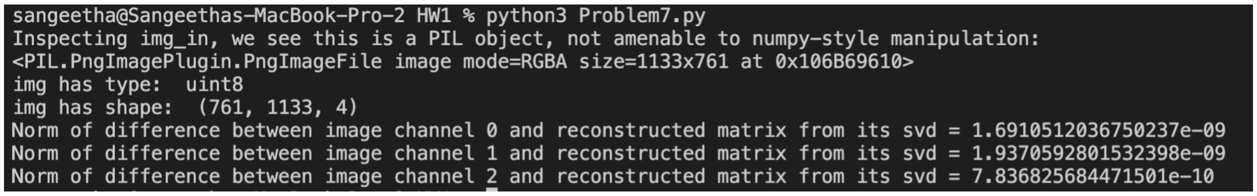
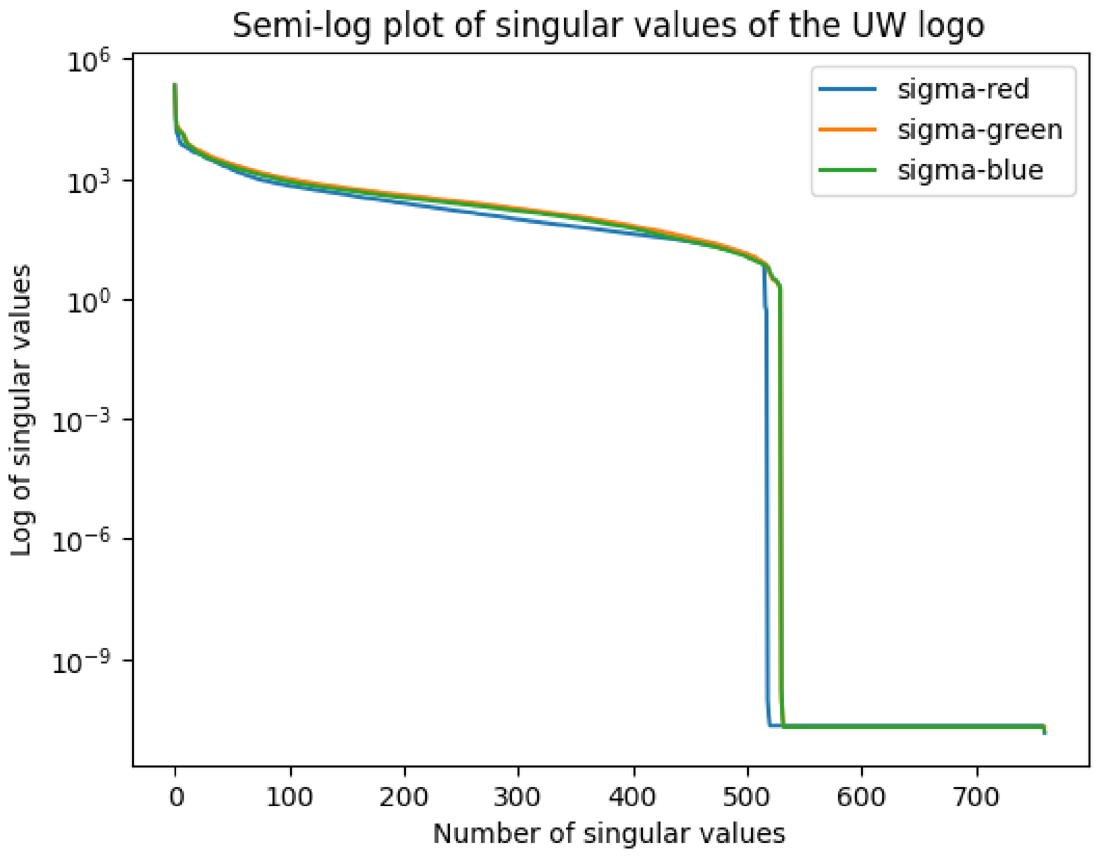
Problem 7

a. Use numpy SVD to decompose each of the 3 channels (R, G, B) and reconstruct the matrix from the SVD and check the error.



```
sangeetha@Sangeethas-MacBook-Pro-2 HW1 % python3 Problem7.py
Inspecting img_in, we see this is a PIL object, not amenable to numpy-style manipulation:
<PIL.PngImagePlugin.PngImageFile image mode=RGBA size=1133x761 at 0x106B69610>
img has type:  uint8
img has shape:  (761, 1133, 4)
Norm of difference between image channel 0 and reconstructed matrix from its svd = 1.6910512036750237e-09
Norm of difference between image channel 1 and reconstructed matrix from its svd = 1.9370592801532398e-09
Norm of difference between image channel 2 and reconstructed matrix from its svd = 7.836825684471501e-10
```

