CS 189: Homework 7

Vinay Maruri

May 8, 2019

1 Regularized and Kernel k-Means

(a) What is the minimum value of the objective when k = n (the number of clusters equals the number of sample points)?
Solution: Please see figure 1 on the next page.

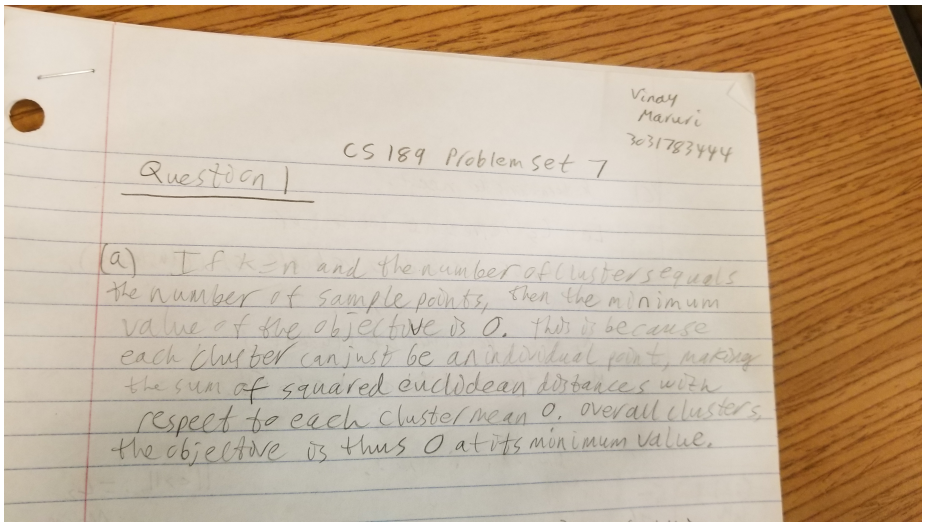

Figure 1: Solution for question 1a.

(b) Show that the optimum of the regularized k-means objective function is $\mu_i = \frac{1}{|C_i|+\lambda} \sum_{x_j \in C_i} x_j$.
Solution: Please see figure 2 on the next page.

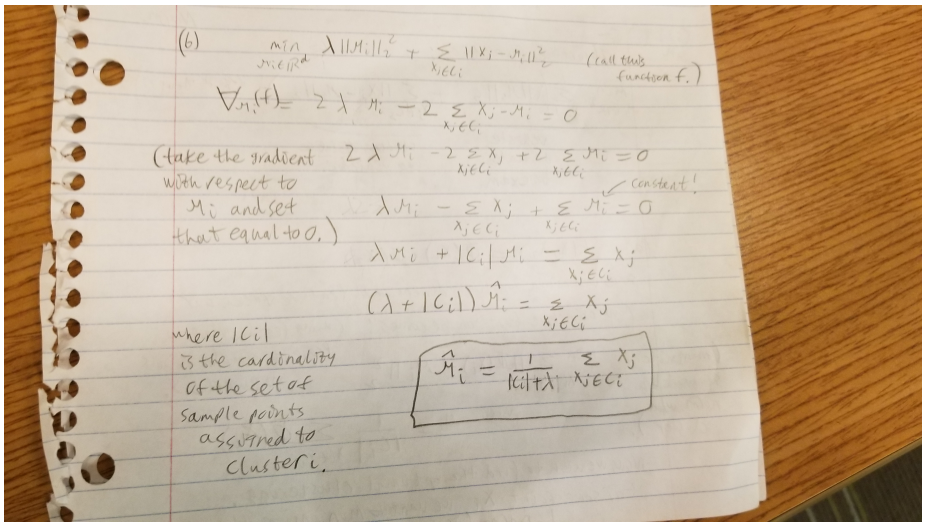

Figure 2: Solution for question 1b.

(c) Write down an appropriate objective function to minimize the total distance that the students and vehicles need to travel.
Solution: Please see figure 3 on the next page.

