# MATH 494A Assignment 1

Yifan Wu

February 9, 2023

Abstract

This document develops the idea of how we can separate the video stream into both foreground video and a background video by using Dynamic mode Decomposition (DMD). Both video clips contain foreground and background objects which foreground objects are present in the video as a fast-moving manner, while the background objects exist in a stationary way in the video. Using Singular value Decomposition (SVD),DMD,and Eigen expansion we are able to successfully determine the different modes in order to separate the video stream.

## 1 Introduction and Overview

### 1.1 DMD

DMD Algorithm is a pure data Driven method which the aim is to take advantage of the low dimensionality in experimental data without having to rely on a given set of governing equations [1]. The goal here is to use DMD reconstructions to separate the foreground and background of 2 videos.

### 1.2 SVD

We will use Singular Value Decomposition (SVD) to reduce the rank of our video to increase the efficiency of producing our DMD matrix.

## 2 Theoretical Background

We start with our data matrices $X, Y \in C^{M\times N}$ being a snapshot of the data. The columns constitute snapshots of the system over time and the rows are the snapshots vector in $C^M$ which contains all the information of each snapshot[1]. The snapshots could also simply represent measurements of different variables in the system, and so we make no assumption of structure on the snapshots. Writing [1]

$$X = [\,X1 \;\; X2 \,\ldots. \quad X_n\,]\,, \quad Y = [\,Y1 \;\; Y2 \,\ldots. \quad Y_n\,] \qquad \textbf{[2].}$$

The critical assumption of DMD is that $Y_n$ is a snapshot of system exactly $\Delta t > 0$ time units in the future from $X_n$ One possibility is that we are collecting data from a single trajectory wherein $Y_n = X_{n+1}$ for all n. [1], Although the underlying process that maps $X_n\ to\ Y_n$ is potentially nonlinear, DMD seeks to find a matrix $A \in C^{M\times M}$ that best maps the snapshots $X_n\ to\ Y_n$, so that A [1]

$$Y_n \approx AX_n \qquad \textbf{[3]}$$

We can simply take $X_n$ pseudo inverse to the left and take SVD of $X_n$ to find the best fit rank to work with Matrix A

$$A = YX^{+} = YVS^{+}U^{*} \qquad \textbf{[4]}$$

With the DMD matrix we obtain on equation [4], we Obtain our Linear discrete dynamical System then we will have.

$$Z_{n+1} \approx AZ_n \qquad \textbf{[5]}$$

Therefore, we can use Eigen Decomposition to find the Eigenvalues and Eigenvectors of A which can tell us the dynamical of A

$$X(t) = \sum_{i=0}^{n} b_i \varphi_i \, e^{-i\omega t} = \Psi diag(e^{-i\omega t})b \qquad \textbf{[6]}$$

We would like to use the eigenvector we found on equation [6] and convert back to our original basis by times U Finally, we would like to find out initial value by just taking the pseudo inverse of $\varphi$ times the first snapshot we have in data. We will get our DMD modes.

## 3 Algorithm Implementation and Development

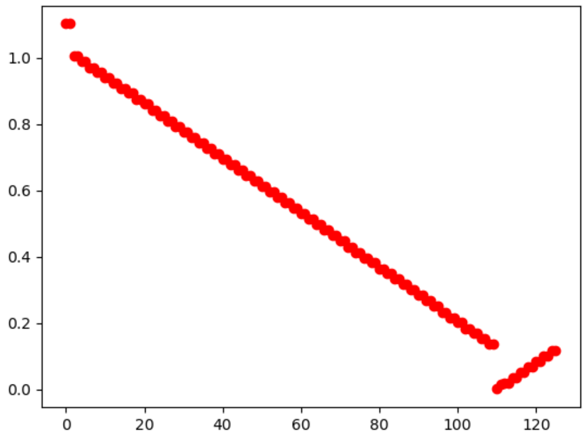
We will first use python opencv package to extract the all the frames from the video and convert them into Matrix. The result we get is 4-dimension Matrix. We will convert RGB to gray to reduce the dimensions to increase the efficiency of our operation. Then we will reshape the Matrix into $C^{M\times N}$ size which N is the each frame we capture from our video and M contains the information of width and length of each frame. Now we can construct Our X and Y Matrix

1. Then compute the SVD of the Matrix X and Choose the rank that you will truncate.
2. Find the low rank Matrix A by using equation 4 and compute the Eigenvalue and Eigenvectors of this low rank Matrix. Then by multiplying the eigenvector $\varphi$ and U, we will get $\Psi$ which is the Eigenvector of the origin Matrix $A^{M\times N}$
3. Compute the $\varpi = \log(eigenvalues)\,/\Delta t$, then we can use the initial snapshot x1 and $\Psi$ to find the coefficients b.
4. Then we will find the $\varpi$ close to zero to determine the background of our video which can construct our low rank $x_{Low}$ then we can compute the Sparse DMD Matrix by subtracting the low rank matrix
5. Find the negative values in X sparse DMD and put them into a Matrix R, then see if adding R into the Low rank and subtract from the Sparse will help our result.

