AMATH 482 Homework 5

Jonah Wu

March 17, 2021

Abstract

We are going to decompose 2 video clips into 2 different parts: the background and the foreground. The first video has a skier skiing down a hill, and the second video has racing cars during a race. We are going to separate the foreground and the background of videos using the techniques of Dynamic Mode Decomposition with MATLAB.

1 Introduction and Overview

Dynamic Mode Decomposition is a dimensionality reduction algorithm. In order to separate the background from the original video, we need to look for objects that are not moving or updating throughout the video. Thus, Dynamic Mode Decomposition is the perfect tool for us since DMD computes a set of modes each of which is associated with a fixed oscillation frequency and growth or decay rate from a given time series of data. Finally, after we have the background we can subtract that from the original video to get the foreground.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

The SVD is a form of factoring a real or complex matrix that generalizes the eigen decomposition of a square normal matrix to any $m \times n$ matrix. Singular value decomposition of the matrix $\mathbf{A}$:

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

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are unitary matrices, and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal.
The values σ_n on the diagonal of Σ are called the singular values of the matrix $\mathbf{A}$. The number of non-zero singular values is equal to the rank of $\mathbf{A}$. The vectors u_n which make up the columns of $\mathbf{U}$ are called the left singular vectors of $\mathbf{A}$. The vectors v_n which make up the columns of $\mathbf{V}$ are called the right singular vectors of $\mathbf{A}$.

2.2 Dynamic Mode Decomposition (DMD)

Dynamic Mode Decomposition (DMD) seeks to find a linear representation of our data, and it is good for capturing simple low-dimensional dynamics, even for nonlinear systems. The first step of DMD is to approximate the modes of the Koopman operator. The Koopman operator $\mathbf{A}$ is a linear, time-independent operator such that:

$$\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j \tag{2}$$

where j indicates each time step from the data, $\mathbf{A}$ is the operator that maps data from time t_j to t_{j+1}, and $\mathbf{x}_j$ is an N-dimensional vector of a snapshot of the data at time j. We first consider the matrix:

$$\mathbf{X}_1^{M-1} = [\mathbf{x}_1 \ \ \mathbf{x}_2 \ \ \cdots \ \ \mathbf{x}_{M-1}] \tag{3}$$

With Koopman operator, we can rewrite the above matrix as:

$$\mathbf{X}_1^{M-1} = [\mathbf{x}_1 \ \ \mathbf{A}\mathbf{x}_1 \ \ \cdots \ \ \mathbf{A}^{M-2}\mathbf{x}_1] \tag{4}$$

Next, we then can then write $\mathbf{X}_2^M$ as:

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T \tag{5}$$

where $\mathbf{r}e_{M-1}^T$ is the error while $\mathbf{r}$ is the residual (error) vector. Since matrices are completely understood by their eigenvalues and eigenvectors, we can find $\mathbf{A}$ by finding another matrix with the same eigenvalues. We first do SVD on $\mathbf{X}_1^{M-1}$:

$$\mathbf{X}_1^{M-1} = \mathbf{U}\Sigma\mathbf{V^*} \tag{6}$$

We then can write $\mathbf{X}_2^M$ as:

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{U}\Sigma\mathbf{V^*} + \mathbf{r}e_{M-1}^T \tag{7}$$

We are going to choose $\mathbf{A}$ such that the columns in $\mathbf{X}_2^M$ can be written as linear combinations of the columns of $\mathbf{U}$, and it is essentially the same as requiring that they can be written as the linear combinations of the POD modes. Since the residual vector $\mathbf{r}$ must be orthogonal to the POD basis, we have $\mathbf{U^*r} = 0$. Multiplying the above equation by $\mathbf{U^*}$ on the left gives us:

$$\mathbf{U^*}\mathbf{X}_2^M = \mathbf{U^*AU}\Sigma\mathbf{V^*} \tag{8}$$

Next, we isolate $\mathbf{U^*AU}$ by by multiplying $\mathbf{V}$ and Σ^{-1} on the right and we will get:

