# Dynamic Mode Decomposition: Separating Foreground from Background

Quinn Luo

March 2019

## Abstract

This report shows how dynamic mode decomposition can be performed on analyzing videos. It is to show you the technique to extract the static background from the moving objects in the foreground. I will introduce with background information, followed by a detailed explanation of algorithms, and finalize my discussion with the results reported.

## Introduction

Modern physical and biological systems are too complex to model because they are often nonlinear dynamic system. However, we can characterize such system with a linear dynamic model. In fact, combining the spectral methods and singular value decomposition, a powerful technique called Dynamic Mode Decomposition is implemented to perform the analysis.

# Theoretical Background

## Dynamic Mode Decomposition (DMD)

Suppose a nonlinear dynamic system is expressed as

$$\frac{dx}{dt} = F(t, x, \cdots) = \mathcal{A}x \tag{1}$$

and we can represent the system above as

$$x_{k+1} = Ax_k \tag{2}$$

where $A$ is the transformation matrix

$$A = e^{\mathcal{A}\Delta t} \tag{3}$$

and the solution to the dynamic system is

$$x_k = \sum \phi_j \lambda_j^k b_j \tag{4}$$

where $\phi$ is the DMD mode and $\lambda$ is the eigenvalues of $A$. Once we obtain the equation (4), we approximate it with

$$x_k \approx \sum \phi_j \omega_k t b_j \tag{5}$$

where $\omega' s$ are frequencies of DMD modes. And low frequencies relate to the modes that are less active, i.e., not changing over time, while modes with the higher frequencies represent the changing content.

## Algorithm Implementation and Results

The following implementation is developed and described in the book by Kutz (2013).

A video can be represented as a matrix by converting each frame into a column and lay them out in a timely order. Equation (6) is the structure of a video stream in a matrix, where each $x_k$ is a time slot.

$$X_{total} = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_k \\ | & | & & | \end{pmatrix} \tag{6}$$

Based on that I split the matrix into two and construct two matrix $X$ and $X^{'}$ where

$$X = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{k-1} \\ | & | & & | \end{pmatrix}, X^{'} = \begin{pmatrix} | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_k \\ | & | & & | \end{pmatrix} \tag{7}$$

The equation (8) describes a dynamic system, where $X^{'}$ is in the system in the next snapshot following $X$ and $A$ is the transformation matrix.

$$X^{'} = AX \tag{8}$$

The objective of DMD is to find the matrix $A$ such that $||X^{'} - AX||_2$ is minimized. With the singular value decomposed form of $X$

$$X = U\Sigma V^* \tag{9}$$

$A$ is approximated by its projection on feature spaces $U$.

$$\tilde{A} = U^* A U \tag{10}$$

and $U$, $\Sigma$, and $V$ are truncated at rank $r$

$$\tilde{A} = U_r^* X^{'} V_r \Sigma_r^* \tag{11}$$

The truncation decision is based on the distribution of SVD modes energy. Figure 1. shows the energy percentage of SVD modes from the analysis of the video. Because the first mode is much more dominant than the rest and the evidence from later cross-validation, I decide the truncate at 1.

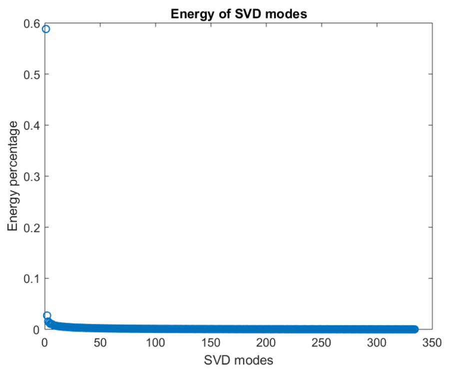

Figure 1: Energy of SVD modes

The next step is to compute the eigenspace of this projection of transformation matrix.

$$\tilde{A} W = W \Lambda \tag{12}$$

where $W$ are eigenvectors and $\Lambda$ is a diagonal matrix containing corresponding eigenvalues (Kutz 2013). Equation (13) shows the procedure of computing DMD modes.

$$\Phi = X^{'} V \Sigma^{'} W \tag{13}$$

$\lambda' s$ are the eigenvalues and I line up the $\omega = \frac{e^{\lambda}}{dt}$ (complex numbers) from smallest to the largest in terms of absolute values and set a threshold where any values less than that respond to the background and any larger $\lambda' s$ are the moving objects in the foreground because the static content is changing across frames less frequently. For test 1, since $r$ is 1, the only choice is to keep the only $\omega$.

