Assignment 5: Á La Dynamic Mode

William Ojemann
AMATH 482
March 14, 2021

Abstract

Non-linear systems have rested on the cutting edge of applied mathematics for as long as the field has existed largely in part because of their inability to be characterized by traditional linear decomposition methods. Dynamic mode decomposition (DMD) is a tool that has enormous capability to model the intricacies of non-linear equations. This report outlines an exploration of DMD and its ability to separate different dynamics of a signal based on their linearity. While this is just one application of DMD, it is a powerful tool that is actively changing the field of medical technology among a plethora of others.

Introduction

DMD was discovered in Soviet era Russia as a tool for analyzing dynamic non-linear systems, but it was not until recently that it gained global popularity as a tool for data analysis. Because of its ability to not only accurately predict chaotic systems but identify the component of the system that makes it chaotic, DMD has a variety of applications ranging from control theory to signal analysis. This assignment explores two videos: formula cars coming around a turn at Monte Carlo and a skier dropping down a powdery chute on a bluebird day. Because these two sequences have an object moving in the foreground and a consistent background, these two features can be isolated using DMD. The following protocol outlines an in-depth analysis of how DMD linearizes a complex signal, and how the decomposition can be used to separate continuous components like the background of an image from the movement in the foreground. While DMD is a method for simplifying systems, the underlying mechanism is quite complex and requires a strong mathematical background to be used in a meaningful way.

Theoretical Background

DMD is a cumulative theory that builds on a series of mathematical ideas commonly used for data analysis and requires specific conditions for the data it is performed on. The data must be organized as snapshots – independent time points – of a system that are equally spaced in time. These snapshots are labeled U such that they form the matrix X (1,2).

$$U(x,t_n) = \begin{matrix} U(x_1,t_n) \\ \vdots \\ U(x_m,t_n) \end{matrix} \qquad \text{eq. 1}$$

$$X_{mxn} = \begin{bmatrix} U(x_1,t_1) & \cdots & U(x_1,t_n) \\ \vdots & \ddots & \vdots \\ U(x_m,t_1) & \cdots & U(x_m,t_n) \end{bmatrix} \qquad \text{eq. 2}$$

The first principal that DMD relies on is the Koopman operator, A, that linearly maps a system at any one time point to the next by the uniform change in time (3).

$$x_{t+1} = Ax_t \qquad \text{eq. 3}$$

While this may resemble simple linear regression, because of the way the Koopman operator is formed it has the ability to approximate the data on a global scale regardless of whether or not the dynamics are actually non-linear. For reference and for the rest of the theoretical background section the matrix containing the snapshots $U(x,t_1 \ldots t_{n-1})$ will be referred to as $\mathrm{X_t}$ or X1 and the matrix containing the snapshots $U(x,t_2 \ldots t_n)$ will be referred to as $\mathrm{X_{t+1}}$ or X2.

Because $x_2 = Ax_1$, and $x_3 = Ax_2$, it can be inferred that $x_3 = \mathrm{A(Ax_1) = A^2x_1}$. This principle can be expanded to create the Krylov subspace of X1 given by equation 4.

$$X_t = [x_1 A x_1 A^2 x_1 \ldots A^{n-2} x_1] \qquad \text{eq. 4}$$

This subspace shows the ability of the Koopman operator to represent the matrix X with the exception of the last value at time t+1. To represent this, we use the residual vector r and a vector e_{n-1} that represents zeros with a one at the n-1 position. Using the residual and the Krylov subspace, X2 can be represented as in equation 5.

$$X_{t+1} = AX_t + re_{n-1}^T \qquad \text{eq. 5}$$

The next step of DMD builds on the highly useful and widely applied mathematical method called singular value decomposition (SVD). The full theoretical background of SVD has been extensively covered in previous reports. SVD in this case is useful because it provides us with an orthonormal basis for the matrix X_t (6). We want A to be a linear combination of U, taken from the SVD of X_t, which means that we know the residual vector is orthogonal to the principal component basis of U and that $U^*r = 0$. Using this principal the equation can be rewritten to remove the residual term (7) and further manipulated to isolate a form of A, Ã (8). One important step to note in the development of the operator Ã is that a low rank version can be formed by only using the first k singular values and singular vectors to construct a low rank operator.

