CS231A: Computer Vision, From 3D Reconstruction to Recognition
Homework #0 Solution

(Winter 2021)

1 Basic Matrix/Vector Manipulation (10 points)

In Python, please calculate the following. Given matrix M and vectors a,b,c such that

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

(a) Define Matrix M and Vectors a,b,c in Python. One helpful Python package that is commonly used for linear algebra related problems is Numpy.

```
    M = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9],
                  [0, 2, 2]])
    a = np.array([ 1, 1, 0])
    b = np.array([-1, 2, 5])
    c = np.array([ 0, 2, 3, 2])

```

Listing 1: Code Snippet

(b) Find the dot product of vectors a and b (i.e. $a^\top b$). Save this value to the variable `aDotb` and write its value in your report.

```
    aDotb = np.dot(np.transpose(a), b)

```

Listing 2: Code Snippet

```
    aDotb = 1

```

Listing 3: Output of the dot product

(c) Find the element-wise product of a and b $[a_1b_1, a_2b_2, a_3b_3]^T$ and write it in your report.

```
    aProdb = np.multiply(a, b)

```

Listing 4: Code Snippet

```
    aProdb = [-1  2  0]

```

Listing 5: Output of the element-wise product

(d) Find $(a^\top b)Ma$ and write it in your report.

```
    aDotb = part_b(a, b)
    result = np.dot(aDotb, np.dot(M, a))

```

Listing 6: Code Snippet

```
    result = [ 3  9 15  2]

```

Listing 7: Output of $(a^\top b)Ma$

(e) Without using a loop, multiply each row of M element-wise by a. (Hint: The function numpy.tile() may come in handy). Write the result in your report.

```
    a_tiled = np.tile(np.transpose(a), (4, 1))
    newM = np.multiply(M, a_tiled)

```

Listing 8: Code Snippet

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

```

Listing 9: Output of the the multiplication

(f) Without using a loop, sort all of the values of the new M from (e) in increasing order and plot them in your report.

```
    sortedM = np.sort(np.reshape(M, -1))

```

Listing 10: Code Snippet

Resulting plot shown in Figure 1.

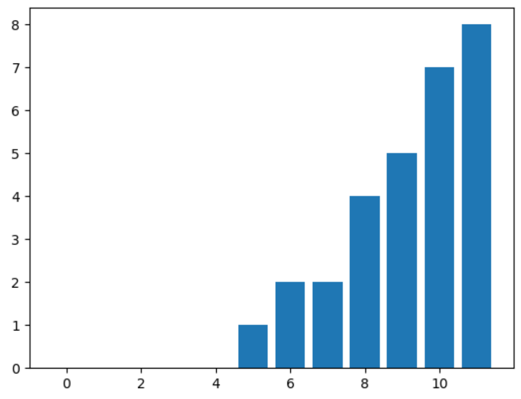

Figure 1: Resulting Plot for Problem 2(f)

2 Basic Image Manipulations (20 points)

(a) Read in the images, image1.jpg and image2.jpg, as color images.

```
    img1 = io.imread('image1.jpg')
    img2 = io.imread('image2.jpg')

```

Listing 11: Code Snippet

(b) Convert the images to double precision and rescale them to stretch from minimum value 0 to maximum value 1.

```
    def normalize_img(img):
        img = np.array(img, dtype='double')
        img = (img - np.min(img)) / (np.max(img) - np.min(img))
        return img

    def part_b(img1, img2):
        # ===== Problem 3b =====
        # Convert the images to double precision and rescale them
        # to stretch from minimum value 0 to maximum value 1.

        # BEGIN YOUR CODE HERE
        img1 = normalize_img(img1)
        img2 = normalize_img(img2)

        # END YOUR CODE HERE
        return img1, img2

```

Listing 12: Code Snippet

(c) Add the images together and re-normalize them to have minimum value 0 and maximum value 1. Display this image in your report.

```
    def part_c(img1, img2):
        # ===== Problem 3c =====
        # Add the images together and re-normalize them
        # to have minimum value 0 and maximum value 1.
        # Display this image.
        sumImage = None

        # BEGIN YOUR CODE HERE
        sumImage = img1 + img2
        sumImage = normalize_img(sumImage)

        # END YOUR CODE HERE
        return sumImage

```

Listing 13: Code Snippet

Resulting figure shown in Figure 2.

(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 image2. Display this image in your report.