After computing $b$ by solving $b \cdot x(0) = \Phi$, I construct a low-rank matrix based on the decomposition. Low-rank matrix $X_{low-rank} = \Phi e^{\omega_j t} b$ contains the background information, and subtracting $X_{low-rank}$ from the original matrix $X$ leaves me with the matrix with foreground information that turns out to be a sparse matrix $X_{sparse}$.

$$X = X_{low-rank} + X_{sparse} \tag{14}$$

Figure 2. below shows the comparison between one frame from original video, the same frame in the low-rank matrix which erases the moving objects, and the sparse matrix with only foreground in it. $X_{sparse}$ has some entries being negative numbers, but adjustment results

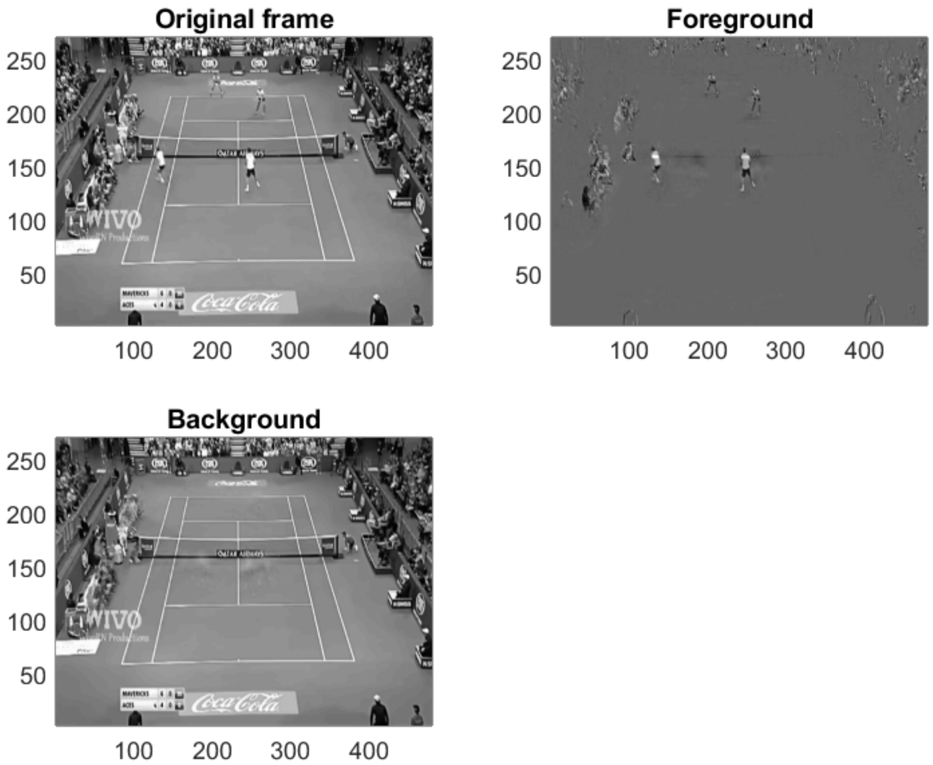

Figure 2: Separation of background and foreground with DMD (test 1)

in indistinguishable visualization and pcolor image plotter handles the negative values well.

Figure 3. illustrates why I made the decision to skip the negative residue and set the truncation at both one for $r$ and $\omega$. They are not as clear as the images in figure 2. Figure

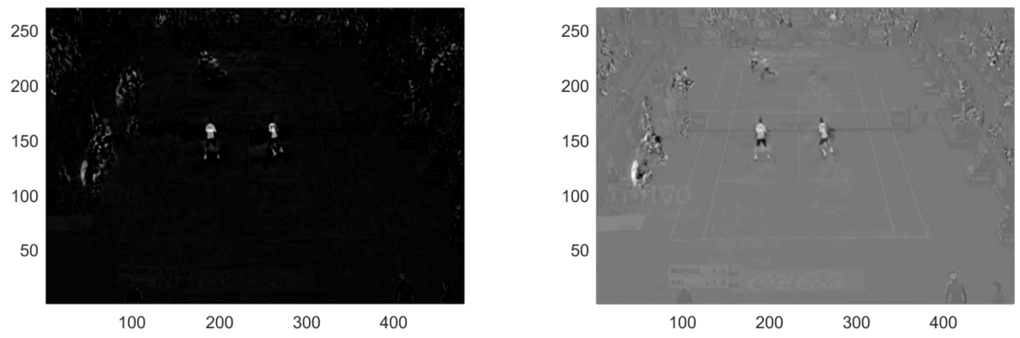

Figure 3: Examples of regularizing negative residue (left) and truncation at $r = 5$ and $\omega = 3$ (right)

4. shows the results from another video. After comparing and validating different number of SVD modes and $omega's$, $r$ is chosen to be 2 and $omega$ is truncated at 1.

