1

Isolating the Background and Foreground of Video Footage Using Dynamic Mode Decomposition

Elliot Jennis

March 2021

Abstract

Many data analysis techniques that rely on the singular value decomposition provide limited insight into non-linear, time dependant dynamics within a given data set. Dynamic mode decomposition (DMD) was investigated as an analytic tool to compensate for that downside by investigating data sets with nonlinear spaciotemporal dynamics. DMD was used to separate the background from the foreground of two video files by isolating the DMD modes corresponding to low frequency content within the data set. This process achieved various levels of success for foreground reconstruction, but proved to be a very robust method to reconstruct the background image.

Introduction and Overview

Dynamic mode decomposition (DMD) is one of the most versatile tools in a data scientist's toolbox. This statistical tool is a purely data driven method, meaning it does not need governing equations to provide insight on a given data set. This makes DMD a popular tool in the fields of fluid dynamics, biology, finance, and many others. In this paper, the strengths of DMD were leveraged to deconstruct two video clips, one of a person skiing down a mountain side and another of track-side clip from the 2019 Formula 1 (F1) Monaco Grand Prix. The end result was full separation between the background and the foreground of the videos on a frame by frame basis.

Theoretical Background

DMD was developed by Peter Shmid in 2008 as a way to decompose data structures collected in the fluid dynamics field into a hierarchical set of 'modes' that provide spaciotemporal information[1]. It has since been broadened and applied to a variety of different fields where highly dimensioned data sets need to be analyzed. Unlike other methods, DMD is purely data driven, making it a versatile tool for data analysis and model creation. At its most basic level, DMD creates a best fit linear operator to describe the behavior of non-linear dynamics within a data set. A requirement of the DMD algorithm is that the data is time resolved. Each 'snapshot' of data must be taken at regular intervals of time. These intervals are defined by $\Delta t = t_m - t_{m-1}$ with the entire time interval starting at t_1 and ending at t_m[2]. To use DMD, the data matrix

$$\boldsymbol{X} = \begin{bmatrix} x_1 & x_2 & x_3 & ... & x_m \end{bmatrix}$$

must be defined from this time resolved data where each column x_m represents a single snapshot. Two matrices are defined from this original data matrix

$$\boldsymbol{X1} = \begin{bmatrix} x_1 & x_2 & ... & x_m - 1 \end{bmatrix} \tag{1}$$

$$\boldsymbol{X2} = \begin{bmatrix} x_2 & x_3 & ... & x_m \end{bmatrix} \tag{2}$$

where $\boldsymbol{X2}$ represents $\boldsymbol{X1}$ moved Δt into the future. The DMD algorithm seeks to approximate the Koopman operator $\boldsymbol{A}$ (equation 3) that will advance $\boldsymbol{X1}$ to $\boldsymbol{X2}$[2]. The Koopman operator is an infinite dimensional operator that can represent nonlinear infinite-dimensional dynamics without resorting to linearizatrion [1].

$$\boldsymbol{X1A = X2} \tag{3}$$

However, the Koopman operator $\boldsymbol{A}$ is an enormous matrix requiring cost-prohibitive levels of computing power[1]. DMD approximates this operator by finding the eigenvalues and eigenvectors of $\boldsymbol{A}$ without having to calculate it[1]. To accomplish this, DMD makes use of an old friend, the Singular Value Decomposition (SVD). $\boldsymbol{A}$ can be represented as a lower rank matrix defined by

$$\boldsymbol{\tilde{A} = U^* X_2^M V \Sigma^{-1}}[1] \tag{4}$$

which approximates the eigenvalues λ of the Koopman operator [1]. The eigenvectors corresponding to $\boldsymbol{y}$ can be calculated using the eigenvalue problem shown in equation 5.

$$\boldsymbol{\tilde{A} y} = \lambda \boldsymbol{y}[1] \tag{5}$$

The eigenvectors are then used to reconstruct the DMD modes

$$\boldsymbol{\Phi} = \boldsymbol{U y}[1] \tag{6}$$

with the columns of $\boldsymbol{\Phi}$ representing the low rank approximation of the eigenvectors of the Koopman operator.

