CS231A

Computer Vision: From 3D Reconstruction to Recognition

Luke Jaffe

Problem Set 0

Due 01/18/2018

1 Piazza

Registration complete.

2 Basic Matrix/Vector Manipulation

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

```
# ===== Problem 2a =====
# Define Matrix M and Vectors a,b,c in Python with NumPy

M, a, b, c = None, None, None, None

# BEGIN YOUR CODE HERE
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])
# END YOUR CODE HERE
```

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

```
# ===== Problem 2b =====
# Find the dot product of vectors a and b, save the value to aDotb

aDotb = None

# BEGIN YOUR CODE HERE
aDotb = np.dot(a, b)
print('Value of aDotb:\n', aDotb)
# END YOUR CODE HERE
```

```
Value of aDotb:
 1
```

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

```
# ===== Problem 2c =====
# Find the element-wise product of a and b

# BEGIN YOUR CODE HERE
p2_c = a*b
print('Element-wise product value:\n', p2_c)
# END YOUR CODE HERE
```

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

d) Find $(a^Tb)M\,a$ and write it in your report.

```
# ===== Problem 2d =====
# Find (a^T b)Ma

# BEGIN YOUR CODE HERE
p2_d = np.dot(a, b)*np.dot(M, a)
print('Dot product value:\n', p2_d)
# END YOUR CODE HERE
```

```
Dot product value:
 [ 3  9 15  2]
```

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.

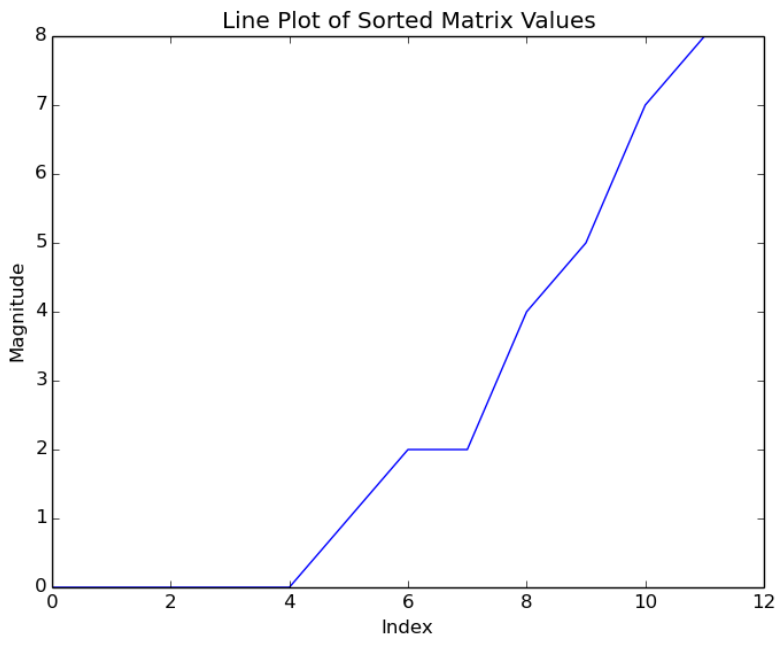
```
# ===== Problem 2e =====
# Without using a loop, multiply each row of M element-wise by a.
# Hint: The function repmat() may come in handy.

newM = None

# BEGIN YOUR CODE HERE
ar = np.tile(a[np.newaxis, :], (M.shape[0], 1))
newM = M*ar
print('Element-wise product value:\n', newM)
# END YOUR CODE HERE
```

```
Element-wise product value:
 [[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 and plot them in your report.

```
    # ===== Problem 2f =====
    # Without using a loop, sort all of the values
    # of M in increasing order and plot them.
    # Note we want you to use newM from e.