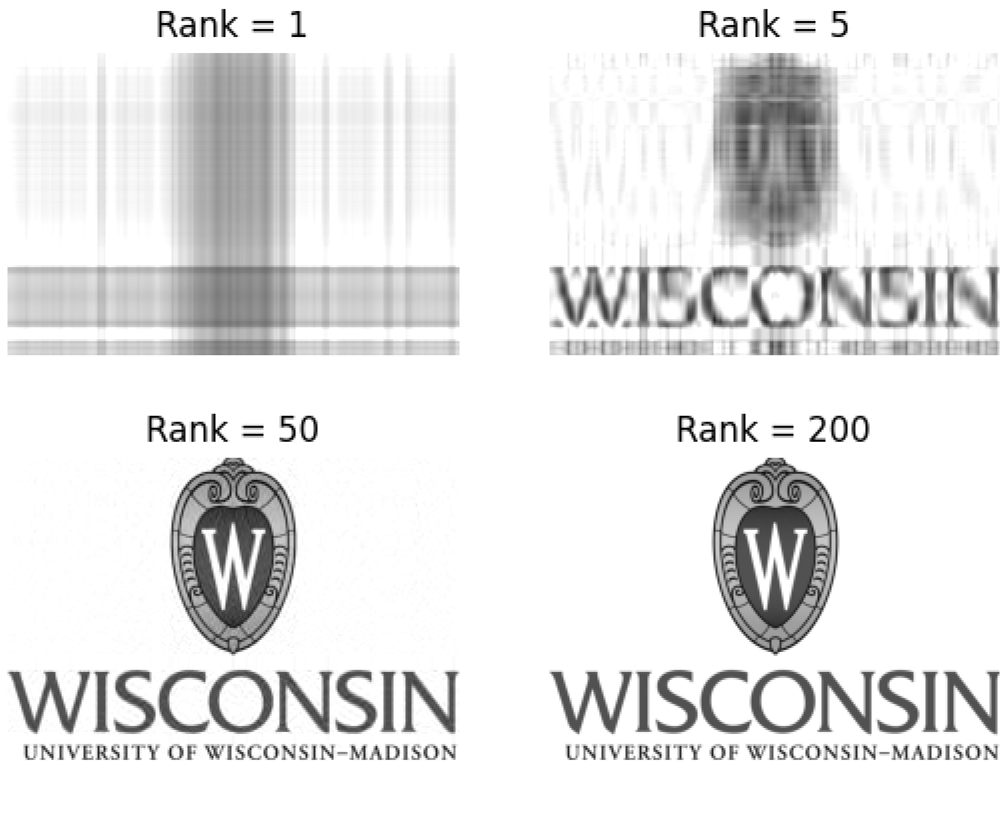
Semilog plot of singular values



b. Convert the UWlogo to grayscale and plot it

W
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Plot the rank - 1, 5, 50, 200 approximations of the SVD



What is the memory cost if you store only the singular values and their corresponding left and right singular vectors?

If M is the grayscale image:
Dimension of M : 761 x 1133
Dimension of the U, Sigma, V matrices (Reduced SVD) = 761 x 761, 761 x 1, 1133 x 761
Dimension of the U, Sigma, V matrices for the rank-k approximation = 761 x k, k x 1, 1133 x k

For a rank-k approximation, we store k singular values, k U vectors of size (761) and k V vectors of size (1133). Each scalar entry in the vectors requires 8 bytes (1 double = 8B).
So the total memory cost for a rank-k approximation is = 8 * (k + 761*k + 1133*k) B.

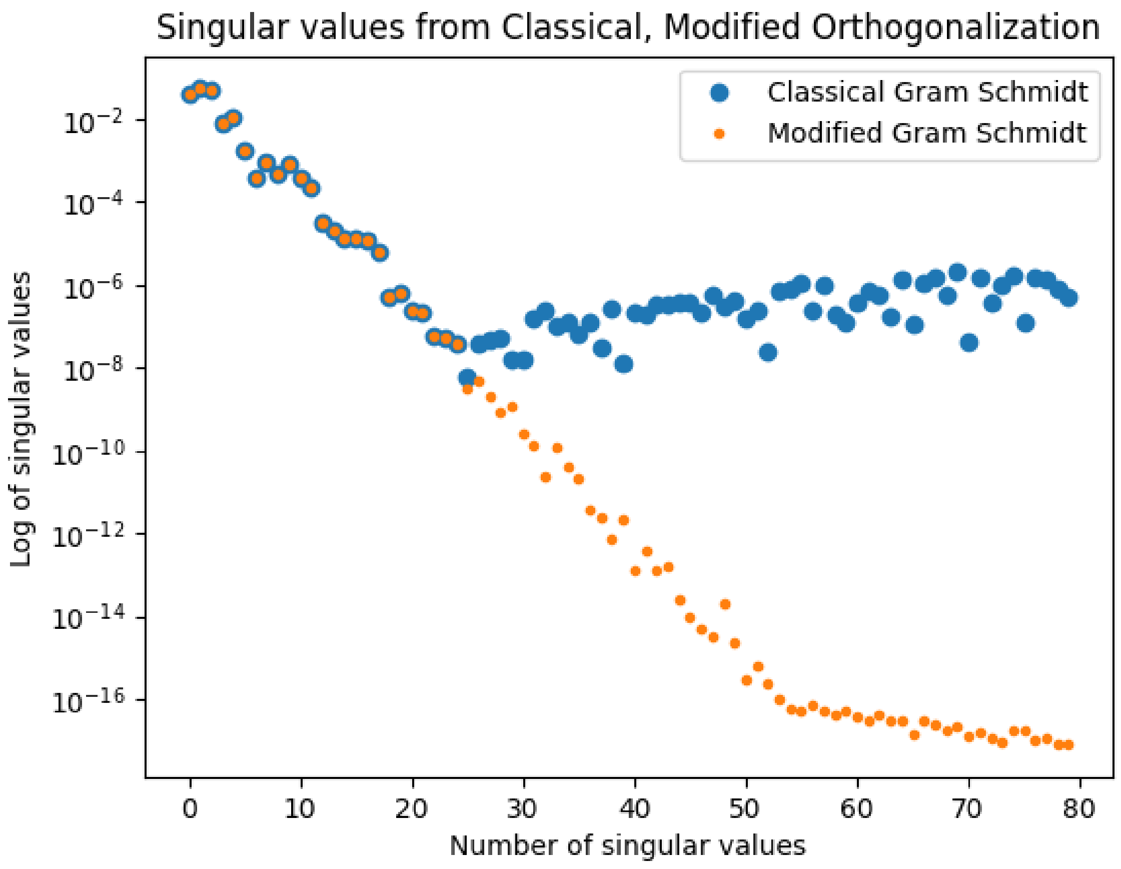
Rank k	Memory Cost
k = 1	8 * (1 + 761 + 1133) = 15160 B
k = 5	8 * (5 + 761 * 5 + 1133 * 5) = 75800 B
k = 50	8 * (50 + 761 * 50 + 1133 * 50) = 758000 B
k = 200	8 * (200 + 761 * 200 + 1133 * 200) = 3032000 B