The eigenvectors and eigenvalues can also be used to approximate future states of the data in question, but that is beyond the scope of this paper.

Algorithm Implementation and Development

The DMD algorithm is fairly straight forward to compute in most programming languages. The first step is initializing the data matrix $\boldsymbol{X}$. Using the code described in Listing 1, each frame from each video was converted into a gray-scale format and cast as a double point variable. This process formed a data matrix of size M x N x $NumberofFrames$ with M and N representing the pixel resolution of each frame.

This matrix was reshaped so that each two dimensional matrix containing the spacial information of the pixels within a single frame was converted into one long column vector of length MN. This formed the typical data matrix that could be used for SVD, POD, or PCA analysis. The next step was to define a Δt value for the videos. Each video was about six seconds long, so a *linspace* command (appendix A) was used to form a vector beginning at zero and ending at six spaced out into $NumFrames$ time steps.

With the data matrix and values initialized, the DMD algorithm could be implemented (Listing 2). $\boldsymbol{X1}$ and $\boldsymbol{X2}$ were initialized corresponding to their definitions in equations 1 and 2. The SVD of $\boldsymbol{X1}$ was computed and a rank truncation value was chosen based on the modal energies present. The truncated SVD matrices were used to compute $\tilde{\boldsymbol{A}}$ following equation 4. The eigenvalues and eigen vectors of $\tilde{\boldsymbol{A}}$ were found, and finally $\boldsymbol{\Phi}$ was computed following equation 6. The final step before the background and foreground seperation could begin, was to create the DMD frequencies. The computation

$$\omega = log(\lambda)/dt$$

computes the frequencies for each eigenvalue calculated from $\tilde{\boldsymbol{A}}$.

To accurately separate the background from the foreground, the assumption was made that all low energy low frequency modes present in the DMD analysis corresponded to the background. This was not always true as problems described later in this paper will show. To find the modes necessary to reconstruct the background, a cutoff value was hard coded. All frequencies below that value would be categorized as background frequencies. The modes present in $\boldsymbol{\Phi}$ that corresponded to those low value frequencies were used to reconstruct the background image (Listing 3). The psudoinverse b of the initial frame of each video was computed using only the background modes. This value b was used to reconstruct the background video by computing

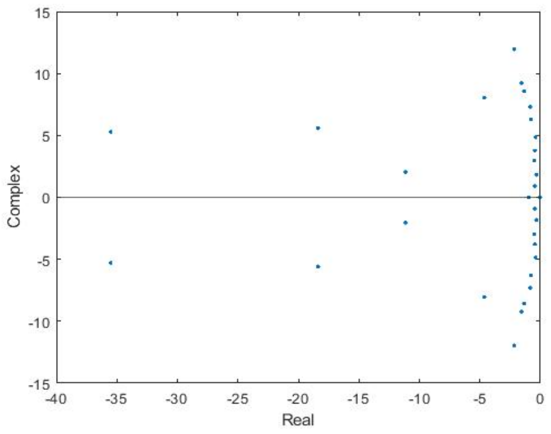
$$\boldsymbol{X_{DMD}} = b_p \boldsymbol{\phi}_p e^{\omega_p t}[1].$$

$\boldsymbol{X_{DMD}}$ is of the same dimensions as the original data matrix $\boldsymbol{X}$. Finally, the simple task of subtracting the absolute value of $\boldsymbol{X_{DMD}}$ from $\boldsymbol{X}$ computes the foreground data matrix $\boldsymbol{X_{for}}$.

