AMATH 482 Homework 5

Zach Zlepper

March, 2021

Abstract

In this paper we will use the Dynamic Mode Decomposition to separate a video into a Foreground and Background representation. This algorithm includes the SVD, investigating eigenvalues, and iterating through time dynamics. The true power of the DMD comes from the fact that we build governing equations from data. This allows us to have a system that describes a video. We successfully separate the videos, however, the results have some shortcomings and I suggest fixes such as filtering and not fully building the low rank matrix by removing R.

1 Introduction and Overview

The question we are trying to answer is if we can separate a video into its foreground and background components. In the following paper, we discuss the DMD algorithm and how to apply it video data. We investigate the DMD on a Monte Carlo video and Skier video.

2 Theoretical Background

We are using the Dynamic Mode Decomposition to find a spatial basis of modes where the time dynamics are just exponential functions. This will enable us the ability to forecast the system using the properties of growth and decay.

2.1 Setup

We need to have data that evolves through time. Let N = number of spacial points per time unit and M = the number of time iterations.

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

Let a snapshot be denoted as

$$\overrightarrow{U}(x, t_m) \tag{2}$$

where $x \in [1, n]$ for each $m = 1, ..., M$.

We can then use these snapshots to form columns of data such that

$$X = [U(x, t_1), ...U(x, t_M)]$$

and

$$X_j^k = [U(x, t_j), ...U(x, t_k)]$$

where X_j^k is the columns j through k of the full snapshot matrix X.

2.2 Koopman Operator

Let A be linear, time dependent operator called a Koopman operator such that

$$x_{j+1} = Ax_j \tag{3}$$

where j are the indices of the specific data collected. A iterates our time from j to j+1. The Koopman operator is a global operation.

2.3 DMD

Lets consider the matrix

$$X_1^M - 1 = [x_1 ... x_M - 1]$$

where x_j is the snapshot at time t_j. The Koopman operator allows us to write,

$$X_1^M - 1 = [x_1 ... A^{M-2} x_1]$$

By applying A to the vector x_1, we form a basis called the Krylov subspace. This way we can connect M-1 snapshots to x_1. We can rewrite the above matrix as

$$X_2^M = AX_1^{M-1} + re_{M-1}^T$$

where $e_M - 1$ is a zero vector with a 1 at the $M - 1$ component. It is worth noting that x_M isn't included in our Krylov subspace so we must ad a residual vector r to account for this. In order to find A we will find another matrix with the same eigenvalues. We will take the SVD and get

$$X_2^M = U\Sigma V^*$$

So,

$$X_2^M = AU\Sigma V^* + re_{M-1}^T$$

Multiplying by U^* and U we get

$$U^* AU = U^* X_2^M V \Sigma^{-1}$$

since $U * r = 0$. Let

$$U^* X_2^M V \Sigma^{-1} =: \tilde{S}$$

Note: We can take the SVD of our data matrix and reduce the dimensions initially. This will reduce the size of our SVD taken in the prior step as well as allow our Σ matrix to be invertable.

We can see that $\tilde{S}$ and our matrix A are similar which means that they have the same eigenvalues. Also, if y is an eigenvector of $\tilde{S}$ then Uy is an eigenvector of A.

$$\tilde{S} y_k = u_k y_k \tag{4}$$

Thus, the DMD modes or the eigenvectors of A are

$$\psi_k = U y_k$$

We can write out our eigenbasis

$$X_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \psi diag(e^{\omega_k t}) b \tag{5}$$

Using our initial conditions we can solve for b

$$x_1 = \psi b \longrightarrow b = \psi^\dagger x_1$$

at t = 0. This step is what allows us to iterate through time.

3 Algorithm Implementation and Development

Our algorithm described in algorithm 1 shows us how to perform a DMD as well as separate the matrix into a sparse representation as well as a low rank approximation. In our computational result we see that the low rank approximation looks like our background and our foreground is represented by the sparse matrix.