$$X_{t+1} = AU\Sigma V^* + re_{n-1}^T \qquad \text{eq. 6}$$

$$U^*X_{t+1} = U^*AU\Sigma V^* \qquad \text{eq. 7}$$

$$\tilde{A} = U^*AU \ = \ U * X_{t+1}\mathrm{V}\Sigma^{-1} \qquad \text{eq. 8}$$

The Ã in equation 8 is a Hessenberg matrix characterized by the Arnoldi iteration and represents a matrix that is similar to A. The characteristic of similarity that is important to the application of DMD is that this means both A and Ã have the same eigenvalues. When the eigenvalues of Ã, labeled y, are projected onto the principal components of X1 they are called the DMD modes (9). These DMD modes can be expanded as an eigenbasis to describe the continual iterative multiplications of A based on the continuous eigenvalues $\omega_k = \ln(\mu_k)/\Delta t$ (10). The sublabel k is used to denote that the rank of the eigenvectors is equal to the rank used in the construction of the Koopman operator.

$$\psi_k = Uy_k \qquad \text{eq. 9}$$

$$X_{dmd}(t) = \sum_{j=1}^{k} b_j\psi_j e^{\omega_j t} = \Psi e^{\omega_k t} b \qquad \text{eq. 10}$$

the initial condition vector b can be calculated by setting the time, t, to zero and solving the linear equation with the initial conditions of $x_{t=1}$ (11).

$$\mathrm{b} = \Psi^{\dagger}x_1 \qquad \text{eq. 11}$$

By multiplying out equation 10, we can use the eigen expansion of the low rank operator to not only reconstruct the original data from the DMD modes, but also extrapolate and predict the future values of the data.

DMD is a powerful tool, and one of the ways that the decomposition can be used is to separate background oscillations from the data. Because the eigenvalues highlight the oscillatory patterns of their corresponding eigen values, eigenvalues with low real and imaginary parts will correspond to DMD modes that represent the background of a signal or system. This concept can be represented mathematically via equation 12.

$$\mathrm{X_{dmd}} = \mathrm{b_p \phi_p e^{\omega_p t}} + \sum_{\mathrm{j \neq p}} \mathrm{b_j \ \phi_j e^{\omega_j t}} \qquad \text{eq. 12}$$

In the above decomposition the single mode subscripted p represents the whole of the background video while the rest of the modes contain the foreground. Practical application of this theory in some cases requires more than one background mode to capture the

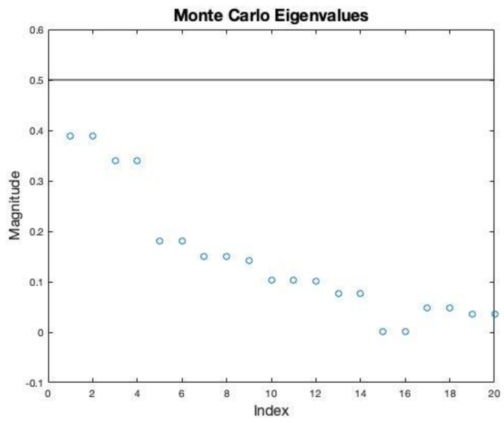


Figure 1: *Eigenvalues of the low rank operator for the Monte Carlo video. Threshold set to 0.5 encompasses all eigenvalues.*

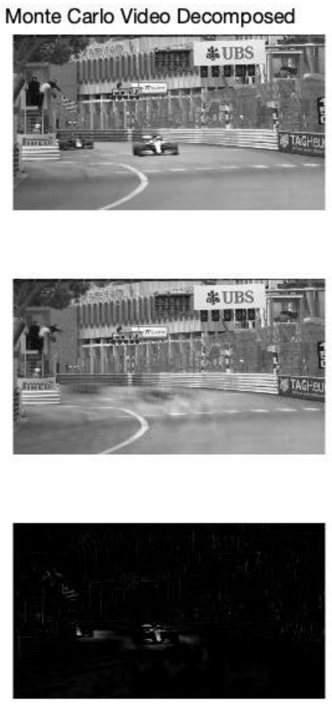
background oscillations and properly isolate the highly oscillatory and sparse foreground.