    # BEGIN YOUR CODE HERE
    s = np.sort(newM.reshape(-1))
    print('Sorted values:\n', s)
    plt.plot(range(len(s)), s)
    plt.title('Line Plot of Sorted Matrix Values')
    plt.xlabel('Index')
    plt.ylabel('Magnitude')
    plt.savefig('p2f.png')
    # END YOUR CODE HERE
```

```
Sorted values:
 [0 0 0 0 0 1 2 2 4 5 7 8]
```



3 Basic Image Manipulation

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

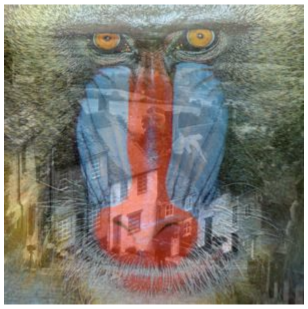
```
# ===== Problem 3a =====
# Read in the images, image1.jpg and image2.jpg, as color images.

img1, img2 = None, None

# BEGIN YOUR CODE HERE
path1 = 'image1.jpg'
path2 = 'image2.jpg'
img1 = misc.imread(path1)
img2 = misc.imread(path2)
# END YOUR CODE HERE
```

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

```
# ===== 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 = img1.astype(np.double)
img1 -= img1.min()
img1 /= img1.max()
img2 = img2.astype(np.double)
img2 -= img2.min()
img2 /= img2.max()
# END YOUR CODE HERE
```

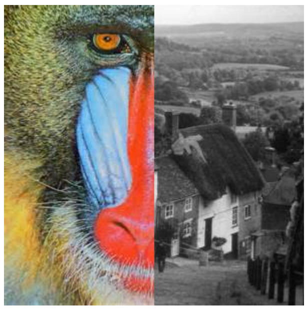
c) Add the images together and re-normalize them to have minimum value 0 and maximum value 1. Display this image in your report.

```
# ===== Problem 3c =====
# Add the images together and re-normalize them
# to have minimum value 0 and maximum value 1.
# Display this image.

# BEGIN YOUR CODE HERE
img3 = img1 + img2
img3 -= img3.min()
img3 /= img3.max()
misc.imsave('p3c.png', img3)
# END YOUR CODE HERE
```



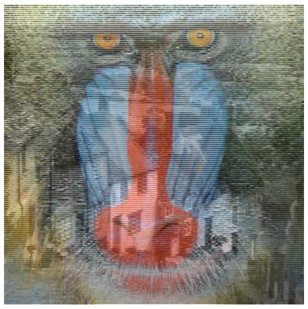
d) Create a new image such that the left half of the image is the left half of image1 and the right half of the image is the right half of image2. Display this image in your report.

```
    # ===== Problem 3d =====
    # 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.

    newImage1 = None

    # BEGIN YOUR CODE HERE
    newImage1 = np.concatenate([img1[:, :img1.shape[1]//2, :], img2[:, 
img2.shape[1]//2:, :]], axis=1)
    misc.imsave('p3d.png', newImage1)
    # END YOUR CODE HERE
```



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. Display this image in your report.

```
    # ===== 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
    row_list = []
    for i in range(img1.shape[0]):
        if i%2 == 0:
            row_list.append(img2[i, :, :][np.newaxis, :, :])
        else:
            row_list.append(img1[i, :, :][np.newaxis, :, :])
    newImage2 = np.concatenate(row_list, axis=0)
    misc.imsave('p3e.png', newImage2)
    # END YOUR CODE HERE
```



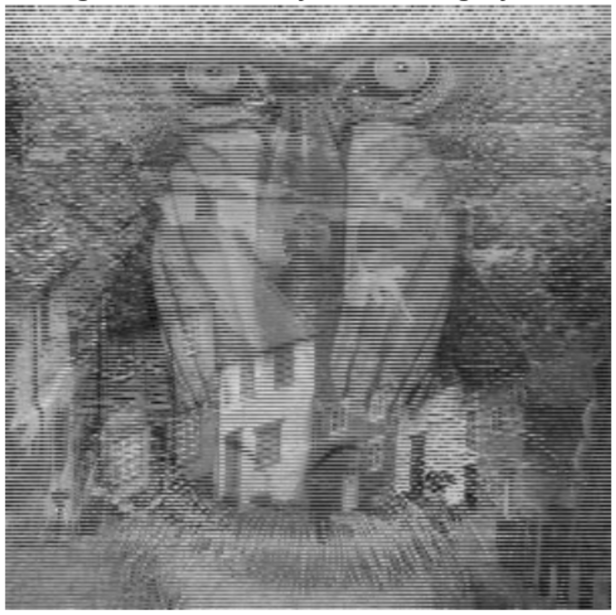
f) Accomplish the same task as part e without using a for-loop.

