AMATH 482 Homework 5

David Warne

March 17, 2021

Abstract

Continuing with the study of Singular Value Decomposition, this assignment delves into a new method called Dynamic Mode Decomposition (or DMD for short). In brief, DMD is a dimensionality reduction that allows us to extract low-rank values from a dataset in order to disambiguate the underlying parts of the data.

1 Introduction and Overview

In this assignment, I am provided with two short movie clips: one of formula one cars coming around a bend in the Monaco Grand Prix, and one of a skier who is descending a large slope. The goal of this assignment is to read the data from these two video clips and perform the DMD process on them so that we can separate the foreground from the background. The final product for each movie after performing DMD will be a matrix containing the low-rank values of the original data, and one that is much more sparse, that contains the foreground data (that we achieve by subtracting the background data from the original dataset).

2 Theoretical Background

The most important part of this assignment is understanding the aforementioned Dynamic Mode Decomposition. To do so, it is important to understand how DMD connects to the Koopman operator. From [1], we can understand that the Koopman operator is "a linear operator that completely describes an autonomous nonlinear dynamical system". What this essentially means is that we can map "finite-dimensional nonlinear systems to infinite-dimensional linear system", which gives us the framework to "enable the synthesis of model reduction" [1]. In more brief terms, we are able to apply the Koopman operator multiple times to an infinitely large linear system in order to describe the behavior of a given nonlinear system. This is applicable to this assignment because that is essentially what we are doing: taking a finite nonlinear dataset (a movie), and reducing the dimensionality through DMD so that we can separate the foreground from the background.

For context, the properties of the videos should be as follows (I used the low resolution ones):

- Both videos should have pixel height of 540 and pixel width of 960
- The Monaco Grand Prix video (monte-carlo) should have 379 frames
- The ski video should have 454 frames

Below is the equation that I use to calculate low-rank DMD matrix (background)

$$X_{DMD}^{Low-Rank} = b_p \varphi_p e^{\omega_p t} \tag{1}$$

Where b is the initial condition for our system that we calculate through implememting a pseudo inverse of $\varphi_p b_p = X$ (where X contains the original spatiotemporal data that was extracted from the movie).

Below is the equation that I use to calculate the sparse DMD matrix (foreground)

$$X_{DMD}^{Sparse} = X - |X_{DMD}^{Low-Rank}| \tag{2}$$

Again, where X denotes the original spatiotemporal data of the movie.

3 Algorithm Implementation and Development

This process can be implemented in a relatively simple manner, and I will describe the approach that I used in 3 algorithms. Note that all three of these algorithms would be for a single video, and have been applied separately to each video for this assignment. Algorithm 1 will explain how to extract the data from the movies and reshape them into a our spatiotemporal matrix X that we can perform SVD/DMD on. Algorithm 2 will describe how to perform the SVD/DMD to calculate our low rank matrix $X_{DMD}^{Low-Rank}$, and sparse matrix X_{DMD}^{Sparse}. Algorithm 3 will explain how we can reshape these matrices to view the separated foreground and background images.

Algorithm 1: Extracting and Setting Up Spatiotemporal Matrix X

- Make sure the video file is in the working directory.
- Load the file using the built-in VideoReader() function
- Extract the frame data from the return of VideoReader() using read()
- Extract pixel width and pixel height, and calculate total number of pixels by multiplying these values together.
- Initialize a Δt variable that is the duration/numFrames
- Create a vector of timestamps using linspace()
- Initialize an empty spatiotemporal matrix X that is numPixels x numFrames
- For each frame, implement rgb2gray() and im2double(), then reshape the result into a vector that can be stored in X.

4 Computational Results

Overall, I was able to get accurate results when separating the foreground and background for both the monte carlo and ski videos.

Originally, I tried to implement the method in the spec that describes how extracting the negative values from X_{DMD}^{Sparse} and adding those values R to $|X_{DMD}^{Low-Rank}|$ would "account for magnitudes of the complex values from the DMD reconstruction"; however, I found that whenever I added those values into $|X_{DMD}^{Low-Rank}|$, the foreground images would appear again when the only thing that I expected was the background.

This occurrence led me to use the built in MATLAB function mat2gray(), which sets all elements in a matrix between 0 and 1. This way, I was able to compensate for the negative values, and as it turned out, get an even better picture that helped me more clearly see the separated foreground images.