$$\mathbf{U^*AU} = \mathbf{U^*}\mathbf{X}_2^M\mathbf{V}\Sigma^{-1} = \tilde{\mathbf{S}} \tag{9}$$

where $\tilde{\mathbf{S}}$ is similar to $\mathbf{A}$ since they are related by applying a matrix on one side and its inverse on the other. Thus, if $\tilde{\mathbf{S}}$ has eigenvectors as $\mathbf{y}$, then $\mathbf{A}$ would have eigenvectors as $\mathbf{Uy}$. Next, we then can write the eigenvector/eigenvalue pairs of $\tilde{\mathbf{S}}$ as:

$$\tilde{\mathbf{S}}\mathbf{y_k} = \mu_k\mathbf{y_k} \tag{10}$$

and therefore the eigenvectors of $\mathbf{A}$, which are called the DMD modes, are defined as:

$$\psi_k = \mathbf{U}\mathbf{y_k} \tag{11}$$

As a result, we got everything we need for $\mathbf{A}$, and we can just expand in our eigenbasis to get the DMD solution, which is essentially the sum of of exponential functions:

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k\psi_k e^{\omega_k t} = \Psi \text{diag}(e^{\omega_k t})\mathbf{b} \tag{12}$$

where K is the rank of $\mathbf{X}_1^{M-1}$, b_k are the initial amplitude of each mode, and the matrix Ψ are the eigenvectors ψ_k in columns. In addition, $\omega_k = \ln(\mu_k)/\Delta t$ because we write the time dynamics as exponential functions. Finally, we can find b_k through initial condition at $t = 0$ using $\mathbf{x_1}$:

$$\mathbf{x_1} = \Psi\mathbf{b} \implies \mathbf{b} = \Psi^\dagger\mathbf{x_1} \tag{13}$$

where $\Psi^\dagger$ is the pseudoinverse of the matrix Ψ.

2.3 Low Rank DMD Subtraction

The DMD spectrum of frequencies can be used to subtract background modes:

$$\mathbf{X}_{DMD} = b_p \varphi_p e^{\omega_p t} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t} \tag{14}$$

where p satisfies $||\omega_p|| \approx 0$, and the first term as Low-Rank DMD of $\mathbf{X}$, which represents the background video while the second term is the sparse DMD of $\mathbf{X}$, which is going to be the foreground video. Finally, the sparse DMD can be calculated with real-valued elements only as:

$$\mathbf{X}_{DMD}^{\text{Sparse}} = \mathbf{X} - |\mathbf{X}_{DMD}^{\text{Low-Rank}}| \tag{15}$$

3 Algorithm Implementation and Development

With the original video files, we will first read and reshape the data into what we need using MATLAB build-in functions, which can be found in **Appendix A**. Next, we will apply the algorithm for Dynamic Mode Decomposition on our data as the following.

Algorithm 1: Dynamic Mode Decomposition (DMD)

```
    X = data
    X1 = X(:, 1:end-1)
    X2 = X(:, 2:end)
    SVD on X1 to get $\mathbf{U\Sigma V^*}$ using svd()
    Calculate $\tilde{\mathbf{S}} = \mathbf{U^*X}_2^M\mathbf{V}\Sigma^{-1}$
    Use eig() to get eigenvectors and eigenvalues of $\tilde{\mathbf{S}}$
    Calculate $\omega = \ln(\mu)/\Delta t$
    Calculate eigenvectors of $\mathbf{A}$ by $\Psi = \mathbf{Uy}$
    Find initial conditions by $\mathbf{b} = \Psi^\dagger \mathbf{x_1}$
    Calculate modes of $\mathbf{U}$ by $\mathbf{x_1}e^{\omega t}$
    Find $\mathbf{X}_{DMD} = \Psi$ using the modes of $\mathbf{U}$
    Subtract the low rank DMD to find background video and foreground video
```

I found out that the 1st column of the DMD solution is the low rank DMD. In addition, when subtracting low rank DMD from the solution to get the sparse DMD, we might get negative values. We can handle this by using MATLAB build-in function `mat2gray()`. Finally, after we subtract the low rank DMD, we can just reshape the data and use `imshow()` to show specific frames of the background and foreground of the videos.