Algorithm Development and Implementation

The analysis of the two movies – F1 cars rounding a turn at Monte Carlo and a skier dropping down a chute – starts with loading the video files using MATLABs built in 'VideoReader' functionality. The resulting four-dimensional tensor was passed through a custom function that converts the frames to gray, linearizes the frames, and scales them from a 0-255 grayscale map to a 0-1 grayscale map. To start implementing DMD, the data for the Monte Carlo video – datam – was split up into two separate matrices X1 and X2 where X1 contained all but the last snapshot and X2 contained all but the first snapshot. These data matrices as well as the number of singular values to be used in the building of the low rank operator were then passed into a custom function. The custom function performed economical SVD, truncated the vectors and values, constructed the low rank operator according to the theoretical background section, and calculated the eigen values and vectors of the Koopman operator-similar matrix. The resulting eigenvectors were projected onto the principal components to build the DMD modes. The DMD modes, Phi, were used to calculate the initial condition for the DMD reconstruction through pseudoinversion. The time evolution of the DMD modes were calculated according to the theoretical background after calculating the continuous time eigenvalues as the natural log of the eigenvalues divided by the constant time step – which in this case is one because time is measured by frames.

The background DMD modes were identified by searching for the eigenvalues whose modulus was below a threshold with the find function. Using just these modes a low rank DMD approximation of the data was calculated following the same procedure as for the initial data approximation. The background video was isolated using the modulus of these low rank modes, and the sparse approximation of the data containing

Figure 2: *(top) raw image of cars rounding turn (middle) background of racetrack (bottom) foreground of cars isolated from the video.*



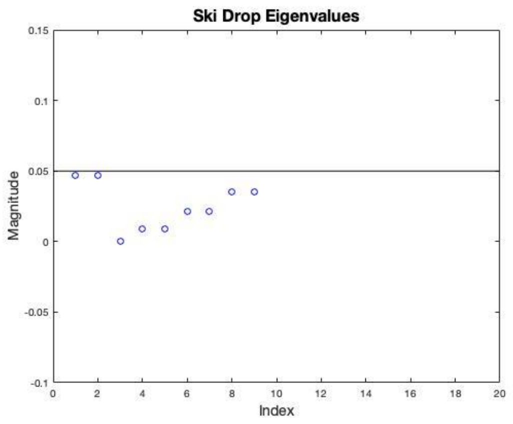


Figure 3: *Eigenvalues of the low rank operator for the Ski drop video. Threshold set to 0.05 encompasses all eigenvalues.*

the foreground was calculated by subtracting the background from the original data. The final background video was simply the modulus of the original to adjust negative values and include information stored in the complex values. The foreground video was calculated to be the background video subtracted from the raw footage with all of the negative values set to zero.

The analysis of the Ski drop video was performed in much the same manner with some exceptions. The final video used to isolate the background consisted of the modulus of the low rank reconstruction of the video. The foreground video was calculated by subtracting the background from the raw data and adding back in the modulus of the minimum value in the film. This last step helped reveal lower contrast moving parts and isolate the moving skier in the foreground.