In terms of numerical computations that might be relevant, I found the following during my implementation of SVD/DMD:

- For the monte carlo video, I used mode 299 to calculate $X_{DMD}^{Low-Rank}$, whose corresponding eigenvalue had minimum real part (closest to 0).
- For the ski video, I used mode 186 to calculate $X_{DMD}^{Low-Rank}$, whose corresponding eigenvalue had minimum real part (closest to 0).

While I could have utilized more than one mode for the background extraction, I found that in both cases there was one predominant mode that accounted for most of the background, and those were the modes that I ended up using in each case.

In terms of the outputs that were produced and how the actual separation of the foreground and background looked in each case, refer to the following figures:

Algorithm 2: Implementing SVD/DMD to get $X_{DMD}^{Low-Rank}$ and X_{DMD}^{Sparse}

- Begin by intializing $X_1^{M-1} = X(:, 1 : end - 1)$ and $X_2^M = X(:, 2 : end)$.
- Perform SVD on X_1^{M-1}, this allows us to compute the eigenvalues of $\tilde{S}$ (which is a matrix that is similar to the Koopman Operator A).
- Compute $\tilde{S} = U * X_2^M * V * \Sigma^{-1}$. Note that you may need to transpose U for the dimensions to work out.
- Use eig() on $\tilde{S}$ to extract the eigenvectors and eigenvalues
- Calculate φ by multiplying U by the eigenvectors of $\tilde{S}$.
- Save the eigenvalues in a vector, apply log() to them and divide by the previously initialized Δt.
- Save the index of the mode whose eigenvalue's real part is the minimum (this is what correlates to the background information since an eigenvalue with 0 real part correlates to no change with time).
- Calculate b (our initial condition) by implementing a pseudoinverse. This can be done by setting $b = \varphi \backslash X(:, 1)$.
- Initialize a matrix to store the DMD modes that is length(b) x numFrames
- For numFrames, compute the time dynamics by multiplying the initial condition by the eigenvalues from $\tilde{S}$. The equation should look something like this: $u_{modes} = be^{\omega t}$. Where omega is the eigenvalue(s) with minimum real part.
- Now that we have b and $e^{\omega t}$, we can plug these values and the previously calculated φ into equation 1 to calculate $X_{DMD}^{Low-Rank}$. Make sure to take the real values of the resulting matrix.
- Subtract the absolute values of $X_{DMD}^{Low-Rank}$ from the original spatiotemporal matrix X to remove the background and achieve the foreground values in X_{DMD}^{Sparse}.

Algorithm 3: Visualizing the Separated Foreground and Background

- Use mat2gray() on X_{DMD}^{Sparse} to cast negative values to positive values between 0 and 1
- For each frame, reshape either X_{DMD}^{Sparse} or $X_{DMD}^{Low-Rank}$ (depending on which one you want to visualize), back into their original dimension of height x width that was previously calculated in Algorithm 1.
- Before the end of the for loop, use imshow() on the reshaped matrix and drawnow. You should get a frame by frame reconstruction of the foreground or background (again, depending on which matrix you choose to visualize).

