AMATH 482 Homework 5

Clayton Heath

March 17, 2021

Abstract

We use the *Dynamic Mode Decomposition* (DMD) in order to removed the background from a Monte Carlo and Ski Drop video, leaving just the foreground of the videos.

1 Introduction and Overview

We will first import our videos. We will then make our videos black and white and make them in the proper form for the use of DMD. We will then apply the DMD to the data of the videos, but only partially recomposing it so that we only have the background data. We will then subtract that background data from the original video, leaving just the foreground.

2 Theoretical Background

The *Dynamical Mode Decomposition* (DMD) uses low-dimensionality in experimental data without having to rely on governing equations. DMD finds a basis of spatial modes where the time dynamics are exponential functions. So the only behavior we can have is oscillations, decay, and growth. Let us have data that evolves in space and time. Then:

$$N = \text{number of spatial points saved per unit time snapshot} M = \text{number of snapshots taken}$$

And we must have time data collected in consistent spaced intervals

$$t_{m+1} = t_m + \Delta t, \qquad m = 1, ..., M-1, \qquad \Delta t > 0$$

Thus we have our snapshots as

$$U(\mathbf{x}, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \vdots \\ U(x_m, t_m) \end{bmatrix}$$

for each $m = 1, ..., M$. We can then use these snapshots to form columns of data matrices

$$\mathbf{X} = [U(\mathbf{x}, t_1) \quad U(\mathbf{x}, t_2) \quad ... \quad U(\mathbf{x}, t_M)]$$

and

$$\mathbf{X}_j^k = [U(\mathbf{x}, t_j) \quad U(\mathbf{x}, t_{j+1}) \quad ... \quad U(\mathbf{x}, t_k)]$$

The DMD approximates the *Koopman operator*. Let the Koopman operator be $\mathbf{A}$, a linear and time-independet operator such that

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

where j is some data collection time and $\mathbf{A}$ maps the data from time t_j to t_{j+1}. And $\mathbf{x}_j$ is an N-dimensional vector of data points collected at time j.
Now, to build our Koppman operator. Consider the matrix

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_{M-1} \end{bmatrix}$$

Then using Equation 1 we have

$$\mathbf{X}_1^{M-1} = \begin{bmatrix}\mathbf{x}_1 & \mathbf{A}\mathbf{x}_1 & \mathbf{A}^2\mathbf{x}_1 & \dots & \mathbf{A}^{M-2}\mathbf{x}_1\end{bmatrix}$$

Which can be written as

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T \tag{2}$$

where e_{M-1} is a vector of all zeros expect at the $M-1$st component, where it is a 1. But since our final point $\mathbf{x}_M$ was not included, so we add residual vector $\mathbf{r}$ to account for it. Our goal is to solve for $\mathbf{A}$, but we will find a matrix that has the same eigenvalues as $\mathbf{A}$. Using SVD, we get $\mathbf{X}_1^{M-1} = \mathbf{U}\Sigma\mathbf{V}^*$. And from Equation 2 we have

$$\begin{aligned}\mathbf{X}_2^M &= \mathbf{A}\mathbf{U}\Sigma\mathbf{V}^* + \mathbf{r}e_{M-1}^T\\ \mathbf{U}^*\mathbf{X}_2^M &= \mathbf{U}^*\mathbf{A}\mathbf{U}\Sigma\mathbf{V}^*\\ \mathbf{U}^*\mathbf{A}\mathbf{U} &= \mathbf{U}*\mathbf{X}_2^M\mathbf{V}\Sigma^{-1} := \mathbf{S}\end{aligned}$$

So we have that $\mathbf{S}$ and $\mathbf{A}$ are similar. This means they share quite a few properties, one of which is that they share the same eigenvalues. And, if $\mathbf{y}$ is an eigenvector of $\mathbf{S}$, then $\mathbf{U}\mathbf{y}$ is an eigenvecotor of $\mathbf{A}$. So, we write the eigenvector/value pairs of $\mathbf{S}$ as

$$\mathbf{S}\mathbf{y}_k = \mu_k\mathbf{y}_k$$

Thus giving $\mathbf{A}$ the eigenvectors or DMD modes

$$\psi_k = \mathbf{U}\mathbf{y}_k$$

So, we now have

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

Where K is the rank of $\mathbf{X_1^{M-1}}$, b_k are the initial amplitude of each mode, and $\omega_k = \ln(\mu_k)/\Delta t$.

3 Algorithm Implementation and Development

First, we start with the Monte Carlo video, under the `Monte Carlo` section of the MATLAB code. We first read in the video, then make the video black and white by going over each frame of the video, giving us our `vid_mat` matrix, this way we have the proper dimensions for our DMD. Then, we make our `X` and `Xshift` matrices, with `X` not having the last frame of our video and `Xshift` not having the first frame of our video, reflecting our matrices from the theoretical section. We then apply our `dmd()` function as explained in the MATLAB functions section in order to get `X_dmd`, our partially reconstructed data matrix, that only contains the background information from the video. We then watch this background video, a frame of which can be seen in Figure 1. We then subtract the background video from the original video `X` and watch that, which is the foreground video, a frame of which can be seen in Figure 2. We then follow the same exact steps for the Ski Drop video under the `Ski Drop` section of the MATLAB code.