```
    # ===== Problem 3f =====
    # Accomplish the same task as part e without using a for-loop.
    # The functions reshape and repmat may be helpful here.

    newImage3 = None

    # BEGIN YOUR CODE HERE
    odd_mask_base = np.concatenate([
            np.zeros((1, img1.shape[1], img1.shape[2])),
            np.ones((1, img1.shape[1], img1.shape[2]))
        ], axis=0)
    odd_mask = np.tile(odd_mask_base, (img1.shape[0]//2, 1, 1))
    even_mask_base = np.concatenate([
            np.ones((1, img2.shape[1], img2.shape[2])),
            np.zeros((1, img2.shape[1], img2.shape[2]))
        ], axis=0)
    even_mask = np.tile(even_mask_base, (img2.shape[0]//2, 1, 1))
    newImage3 = odd_mask*img1 + even_mask*img2
    misc.imsave('p3f.png', newImage3)
    # END YOUR CODE HERE
```

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

```
    # ===== Problem 3g =====
    # Convert the result from part f to a grayscale image.
    # Display the grayscale image with a title.

    # BEGIN YOUR CODE HERE
    grayImage3 = misc.imread('p3f.png', flatten=True)
    plt.axis('off')
    plt.imshow(grayImage3, cmap='gray')
    plt.title('Images Interleaved by Pixel Row (grayscale)')
    plt.savefig('p3g.png')
    # END YOUR CODE HERE
```

Images Interleaved by Pixel Row (grayscale)



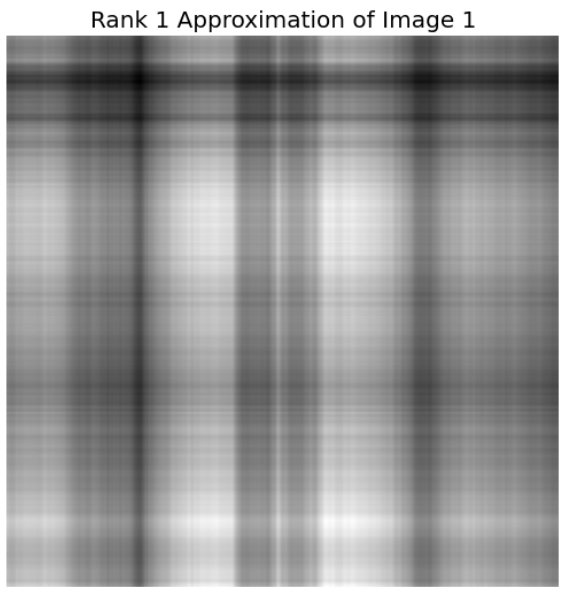
4 Singular Value Decomposition

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

```
    # ===== Problem 4a =====
    # Read in image1 as a grayscale image. Take the singular value
    # decomposition of the image.

    img1 = None

    # BEGIN YOUR CODE HERE
    img1 = misc.imread('image1.jpg', flatten=True)
    u, s, v = np.linalg.svd(img1)
    # END YOUR CODE HERE
```

b) Save and display the best rank 1 approximation of the (grayscale) image1 in your report

```
# ===== Problem 4b =====
# Save and display the best rank 1 approximation
# of the (grayscale) image1.

rank1approx = None

# BEGIN YOUR CODE HERE
rank = 1
rank1approx = np.zeros((u.shape[0], v.shape[0]))
for i in range(rank):
    rank1approx += np.outer(u.T[i]*s[i], v[i])
plt.axis('off')
plt.imshow(rank1approx, cmap='gray')
plt.title('Rank 1 Approximation of Image 1')
plt.savefig('p4b.png')
# END YOUR CODE HERE
```



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

```
# ===== Problem 4c =====
# Save and display the best rank 20 approximation
# of the (grayscale) image1.

rank20approx = None

# BEGIN YOUR CODE HERE
rank = 20
rank20approx = np.zeros((u.shape[0], v.shape[0]))
for i in range(rank):
    rank20approx += np.outer(u.T[i]*s[i], v[i])
plt.axis('off')
plt.imshow(rank20approx, cmap='gray')
plt.title('Rank 20 Approximation of Image 1')
plt.savefig('p4c.png')
# END YOUR CODE HERE
```