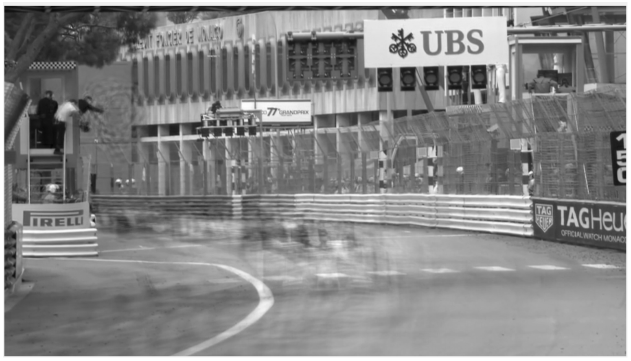

Figure 1: Background of Monte Carlo Video

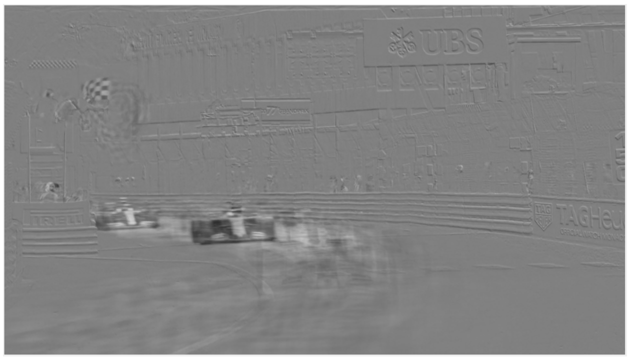

Figure 2: Foreground of Monte Carlo Video: Frame 150

- See Figure 1 for the background that was extracted from the monte carlo video. This is essentially a still frame for all of the frames since the eigenvalue that we used to calculate this has real part 0, meaning that the image will not change with time. We can also see blurred motion, which is expected since the background can also be thought of as an average of all of the moving foreground.
- See Figures 2, 3, and 4 for foreground snapshots of frames 150, 250, and 350 of the monte carlo video, respectively. We can see that the background is almost completely eliminated, but the subject of the foreground (the formula 1 cars) are still evident and moving as time progresses.
- See Figure 5 for the background that was extracted from the ski video. Similar to the monte carlo video, this background is essentially a still image since the eigenvalue that we used to calculate it has real part 0, again, meaning that it will not change with time. We can also see how the skier is not present, which is a good thing because that means we were able to properly exclude the foreground.
- See Figures 6, 7, and 8 for foreground snapshots of frames 198, 295, and 402 of the ski video, respectively. We can see that the background is almost completely eliminated, but the subject of the foreground (the skier) is still evident and moving as time progresses.

5 Summary and Conclusions

In conclusion, I found the implementation of Dynamic Mode Decomposition to be extremely rewarding. At first, I was intimidated by the task at hand, but after successfully separating the foreground from the background in both of the given videos, I can say that I am blown away by the power of this method and the intuitive nature that it possesses. If I were to improve anything about my approach, I would maybe include more time-insensitive modes to see if it alters the final result, but for just using 1 eigenvalue I am still thoroughly impressed by the accuracy of separation.

