CS 189 Homework7

Xu Zhihao

August 8, 2019

I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted.

Signature: Zhihao Xu

Contents

1 Regularized and Kernel k-Means

(a) When k=n,

$$\min_{C_1,C_2,\ldots,C_k} \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - u_i\|_2^2 = 0$$

Since all the points' clusters are themselves.

(b) The target cost function is

$$f(\mu_i) = \left(\sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2 \right) + \lambda \|\mu_i\|_2^2$$

Take the derivative,

$$\begin{aligned} \frac{\partial f(\mu_i)}{\partial \mu_i} &= -2 \sum_{x_j \in C_i} (x_j - \mu_i) + 2\lambda \mu_i \\ &= -2 \sum_{x_j \in C_i} x_j + 2|C_i|\mu_i + 2\lambda \mu_i \\ &= -2 \left(\sum_{x_j \in C_i} x_j - (\lambda + |C_i|)\mu_i \right) \\ &= 0 \end{aligned}$$

We can get

$$\mu_i = \frac{1}{\lambda + |C_i|} \sum_{x_j \in C_i} x_j$$

where $|C_i|$ represents the # of points in cluster C_i. Since $f(\mu_i)$ is convex, the optimum can be obtained at $\mu_i = \frac{1}{\lambda+|C_i|} \sum_{x_j \in C_i} x_j$.

(c) Since we need minimize the distance that the students and vehicles need to travel, the objective function is

$$\min_{\mu_i} \sum_{i=1}^{K} \left(\|\mu_i\|_2^2 + \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2 \right)$$

(d) First we need to compute the center of each cluster, we need to solve

$$\min \sum_{x_j \in S_i} \left\| \phi(x_j) - \mu_i \right\|_2^2$$

The result is

$$\mu_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} \phi(x_j)$$

Next we are going to simplify the object function

$$\begin{aligned}
f(i,k) &= \|\phi(x_i) - \mu_k\|_2^2 \\
&= \langle \phi(x_i), \phi(x_i) \rangle - 2 \langle \phi(x_i), \mu_k \rangle + \langle \mu_k, \mu_k \rangle \\
&= \langle \phi(x_i), \phi(x_i) \rangle - \frac{2}{|S_k|} \sum_{x_j \in S_k} \left\langle \phi(x_i), \phi(x_j) \right\rangle + \frac{1}{|S_k|^2} \sum_{x_j, x_l \in S_k} \left\langle \phi(x_j), \phi(x_l) \right\rangle \\
&= \kappa(x_i, x_i) - \frac{2}{|S_k|} \sum_{x_j \in S_k} \kappa(x_i, x_j) + \frac{1}{|S_k|^2} \sum_{x_j, x_l \in S_k} \kappa(x_j, x_l)
\end{aligned}$$

Since our target is to compute

$$\arg\min_k f(i,k)$$

The final result is

$$\text{Set class(j)} = \arg\min_k \kappa(x_i, x_i) - \frac{2}{|S_k|} \sum_{x_j \in S_k} \kappa(x_i, x_j) + \frac{1}{|S_k|^2} \sum_{x_j, x_l \in S_k} \kappa(x_j, x_l)$$

(e) The expression I derived is

$$\text{Set class(j)} = \arg\min_k \kappa(x_i, x_i) - \frac{2}{|S_k|} \sum_{x_j \in S_k} \kappa(x_i, x_j) + \frac{1}{|S_k|^2} \sum_{x_j, x_l \in S_k} \kappa(x_j, x_l)$$

We can notice that the first term $\kappa(x_i, x_i)$ is unnecessary term, which does not influenced by k, we can just eliminate it. And for the rest of the redundant kernel computations, we can compute the kernel matrix first and just access it in the later computation. There is no need to compute them again and again, which can help us perform the computation quickly. The expression can be simplified as

$$\text{Set class(j)} = \arg\min_k -\frac{2}{|S_k|} \sum_{x_j \in S_k} \kappa(x_i, x_j) + \frac{1}{|S_k|^2} \sum_{x_j, x_l \in S_k} \kappa(x_j, x_l)$$