4 Computational Results

A frame from the background of the Monte Carlo video can be seen in Figure 1. This one frame is very representative of the entire video, it essentially looks like this the entire time, which is good since that is what we would expect from the background. Though the beginning of the video is every clear and in focus, then it gets blurry (as seen in the frame), then it gets clear again. Looking at Figure 2 we can see a frame of the foreground from the video. Here we see pretty much all of the background is gone, except the edges of buildings and signs can still be seen. This is most likely because the video is shaking, so the computer 'sees' these edges moving and puts them in the foreground. Through the video, we can only see line on the racetrack through the cars when they drive by, which is very interesting, mus have to do with how hard of an edge there is between the paint of the line and the track.

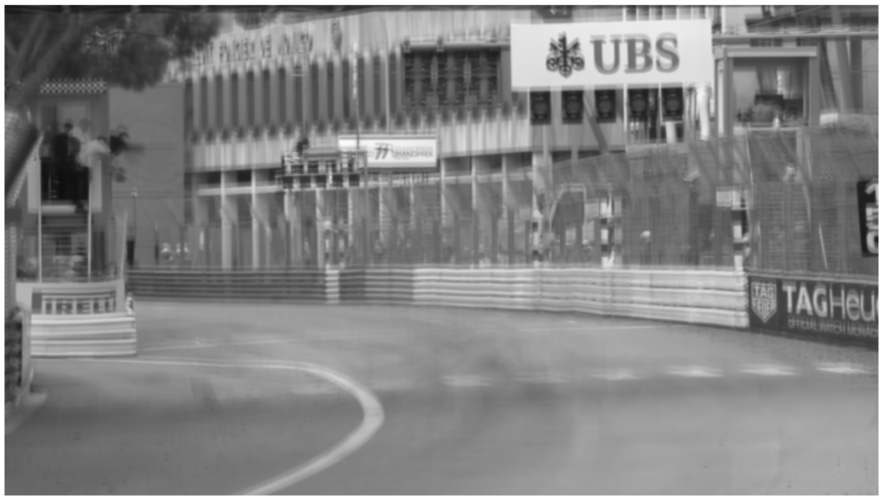

Figure 1: This is a frame from the video of the Monte Carlo background.

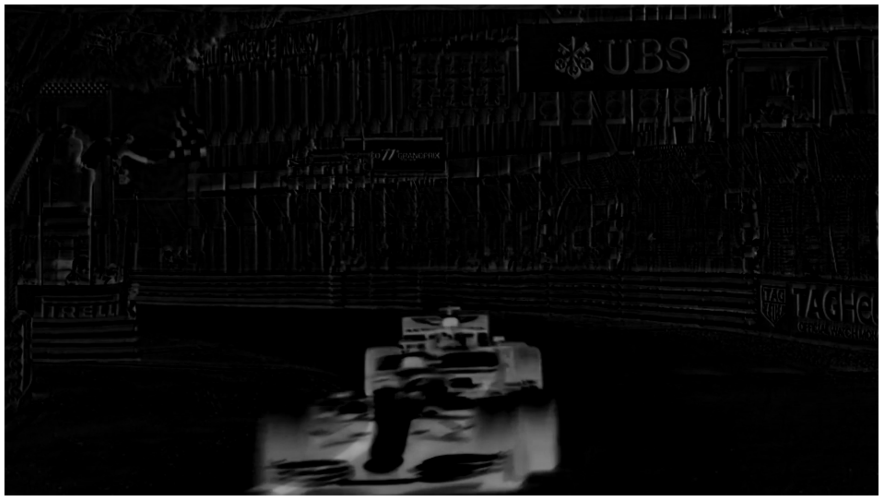

Figure 2: This is a frame from the video of the Monte Carlo foreground.

For the Ski Drop background, we can see a frame of it in Figure 3. Similar to the Monte Carlo background, this frame pretty stays the same throughout the entire video. Though this one does not get blurry, it stays the same the entire time. And for the foreground, we have a frame of it in Figure 4. It has the skier highlighted for the whole video pretty much surrounded in black. Then when the skier makes the drop and show puffs around them, they disappear from the foreground video then reappear a second later. Since the video is not shaking like the Monte Carlo was, we do not get all of the edges appearing in the foreground.

5 Summary and Conclusions

We were able to successfully use Dynamic Mode Decomposition (DMD) to remove the background from our Monte Carlo and Ski Drop videos, leaving only the foregrounds.

