Lab 11

Image Compression (SVD)

Lab Objective: *Explore the SVD as a method of image compression*

In this lab, we are going to explore how the SVD can be used to compress image data. Recall that the SVD is a decomposition of an $m \times n$ matrix A of rank r into the product $A = U\Sigma V^H$, where U and V are unitary matrices having dimensions $m \times m$ and $n \times n$, respectively, and Σ is an $m \times n$ diagonal matrix.

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r, 0, \ldots, 0)$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ are the singular values of A. Upon closer inspection, we see that we can write

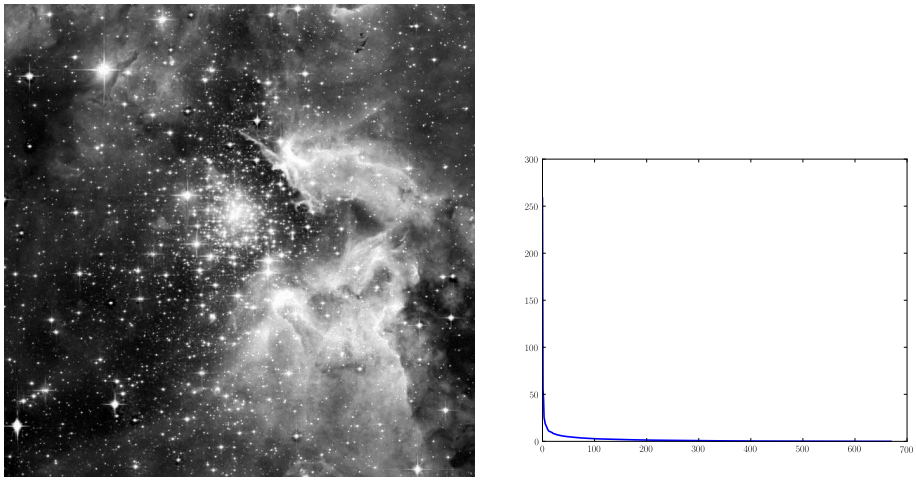
$$U = \begin{pmatrix} U_1 & U_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix}, \quad V = \begin{pmatrix} V_1 & V_2 \end{pmatrix},$$

where U_1 and V_1 have dimensions $m \times r$ and $n \times r$ respectively and Σ_r is the $r \times r$ diagonal matrix of (nonzero) singular values. Multiplying this out yields the reduced form of the SVD

$$A = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^H \\ V_2^H \end{pmatrix} = U_1 \Sigma_r V_1^H$$

Low rank data storage

If the rank of a given matrix is significantly smaller than its dimensions, the reduced form of the SVD offers a way to store A with less memory. Without the SVD, an $m \times n$ matrix requires storing mn values. By decomposing the original matrix into the SVD reduced form, U_1, Σ_r and V_1 together require $mr + r + nr$ values. Thus if r is much smaller than both m and n, we can obtain considerable efficiency. For example, suppose $m = 100$, $n = 200$ and $r = 20$. Then the original matrix would require storing $20,000$ values whereas the reduces form of the SVD only requires storing 6020 values.



(a) NGC 3603 (Hubble Space Telescope). (b) Singular values from greatest to smallest.

Figure 11.1: An image and its singular values.

Low rank approximation

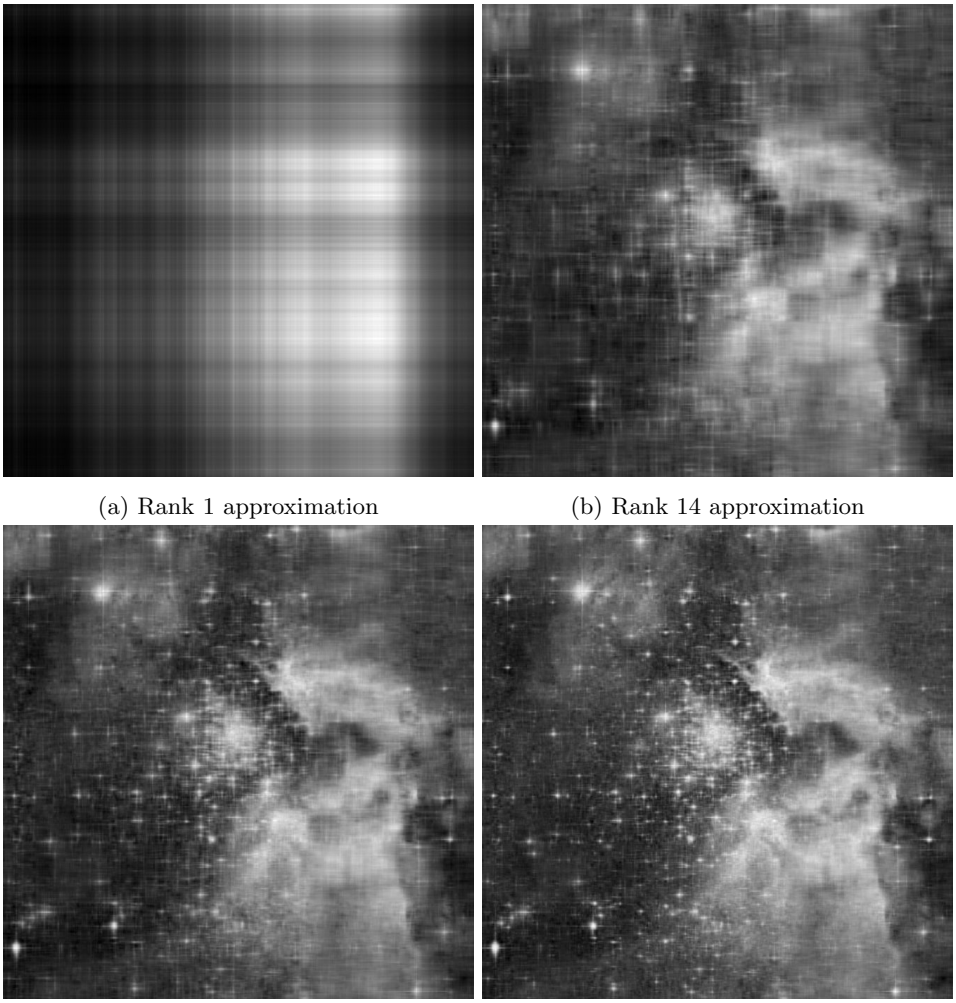
The reduced form of the SVD also provides a way to approximate a matrix with another one of lower rank. This idea is used in many areas of applied mathematics including signal processing, statistics, semantic indexing (search engines), and control theory. If we are given a matrix A of rank r, we can find an approximate matrix $\widehat{A}$ of rank $s < r$ by taking the SVD of A and setting all of its singular values after σ_s to zero, that is,