2 Low-Rank Approximation

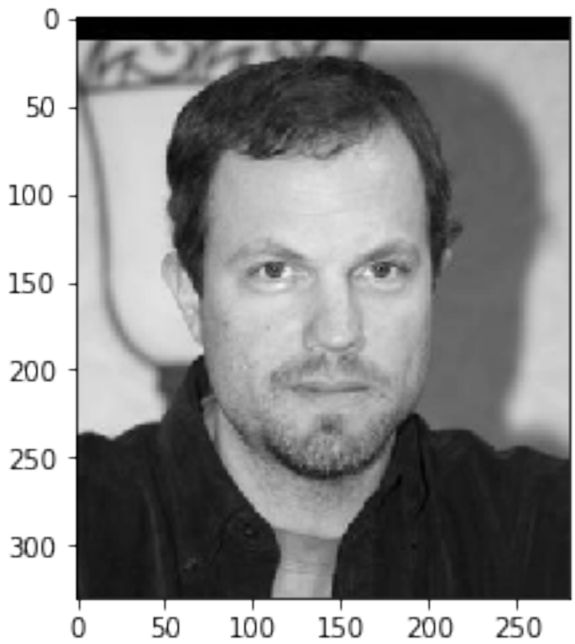
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from imageio import imread

        %matplotlib inline
```

2.1 Part a)

```
In [6]: face = imread("./data/face.jpg")
        plt.set_cmap('gray')
        plt.imshow(face)
```

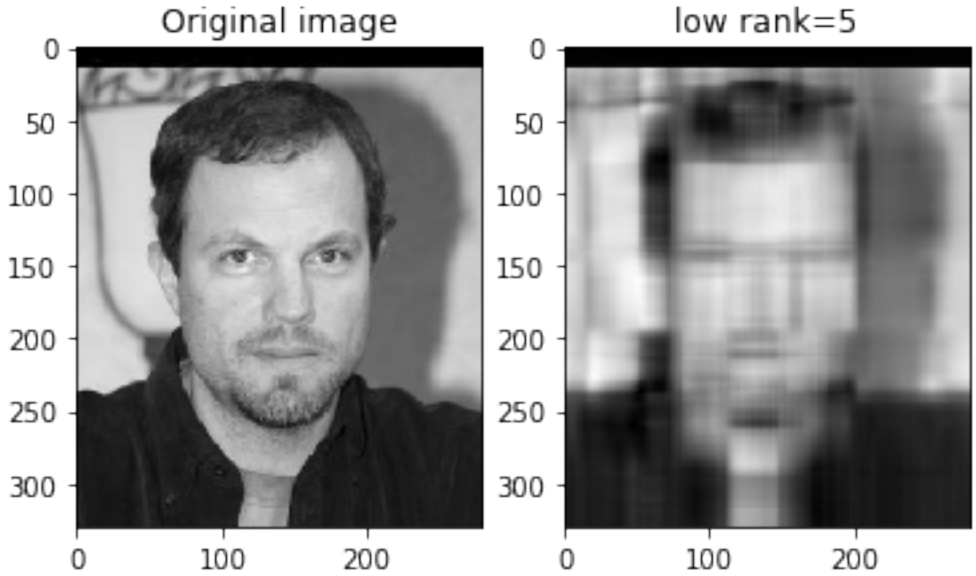
```
Out[6]: <matplotlib.image.AxesImage at 0x120e9dcf8>
```



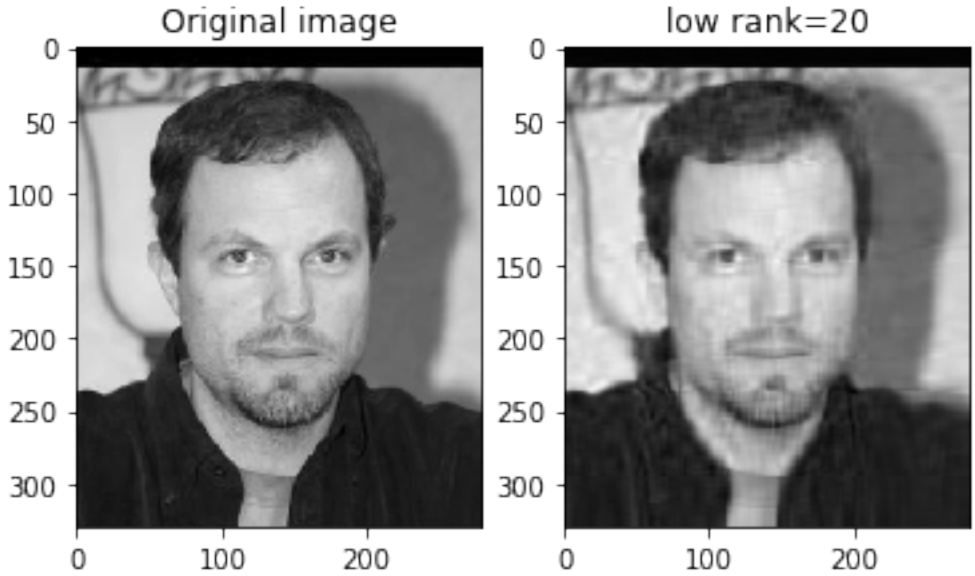