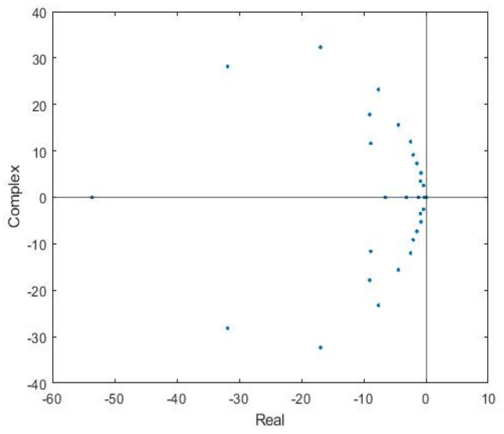
Computational Results

The SVD modal energy plot clearly shows (Figure 4 Appendix B) that for both videos, most of the energy within the videos was located within the first 10 or 15 modes. It was decided that $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V}$ would be truncated to 30 modes for the subsequent DMD analysis. This included plenty of detail from the videos while keeping the computational time relatively short.

Once the DMD algorithm had finished and the DMD frequencies (Omega) were calculated and plotted, the background frequencies had to be isolated from the rest. The metric used to divide the foreground and background frequencies was the same for both videos with values under .01 chosen as the background. These values are clearly shown (Figure 1) to be centered near the origin on the complex plane. For both videos there was only one omega value that met this criteria.



(a) Ski video



(b) F1 video

Figure 1: Real and imaginary components of DMD frequencies

With the background frequencies isolated, the background image was reconstructed using the DMD modes. This reconstructed image was then subtracted from the original data matrix to form the foreground image. For each video three frames spaced at intervals of 40 frames apart to make movement perceivable were plotted (Figures 2 and 3). The separation effect is particularly apparent when comparing Figures 2 and 3 to the original video frames (Figure 5 Appendix B). The foreground images were plotted using the negative of the reconstructed image to make the movement more apparent.

Summary and Conclusions

The images DMD outputs have varying degrees of success. For the skier video, while the background reconstruction is immaculate, the foreground is not perfect. There are clearly some spots where no movement is occurring within that frame that are still highlighted (bottom center of frame). This could be due to future movement occurring at those locations. Additionally parts of the background are still highlighted such as the clouds in the top left corner. This is also due to the movement of those clouds picked up by the DMD modes as this code assumes that stationary objects are background while moving objects are foreground. The most challenging part about working with this video is the small size of the skier. Future versions of this analysis would included extra preprocessing steps such as image cropping to make the skier's movement more apparent. The F1 video had a different set of challenges to overcome compared to the skier video. The moving objects are much larger in this video making the resulting foreground movement much more apparent. However there are some artifacts left over in the foreground image that belong to the background image. These include blurry areas on the road that were not fully removed from the background. Additionally there are clear lines present in the foreground reconstruction leftover from the background. After changing settings and rewatching the video, it was observed that in the original video file there is some amount of camera shake due to the car's movement. This camera shake coupled with the highly detailed background image could fool the DMD