$$\Sigma_{\widehat{A}} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_s, 0, \ldots, 0)$$

and then multiplying the matrix back together again. The more singular values we keep, the closer our approximation is to A. The number of singular values we decide to preserve depends on how close of an approximation we need and what our size requirements are for U_1, $\Sigma_{\widehat{A}}$, and V_1. Try plotting the the singular values. In Figure 11.1 we present an image and its singular values. Matrix rank is on the x-axis and the singular values are the y-axis. Note that SVD orders the singular values from greatest to least. The greatest singular values contribute most to the image while the smallest singular values hardly contribute anything to the final approximation. By looking at the graph in Figure 11.1b we can have a rough idea of how many singular values we need to preserve to have a good approximation of A. The matrix rank of the image below is 670. However, as the plot shows, we could easily approximate the image using only the first half of the singular values. In Figure 11.2, we can see different rank approximations of the image in Figure 11.1.

The `scipy.linalg` module has a convenient method to calculate the SVD of a given matrix. We can use this method to create a lower-rank approximation of a given matrix. Execute the following code.

```
>>> import numpy as np
```



(a) Rank 1 approximation

(b) Rank 14 approximation

(c) Rank 27 approximation

(d) Rank 40 approximation

Figure 11.2: Different rank approximations for SVD based compression. Notice that higher rank is needed to resolve finer detail.

```
>>> from scipy.linalg import svd, norm
>>> A = np.array([[1,1,3,4], [5,4,3,7], [9,10,10,12], [13,14,15,16], ↩
    [17,18,19,20]])
>>> U,s,Vt = svd(A, full_matrices=False)
```

In that last line of code, we included the keyword argument `full_matrices=False` to calculate the reduced SVD rather than the full SVD. The arrays `U` and `Vt` correspond to the matrices U_1 and V_1^H discussed earlier in the lab. The array `s` simply gives the nonzero singular values of the matrix `A`, and we can find the rank of `A` by inspecting the number of entries in `s` (in this example, we have a rank 4 matrix).

Next, we calculate a rank 3 approximation. Instead of setting the smallest singular value to 0, we simply omit it from the calculation. Note that we also omit

the last column of `U` and last row of `Vt`, as they correspond to the smallest eigenvalue.

```
>>> S = np.diag(s[:-1])
>>> Ahat = U[:,:-1].dot(S).dot(Vt[:-1,:])
>>> norm(A-Ahat)
```

Note that $\widehat{A}$ is "close" to the original matrix A, but that its rank is 3 instead of 4. More precisely, $\widehat{A}$ is the best rank 3 approximation of A with respect to both the induced 2-norm and the Frobenius norm.

Problem 1. Write a function `svd_approx` that takes as input a matrix A and a positive integer k and returns the best rank k approximation to A with respect to the induced 2-norm.

Application to Imaging

Enter the following into IPython (note that any image you might have will work):

```
>>> import matplotlib.pyplot as plt
>>> X = plt.imread('fingerprint.png')[:,:,0].astype(float)
>>> X.nbytes      #number of bytes needed to store X
>>> plt.imshow(X)
>>> plt.show()
```

Computing the SVD of your image is simple. Remember to make the singular values a diagonal matrix before multiplying.

```
>>> U,s,Vt = svd(X, full_matrices=False)
>>> S = sp.diag(s)
```

In the next code block, n represents the desired rank of the output.

```
>>> n = 50
>>> u1, s1, vt1 = U[:,0:n], S[0:n,0:n], Vt[0:n,:]
>>> Xhat = u1.dot(s1).dot(vt1)
>>> (u1.nbytes + np.diag(s1).nbytes + vt1.nbytes) - X.nbytes   #should be ↩
    negative
>>> plt.imshow(Xhat)
>>> plt.show()
```

Recall that the error between the best rank s approximation $\widehat{A_s}$ to A with respect to the induced 2-norm is given by

$$\|A - \widehat{A_s}\|_2 = \sigma_{s+1},$$

where σ_{s+1} is the $(s+1)$-th singular value of A.

Problem 2. Sometimes there is not enough available bandwidth to transmit a full resolution photograph. You aim to reduce the amount of data that needs to be transmitted from a remote location such that loss of image detail is minimal, but the amount of data that needs to be sent has reduced as much as possible. Write a function `lowest_rank_approx` that takes as input a matrix A and a positive number e and returns the lowest rank approximation of A with error less than e (with respect to the induced 2-norm).