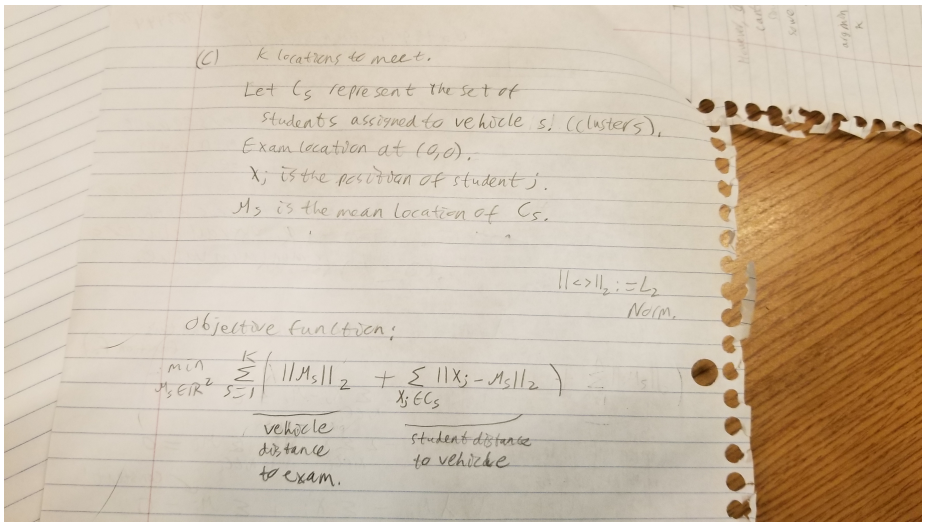

Figure 3: Solution for question 1c.

(d) Derive the underlined portion of this algorithm, and show your work in deriving it.
Solution: Please see figures 4 and 5 on the next 2 pages.

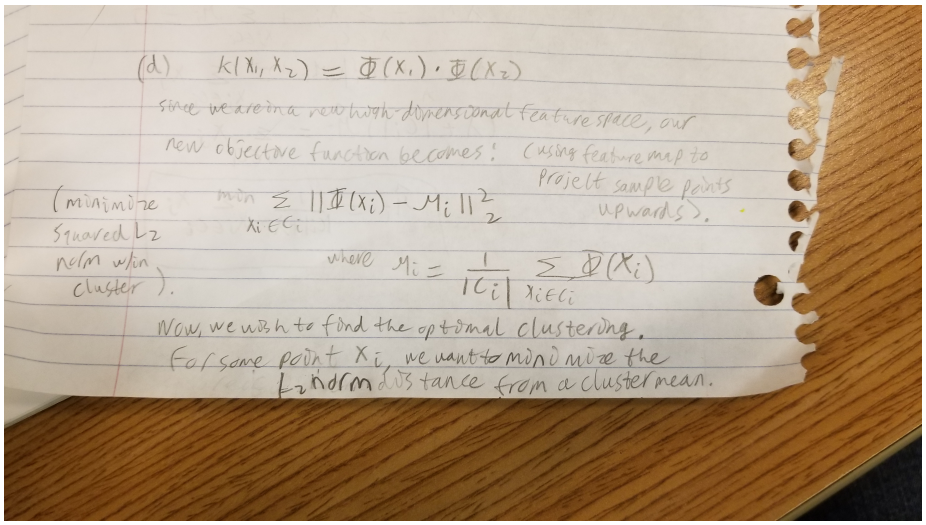

Figure 4: Solution for question 1d.