4 Computational Results

4.1 Ski Drop

After we subtract the low rank DMD, we can find the sparse DMD for the foreground of the video, and I pick the 200th frame to show what the foreground looks like, which is going to be a skier as below.

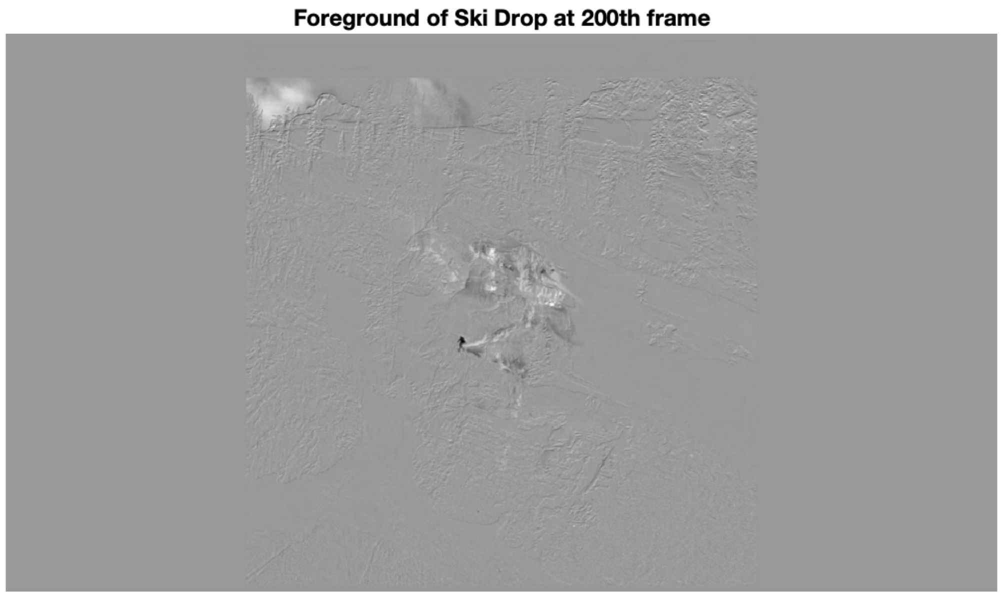

Figure 1: Ski drop foreground

With the subtracted low rank DMD, we can reconstruct the background of the video, which is going to be just the mountain as below without the skier.