```
    def part_d(img1, img2):
        # ===== Problem 3d =====
        # Create a new image such that the left half of
        # the image is the left half of image1 and the
```

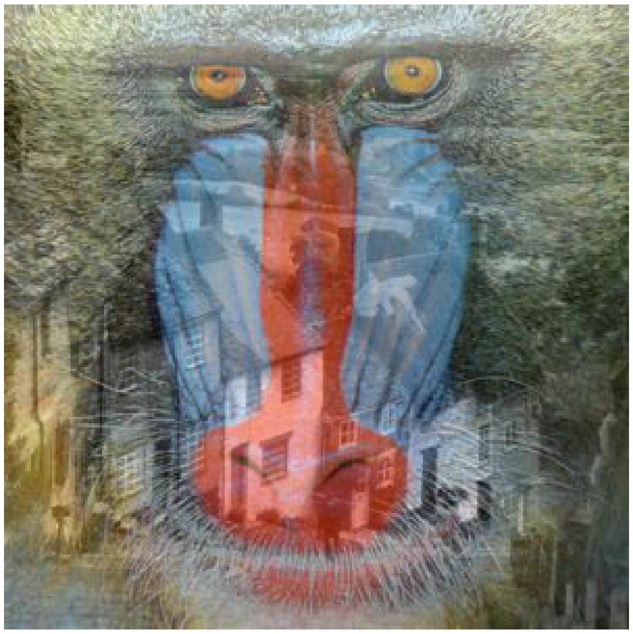

Figure 2: Resulting figure for Problem 3(c)

```
    # right half of the image is the right half of image2.

    newImage1 = None

    # BEGIN YOUR CODE HERE
    newImage1 = np.zeros_like(img1)
    newImage1[:,:int(newImage1.shape[1]/2)] = img1[:,:int(newImage1.shape
[1]/2)]
    newImage1[:,int(newImage1.shape[1]/2):] = img2[:,int(newImage1.shape
[1]/2):]

    # END YOUR CODE HERE
    return newImage1

```

Listing 14: Code Snippet

Resulting figure shown in Figure 3.

(e) Using a for loop, create a new image such that every odd numbered row is the corresponding row from image1 and the every even row is the corresponding row from image2 (Hint: Remember that indices start at 0 and not 1 in Python). Display this image in your report.

```
def part_e(img1, img2):
    # ===== Problem 3e =====
    # Using a for loop, create a new image such that every odd
    # numbered row is the corresponding row from image1 and the
    # every even row is the corresponding row from image2.
    # Hint: Remember that indices start at 0 and not 1 in Python.

    newImage2 = None

    # BEGIN YOUR CODE HERE
    newImage2 = img1
    for i in range(0,img1.shape[0],2):
        newImage2[i,:] = img2[i,:]

```

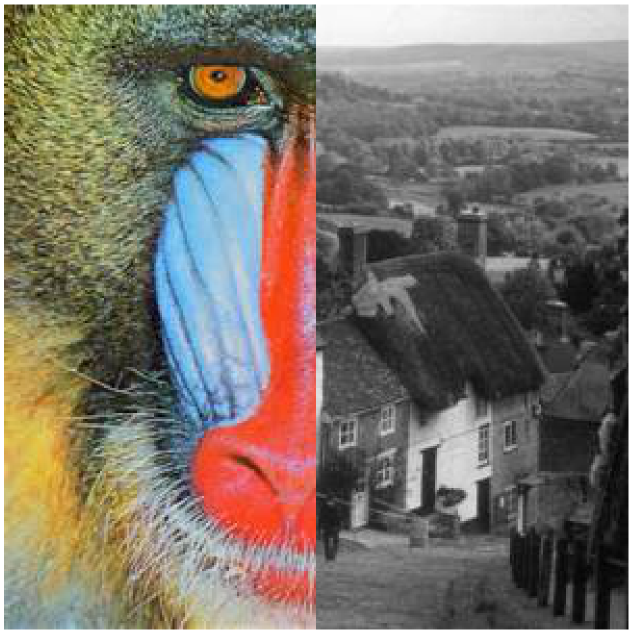

Figure 3: Resulting figure for Problem 3(d)

```
        # END YOUR CODE HERE
        return newImage2

```

Listing 15: Code Snippet

Resulting figure shown in Figure 4.

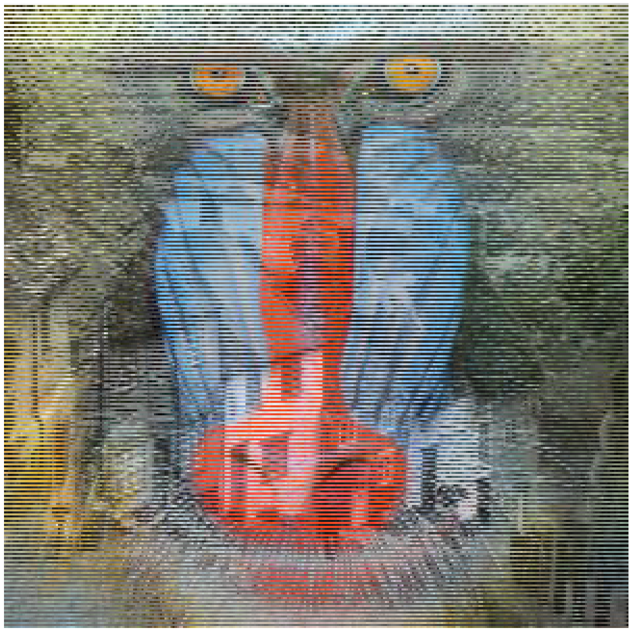

Figure 4: Resulting figure for Problem 3(e)

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