Why does lower-rank approximation require more space?
The reasons might be as follows:

1. The lower the rank approximation, the noisier the image. This means that the number of distinct pixel values in the image are too many. So the number of distinct values to be stored in the lossless compression of the image might be more of the lower rank approximation when compared to the higher rank approximations.
2. The library used to save the image need not be storing the non-zero singular values and the corresponding left and right singular vectors. It might be storing a compressed version of the product.

Problem 8

Plot for Experiment 2: Classical vs Modified GS



Code:

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

def classical_gram_schmidt(A):
   m = A.shape[0]
   n = A.shape[1]
   assert(m >= n)
   Q = np.zeros((m, n))
   R = np.zeros((n, n))
   for j in range(n):
       v_j = A[:, j]
       for i in range(j):
           R[i][j] = np.dot(Q[:, i], A[:, j])
           v_j = v_j - R[i][j] * Q[:, i]
       R[j][j] = np.linalg.norm(v_j, 2)
       Q[:, j] = v_j/R[j][j]
   return Q, R

def modified_gram_schmidt(A):
   m = A.shape[0]
   n = A.shape[1]
   assert(m >= n)
   Q = np.zeros((m, n))
   R = np.zeros((n, n))
   V = np.zeros((m, n))
   for i in range(n):
       V[:, i] = np.copy(A[:, i])
   for i in range(n):
       R[i][i] = np.linalg.norm(V[:, i], 2)
       Q[:, i] = V[:, i]/R[i][i]
       for j in range(i+1, n):
           R[i][j] = np.dot(Q[:, i], V[:, j])
           V[:, j] = V[:, j] - R[i][j] * Q[:, i]
   return Q, R

def verify_classical_QR(m, n):
   A = np.random.rand(m, n)
   Q, R = classical_gram_schmidt(A)
   print( "Reconstruction error for A with Classical Gram Schmidt = {}".format(
np.linalg.norm(A - np.matmul(Q, R))))

def verify_modified_QR(m, n):
   A = np.random.rand(m, n)
   Q, R = modified_gram_schmidt(A)
   print( "Reconstruction error for A with Modified Gram Schmidt = {}".format(
np.linalg.norm(A - np.matmul(Q, R))))

def run_experiment2():
   m = 80
   sigma = np.linspace(-1, -80, m)
   sigma = 2**sigma

   A = np.random.rand(m, m)
   U, _ = np.linalg.qr(A)
   A = np.random.rand(m, m)
   V, _ = np.linalg.qr(A)

   A = np.matmul(np.matmul(U, np.diag(sigma)), np.transpose(V))

   _, Rc = classical_gram_schmidt(A)
   _, Rm = modified_gram_schmidt(A)

   plt.plot(np.diagonal(Rc), 'o', label='Classical Gram Schmidt')
   plt.plot(np.diagonal(Rm), '.',label='Modified Gram Schmidt')
   plt.yscale('log')
   plt.legend()
   plt.xlabel('Number of singular values')
   plt.title('Singular values from Classical, Modified Orthogonalization')
   plt.savefig('CGS_vs_MGS.png')
   return

if __name__ == '__main__':
   verify_classical_QR(100, 80)
   verify_modified_QR(100, 80)
   run_experiment2()
```

Acknowledgement - I have discussed the homework with Dyah Adila. (adila@wisc.edu)