AMATH 482 Homework 5

Vineet Palepu

March 17, 2021

1. Abstract

In this paper, we explore the use of dynamic mode decomposition in the area of foreground / background isolation. We use a low rank approximation to construct the decomposition and then extract the background from a series of video clips with the hopes of splitting the video into the static background and dynamic foreground.

2. Introduction and Overview

In this paper, we are given video files and are tasked with separating the background and the foreground. The background consists of a mostly static image, and the foreground consists of moving objects. We perform dynamic mode decomposition on the video data in order to extract the background image. After this, we subtract the background from our image data.

3. Theoretical Background

3.1. SVD

The Singular Value Decomposition (SVD) is a matrix factorization of the form shown in Equation (1), where U and V are unitary matrices and Σ is a diagonal matrix. Because unitary matrices do not stretch or compress a vector and diagonal matrices only stretch vectors, the SVD can be visualized as a rotation, followed by a dilation, followed by another rotation.

$$A = U\Sigma V^* \tag{1}$$

$$A \in \mathbb{R}^{m\times n}, U \in \mathbb{R}^{m\times m}, \Sigma \in \mathbb{R}^{m\times n}, V \in \mathbb{R}^{n\times n}$$

In the SVD, U is the matrix of left singular vectors of A, V is the matrix of right singular vectors of A, and Σ is the matrix of singular values of A, with the singular values located on the diagonal. Additionally, the singular values are ordered from largest to smallest, thus ensuring that every matrix has a unique SVD. Important to note is that all matrices, no matter the size or shape, have a singular value decomposition.

Further, the SVD is useful for forming low-rank approximations of a matrix. If the matrix A is $m \times n$ and $rank(A) = r$, then it can be expressed as shown in Equation (2), where $u_i v_i^*$ is the outer product of $m \times 1$ and $1 \times n$ matrices. Further, if we instead sum up to N instead of r, where $N \leq r$, we can form the best rank N approximation of A.

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^* \tag{2}$$

In order to compute the SVD of a matrix, Equation (3) shows us that V and Σ^2 are the eigenvalues and eigenvectors of A^*A. This can also be done for AA^* as shown in Equation (4), showing that U and Σ^2 are the eigenvalues and eigenvectors of AA^*.

$$\begin{aligned} A^*A &= (U\Sigma V^*)^*(U\Sigma V^*) \\ &= V\Sigma U^* U\Sigma V^* \\ &= V\Sigma^2 V^* \\ A^*AV &= V\Sigma^2 \end{aligned} \tag{3}$$

$$AA^* = U\Sigma^2 U^* \tag{4}$$
$$AA^*U = U\Sigma^2$$

The singular value decomposition is useful in this application because it provides us with a way of calculating the dynamic mode decomposition.

3.2. Dynamic Mode Decomposition

Dynamic mode decomposition (DMD) is a technique based on the SVD that is aimed at dimensionality reduction. Unlike other algorithms, DMD is a data based technique that doesn't need an underlying understanding of the model. DMD finds a basis of spatial modes where the time dynamics can be described by exponential functions with complex exponents.

DMD requires snapshots of spatio-temporal data at evenly spaced points. That is, for a given time point t_i, it must hold that $t_{i+1} = t_i + \Delta t$ where Δt is some constant. These snapshots in time are denoted as in Equation 1. Further, these snapshots can be arranged into the overall data matrix X given by Equation 2.

$$U(x, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \dots \\ U(x_n, t_m) \end{bmatrix} \tag{1}$$

$$X = [U(x, t_1) \quad U(x, t_2) \quad \cdots \quad U(x, t_m)] \tag{2}$$

From this point, DMD then attempts to approximate the Koopman operator. The Koopman operator is a linear time independent operator that maps the data from time t_i to t_{i+1}. The Koopman operator A is given in Equation 3.