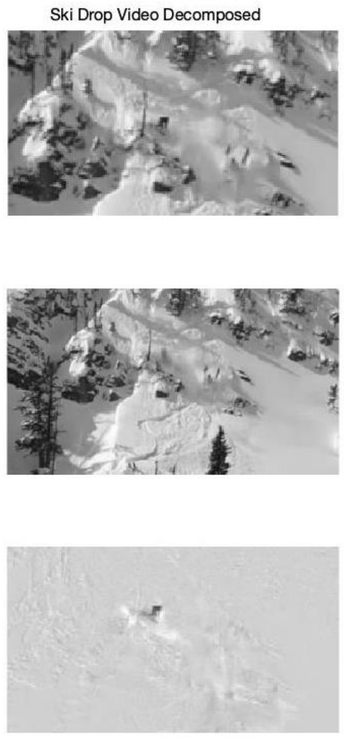
Computational Results

When analyzing the video of cars racing in Monte Carlo, the foreground and background were separated using 20 components for approximating the Koopman operator and an eigenvalue threshold of 0.5 (**figure 1**). These eigenvalues effectively encompassed the background footage, allowing for the successful extraction of the moving cars in the foreground (**figure 2**). For the ski drop video, the foreground and background were successfully isolated using 20 components to reconstruct the data and an eigenvalue threshold of 0.05 (**figure 3**). Using the low rank and sparse approximations of the video the foreground and the background of the video were successfully extracted (**figure 4**).

Conclusion and Summary

Through the application of DMD and SVD, two different videos were successfully decomposed into their foregrounds and backgrounds. While this is one particularly fascinating application of DMD, it has a potential to change any field that relies on data science and analysis for the better.

Figure 4: *(top) raw image of skier on slope (middle) background of mountain and snow (bottom) foreground of skier isolated from the surrounding terrain.*



Not only does DMD have a variety of applications, but it represents a powerful movement of reaching back into the wealth of mathematical tools that have been developed over the past century and applying them to incredibly modern and futuristic problems.

Appendix A – MATLAB Functions

size(x)– returns an array containing the number of elements in each dimension of x, used to find the dimensions of the imported video tensors.
x = zeros(m,n) – Initializes a m x n matrix of zeros, used to initialize matrices
diag(x) – takes the diagonal values of a matrix and returns an array or turns an array into a diagonal matrix, used to extract the eigenvalues from the output of eig.
Xgray = rgb2gray(X) – converts an m x n x 3 color image to an m x n grayscale image.
[Vecs,Eigs] = eig(x) – returns the eigenvectors and eigenvalues of the matrix x, used to calculate the basis of the low rank similar Koopman operator.
[x,y] = find(X == a) – Takes a matrix X and returns the x and y indices of any coordinates that satisfy the input Boolean condition, used to locate the eigenvalues below the threshold a.
plot(t,y) – plots y against t, used to plot the eigenvalues.
abs(x) – calculates the modulus of each value in the matrix x, used to calculate the modulus of complex matrices approximated with the DMD modes.
min(x,[],'all') – calculates the minimum element in the matrix x, used to reformat videos after background isolation.
sum(x) – returns the sum of the values in x with the option to sum along one or more dimensions, used to calculate errors.
[U,S,V] = sum(x, 'econ') – returns the economical singular value decomposition of the matrix x, used to calculate the SVD of the X1 matrix.
[data, m, n] = parametrize(predata) – takes in a color video and reformats the video to a linearized grayscale scaled version, used for initial formatting.
[U, Vs, l] = operator(X1,X2,num_components) – takes in the X1 and X2 matrices as well as the number of components to use for SVD truncation and calculates the eigen-decomposition of the Koopman operator.

