AMATH 582 Homework 5

Joseph David

March 17, 2021

Abstract

We apply Dynamic Mode Decomposition (DMD) to two videos, `monte-carlo.mp4` and `ski-drop.mp4` in order to separate the objects in the foreground from the background.

1 Introduction and Overview

In this project we explore the data analytic technique of Dynamic Mode Decomposition, which is primarily used to exploit low-dimensionality, and apply it to two videos. In the Theoretical Background section, we describe the mathematics of DMD and how it achieves the desired goal. In the Algorithm Implementation section, we discuss how DMD was implemented in `MATLAB` to analyze the videos. In the Computational Results section we show how well DMD performed in separating foreground from background objects by presenting frames obtained from the `MATLAB` program.

2 Theoretical Background

The fundamental idea behind DMD is that we have experimental data vectors $x_1, \dots x_M$ each of length N, such that each vector is a "snapshot" of our system at a given time, and such that the timing between each snapshot is ssome constant Δt. We would like to find the best matrix A such that $Ax_j \approx x_{j+1}$ for all j, thus imposing the condition that the change in the system is linear.

Let $X = [x_1 \cdots x_M]$ be the $N \times M$ matrix that contains the data vectors as columns, and define $X_1^{M-1} = [x_1 \cdots x_{M-1}]$ and $X_2^M = [x_2 \cdots x_M]$. We may express the problem as finding A such that

$$AX_1^{M-1} = X_2^M. \tag{1}$$

Using pseudo-inverses, one may compute $A = X_2^M (X_1^{M-1})^+$. However, this would result in an extremely large matrix A ($(N-1) \times (N-1)$, and we are generally working with $N >> M$). Therefore, we compute the rank r truncation of the SVD of X_1^{M-1}

$$X_1^{M-1} = U_r \Sigma_r V_r^*. \tag{2}$$

Using this, we may find a matrix $\widetilde{A}$ that is similar to A but with rank r,

$$\widetilde{A} = U^* A U = U^* X_2^M V \Sigma^{-1}. \tag{3}$$

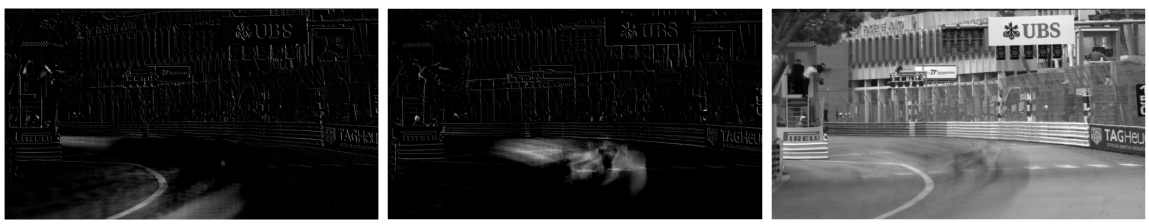

Figure 1: Select frames from the dynamic mode decomposition of `monte-carlo.mp4`. The first two are foreground and the last is background. The first and last correspond to the same frame.

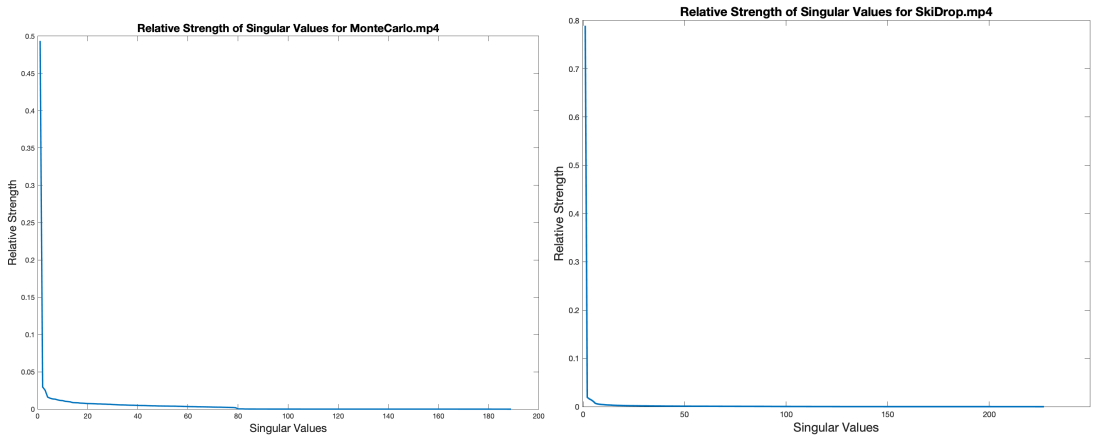

Figure 2: The relative singular values for both videos.