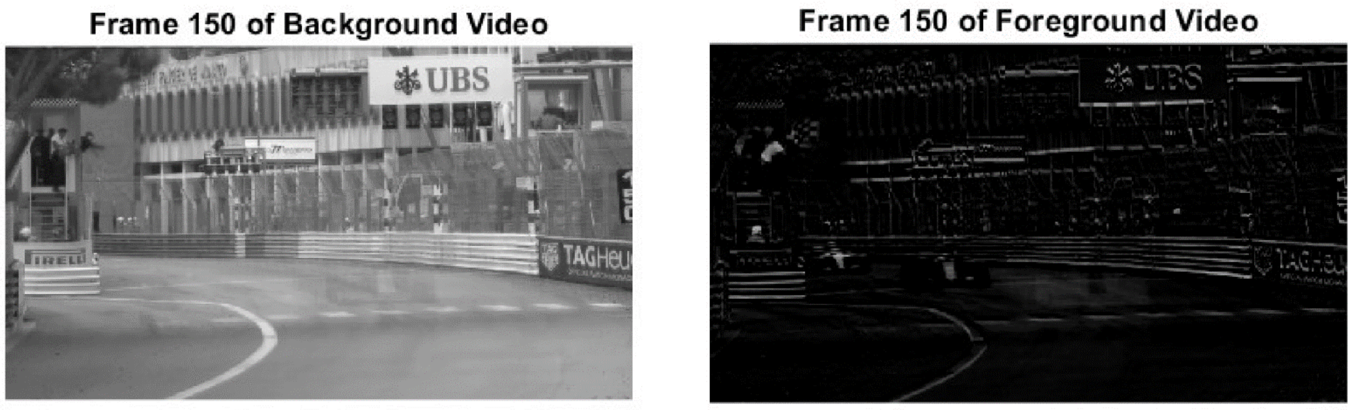
## 4 Computational Results

### 4.1 result of Monte Carlo

By using the energy Function, we choose 95% energy of the data for the SVD computation which the rank here is r = 127. Then we use the Algorithms [1] and [2] to compute our low rank Matrix with associate Eigenvalues and Eigenvectors. By using Algorithm [3], we can find our $\varpi$, plot them to see the values on Figure 2



The threshold of $\varpi$ here for separate the Background is 0.02, then after Algorithm [4] and [5], we will have the reconstruct DMD Matrix



Above Figure 3 is Frame 150 of the Background on the left and Foreground on the right after we separate it from the original video. The background result here is by adding R in the low rank X, as we can see on the figure, there is the shadow of a slightly blurred car.

## 4.2 Result of Ski Drop

By using the energy Function, we choose 95% energy of the data for the SVD computation which the rank here is r = 124. Then we use the Algorithms [1] and [2] to compute our low rank Matrix with associate Eigenvalues and Eigenvectors. Compared to the original data size, which is 454, the rank is obviously much lower here, which helps us boost our operation even more.

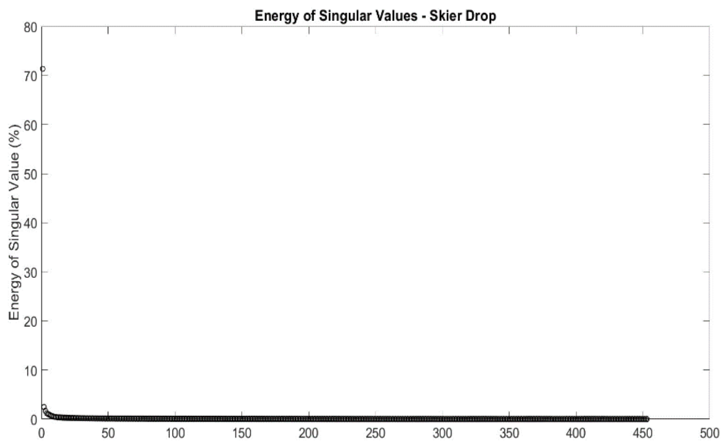
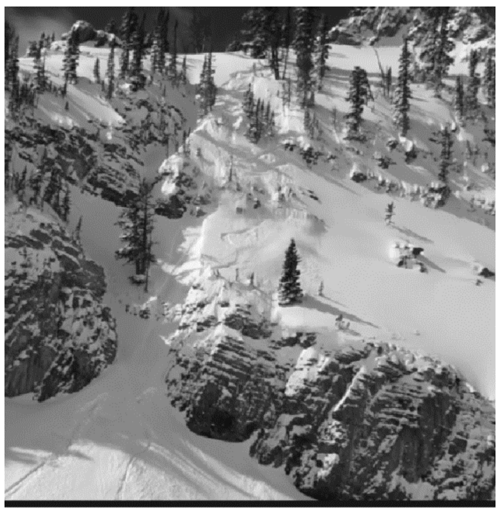