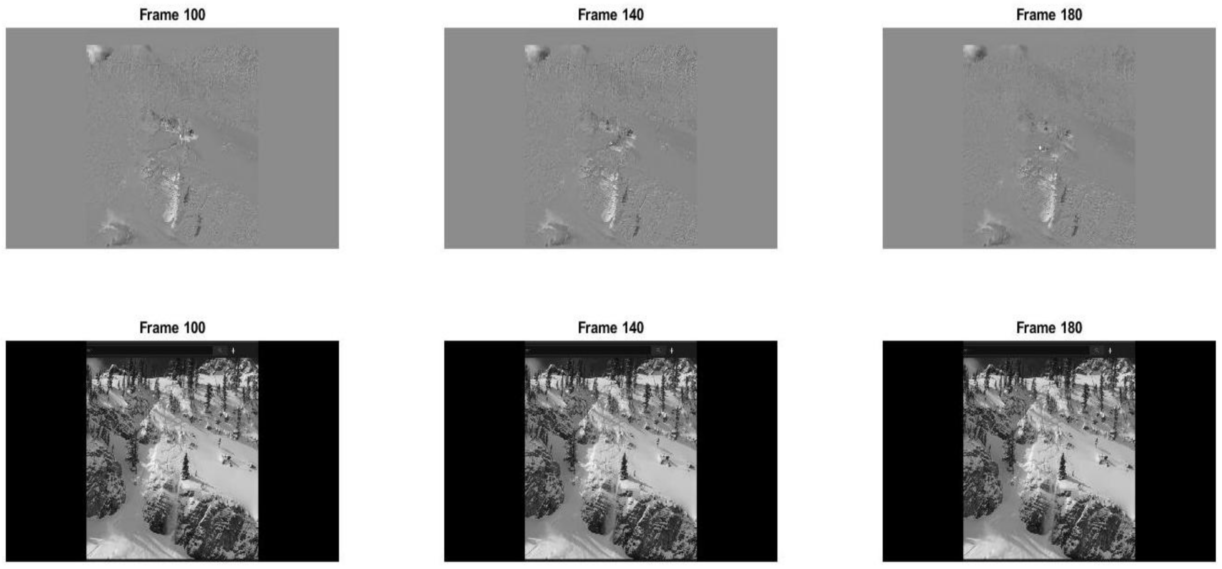


Figure 2: Foreground and background frame comparison for skiing video

algorithm into thinking the movement represent foreground artifacts when in reality it does not.

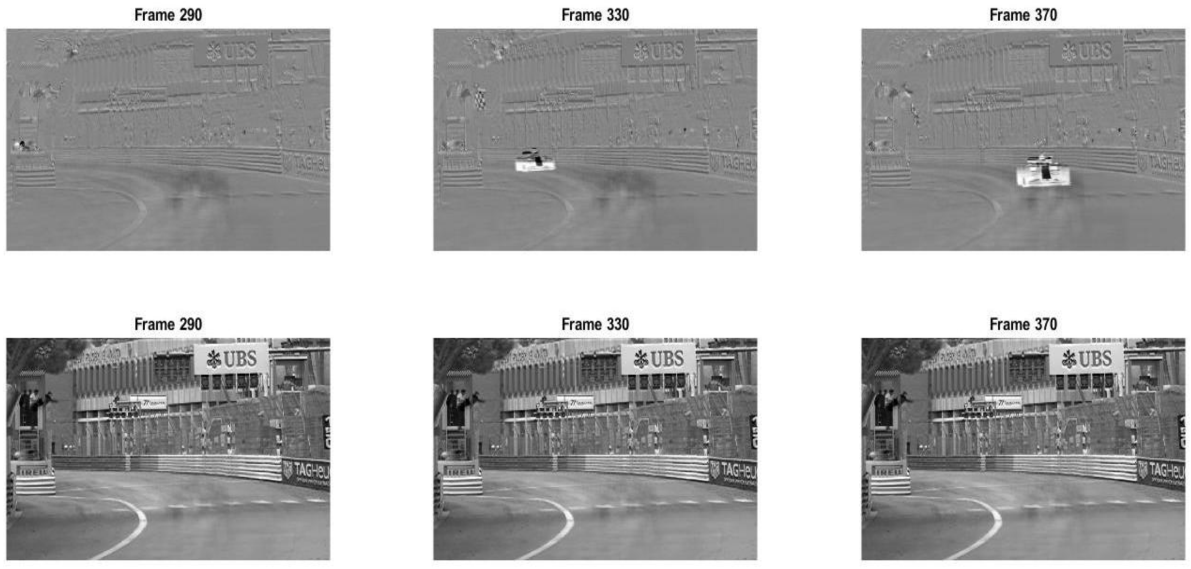


Figure 3: Foreground and background frame comparison for F1 video

References

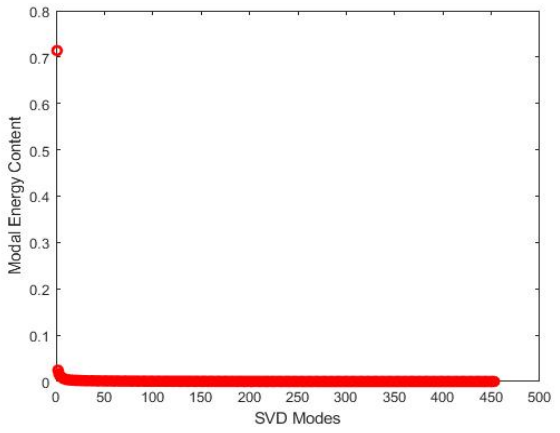
[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[2] Jonathan H Tu et al. "On dynamic mode decomposition: Theory and applications". In: *arXiv preprint arXiv:1312.0041* (2013).