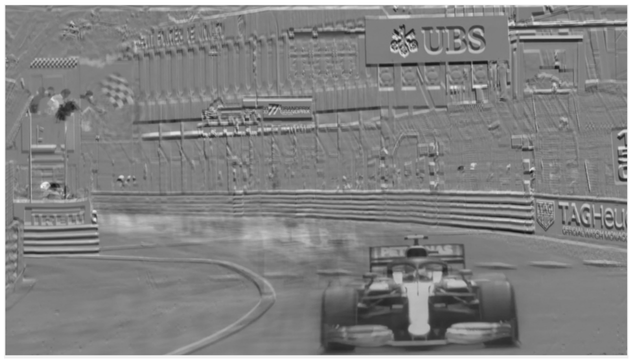

Figure 3: Foreground of Monte Carlo Video: Frame 250

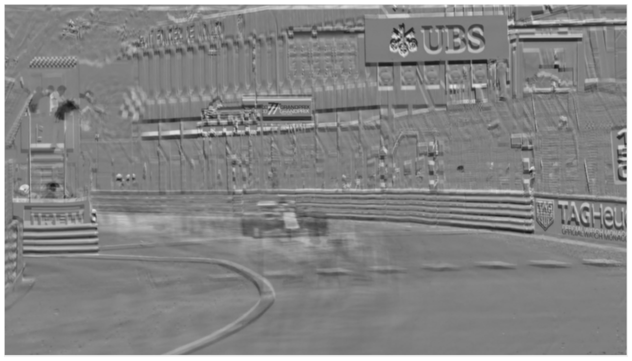

Figure 4: Foreground of Monte Carlo Video: Frame 350

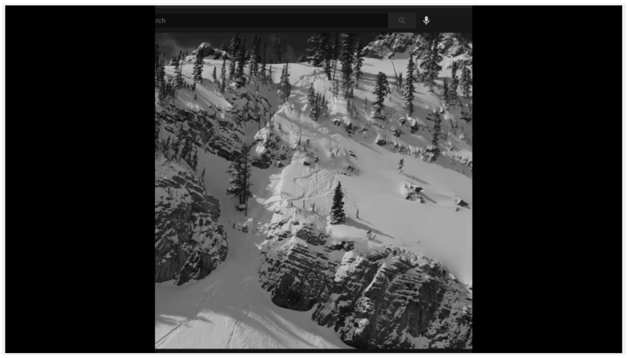

Figure 5: Background of Ski Video

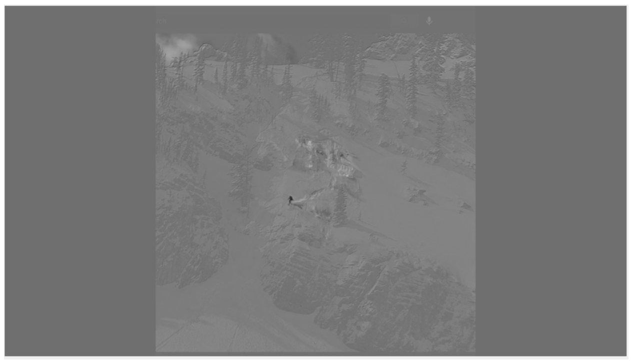

Figure 6: Foreground of Ski Video: Frame 198

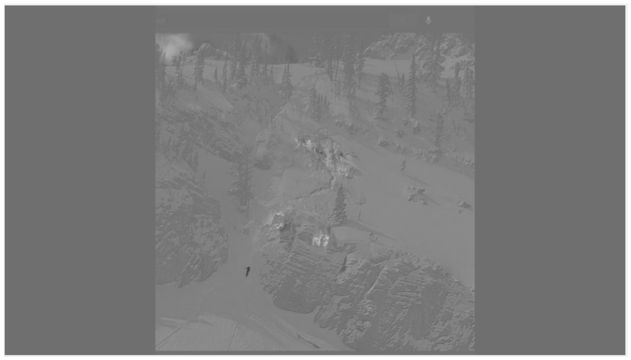

Figure 7: Foreground of Ski Video: Frame 295

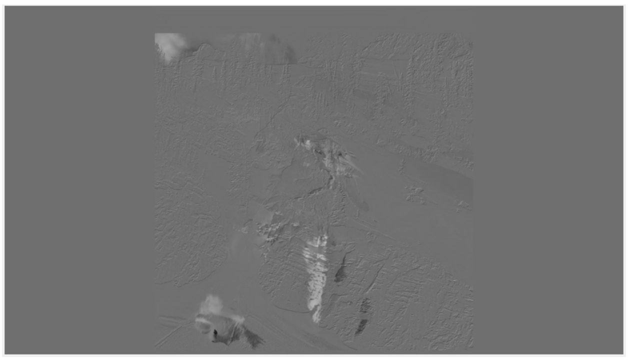

Figure 8: Foreground of Ski Video: Frame 402

References

[1] *Connecting Dynamic Mode Decomposition and Coopman Operator*. URL: `https://faculty.washington.edu/kutz/page1/page13/`.

Appendix A MATLAB Functions

Here are the main functions that I used in this assignment:

- `[U,S,V] = svd() allows us to compute singular value decomposition on our matrix of x and y components.`
- `OBJ = VideoReader(FILENAME) constructs a multimedia reader object, OBJ, that can read in video data from a multimedia file.`
- `[V,D] = eig(A) produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that A*V = V*D.`
- `I = mat2gray(A,[AMIN AMAX]) converts the matrix A to the intensity image I. The returned matrix I contains values in the range 0.0 (black) to 1.0`
- `reshape(X,M,N) or reshape(X,[M,N]) returns the M-by-N matrix whose elements are taken columnwise from X.`