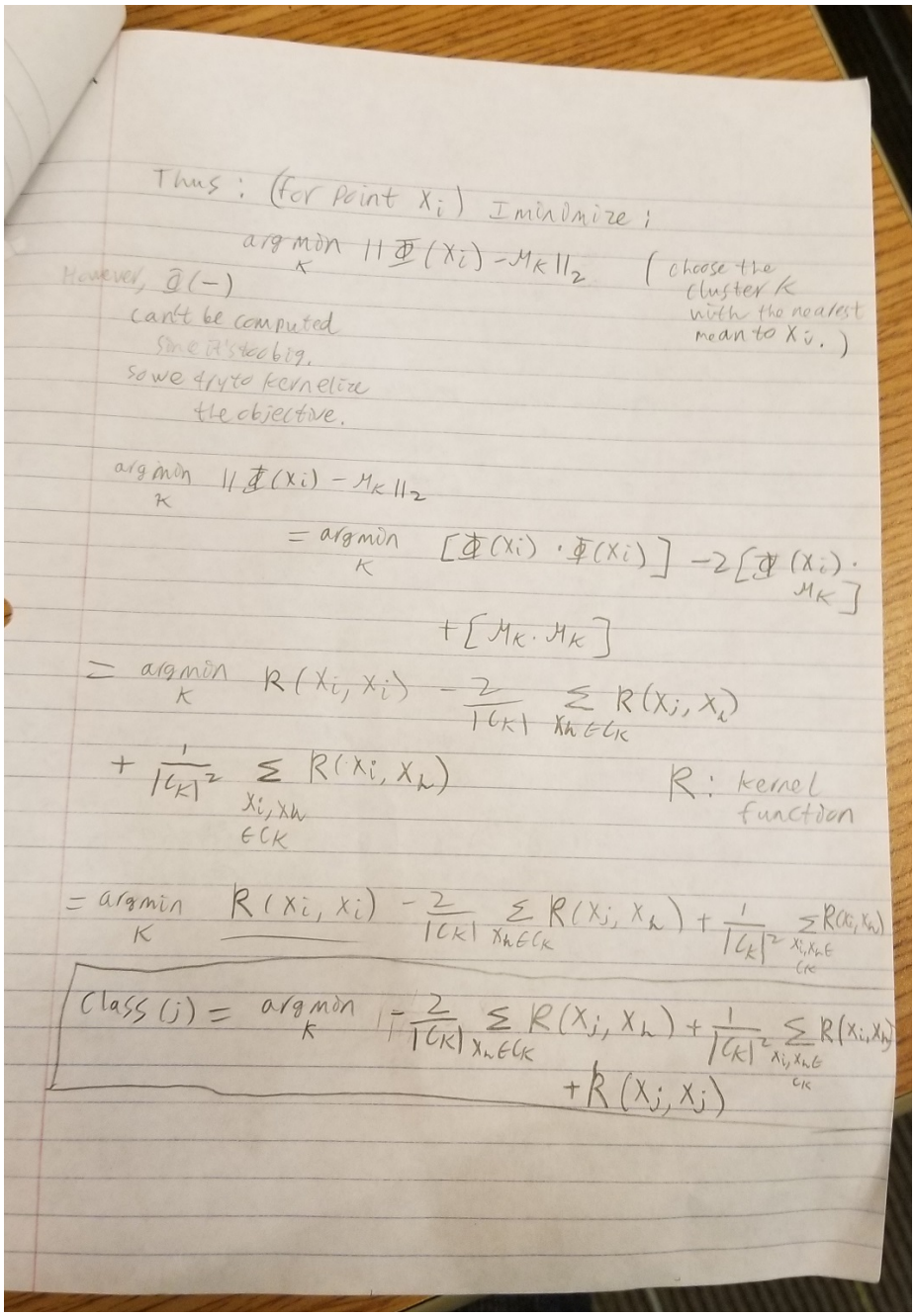

Figure 5: Solution for question 1d, continued.

(e) The expression you derived may have unnecessary terms or redundant kernel computations. Explain how to eliminate them; that is, how to perform the computation quickly without doing irrelevant computations or redoing computations already done.

Solution: Please see figure 6 on the next page.