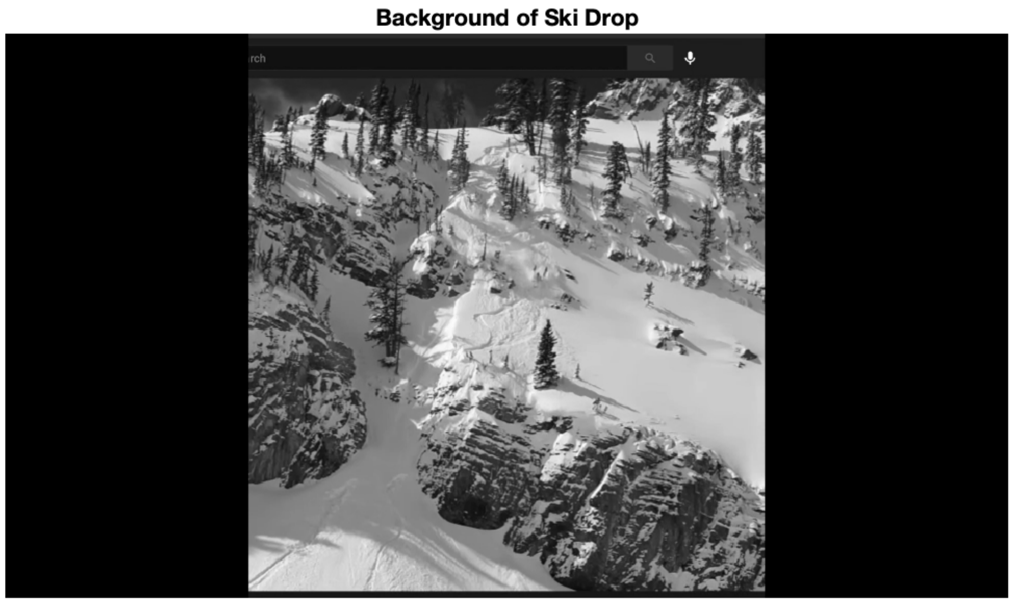

Figure 2: Ski drop background

4.2 Monte Carlo

After we subtract the low rank DMD, we can find the sparse DMD for the foreground of the video, and I pick the 200th frame to show what the foreground looks like, which is going to be the race cars as below.

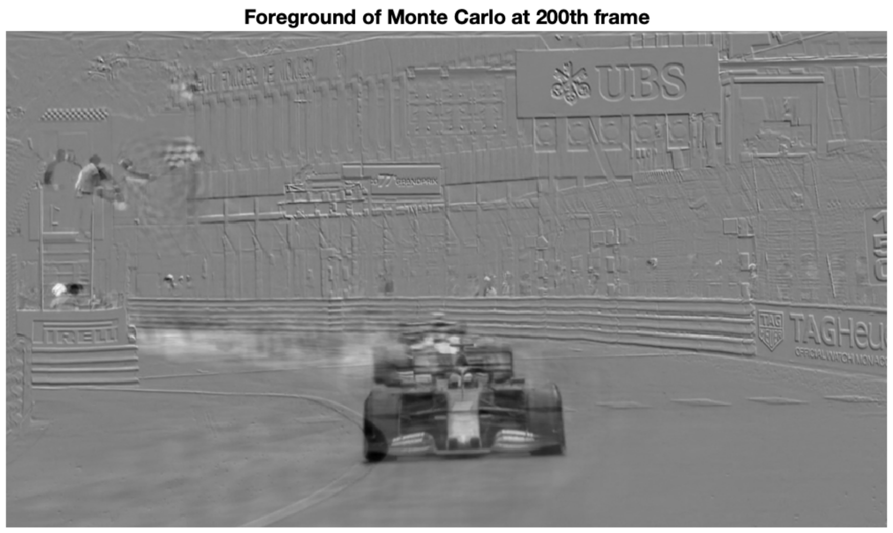

Figure 3: Monte carlo foreground

With the subtracted low rank DMD, we can reconstruct the background of the video, which is going to be just the mountain as below without the skier.

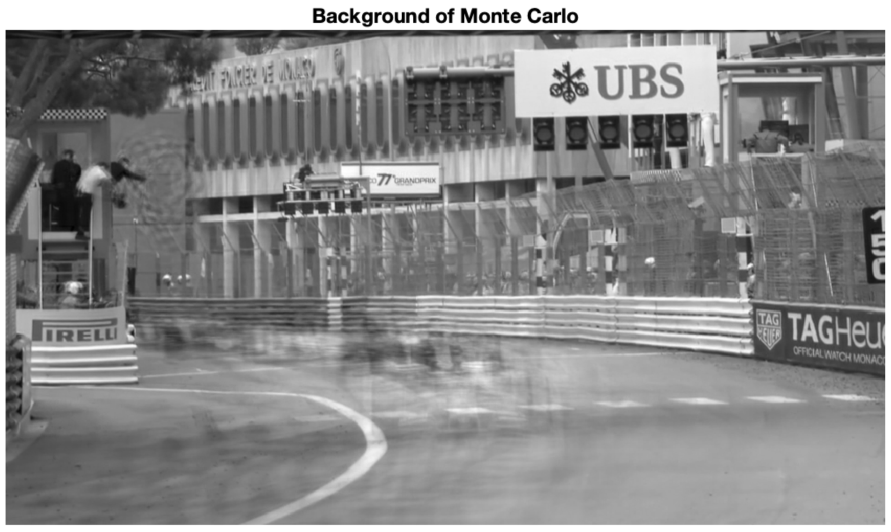

Figure 4: Monte carlo background

5 Summary and Conclusions

Overall, we can see that DMD is a powerful tool for separating the background and foreground from videos since DMD computes a set of modes each of which is associated with a fixed oscillation frequency and growth or decay rate from a given time series of data, and we can subtract the low rank DMD as the background video, and the leftover sparse DMD as the foreground video.