Then, we may compute the eigenvector matrix W and eigenvalue matrix D of $\widetilde{A}$. Finally, we compute the DMD modes

$$\Phi = X_2^M V \Sigma^{-1} W. \tag{4}$$

Thus, our model becomes

$$x(t) = \Phi \exp(Dt) b = \sum_{k=1}^{r} \phi_k \exp(\omega_k t) b_k, \tag{5}$$

where ϕ_k are the eigenvectors and ω_k the eigenvalues of $\widetilde{A}$. To compute b, we simply compute the model at $t = 0$ to obtain $x(0) = \Phi b$ so $b = \Phi x(0)^{-1}$. Thus, for any time t this will give us a prediction for the state of the system at t.

3 Algorithm Implementation and Development

We used `MATLAB`'s built in `svd` function to compute the SVD of X_1^{M-1}. We also used `VideoReader` in order to load the mp4 files into `MATLAB` as well as extract the video information such as number of frames and the dimensions of the frames. We also used `imshow` to view the foreground and background frames, as well as built-in functionality for automatically scaling the pixel intensities in order to create more contrast. This was useful for viewing the efficacy of DMD on the videos.

Algorithm 1: Dynamic Mode Decomposition Algorithm

Load mp4 files, convert to black and white, and reshape into columns of data matrix X.
Compute SVD decomposition of X and determine what rank r to use for DMD.
Compute dynamic mode decomposition $u.dmd$.
From eigenvalues of $Atilde$, determine which has smallest norm to compute $u.dmd.lr$.
Compute $u.dmd.sparse = u.dmd - |u.dmd.lr|$.
for i = 1 : height*width **do**
  for i = j : numFrames **do**
    if $u.dmd(i, j) < 0$, then set $R(i, j) = u.dmd.sparse(i, j)$.
  end for
end for
Set $u.dmd.lr = |u.dmd.lr| + R$ and $u.dmd.sparse = u.dmd.sparse - R$. These are the background and foreground frames, respectively.

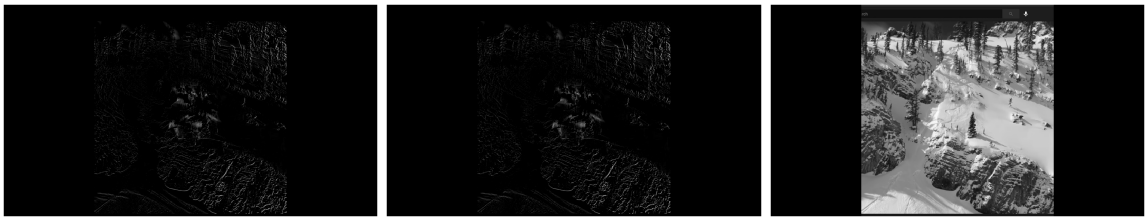

Figure 3: Select frames from the dynamic mode decomposition of `ski-drop.mp4`. The first two are foreground and the last is background.

4 Computational Results

We first computed the SVD of X_1^{M-1} for both videos and plotted the relative strength of their singular values in order to determine which rank r to project on to. In Figure 2, we see that for both videos, there was one dominant mode, so we chose $r = 3$ in both cases.

In Figure 4, we show the three eigenvalues obtained for $\widetilde{A}$ in each video. In both, we see there is one clear smallest eigenvalue, which is selected as the background mode.

Lastly, in Figures 1 and 3 we see the results of dynamic mode decomposition in separating the foreground and backgrounds in `monte-carlo.mp4` and `ski-drop.mp4`, respectively. As was expected, the cars in `monte-carlo.mp4` were the brightest part of the frames for the foreground. Furthermore, comparing that same frame to its background frame we see that the car was mostly removed from the image as expected. DMD appeared successful for `ski-drop.mp4` as well, but since the foreground (the skier) was small, it is difficult to see it. However, inspecting the background frames we see that the skier was successfully removed from the video, suggesting that the method was at least partially successful.