Algorithm 1: DMD

1.Sample data at N locations M times. These snapshots should be evenly spaced in time.
2.Create Matrix X with sampling
3.Reduce rank of matrix by taking SVD and finding significant energy
4.From X, construct X_1^{M-1} and X_2^M
5.Compute the SVD of X_1^{M-1}
6.Create matrix $U^* X_2^M V \Sigma^{-1} = \tilde{S}$ and find its eigenvalues and eigenvectors
7.Identify eigenvalues close to zero which represent the background
8.Use initial snapshot x_1 and pseudoinverse of ψ to find b_k
9.Compute future solutions with the DMD modes with their projections from the initial conditions and the time dynamics in the eigenvalues close to zero of the $\tilde{S}$ Matrix
10.Subtract the absolute value of our background time dynamics matrix from our X_1^{M-1} matrix to create a sparse matrix
11.Calculate the residual and that to the absolute value of our background matrix
12.Subtract the residual from the sparse matrix
13.Plot

Figure 1: Background of Monte Carlo

Figure 2: Foreground of Monte Carlo

4 Computational Results

We calculated a sparse matrix which corresponds to our background and a low rank matrix which represents our foreground. In order to build our low rank matrix, we needed to find out which eigenvalues were close to zero which we see in the Eigenvalues plots. In our building of the sparse matrix and low rank matrices, we need to add residuals back in. Our results are a little strange since the low rank matrix without the residual seems to do better than the low rank matrix with the residual.

We show images from the Monte Carlo Video in Figures 1-4. In the Monte Carlo video, we only have one low rank mode near zero. Our low rank approximation does a really good job of selecting the background. However, when we add R back in some of the foreground appears. The sparse matrix with R subtracted does an ok job of finding the car but it is pretty dark and hard to see.

We show images from the Skier Video in Figures 5-8. In the skier video, we see we have a few eigenvalues near zero in our eigenvalue plot. This means we will have more modes than the Monte Carlo Video for the low rank matrix. For the skier image, I applied a filter to bring them out in the foreground. Without the filter, the image is black. With the filter, we can see the skier in a clear fashion. Again, we see that the low rank matrix without R is better than with R. Parts of the foreground seem to come back in the image when we add R to the background.

5 Summary and Conclusions

Overall DMD, does a decent job of separating the foreground and the background from images. We showed this algorithm on two videos and described some fixes to improve the results such as filters and not putting R back into the low rank approximation.