Figure 4 is the Energy of Singular values of ski Drop video. I tried the first 10 rank of singular values which contains 80% of the energy but the results are not good as 95% energy of the data. Although the 10 rank of data looks tempting.



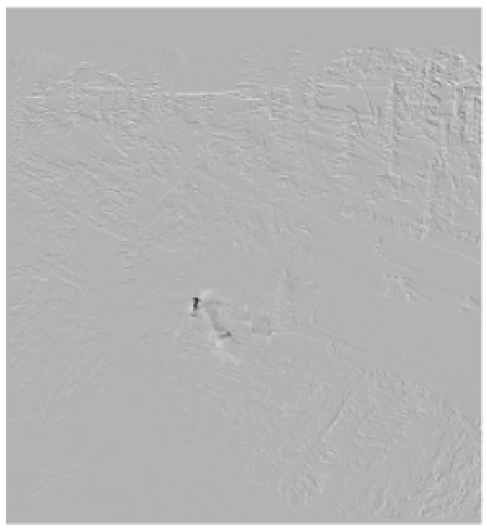


Figure 5 on the left is the background of the video on Frame 248, and the right side is the foreground on Frame 248. The threshold I choose for $\varpi$ is 0.1. I subtracted the R values from the Sparse Matrix this time and it seems the foreground here has a better result than Monte Carlo.

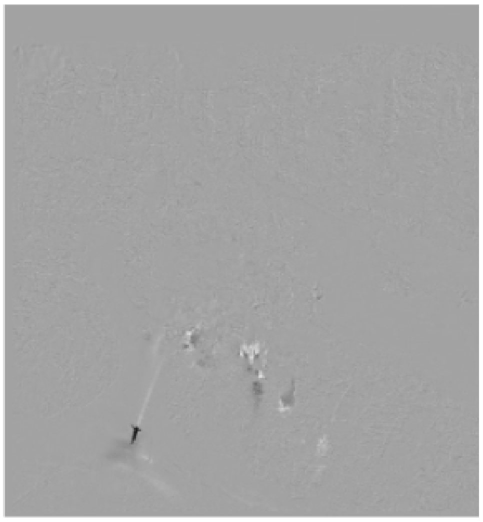
## 5 Summary and Conclusions

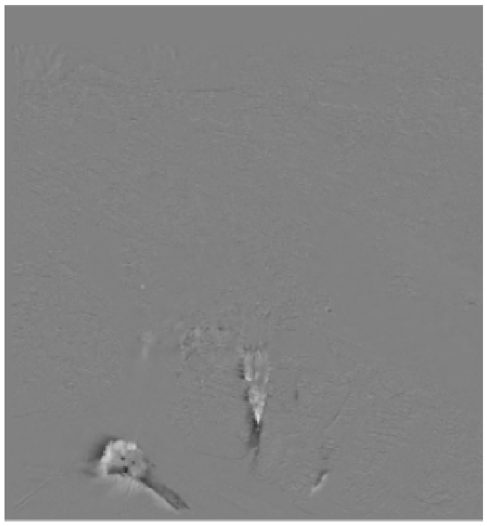
By seeing the result of the background of Monte Carlo, adding R into the X (low rank) will deteriorate our result. From another perspective. Subtracting R from the Sparse in Ski drop video has the better result in terms of separate foreground. All in all, DMD is an excellent tool that can separate the foreground and background from a video. DMD algorithm has a lot of potential for application in other fields as well, I hope that I have the opportunity to use the DMD algorithm to try more projects in the future.