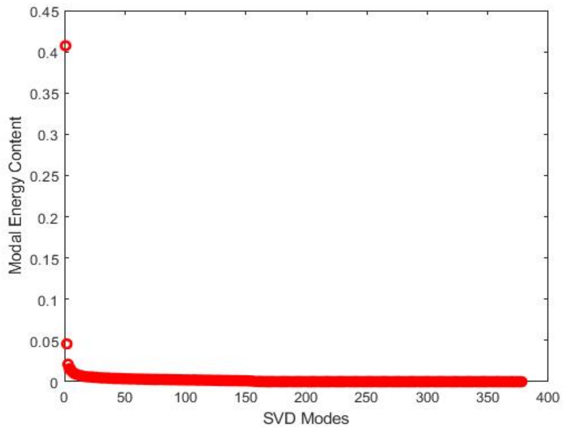
A MATLAB Functions

- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `data = read(ds)` returns data from a datastore. Subsequent calls to the read function continue reading from the endpoint of the previous call.
- `y = linspace(x1,x2,n)` generates n points. The spacing between the points is (x2-x1)/(n-1). linspace is similar to the colon operator, ":", but gives direct control over the number of points and always includes the endpoints. "lin" in the name "linspace" refers to generating linearly spaced values as opposed to the sibling function logspace, which generates logarithmically spaced values.
- `B = reshape(A,sz1,...,szN)` reshapes A into a sz1-by-...-by-szN array where sz1,...,szN indicates the size of each dimension. You can specify a single dimension size of [] to have the dimension size automatically calculated, such that the number of elements in B matches the number of elements in A. For example, if A is a 10-by-10 matrix, then reshape(A,2,2,[]) reshapes the 100 elements of A into a 2-by-2-by-25 array.
- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of m-by-n matrix A. The economy-size decomposition removes extra rows or columns of zeros from the diagonal matrix of singular values, S, along with the columns in either U or V that multiply those zeros in the expression A = U*S*V'. Removing these zeros and columns can improve execution time and reduce storage requirements without compromising the accuracy of the decomposition.
- `[V,D] = eig(A)` returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that A*V = V*D.
- `C = setdiff(A,B)` returns the data in A that is not in B, with no repetitions. C is in sorted order. If A and B are tables or timetables, then setdiff returns the rows from A that are not in B. For timetables, setdiff takes row times into account to determine equality, and sorts the output timetable C by row times.