Appendix A MATLAB Functions

- `[U, S, V]` = `svd(X)`: performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `VideoReader(X)` returns a video object from input file X.
- `video = read(v)` reads all video frames from the file associated with v.
- `zeros(m,n)` returns an $m \times n$ matrix of zeroes.
- `y = linspace(x1,x2,n)` generates n points and the spacing between the points is `(x2-x1)/(n-1)`.
- `exp(X)` returns e^X.
- `Y = log(X)` returns the natural logarithm ln(x) of each element in array X.
- `abs(X)` returns the absolute value of a given input X.
- `diag(X)` returns the diagonal entries of a given a matrix X.
- `eig(X)` returns a column vector of eigenvalues corresponding to the matrix X.
- `reshape(X, [m n])` returns an $m \times n$ matrix given an inputted array X.
- `im2double(X)` returns an image converted to double precision, given an image X.
- `mat2gray(X)` returns a gray-scale image with values bounded by (0, 1) that is converted with input matrix X.
- `rgb2gray(X)` returns a gray-scale image converted from the input image X.
- `imshow(I)` displays the gray-scale image I in a figure.

Appendix B MATLAB Code

```
%% Read Ski Drop video
video = VideoReader('ski_drop_low.mp4');
t = linspace(0,video.duration,video.NumFrames);
dt = video.duration / video.NumFrames;
frames = read(video);
for j = 1:video.NumFrames
    frame = rgb2gray(frames(:,:,:,j));
    X(:,j) = frame(:);
end


%% DMD
X = im2double(X);
X1 = X(:,1:end-1);
X2 = X(:,2:end);

[U, Sigma, V] = svd(X1,'econ');
S = U'*X2*V*diag(1./diag(Sigma));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV;

y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),video.NumFrames);
for iter = 1: video.NumFrames
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end
```

```
u_dmd = Phi*u_modes;


%% Foreground and Background
sparse = X-abs(u_dmd(:,1));
foreground = mat2gray(reshape(sparse(:,200), [video.height video.width]));
figure()
imshow(foreground);
title('Foreground of Ski Drop at 200th frame')
saveas(gcf, 'ski_drop_foreground.png')

background = reshape(u_dmd(:,1), [video.height video.width]);
figure()
imshow(background);
title('Background of Ski Drop')
saveas(gcf, 'ski_drop_background.png')



%% Read Monte Carlo video
clear;

video = VideoReader('monte_carlo_low.mp4');
t = linspace(0,video.duration,video.NumFrames);
dt = video.duration / video.NumFrames;
frames = read(video);
for j = 1:video.NumFrames
    frame = rgb2gray(frames(:,:,:,j));
    X(:,j) = frame(:);
end


%% DMD
X = im2double(X);
X1 = X(:,1:end-1);
X2 = X(:,2:end);

[U, Sigma, V] = svd(X1,'econ');
S = U'*X2*V*diag(1./diag(Sigma));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV;


y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),video.NumFrames);
for iter = 1: video.NumFrames
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;


%% Foreground and Background
sparse = X-abs(u_dmd(:,1));
foreground = mat2gray(reshape(sparse(:,200), [video.height video.width]));
figure()
imshow(foreground);
title('Foreground of Monte Carlo at 200th frame')
saveas(gcf, 'monte_carlo_foreground.png')

background = reshape(u_dmd(:,1), [video.height video.width]);
figure()
imshow(background);
title('Background of Monte Carlo')
saveas(gcf, 'monte_carlo_background.png')
```