$$x_{j+1} = Ax_j \tag{3}$$

Using our data matrix X, we can construct the Koopman operator as shown in Equation 4. The Koopman operator can be calculated as shown in Equation 5, using the SVD. We can then require that r is orthogonal to U giving $U \cdot r = 0$. Thus, we can multiply by U^* and get the result in Equation 6. Next, we can further isolate A bu multiplying by V and Σ^{-1}, shown in Equation 7 and denote the result as $\tilde{S}$. We can then solve the eigenvalue equation in Equation 8 which lets us calculate the eigenvalues of A, called the DMD modes, as shown in Equation 9. Finally, we can express continual multiplications by A as shown in Equation 10.

$$X_2^M = AX_1^{M-1} + re_{M-1}^T \tag{4}$$

$$X_2^M = AU\Sigma V^* + re_{M-1}^T \tag{5}$$

$$U^*X_2^M = U^*AU\Sigma V^* \tag{6}$$

$$U^*AU = U^*X_2^M V\Sigma^{-1} = \tilde{S} \tag{7}$$

$$\tilde{S}y_k = \mu_k y_k \tag{8}$$

$$\psi_k = Uy_k \tag{9}$$

$$X_{DMD}(t) = \sum b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t})b \tag{10}$$

4. Algorithm Implementation and Development

First, we need to load the video file and convert it into a matrix. Once we have done this, we can then create our data matrix X by reshaping the data to flatten out each image into a column vector. Once we have done this, we then form the X_1 and X_2 matrices as described in the background for DMD, and compute the SVD on X_1. Then, we form a low rank approximation by discarding the unneeded data. Then, we calculate $\tilde{S}$ and calculate the eigenvalues and eigenvectors. We can then form our ω vector from the eigenvalues as well as compute our matrix ψ from U and the eigenvectors. Once we have done this, we can compute the initial conditions y_0 by solving the system $\psi y_0 = X_1$. Now that we have done this, we need to extract the mode that represents the background. In order to do so, we assume that $\|\omega_p\| \approx 0$ where ω_p is the value corresponding to the background mode. Similarly, we assume that all the other modes that aren't the background will have a norm larger than zero. That is $\|\omega_k\| \gg 0$ for $k \neq p$. At this point, we can now reconstruct the background image by multiplying $y_{0_p} * \psi_p * \exp(\omega_p * t)$. Note that this matches the procedure in Equation 10. We then normalize these vectors so that they are similarly scaled for image subtraction. We can then extract the foreground image by subtracting the background from every frame in our original video data. However, when performing background subtraction, we may end up where the reconstruction has negative elements. To avoid this, we construct the sparse DMD approximation as show in Equation 11. Then, we find the residual negative values in the sparse DMD and add the residuals to our low rank DMD as in Equation 12. Finally, to get our sparse DMD we subtract the residuals from the current sparse DMD as in Equation 13.

$$X_{DMD}^{Sparse} = X - \left|X_{DMD}^{Low\ Rank}\right| \tag{11}$$

$$X_{DMD}^{Low\ Rank} = \left|X_{DMD}^{Low\ Rank}\right| + R \tag{12}$$

$$X_{DMD}^{Sparse} = X_{DMD}^{Sparse} - R \tag{13}$$

5. Computational Results

For the Monte Carlo clip, we found that only a rank 1 approximation was necessary to construct the background. The singular values decreased very rapidly, dropping to below 1 on only the second singular value. The plot of singular values for the Monte Carlo clip is shown in Figure 1.

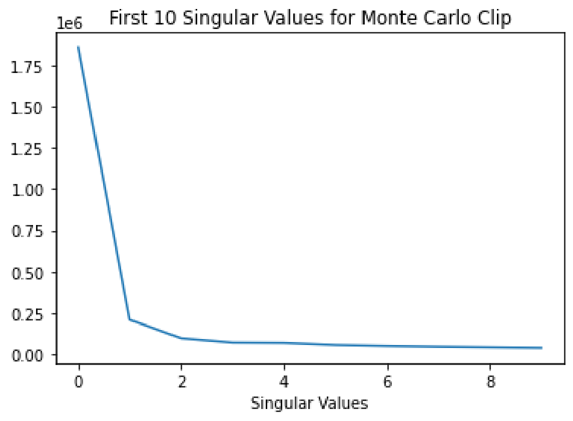

Figure 1. Here we plot the first 10 singular values for the Monte Carlo clip. Note that the first is much larger than all the other singular values.