Appendix B MATLAB Code

Here is the MATLAB code that was used to produce the information in this assignment.

```matlab
% Assignment 5 Script
%%
clear all; close all; clc;

%% Load in the Data

% read the data
monte = VideoReader('monte_carlo_low.mp4');
vidFrames = read(monte);

% store important information
num_frames = size(vidFrames,4);
width = monte.Width;
height = monte.Height;
num_pixels = width*height;

% calculate our dt variable
total_time = monte.Duration;
dt = total_time/num_frames;

% create a time vector to use later when computing fourier values
t = linspace(0, total_time, num_frames);

% intialize a matrix for reshaping purposes
X_vec = zeros(num_pixels, num_frames);

% access each individual frame and convert it into a vector that stores the
% total number of pixels, then successively add these vectors into a
% matrix that we can perform SVD on

for j = 1:num_frames
    X = vidFrames(:,:,:,j);
    I = rgb2gray(X);
    I = im2double(I);
    vectorized = reshape(I(:,:), num_pixels, 1);
    X_vec(:,j) = vectorized;
end

%% Apply DMD
% implement method from lecture
X1 = X_vec(:, 1:end-1);
X2 = X_vec(:, 2:end);

[U, S, V] = svd(X1, 'econ');

Stilde = U'*X2*V*diag(1./diag(S));
[eV, D] = eig(Stilde); % compute eigenvalues + eigenvectors
Phi = U*eV;

mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;


%% Finding the eigenvalues whose real parts are basically 0

[value, background_modes_idx] = min(abs(real(omega)));

%% Calculating DMD

y0 = Phi\X_vec(:,1); % pseudoinverse to get initial conditions

u_modes = zeros(length(y0),length(t));
for iter = 1:num_frames
   u_modes(:,iter) = y0.*exp(omega(background_modes_idx)*t(iter));
end
u_dmd = real(Phi*u_modes);


%% Find the sparse matrix by subtracting the low-rank matrix from our original x_vec
%
x_sparse = X_vec - abs(u_dmd);
x_sparse = mat2gray(x_sparse);

%% Visualizing the foreground

for j = 1:num_frames
    visual = reshape(x_sparse(:, j), height, width);
    imshow(visual); drawnow
end

%% Visualizing the background
```

```
for j = 1:num_frames
    visual2 = reshape(u_dmd(:, j), height, width);
    imshow(visual2); drawnow
end

%% Plotting several frames from the foreground to show that the cars exist
figure(1)
imshow(reshape(x_sparse(:, 150), height, width));
figure(2)
imshow(reshape(x_sparse(:, 250), height, width));
figure(3)
imshow(reshape(x_sparse(:, 350), height, width));


%% Ski Video
%%
clear all; close all; clc;

%% Load in the Data

% read the data
ski = VideoReader('ski_drop_low.mp4');
vidFrames = read(ski);

% store important information
num_frames = size(vidFrames,4);
width = ski.Width;
height = ski.Height;
num_pixels = width*height;

% calculate our dt variable
total_time = ski.Duration;
dt = total_time/num_frames;

% create a time vector to use later when computing fourier values
t = linspace(0, total_time, num_frames);

% intialize a matrix for reshaping purposes
X_vec = zeros(num_pixels, num_frames);

% access each individual frame and convert it into a vector that stores the
% total number of pixels, then successively add these vectors into a
% matrix that we can perform SVD on

for j = 1:num_frames
    X = vidFrames(:,:,:,j);
    I = rgb2gray(X);
    I = im2double(I);
    vectorized = reshape(I(:,:), num_pixels, 1);
    X_vec(:,j) = vectorized;
end

%% Apply DMD
% implement method from lecture
```

```
X1 = X_vec(:, 1:end-1);
X2 = X_vec(:, 2:end);

[U, S, V] = svd(X1, 'econ');

Stilde = U'*X2*V*diag(1./diag(S));
[eV, D] = eig(Stilde); % compute eigenvalues + eigenvectors
Phi = U*eV;

mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;

%% Finding the eigenvalue that is closest to 0

[value, background_mode] = min(abs(real(omega)));

%% Finding DMD modes
y0 = Phi\X_vec(:,1); % pseudoinverse to get initial conditions

u_modes = zeros(length(y0),length(t));
for iter = 1:num_frames
   u_modes(:,iter) = y0.*exp(omega(background_mode)*t(iter));
end
u_dmd = real(Phi*u_modes);


%% Find the sparse matrix by subtracting the low-rank matrix from our original x_vec
%
x_sparse = X_vec - abs(u_dmd);
x_sparse = mat2gray(x_sparse);

%% Visualizing the foreground

for j = 1:num_frames
    visual = reshape(x_sparse(:, j), height, width);
    imshow(visual); drawnow
end

%% Visualizing the background
for j = 1:num_frames
    visual2 = reshape(u_dmd(:, j), height, width);
    imshow(visual2); drawnow
end

%% Plotting several frames to show that the background has gotten rid of the skiier
figure(1);
imshow(reshape(u_dmd(:, 198), height, width));
figure(2);
imshow(reshape(u_dmd(:, 295), height, width));
figure(3);
imshow(reshape(u_dmd(:, 402), height, width));

%% Plotting several frames from the foreground to show that the skier exists
figure(4)
```

```
imshow(reshape(x_sparse(:, 198), height, width));
figure(5)
imshow(reshape(x_sparse(:, 295), height, width));
figure(6)
imshow(reshape(x_sparse(:, 402), height, width));
```

Script for Assignment 5