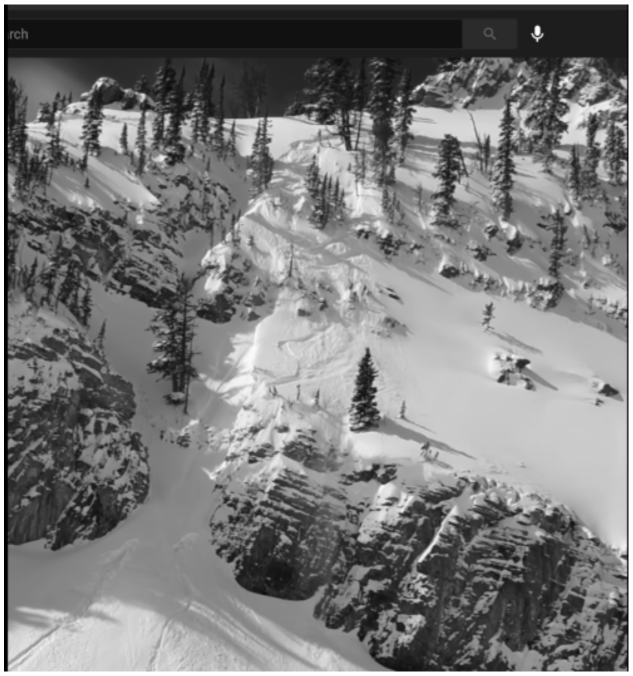

Figure 3: This is a frame from the video of the Ski Drop background.

Figure 4: This is a frame from the video of the Ski Drop foreground.

Appendix A MATLAB Functions

- `svd(A,'econ')` returns the SVD of matrix `A`, $[U, \Sigma, V^*]$, without the rows of zeros padding the Σ matrix.
- `dmd(X,X2,r,dt)` takes in `X` (data matrix), `Xshift` (shifted data matrix), `r` (target rank of SVD), and `dt` (time step size). Then applies DMD to our data matrix and returns a partial DMD reconstruction that only contains the background information.

Appendix B MATLAB Code

```
%% Prep
clear all; close all; clc
%% Monte Carlo
vidmp4 = VideoReader('monte_carlo_low.mp4');

video = read(vidmp4);
numFrames = get(vidmp4, 'NumFrames');
height = get(vidmp4, 'Height');
width = get(vidmp4, 'Width');


for j=1:numFrames
    vid_mat(:,j) = double(reshape(rgb2gray(video(:,:,:,j)), [], 1));
end

X = vid_mat(:,1:end-1);
Xshift = vid_mat(:,2:end);
r = 2;
t = linspace(0,numFrames, 2*numFrames);
dt = t(2) - t(1);

X_dmd = dmd(X,Xshift,r,dt);

 for j=1:numFrames - 1
 frame = reshape(X_dmd(:,j), height, width);
 frame = uint8(real(frame));
 imshow(frame); drawnow
 end

 foreground = X - X_dmd;
 for j=1:numFrames - 1
 frame = reshape(-foreground(:,j), height, width);
 frame = uint8(real(frame));
 imshow(frame); drawnow
 end

 %% Ski Drop
vidmp4 = VideoReader('ski_drop_low.mp4');

video = read(vidmp4);
numFrames = get(vidmp4, 'NumFrames');
height = get(vidmp4, 'Height');
width = get(vidmp4, 'Width');
```

```
for j=1:numFrames
    vid_mat(:,j) = double(reshape(rgb2gray(video(:,:,:,j)), [], 1));
end

X = vid_mat(:,1:end-1);
Xshift = vid_mat(:,2:end);
r = 2;
t = linspace(0,numFrames, 2*numFrames);
dt = t(2) - t(1);

X_dmd = dmd(X,Xshift,r,dt);

 for j=1:numFrames - 1
 frame = reshape(X_dmd(:,j), height, width);
 frame = uint8(real(frame));
 imshow(frame); drawnow
 end

 foreground = X - X_dmd;
 for j=1:numFrames - 1
 frame = reshape(-foreground(:,j), height, width);
 frame = uint8(real(frame));
 imshow(frame); drawnow
 end

%% function
function X_dmd = dmd(X,Xshift,r,dt)
[U, S, V] = svd(X, 'econ');
r = min(r, size(U,2));
U_r = U(:, 1:r);
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
A_tilde = U_r' * Xshift * V_r / S_r;
[W_r, D] = eig(A_tilde);
Phi = Xshift * V_r / S_r * W_r;
lambda = diag(D);
omega = log(lambda)/dt;
x1 =X(:, 1);
b = Phi\x1;
measurements_1 = size(X, 2);
time_dynamics = zeros(r, measurements_1);
t = (0:measurements_1 - 1).*dt;
for i = 1:measurements_1
    time_dynamics(:, i) = (b.*exp(omega*t(i)));
end
X_dmd = Phi * time_dynamics; % only background information
end
```