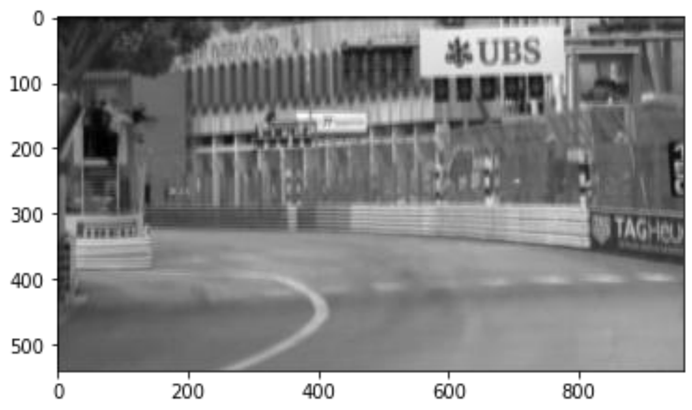
Figure 2. Here we have the isolated background as a result of the DMD. Note that there is slight blurring.

We then found the background image constructed, shown in Figure 2. Overall, there is some slight blurring to the background, but aside from that it is a very good reconstruction of the background.

Finally, we can compare the background subtracted images to the full images, seen in Figure 3. Note that while the background does have a value of 0 indicating it was properly subtracted, some of the pixels of the foreground have negative values. We are unsure as to why this occurs. Attempts to rescale the ranges of these pixels proved unsuccessful. Further, note that some outlines are still visible in the background. These are likely due to slight shakes in the camera during recording. This is supported by the fact that the background image, while accurate, is slightly blurry. This means that the subtraction won't be perfect, and will show artifacts along sharp edges, as seen below.

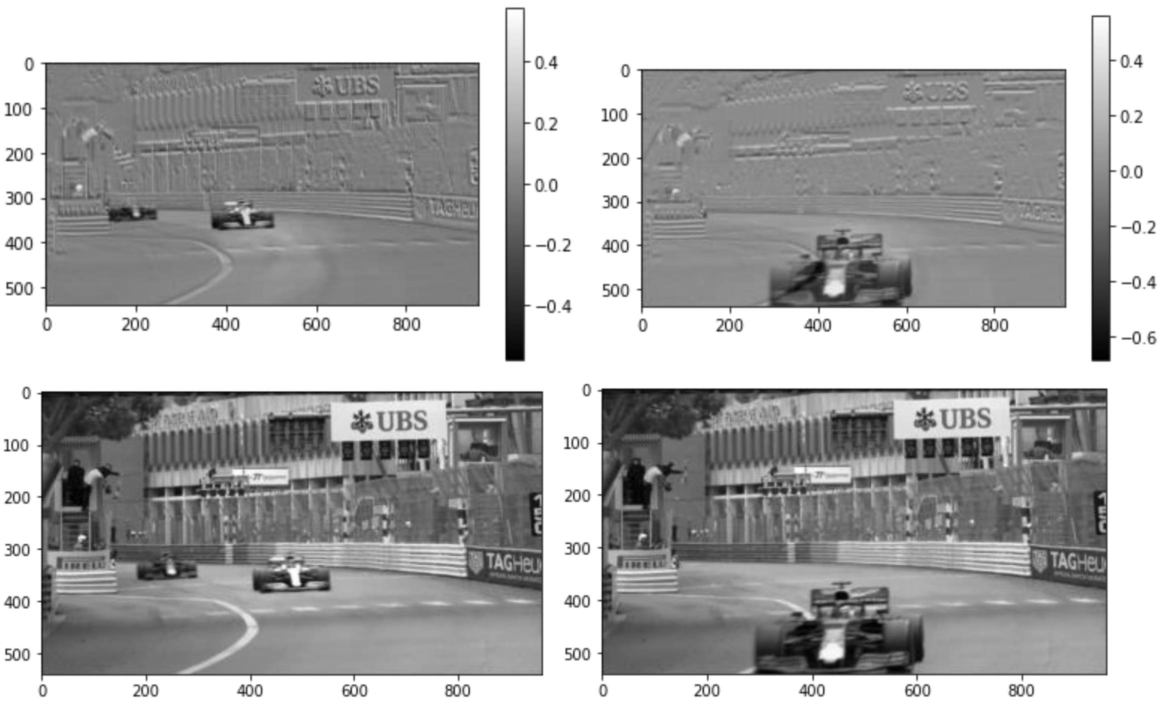
Figure 3. Here we have the foreground isolated (above) and the full images (below). Note that while the foreground objects are isolated, some pixel values are negative. Despite this, the background has pixel values near 0, indicating successful background subtraction. Also note the visibility of edges present in the background.