```
In [3]: def low_rank_approximation(image, r):
            u, s, v = np.linalg.svd(image,full_matrices=False)
            s[r:] = 0
            return np.dot(u * s, v)
```

```
In [53]: face_lr5 = low_rank_approximation(face, 5)
         plt.subplot(121)
         plt.imshow(face)
         plt.title('Original image')
```

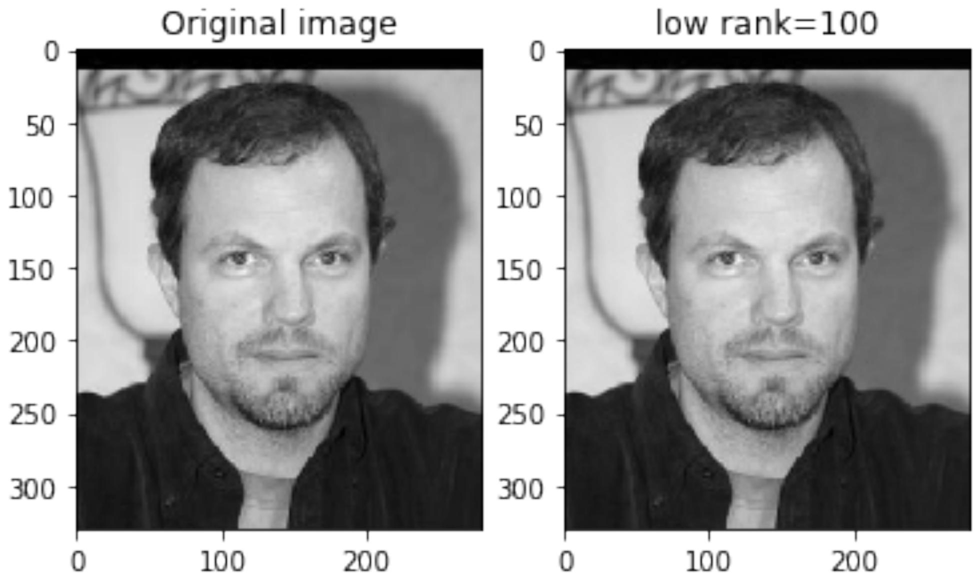
```
plt.subplot(122)
plt.imshow(face_lr5)
plt.title('low rank=%d' % 5)
plt.show()
```




```
In [54]: face_lr20 = low_rank_approximation(face, 20)
         plt.set_cmap('gray')
         plt.subplot(121)
         plt.imshow(face)
         plt.title('Original image')
         plt.subplot(122)
         plt.imshow(face_lr20)
         plt.title('low rank=%d' % 20)
         plt.show()
```



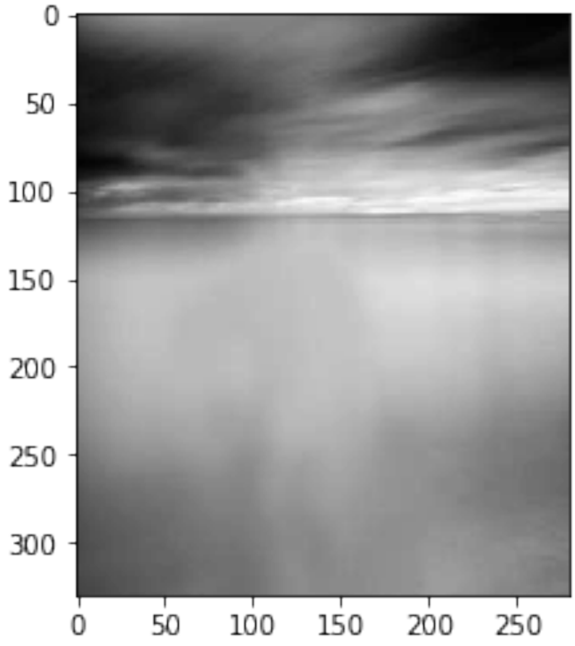