Figure 3: Background without R

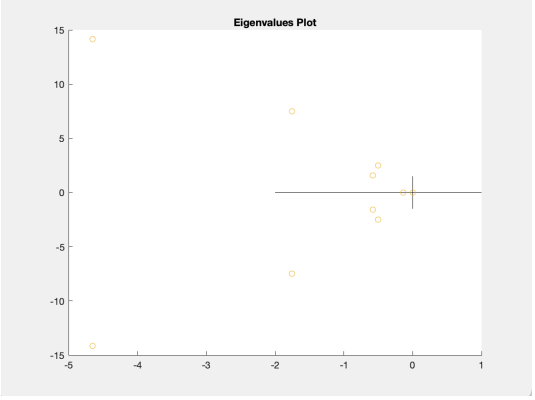

Figure 4: Eigenvalues for Monte Carlo

Figure 5: Background of ski

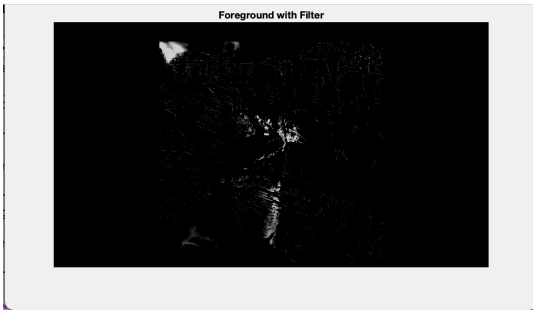

Figure 6: Foreground of Ski with Filter

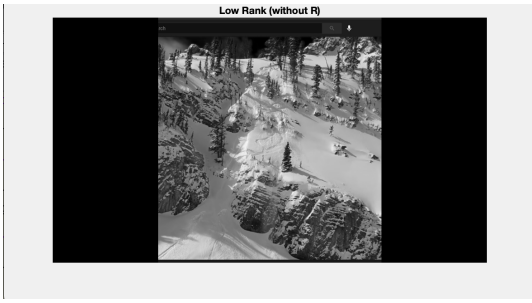

Figure 7: Background of ski without R

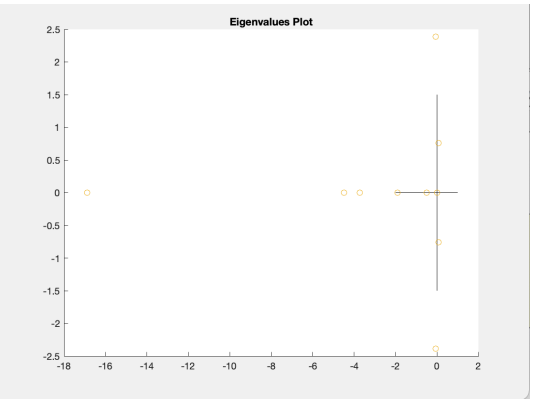

Figure 8: Eigenvalues for Ski

Appendix A MATLAB Functions

"diag(X)" : Returns a column vector of the diagonal values of a matrix.
"eig(X)" : Finds the Eigenvalues and Eigenvectors of X
"find(A)" : Finds indices and values of nonzero elements
"imshow(A)" : shows image A
"svd(X, 'econ') : computes the SVD of the matrix X.
"VideoReader('name') : reads in Video
"zeros(A,B,C)" : Creates a matrix filled with zeros of size "A x B x C".

Appendix B MATLAB Code

```
clear all; close all; clc;

%% Loading the videos, find svd spectrum
ski = VideoReader('ski_drop_low.mp4');
video = read(ski);
frames = ski.NumFrames;
[n m h1 h2] = size(video);
dt = 1/ski.Framerate;
t = 0:dt:ski.Duration;
%%

for j = 1:frames(1)
    ski_reshape = double(reshape(video(:,:,1,j), n*m, 1));
    v_ski(:,j) = ski_reshape;
end


%%

%Reduce data size
rank = 10;

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

[u, s, v] = svd(X1,'econ');
U = u(:,1:rank);
Sigma = s(1:rank, 1:rank);
V = v(:, 1:rank);

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

%%
figure()
hold on
plot([-2 1], [0 0], 'k')
plot ([0 0], [-1.5 1.5], 'k')
plot(real(omega), imag(omega),'o')
title('Eigenvalues Plot')
%%
thresh = .008;
bg = find(abs(omega) < thresh);
omega_bg = omega(bg);
phi_bg = Phi(:,bg);
%%
```

Listing 1: DMD code

```
y0 = phi_bg\X1(:,1);

mn1 = size(X1, 2);

t = (0:mn1-1)*dt;
u_modes = zeros(1,mn1);

for iter = 1:mn1
    u_modes(:,iter) = y0.*exp(omega_bg*t(iter));
end
u_dmd = phi_bg*u_modes;

%%
Xsparse = X1 - abs(u_dmd);

R = Xsparse.*(Xsparse<0);

X_bg = R + abs(u_dmd);
X_fg = Xsparse - R;

X_reconstructed =  X_fg + X_bg;

filterXS = uint8((Xsparse.*(Xsparse>500) - R)).* 5-uint8(50);
%%
figure(2)
    imshow(uint8(reshape(X_reconstructed(:,25), [], 960)))
    drawnow
    title("Full Video");
figure(3)
    imshow(uint8(reshape(X_bg(:,25), [], 960)))
    drawnow
    title("Background (R added)");
figure(4)
    imshow(uint8(reshape(X_fg(:,25), [], 960)))
    drawnow
    title("Foreground (R Subtracted)");
figure(5)
    fig = reshape(filterXS(:,25), [], 960);
    imshow(fig)
    drawnow
    title("Foreground with Filter");
figure(6)
    imshow(uint8(reshape(u_dmd(:,25), [], 960)))
    drawnow
    title("Low Rank (without R)");
figure(7)
    imshow(uint8(reshape(Xsparse(:,25), [], 960)))
    drawnow
    title("Sparse");
```

Listing 2: DMD code

```
%%
figure(2)
for j = 1:100
    imshow(uint8(reshape(X_reconstructed(:,j), [], 960)))
    drawnow
end
%%
figure(3)
for j = 1:100
    imshow(uint8(reshape(X_bg(:,j), [], 960)))
    drawnow
end
%%
figure(4)
for j = 1:25
    imshow(uint8(reshape(X_fg(:,j), [], 960)))
    drawnow
end
%%
figure(2)
for j = 1:100
    fig = reshape(filterXS(:,j), [], 960);
    imshow(fig)
    drawnow
end
%%
figure(2)
for j = 1:10
    imshow(uint8(reshape(u_dmd(:,j), [], 960)))
    drawnow
end
%%
figure(2)
for j = 1:100
    fig = reshape(Xsparse(:,j), [], 960);
    imshow(fig)
    drawnow
end
```

Listing 3: DMD code