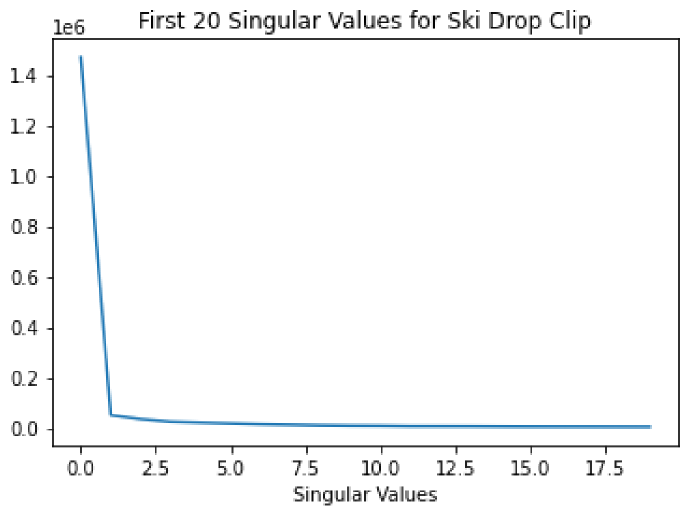

Figure 4. Here we have the first 20 singular values from the Ski Drop clip. Again, note that these significantly decrease after the first value.

Next, we will show the results for the Ski Drop clip. Figure 4 shows the singular values for the clip. Like the Monte Carlo clip, the first singular value was significantly larger than the others. However, we encountered numerical errors that occurred when using an approximation lower than rank 10, possibly due to specific modes with largely positive eigenvalues that caused the reconstructed images' pixel values to blow up.

The results of background subtraction for the Ski Drop clip are shown in Figure 5. While difficult to tell due to the foreground's relatively small size, the background image found was very accurate, and had very little blurring. We found it difficult to tell how successful foreground isolation was due to this.