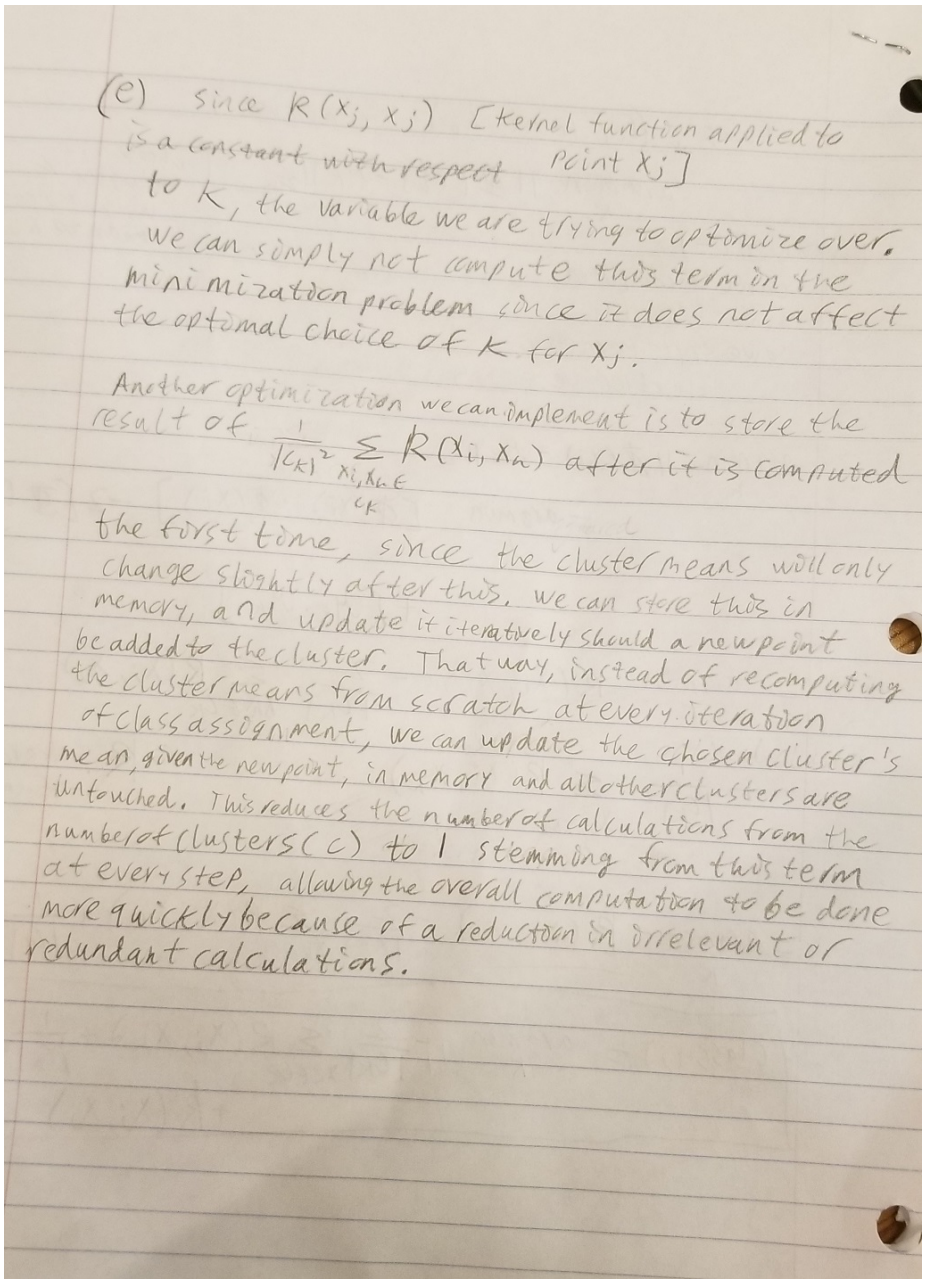

Figure 6: Solution for question 1e.

2 Low-Rank Approximation

(a) Using the image low-rank data/face.jpg, perform a rank-5, rank-20, and rank-100 approximation on the image. Show both the original image as well as

the low-rank images you obtain in your report.
Solution:

```
def low_rank_approximation(X, rank):
    # YOUR CODE GOES HERE
    #step 1:
    #compute the SVD of the image data matrix
    #step 2: take the first rank singular values
    #step 3: generate a diagonal matrix of singular values, and then matrix mult
    #to generate the approximation
    #step 4: generate and show the new image.
    u, s, vh = np.linalg.svd(X, full_matrices = False)
    s[rank:] = 0
    s = np.diag(s)
    return u @ s @ vh

face = imread("./data/face.jpg")
plt.imshow(face, cmap = 'gray')
rank5face = low_rank_approximation(face, 5)
plt.imshow(rank5face, cmap = 'gray')
rank20face = low_rank_approximation(face, 20)
plt.imshow(rank20face, cmap = 'gray')
rank100face = low_rank_approximation(face, 100)
plt.imshow(rank100face, cmap = 'gray')
```

```
Out[4]: <matplotlib.image.AxesImage at 0x2d3db34b860>
```

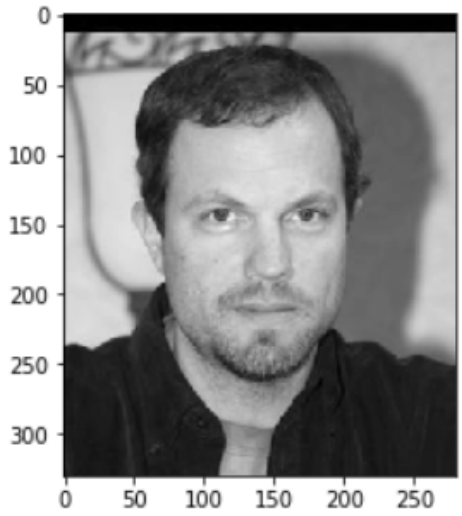

Figure 7: The original face.jpg image.

```
Out[6]: <matplotlib.image.AxesImage at 0x2d3de830208>
```