```
In [55]: face_lr100 = low_rank_approximation(face, 100)
         plt.subplot(121)
         plt.imshow(face)
         plt.title('Original image')
         plt.subplot(122)
         plt.imshow(face_lr100)
         plt.title('low rank=%d' % 100)
         plt.show()
```



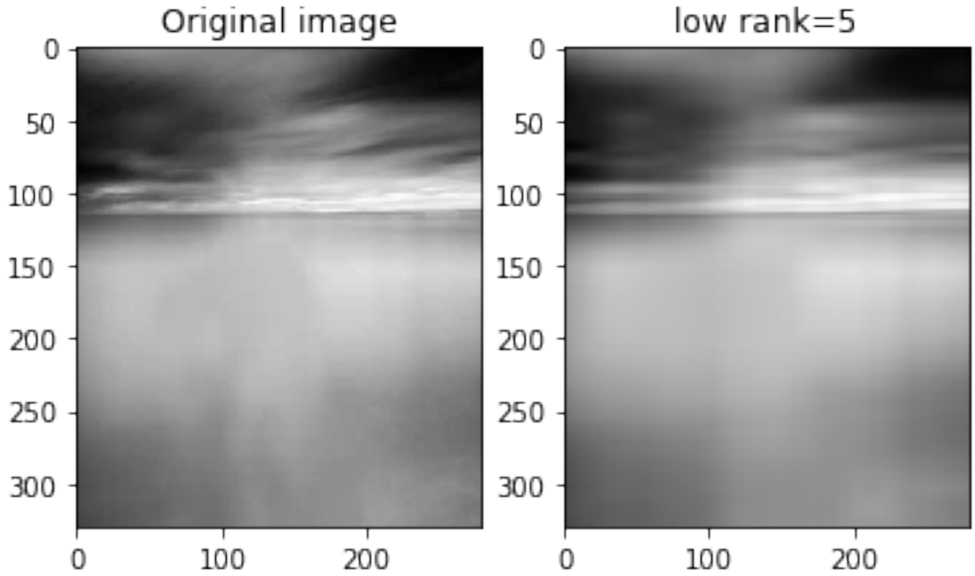
2.2 Part b)