```python
    def part_f(img1, img2):
        # ===== Problem 3f =====
        # Accomplish the same task as part e without using a for-loop.
        # The functions reshape and tile may be helpful here.

        newImage3 = None

        # BEGIN YOUR CODE HERE
        newImage3 = img1
        newImage3[::2] = img2[::2]

        # END YOUR CODE HERE
        return newImage3

```

Listing 16: Code Snippet

Resulting figure shown in Figure 5.

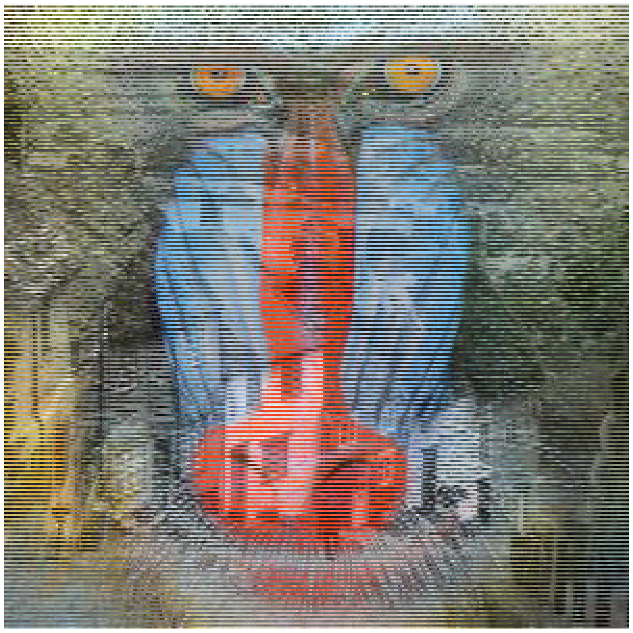

Figure 5: Resulting figure for Problem 3(f)

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

```python
    def part_g(img):
        # ===== Problem 3g =====
        # Convert the result from part f to a grayscale image.
        # Display the grayscale image with a title.

        # BEGIN YOUR CODE HERE
        R = img[:,:,0]
        G = img[:,:,1]
        B = img[:,:,2]
        img = R * 299/1000 + G * 587/1000 + B * 114/1000

        # END YOUR CODE HERE
        return img

```

Listing 17: Code Snippet

Resulting figure shown in Figure 6.

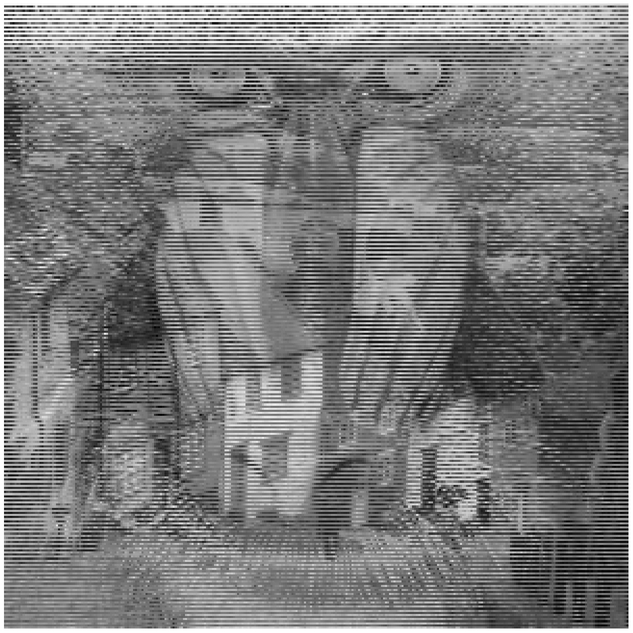
Figure 6: Resulting figure for Problem 3(g)

3 Singular Value Decomposition (20 points)

(a) Read in image1 as a grayscale image. Take the singular value decomposition of the image.