Appendix B – MATLAB Code

```
% William Ojemann
% Assignment 5
% AMATH 482
% 10 MAR 2021
clear; close all; clc;
%% Initial Processing - Do Not Run
% % % monte = read(VideoReader('monte_carlo_low.mp4'));
% % % ski = read(VideoReader('ski_drop_low.mp4'));
% % % [datam, m1, n1] = parametrize(monte);
% % % [datas, m2, n2] = parametrize(ski);
% % % save('Ski_data.mat','datas','m2','n2');
% % % save('Monte_data.mat','datam','m1','n1');
%% Loading
load('Monte_data.mat'); load('Ski_data.mat');
%% Monte Carlo Analysis
% num_components -- 20
% eig threshold  -- 0.5
% Background isolated in datalow_inter
data = datam; j = m1; k = n1;
t = size(data,2);
n = length(data);
X1 = data(:,1:end-1); X2 = data(:,2:end);
num_components = 20;
[U,Vs,l] = operator(X1,X2,num_components);
mu = diag(l);
omega = log(mu);
Phi = U*Vs;
Y0 = Phi\X1(:,1);
modes = zeros(length(Y0),t);
for i = 1:t
    modes(:,i) = Y0.*exp(omega*i);
end
dataDMD = Phi*modes;
omegano = abs(omega);
figure(2)
thresh = 0.5;
I = find(omegano < thresh);
plot(omegano,'o');
hold on
plot([0 length(omegano)],[thresh thresh],'-k');
ylim([-.1 (thresh + .1)])
title('Monte Carlo Eigenvalues','Fontsize',16);
ylabel('Magnitude','Fontsize',14);
xlabel('Index','Fontsize',14);
datalow_inter = Phi(:,I)*modes(I,:);
datasparse_inter = data - abs(datalow_inter);
rcheck = datasparse_inter < 0;
R = datasparse_inter.*rcheck;
datalow = abs(datalow_inter);
datasparse = datasparse_inter-R;
figure(100)
i = 1;
subplot(3,1,i)
imshow(reshape(data(:,i),j,k));
subplot(3,1,i+1)
```

```
imshow(reshape(datalow(:,i),j,k));
subplot(3,1,i+2)
imshow(reshape(datasparse(:,i),j,k));
sgtitle('Monte Carlo Video Decomposed','Fontsize',16)
%% Ski Drop Analysis
% num_components -- 20
% eig threshold  -- 0.05
% add abs(min) for sparse instead of R
% don't add R for low rank
data = datas; j = m2; k = n2;
t = size(data,2);
n = length(data);
X1 = data(:,1:(end-1)); X2 = data(:,2:end);
num_components = 20;
[U,Vs,l] = operator(X1,X2,num_components);
mu = diag(l);
omega = log(mu);
Phi = U*Vs;
Y0 = Phi\X1(:,1);
modes = zeros(length(Y0),t);
for i = 1:t
    modes(:,i) = Y0.*exp(omega*i);
end
dataDMD = Phi*modes;
omegano = abs(omega);
figure(1)
thresh = 0.05;
I = find(omegano < thresh);
plot(omegano(I),'bo'); drawnow
hold on
plot([0 length(omegano)],[thresh thresh],'-k');
%ylim([-.1 (thresh + .1)])
title('Ski Drop Eigenvalues','Fontsize',16);
ylabel('Magnitude','Fontsize',14);
xlabel('Index','Fontsize',14);
datalow_inter = Phi(:,I)*modes(I,:);
datasparse_inter = data - abs(datalow_inter);
rcheck = datasparse_inter < 0;
R = datasparse_inter.*rcheck;
datalow = abs(datalow_inter);
datasparse = datasparse_inter + abs(min(datasparse_inter,[],'all'));
figure(101)
i = 1;
subplot(3,1,i)
imshow(reshape(data(:,i),j,k));
subplot(3,1,i+1)
imshow(reshape(datalow(:,i),j,k));
subplot(3,1,i+2)
imshow(reshape(datasparse(:,i),j,k));
sgtitle('Ski Drop Video Decomposed','Fontsize',16)
%% Functions
% Internal Function
function [data, m, n] = parametrize(predata)
    shapes = size(predata);
    m = shapes(1); n = shapes(2);
    num_frames = shapes(4);
    data = zeros(m*n,num_frames);
    for f = 1:num_frames
        data(:,f) = reshape(rgb2gray(predata(:,:,:,f)),...
            m*n,1);
    end
    data = data/255;
end
function [U,Vs,l] = operator(X1,X2,num_components)
    [U,s,V] = svd(X1,'econ');
    U = U(:,1:num_components);
    s = s(1:num_components,1:num_components);
    V = V(:,1:num_components);
    S = U'*X2*V*diag(1./diag(s));
    [Vs,l] = eig(S);
end
```