```
In [12]: sky = imread("./data/sky.jpg")
         plt.set_cmap('gray')
         plt.imshow(sky)
```

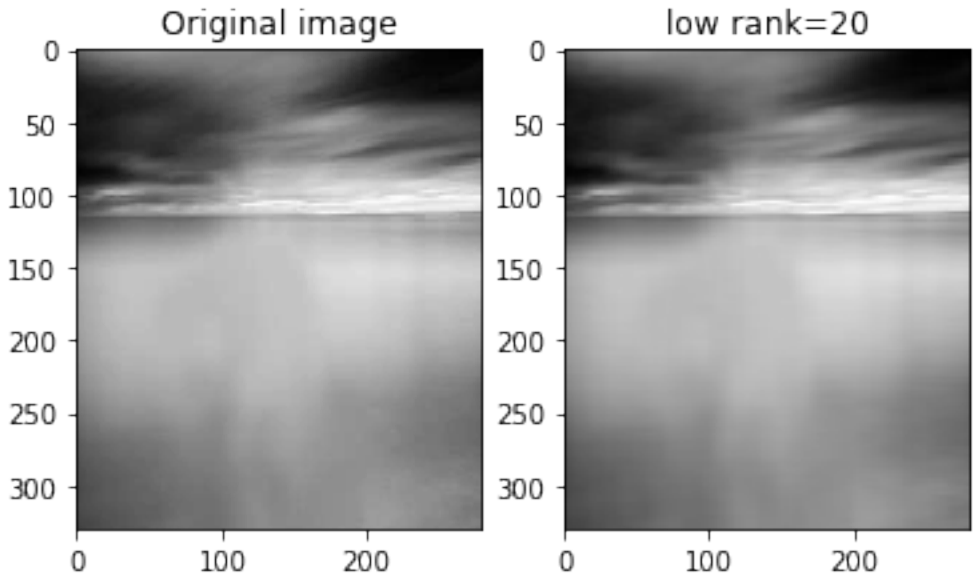
```
Out[12]: <matplotlib.image.AxesImage at 0x121d555c0>
```



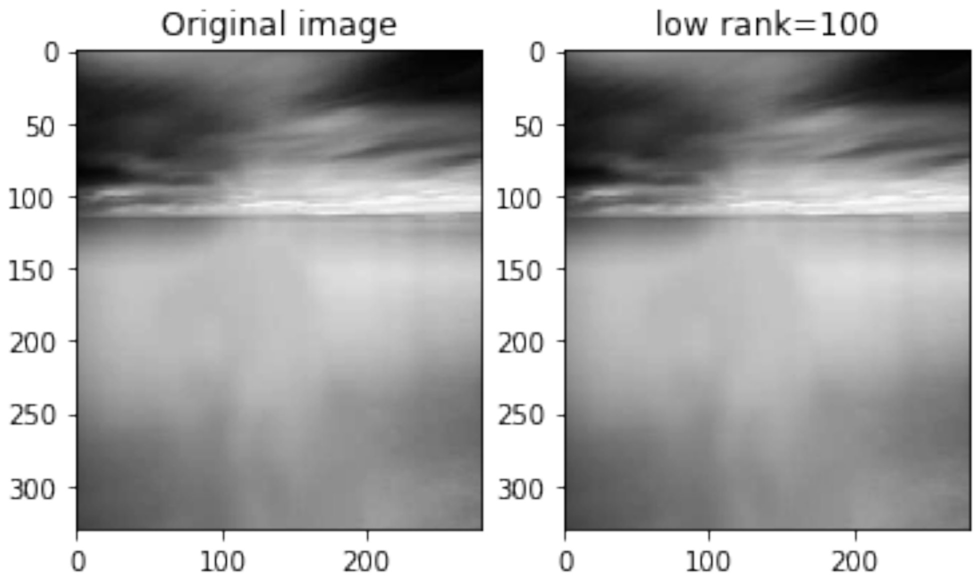

```
In [56]: sky_lr5 = low_rank_approximation(sky, 5)
         plt.subplot(121)
         plt.imshow(sky)
         plt.title('Original image')
         plt.subplot(122)
         plt.imshow(sky_lr5)
         plt.title('low rank=%d' % 5)
         plt.show()
```



In [57]:
```
sky_lr20 = low_rank_approximation(sky, 20)
plt.subplot(121)
plt.imshow(sky)
plt.title('Original image')
plt.subplot(122)
plt.imshow(sky_lr20)
plt.title('low rank=%d' % 20)
plt.show()
```



```
In [58]: sky_lr100 = low_rank_approximation(sky, 100)
         plt.subplot(121)
         plt.imshow(sky)
         plt.title('Original image')
         plt.subplot(122)
         plt.imshow(sky_lr100)
         plt.title('low rank=%d' % 100)
         plt.show()
```

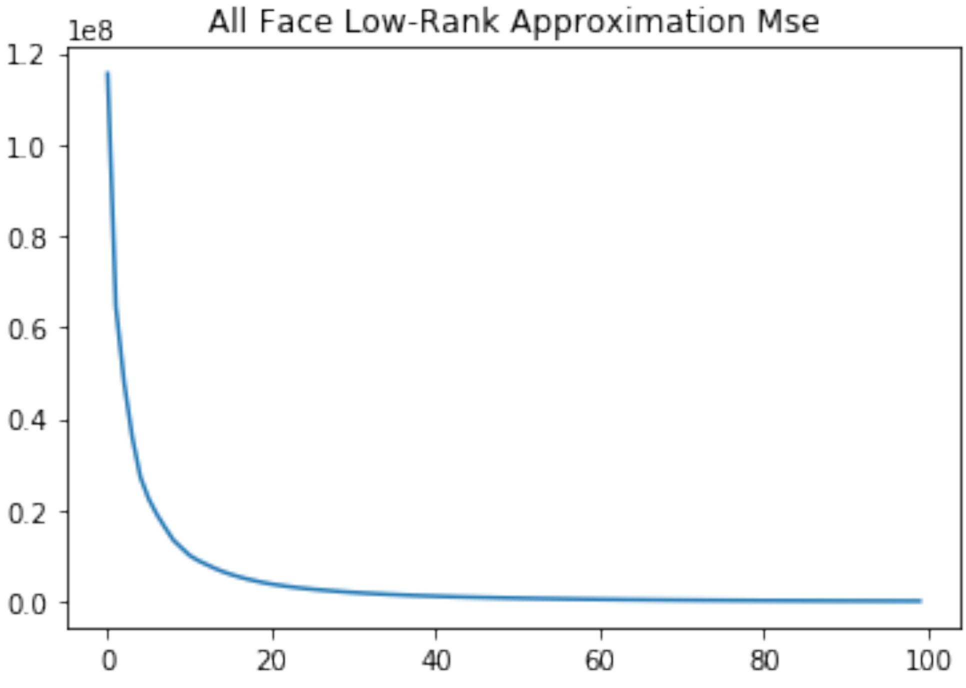


2.3 Part c)