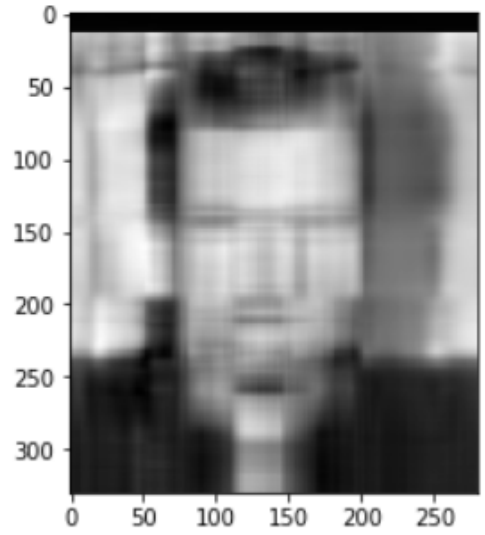

Figure 8: The rank-5 approximation of face.jpg.

```
Out[7]: <matplotlib.image.AxesImage at 0x2d3de894908>
```

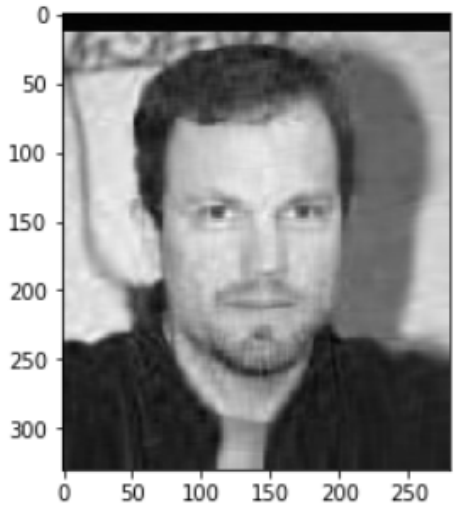

Figure 9: The rank-20 approximation of face.jpg.

```
Out[8]: <matplotlib.image.AxesImage at 0x2d3de8f1a58>
```

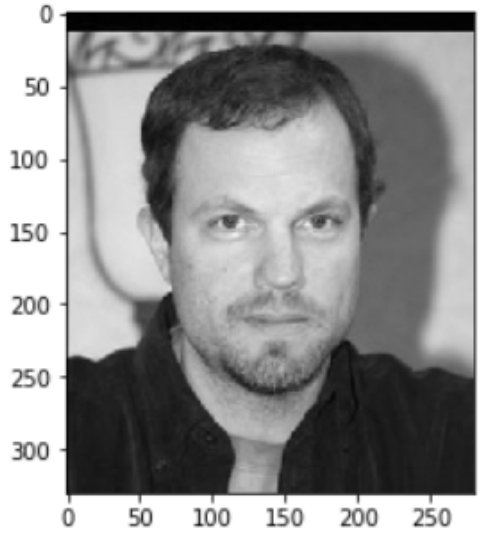

Figure 10: The rank-100 approximation of face.jpg.

(b) Now perform the same rank-5, rank-20, and rank-100 approximation on low-rank data/sky.jpg. Show both the original image as well as the low-rank images you obtain in your report.

Solution:

```
sky = imread("./data/sky.jpg")
plt.imshow(sky, cmap = 'gray')
rank5sky = low_rank_approximation(sky, 5)
plt.imshow(rank5sky, cmap = 'gray')
rank20sky = low_rank_approximation(sky, 20)
plt.imshow(rank20sky, cmap = 'gray')
rank100sky = low_rank_approximation(sky, 100)
plt.imshow(rank100sky, cmap = 'gray')
```

```
Out[11]: <matplotlib.image.AxesImage at 0x2d3de94ecf8>
```