```python
    def part_a():
        # ===== Problem 4a =====
        # Read in image1 as a grayscale image. Take the singular value
        # decomposition of the image.

        img1 = None

        # BEGIN YOUR CODE HERE
        img1 = np.array(io.imread('image1.jpg', as_gray=True))
        u, s, v = np.linalg.svd(img1, full_matrices=True)

        # END YOUR CODE HERE
        return u, s, v

```

Listing 18: Code Snippet

(b) Recall from the discussion section that the best rank n approximation of a matrix is $\sum_{i=1}^{i=n} u_i \sigma_i v_i^\top$, where u_i, σ_i, and v_i are the ith left singular vector, singular value, and right singular vector respectively. Save and display the best rank 1 approximation of the (grayscale) image1 in your report.

```python
    def part_b(u, s, v):
        # ===== Problem 4b =====
        # Save and display the best rank 1 approximation
        # of the (grayscale) image1.

        rank1approx = None

        # BEGIN YOUR CODE HERE
        k = 1
        # extract top k vectors in singular matrices and top k values in
    singular values
        uk = u[:,:k]
        sk = s[:k]
        vk = v[:k,:]
        # perform low-rank approximation
        sk_tiled = np.transpose(np.tile(sk, (vk.shape[1], 1)))
        sv = np.multiply(sk_tiled, vk)
        rank1approx = np.dot(uk, sv)

        # END YOUR CODE HERE
        return rank1approx

```

Listing 19: Code Snippet

Resulting figure shown in Figure 7.

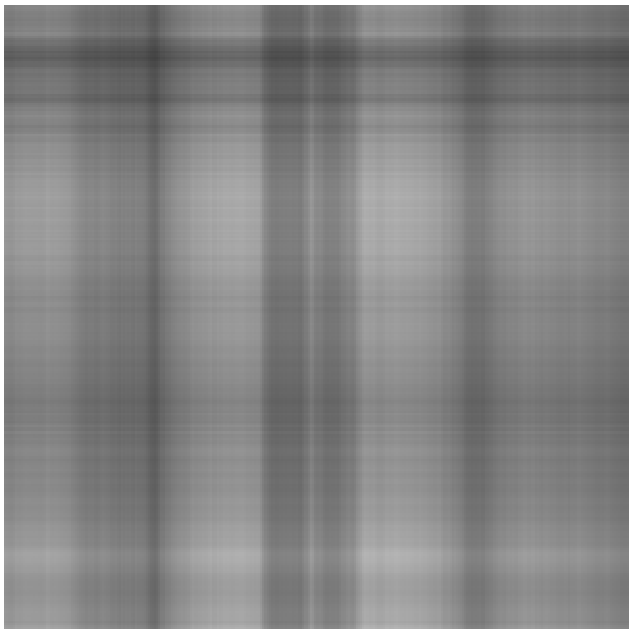

Figure 7: Resulting figure for Problem 4(b)

(c) Save and display the best rank 20 approximation of the (grayscale) image1 in your report.

```
    def part_c(u, s, v):
        # ===== Problem 4c =====
        # Save and display the best rank 20 approximation
        # of the (grayscale) image1.

        rank20approx = None

        # BEGIN YOUR CODE HERE
        k = 20
        # extract top k vectors in singular matrices and top k values in
    singular values
        uk = u[:,:k]
        sk = s[:k]
        vk = v[:k,:]
        # perform low-rank approximation
        sk_tiled = np.transpose(np.tile(sk, (vk.shape[1], 1)))
        sv = np.multiply(sk_tiled, vk)
        rank20approx = np.dot(uk, sv)

        # END YOUR CODE HERE
        return rank20approx

```

Listing 20: Code Snippet

Resulting figure shown in Figure 8.

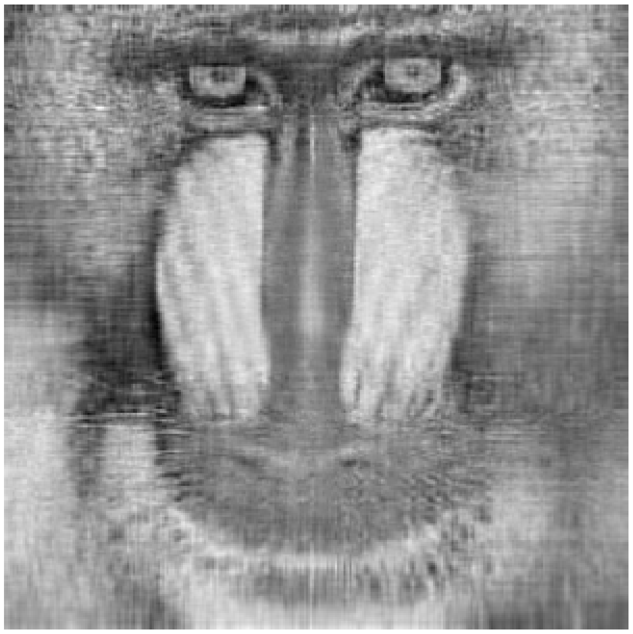

Figure 8: Resulting figure for Problem 4(c)