the variance of X?

5.) $P(A) = x \quad P(B) = y$

A.) a) A & B indep

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$ since they're indep.,

$$P(B|A) = P(B)$$

so $P(A|B) = \frac{P(B)P(A)}{P(B)} = \boxed{P(A) = x}$

b.) $\underline{P(AB)} = P(A \cap B) = \underline{P(A)P(B) = xy}$

c.) $P(A \cup B)$? since they're indep events,

$$P(A \cup B) = P(A) + P(B) + P(A \cap B) = x + y + xy$$

B.) Mutually exclusive.

a.) $P(A|B) = 0$ (both cannot happen by def of mutually exclusive)

b.) $P(AB) = 0$

c.) $P(A \cup B) = x + y$ (b/c no overlap b/c mutually exclusive)

6.) X, $E(x) = 0.2$, $E(x^2) = 0.5$

Variance of x?

$$\boxed{Var(X) = E(x^2) - E(x)^2 = 0.5 - 0.04 = 4.96}$$

Submission Instructions

Remember to submit a PDF version of this notebook to Gradescope. You can create a PDF via **File > Download as > PDF via LaTeX** (preferred, if possible), or by downloading as an HTML page and then "printing" the HTML page to a PDF (by opening the print dialog and then choosing the "Save as PDF" option).