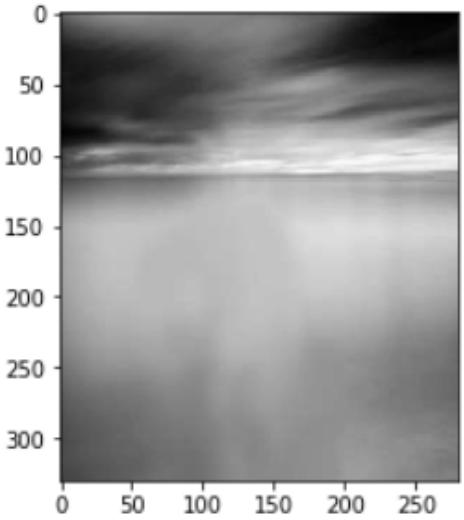

Figure 11: The original sky.jpg image.

```
Out[12]: <matplotlib.image.AxesImage at 0x2d3de9afe48>
```

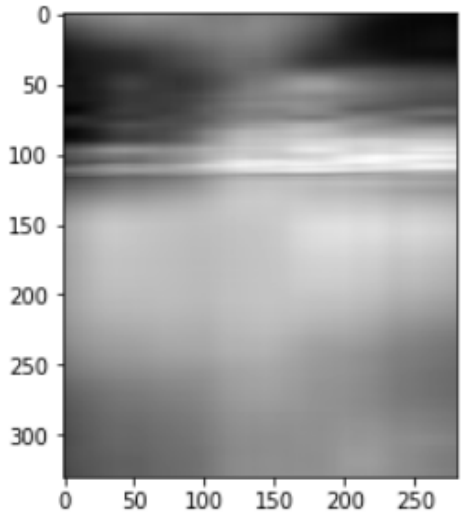

Figure 12: The rank-5 approximation of sky.jpg.

```
Out[13]: <matplotlib.image.AxesImage at 0x2d3dea0df60>
```

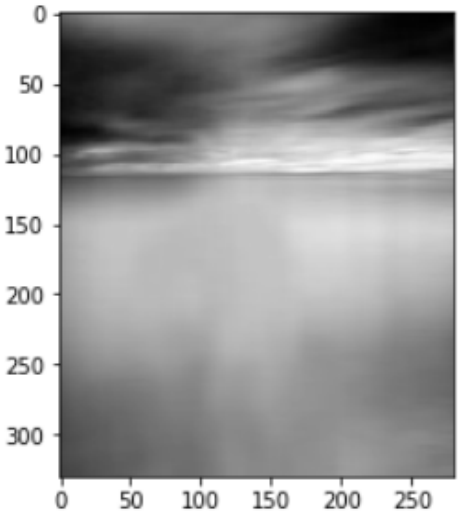

Figure 13: The rank-20 approximation of sky.jpg.

```
Out[14]: <matplotlib.image.AxesImage at 0x2d3dea740f0>
```

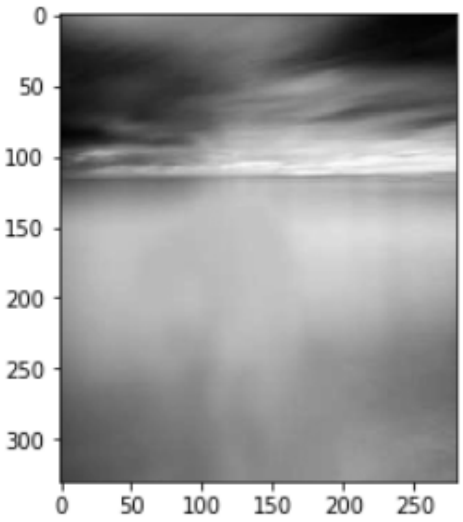

Figure 14: The rank-100 approximation of sky.jpg.

(c) In one plot, plot the Mean Squared Error (MSE) between the rank-k approximation and the original image for both low-rank data/face.jpg and low-rank data/sky.jpg, for k ranging from 1 to 100. Be sure to label each curve in the plot.