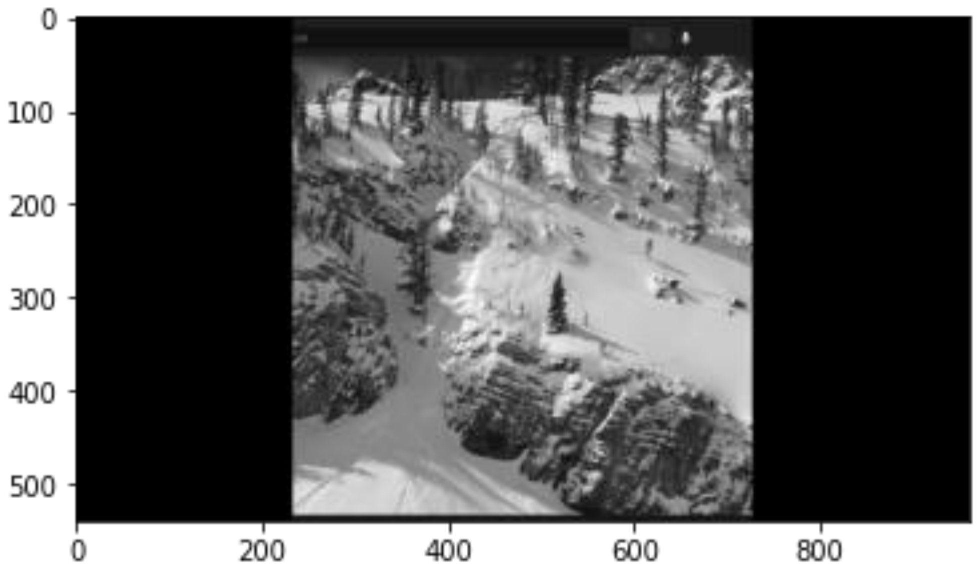

Figure 5. Here we have the isolated background from the ski drop clip. While may be difficult to see, there is no skier present in the clip, thus indicating successful background isolation.

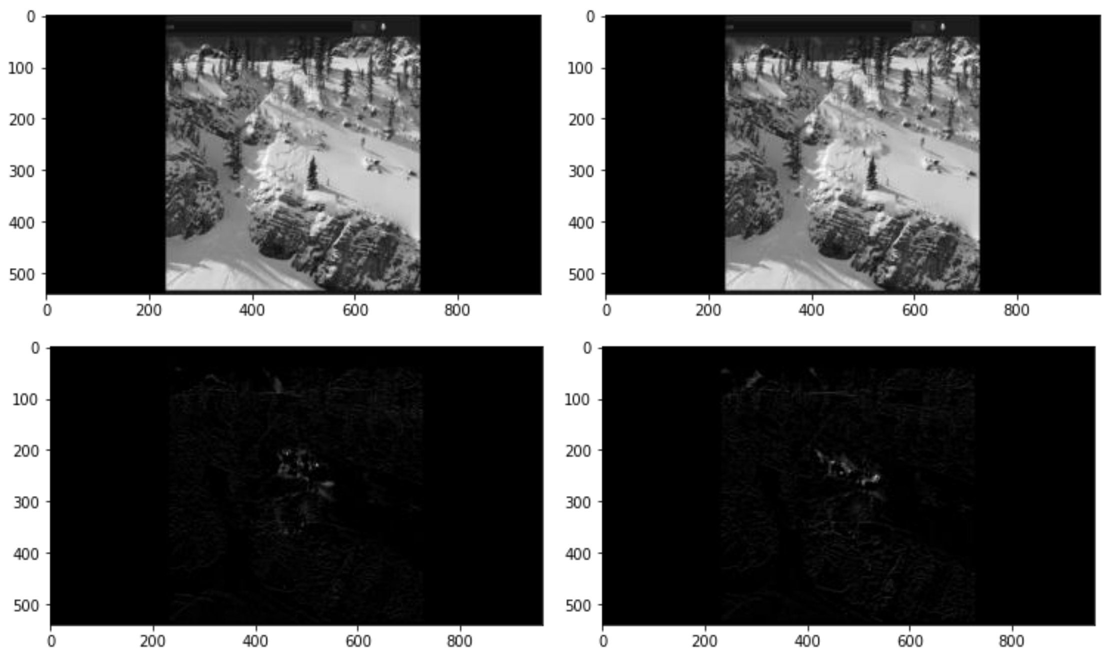

Figure 6. Here we have the results of the foreground isolation. Note that the foreground is incomprehensible, likely due to the small size of the foreground.

Because of the small size of the skier in the clip, it was difficult to ascertain if our results were successful. Of note is the fact that the locations with the small white spots are places where the skier is, so there may at least have been some success. However, the contrast needs to be greatly improved somehow for these results to be more useful.