B Figures

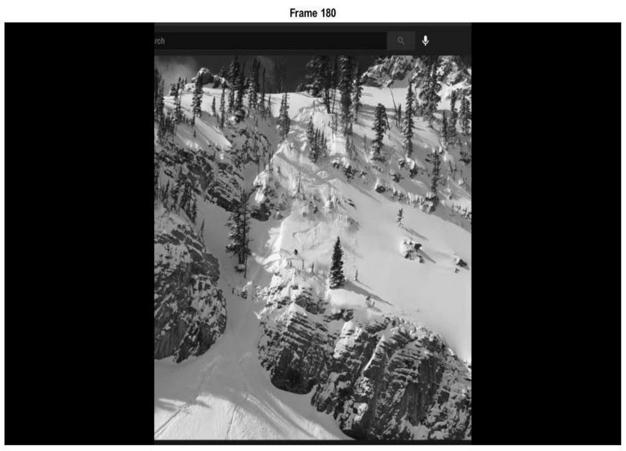


(a) Original ski video

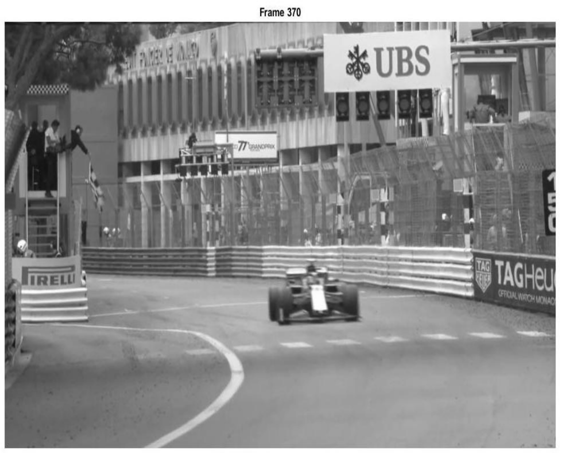


(b) Original F1 video

Figure 4: SVD modal energies



(a) Ski video



(b) F1 video

Figure 5: Original frames from unaltered videos

C MATLAB Code

```
%% Load Videos
ski_data = VideoReader('ski_drop_low.mp4');

vidFrames = read(ski_data);
numFrames = get(ski_data,'numberOfFrames');
for k = 1 : numFrames
mov(k).cdata = vidFrames(:,:,:,k);
mov(k).colormap = [];
end

for j=1:numFrames
Xg=rgb2gray(frame2im(mov(j)));
Xski(:,:,j) = double(Xg);
%imshow(Xg); drawnow
end
```

Listing 1: Function used to load video data

```
%% DMD analysis F1 Video
%defining spaciotemporal domain
t = linspace(0 ,6,numFrames);
dt = t(2)-t(1);
x = linspace(1,51800);

%initializing data matrix
for j = 1:numFrames
Xdat(:,j) = reshape(Xski(:,:,j),[518400 1]);
end
%creation of DMD data matrices
Xa = Xdat(:,1:end-1);
Xb = Xdat(:,2:end);
%SVD of Xa
[U,S,V] = svd(Xa,'econ');

figure(1)
plot(diag(S)/sum(diag(S)),'ro','Linewidth',[2])
xlabel('SVD Modes')
ylabel('Modal Energy Content')
r = 30;
%rank truncation of SVD matricies
Ur = U(:,1:r);
Sr = S(1:r,1:r);
Vr = V(:,1:r);

Atilde = Ur'*Xb*Vr*inv(Sr);
[W,D] = eig(Atilde);
Phi = Xb*Vr*inv(Sr)*W; %DMD modes (eigenvectors)

lambda = diag(D); %eigenvalues
omega = log(lambda)/dt; %frequencies

%plot omega to determine background modes
figure(2)
plot(omega,'.','LineWidth',7);
hold on
xline(0);
yline(0);
xlabel('Real')
ylabel('Complex')
```

Listing 2: DMD algorithm including SVD energies plot and DMD frequency plot

```
%% Background Subtracktion
%find indicies of low value frequencies
Index_b = find(abs(omega)<.01);
Ingex_f = setdiff(1:r, Index_b);

omega_b = omega(Index_b);
Phi_b = Phi(:,Index_b);
B = Phi_b\Xdat(:,1);

for t2 = 1:length(t)
    time_dynamics(:,t2) = B.*exp(omega_b.* t(t2));
end
X_dmd = Phi_b*time_dynamics;
%Background image reconstruction
X_for = Xdat-abs(X_dmd);
%Forground image reconstruction

figure(3)
for j = 1:3
    frame2 = reshape(X_for(:,100 +(j-1)*40),[540 960]);
    frame2 = uint8(140-real(frame2));
    subplot(2,3,j)
    imshow(frame2);
    title(['Frame ', num2str(100+(j-1)*40)])
    axis('equal','tight')
end
for j = 1:3
    frame1 = reshape(X_dmd(:,100 +(j-1)*40),[540 960]);
    frame1 = uint8(real(frame1));
    subplot(2,3,3+j)
    imshow(frame1);
    title(['Frame ', num2str(100+(j-1)*40)])
    axis('equal','tight')
end
```

Listing 3: Algorithm used to subtract background image from the foreground