5 Summary and Conclusions

In conclusion, DMD proved to be mostly successful at separating the videos into its foreground and background. In both experiments, the background is clear and most of the objects in the foreground had been removed from the picture. The foreground in `ski-drop.mp4` is not so clear, but I believe with the correct image processing tools this could be improved.

Once again, SVD was central to the data analysis method. In this instance, SVD was used to select a few of the dominant modes which drastically reduces the size of the system and allows for much faster computations of eigenvectors and eigenvalues.

Appendix A MATLAB Functions

- `VideoReader(filename)` returns an object `vid` from which one has functions such as `.NumFrames`, `.Duration`, `.Height`, `.Width`, and `readFrame` to extract video data.
- `svd(A)` computes the SVD of matrix `A` and returns the decomposition matrices `U`, `S`, and `V`.

```
omega =

  -0.0009 + 0.0000i
  -0.4636 + 2.4841i
  -0.4636 - 2.4841i
```

```
omega =

   0.0028
  -1.7639
  -4.1399
```

Figure 4: Eigenvalues of $\widetilde{A}$ for both videos.

- `eig(A)` computes the eigenvectors and eigenvalues of `A` and returns the corresponding matrices.
- `imshow(A,[])` plots the matrix `A` as an image and scales the pixel intensity so that the minimum is black and the maximum is white.

Appendix B MATLAB Code

```
clear all; close all; clc

%% load video into matrix X and store video attributes
vid = VideoReader('monte_carlo_low.mp4');

numFrames = floor(vid.NumFrames/2);
duration = vid.Duration/2;
height = vid.Height;
width = vid.Width;

t = linspace(0, duration, numFrames); dt = t(2)-t(1);
X=zeros(width*height,numFrames);
for j = 1 : numFrames
    X(:,j) = (reshape(im2double(rgb2gray(readFrame(vid))),width*height,1));
end
clear vid;

%% find singular value spectrum
[u,s,v] = svd(X,'econ');
figure(1)
plot(diag(s)/sum(diag(s)),'Linewidth',[2])
title('Relative Strength of Singular Values for SkiDrop.mp4','FontSize',16)
xlabel('Singular Values','FontSize',16); ylabel('Relative Strength','FontSize',16)
clear u; clear s; clear v;

%% from singular values, determine how many modes r to take and compute DMD
r=3;
X1 = X(:,1:end-1); X2 = X(:,2:end);
[U2,Sigma2,V2] = svd(X1,'econ');
U = U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);
clear U2; clear Sigma2; clear V2;

Atilde = U'*X2*V/Sigma;
[W,D] = eig(Atilde);
Phi = X2*V/Sigma*W;

mu = diag(D);
omega = log(mu)/dt;

clear U; clear Sigma; clear V;

u0=X(:,1);
y0 = Phi\u0;  % pseudo-inverse initial conditions
u_modes = zeros(r,length(t));
for iter = 1:length(t)
     u_modes(:,iter) =(y0.*exp(omega*t(iter)));
end
u_dmd = Phi*u_modes;

%% look at omega and determine which mode has smallest norm, use as background mode
u_dmd_lr = y0(1) * Phi(:,1) * exp(omega(1)*t);
u_dmd_sparse = u_dmd - abs(u_dmd_lr);
```

Listing 1: MATLAB code.

```
R = zeros(height*width,numFrames);
for i = 1 : height*width
    for j = 1 : numFrames
        if u_dmd_sparse(i,j) < 0
            R(i,j) = u_dmd_sparse(i,j);
        end
    end
end

u_dmd_lr = abs(u_dmd_lr)+ R;
u_dmd_sparse = u_dmd_sparse - R;

clear u_modes; clear y0; clear u0; clear mu; clear D; clear W; clear Atilde;

for j = 120
    %minPix = min(real(u_dmd_sparse(:,j))); maxPix = max(real(u_dmd_sparse(:,j)));
    figure(2)
    imshow(reshape(real(u_dmd_sparse(:,j)),height,width), [])
    figure(3)
    imshow(reshape(real(u_dmd_lr(:,j)),height,width), [])
end
```

Listing 2: MATLAB code (continued).