Solution: I am including 2 plots with my submission for part (c). The first uses a linear scale for the Mean Squared Error values on the y-axis. I was unsatisfied with this plot as it did not adequately reflect the differences between the two image types. To tease out the difference, I created a second plot where I log scale the Mean Squared Error values on the y-axis. This does a better job of illustrating the differences between the two image types.

```
def mse(img1, img2):
    # YOUR CODE GOES HERE
    #NOTE: This function actually uses the Mean Squared Error, not the spec's de
    #Clarification here: (https://piazza.com/class/jpdq1epqrr3231?cid=695)
    error = 0
    for i in range(img1.shape[0]):
        for j in range(img1.shape[1]):
            error += (img1[i][j] - img2[i][j])**2
    return error/(img1.shape[0] * img1.shape[1])

#I wrote this helper function to compute all sky and face errors for each rank-k
def plot_errors():
```

```
    face_err = []
    k = 1
    for k in range(100):
        approx = low_rank_approximation(face, k)
        face_err.append(mse(face, approx))
    sky_err = []
    z = 1
    for z in range(100):
        approx = low_rank_approximation(sky, z)
        sky_err.append(mse(sky, approx))
    return face_err, sky_err
f, s = plot_errors()
x_vals = np.arange(100)
plt.plot(x_vals, f, label = 'face')
#plotting face errors
plt.plot(x_vals, s, label = 'sky')
#plotting sky errors
plt.legend()
plt.ylabel("Mean Squared Error")
plt.xlabel("Rank of Approximated Image")
plt.title("Mean Squared Error between original image and rank k approximation")
#linear y-scale
plt.plot(x_vals, f, label = 'face')
#plotting face errors
plt.plot(x_vals, s, label = 'sky')
#plotting sky errors
plt.legend()
plt.yscale(value = "log")
#log y-scale
plt.ylabel("Mean Squared Error (logarithmic scale)")
plt.xlabel("Rank of Approximated Image")
plt.title("Mean Squared Error (logarithmic) between original image and rank k ap
```

```
Out[41]: Text(0.5, 1.0, 'Mean Squared Error between original image and rank k approximation')
```

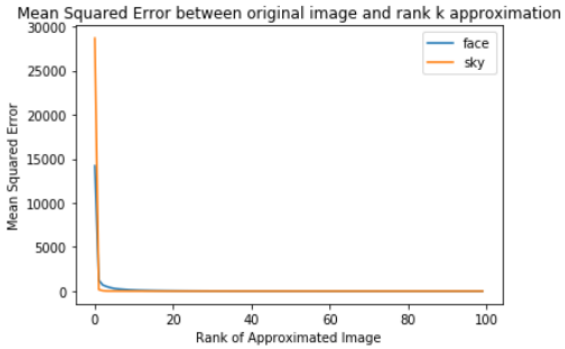

Figure 15: Mean Squared Error between original image and rank k approximation.

```
Out[42]: Text(0.5, 1.0, 'Mean Squared Error (logarithmic) between original image and rank k approximation')
```

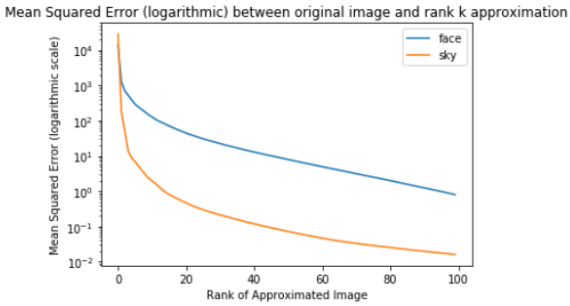

Figure 16: Mean Squared Error (logarithmic) between original image and rank k approximation.

(d) Find the lowest-rank approximation for which you begin to have a hard time differentiating the original and the approximated images. Compare your results for the face and the sky image. What are the possible reasons for the difference?

Solution:

```
rank50face = low_rank_approximation(face, 50)
plt.imshow(rank50face, cmap = 'gray')
rank25sky = low_rank_approximation(sky, 25)
plt.imshow(rank30sky, cmap = 'gray')
```

Subjectively, I began to have a hard time differentiating between the original and approximated images at rank-50 for the face image and at rank-25 for the sky image. The sky image is able to be approximated at a lower rank than the face image. Some possible reasons for this difference are the different levels

of detail present in the two images, and the difference in color/color channels between the two images. I suspect that there is more detail and information in the face image than in the sky image, thus making the face image harder to compress, which is why I found that in general, higher-rank approximations were necessary for the face image to preserve the image's information (in matrix form) and look more like the original image. There are many more distinct lines, shapes and details on the man's face, different skin/color tones, and also the white background wall details are intricate with a drawing and the man's shadow. This in contrast to the sky image which has a lot less detail and distinct lines, less color channels, and a large space of virtually the same color (meaning not much information is there). Thus, the sky image can be compressed more tightly (small rank approximations) than the face image (requiring higher rank approximations).