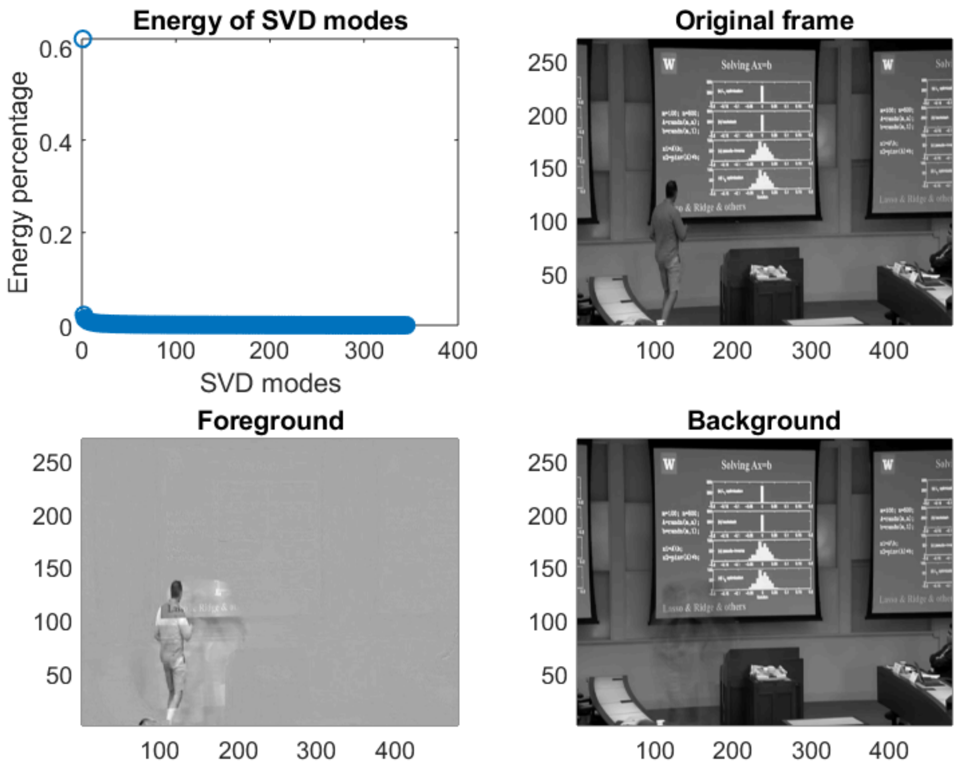

Figure 4: Separation of background and foreground with DMD (test 2)

### Results

1. Every video (every dataset given) has an optimal truncation threshold at SVD modes and eigenvalues.
2. DMD displays varying power of breaking down the data for different dataset.

## Conclusion

By performing DMD algorithms, I am able to separate the foreground moving objects and static background through extracting low frequency content from the high frequency one. Dynamic mode decomposition is a powerful technique to model complex high-dimensional system with low-dimension behaviors. It is also capable to predict in the near future, for instance, how the objects will be moving in the next second.

## Appendix I: MATLAB commands explained

`hasFrame(v)` checks if the video file v has next frame

`readFrame(frame)` stores a frame from video to matrix

`imresize(X, s)` resizes a matrix to s proportion of the original

`[W, D]=eig(A)` eigen-decomposes matrix A and stores the eigenvectors in W and corresponding eigenvalues in the diagonal of D

`diag(D)` pulling out the diagonal from a matrix or shapes a vector into a matrix by putting the vector on the matrix

`[B, I]=sort(v)` sorts the values of the vector v from the in ascending order into B and corresponding indices into I

## Appendix II: MATLAB code

```
clear all; close all; clc
load = VideoReader('Kutz.mp4');
video = [];
i = 1;
while hasFrame(load)
    frame = double(rgb2gray(readFrame(load)));
    frame = imresize(frame, 0.5);
    video = [video reshape(frame, [270*480 1])];
    i = i + 1;
end
X = video(:, 1:end-1);
X_prime = video(:, 2:end);

%%
[u, s, v] = svd(X, 'econ');
r = 5;
```

```
ur = u(:, 1:r);
sr = s(1:r, 1:r);
vr = v(:, 1:r);
Ar = ur' * X_prime * vr / sr;

[w, d] = eig(Ar);
fi = X_prime * vr / sr * w;
lambda = diag(d);
dt = 1/load.FrameRate;

omega = log(lambda)/dt;
[B, I] = sort(abs(omega));
Is = I(1:3);

x1 = video(:, 3);
b = fi \ x1;
frames = size(X, 2);
t = dt * 1:frames;
dynamics = zeros(r, length(t));

for i = 1:length(t)
    dynamics(:,i) = sum(b.*exp(omega(Is)*t(i))', 2);
end
Xd = fi*dynamics;
Xs = X - abs(Xd);

%%
neg = Xs;
neg(neg >= 0) = 0;
Xs = Xs - neg;

%%
% videoPlayer = vision.VideoPlayer;
```

```
for j = 1:frames
  videoFrame = reshape(Xs(:, j), [270 480]);
%   imshow(uint8(videoFrame));
  pcolor(flip(videoFrame)); shading interp; colormap(gray);
  pause(dt)
end

%%
subplot(2, 2, 1)
plot(diag(s)/sum(diag(s)), 'o', 'LineWidth', 1);
xlabel('SVD modes'); ylabel('Energy percentage'); title('Energy of SVD modes')
subplot(2, 2, 2)
pcolor(flip(reshape(X(:, 80), [270 480]))); shading interp; colormap(gray);
title('Original frame')
subplot(2, 2, 3)
pcolor(flip(reshape(Xs(:, 80), [270 480]))); shading interp; colormap(gray);
title('Foreground')
subplot(2, 2, 4)
pcolor(flip(reshape(Xd(:, 80), [270 480]))); shading interp; colormap(gray);
title('Background')
```

## References

Kutz, J. N., (2013). Data-driven modeling & scientific computation: Methods for complex systems & big data. Oxford: Oxford University Press.