6. Summary and Conclusions

In this paper, we were able to isolate the moving foreground from the stationary background from video clips with varying degrees of success. With the Monte Carlo clip, which consisted of medium sized cars driving on a race track, we were able to successfully isolate the background, and somewhat successfully isolate the foreground, though with some issues. For the Ski Drop clip, which consisted of a very small skier on a ski slope, we were able to isolate the background image again, but the isolated foreground wasn't easily comprehensible. With further efforts at increasing the contrast and improving the accuracy of the background subtraction, it should be possible to improve these results. These results represent a promising step towards fully isolating the foreground and background of an image. Further areas of research could include looking at predictions of future data. One benefit of DMD is it allows future data to be predicted. While this is normally used on simpler data such as functions or time-series data, it could be interesting to see with what degree of success prediction could occur.

7. Appendix A: Functions Used

Python

NumPy

`np.abs` : Returns the absolute value of a matrix.

`np.amax` : Returns the largest value of a matrix. Used to normalize certain matrices.

`np.arange` : Generates a vector with consecutive integer values.

`np.argmax` : Gets the index of the maximal element

`np.argmin` : Gets the index of the minimal element

`np.linalg.svd` : Calculates the SVD of a matrix. Used to get the singular values and U, V matrices

`np.linalg.eig` : Calculates the eigenvalues and eigenvectors of a matrix

SciKitVideo

`skvideo.io.vread` : Used to read a video file into a data matrix

8. Appendix B: Code

```python
import numpy as np
import matplotlib.pyplot as plt
import skvideo
import skvideo.io

video = skvideo.io.vread("monte_carlo_low.mp4", as_grey=True)
video = np.moveaxis(video.reshape(video.shape[:-1]), 0, -1)

dt = 1 / 60 # 60 fps

print(video.shape)

plt.imshow(video[:,:,5], cmap="gray")
plt.show()

X = video.reshape(video.shape[0] * video.shape[1], -1)
X1 = X[:, 0:-2]
X2 = X[:, 1:-1]
U, Sigma, V = np.linalg.svd(X1, full_matrices=False)

plt.plot(Sigma[:20])
plt.xlabel("Singular Values")
plt.title("First 20 Singular Values for Ski Drop Clip")
plt.show()

# mc: rank = 1
#ski drop: rank = 10
rank = 10
S = U[:, 0:rank].T @ X2 @ V[:, 0:rank] @ np.diag(1. / Sigma[0:rank])
eigenvalues, eigenvectors = np.linalg.eig(S)
omega = np.log(eigenvalues) / dt
phi = U[:, 0:rank] @ eigenvectors

y0, _, _, _ = np.linalg.lstsq(phi, X1[:, 0], rcond=None)

ind = np.argmin(np.abs(omega))
X_low = y0[ind] * phi[:, ind] * np.exp(omega[ind] * (ind * dt))
X_sparse = X - np.abs(X_low)[:, None]


## Maybe move these? idk
X_low_norm = X_low / np.amax(np.abs(X_low))
```

```
X_sparse_norm = (X / np.abs(np.amax(np.abs(X), axis=0))) - np.abs(X_low_norm[:, N
one])


R = X_sparse_norm * (X_sparse_norm < 0)
X_low_imp = R[:, 0] + np.abs(X_low_norm)
X_sparse_imp = X_sparse_norm - R

plt.imshow(X[:, 10].reshape(video.shape[0], video.shape[1]), cmap="gray")
plt.show()
plt.imshow((X_sparse_imp[:, 10]).reshape(video.shape[0], video.shape[1]), cmap="g
ray")
plt.show()

# Check ranges of data
print(np.amin(np.abs(X_low_norm)))
print(np.amax(np.abs(X_low_norm)))
plt.imshow(np.abs(X_low_norm).reshape(video.shape[0], video.shape[1]), cmap="gray
")
plt.show()

test_img = X[:, 0] / np.amax(X[:, 0])
print(np.amin(np.abs(test_img)))
print(np.amax(np.abs(test_img)))
plt.imshow(test_img.reshape(video.shape[0], video.shape[1]), cmap="gray")
plt.show()

fg = test_img - X_low_norm
print(np.amin(fg))
print(np.amax(fg))
plt.imshow(np.abs(fg).reshape(video.shape[0], video.shape[1]), cmap="gray")
plt.colorbar()
plt.show()
```