```
In [26]: def mse(img1, img2):
             return np.sum(np.square(img1-img2))
```

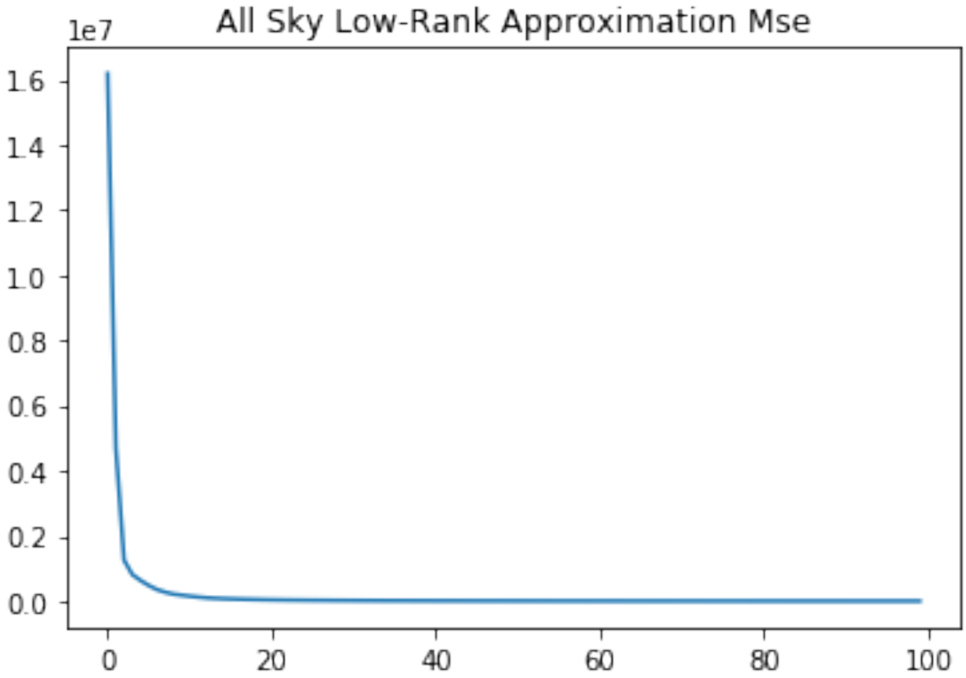
```
In [32]: allFaceMSE = []
         for r in range(100):
             img_lr = low_rank_approximation(face, r+1)
             allFaceMSE.append(mse(img_lr,face))
         plt.title("All Face Low-Rank Approximation Mse")
         plt.plot(allFaceMSE)
```