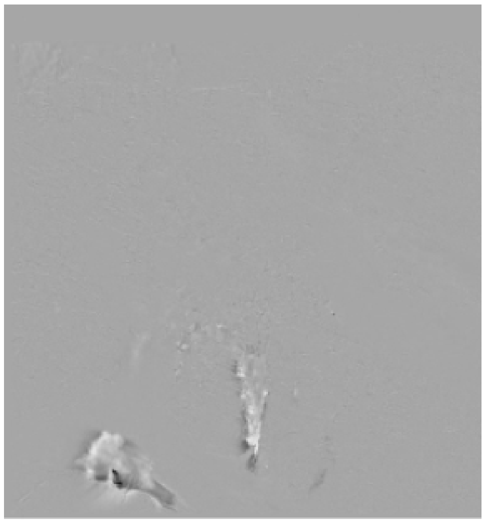
## References

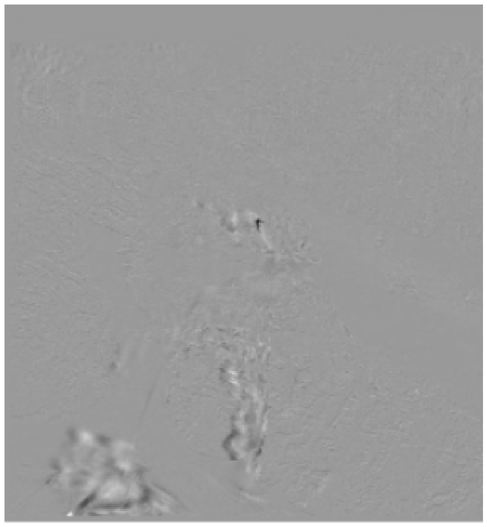
[1] Jason J. Bramburger. *Data-Driven Methods for Dynamic Systems: A new perspective on old problems First Edition*. Concordia University, 2023.

## Appendix Additional Figure of Ski_Drop Foreground









## Appendix B Python Code

Add your python code here. This section will not be included in your page limit of six pages.

```
import cv2
import numpy as np

# Load the video
video = cv2.VideoCapture("E:\Math_494\Assignments\ski_drop_low.mp4")

# Initialize a list to store the grayscale matrices
frames = []

# Read the first frame
success, frame = video.read()

while success:
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Convert the grayscale frame to a matrix
    gray_matrix = np.array(gray)

    # Append the grayscale matrix to the list of frames
    frames.append(gray_matrix)

    # Read the next frame
    success, frame = video.read()

# Release the video
video.release()

vid=np.array(frames)
# Reshape the numpy array
vid_reshaped = vid.reshape(518400, 454)
vid_reshaped.shape
# Create the matrices X and Y
X = vid_reshaped[:, :453]
Y = vid_reshaped[:, 1:454]

# Perform SVD on the matrix X
U, s, Vh = np.linalg.svd(X, full_matrices=False)
```

```
# Set the truncation rank r
r = 124

# Truncate the matrices U, s, and Vh
U = U[:, :r]
s = s[:r]
Vh = Vh[:r, :]

# Compute the low-rank DMD matrix A
S = np.diag(s)
A = np.dot(np.dot(np.dot(np.conj(U).T, Y), np.conj(Vh).T),np.linalg.inv(S))

# Find the eigenvalues and eigenvectors of the matrix A
eigenvalues, eigenvectors = np.linalg.eig(A)

phi = np.dot(U, eigenvectors)

eigenvalues_diag = np.diag(eigenvalues)
eigenvalues_abs_log = np.abs(np.log(np.diag(eigenvalues_diag)) / 1)
mode_threshold = 0.1

close_to_zero_modes = np.where(eigenvalues_abs_log < mode_threshold)
close_to_zero_modes = np.array(close_to_zero_modes)
plt.plot(close_to_zero_modes,'ro')

def extract_background(video_path, singular_value_ratio=0.0):
    cap = cv2.VideoCapture(video_path)
    frames = []
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        frame = frame.astype(float) / 255.0
        frames.append(frame.flatten())
    frames = np.array(frames)
    X = frames[:-1, :]
    Y = frames[1:, :]
    modes, eigenvalues, eigenvectors = dmd(X, Y, singular_value_ratio)
    rank = np.sum(S > singular_value_ratio * S[0])
    background = np.dot(modes[:, :rank], eigenvectors[:rank, :]).real
    return np.median(background, axis=0)
```

```
def show_image(image):
    cv2.imshow("Background", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

background = extract_background("E:\Math_494\Assignments\monte_carlo_low.mp4",singular_value_ratio=0.99)
show_image(background)
```