```
Out[32]: [<matplotlib.lines.Line2D at 0x121a816d8>]
```



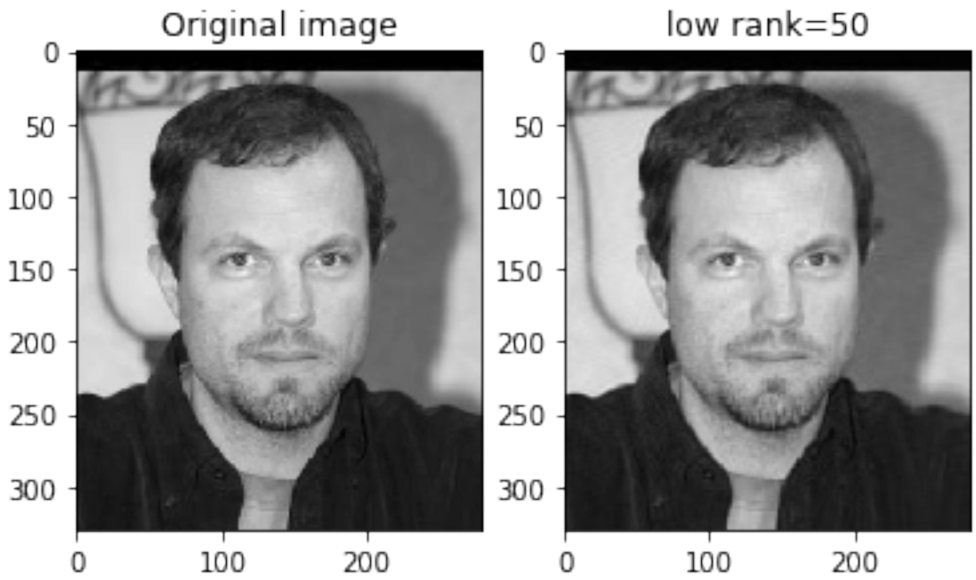

```
In [33]: allSkyMSE = []
         for r in range(100):
             img_lr = low_rank_approximation(sky, r+1)
             allSkyMSE.append(mse(img_lr,sky))
         plt.title("All Sky Low-Rank Approximation Mse")
         plt.plot(allSkyMSE)
```

```
Out[33]: [<matplotlib.lines.Line2D at 0x123cbc9e8>]
```

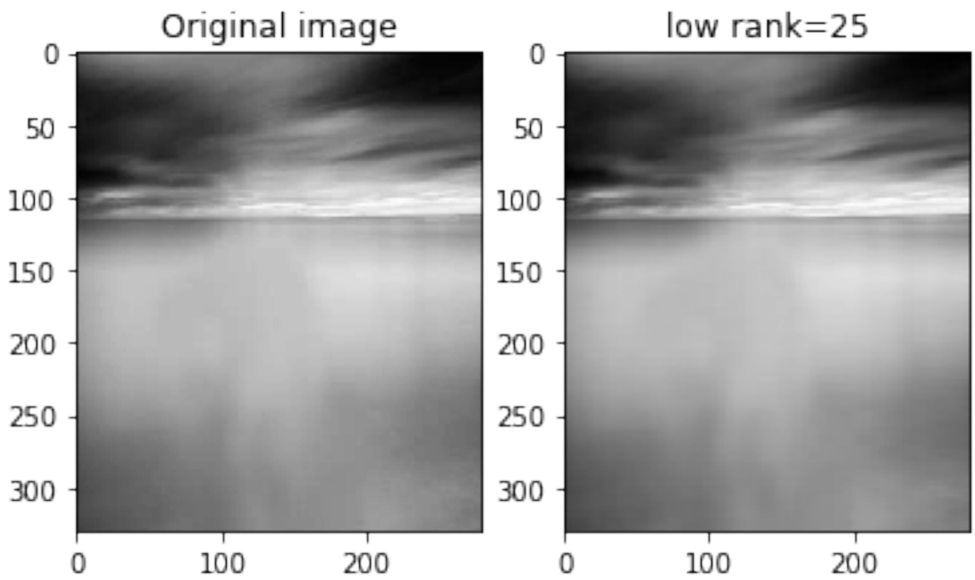

All Sky Low-Rank Approximation Mse
1e7
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

2.4 Part d)

In [48]:
```
rank = 50
face_lr = low_rank_approximation(face, rank)
plt.subplot(121)
plt.imshow(face)
plt.title('Original image')
plt.subplot(122)
plt.imshow(face_lr)
plt.title('low rank=%d' % rank)
plt.show()
```



In [52]:
```
rank = 25
sky_lr = low_rank_approximation(sky, rank)
plt.subplot(121)
plt.imshow(sky)
plt.title('Original image')
plt.subplot(122)
plt.imshow(sky_lr)
plt.title('low rank=%d' % rank)
plt.show()
```



We can see that for the face figure, when rank≈50, we began to have a hard time differentiating the original and the approximated images. For sky figure, rank≈25. The reason why the face figure needs higher rank might be face figure has more detailed information like nose, eye, mouth, etc. But the information in sky figure is much more rough.