Background Subtraction in Video Streams

Andrei Karavanov

Friday, March 15, 2019

Abstract

In this project we will use Dynamic Mode Decomposition to take several video clips containing a foreground and background objects and separate the video stream to both the foreground video and a background.

1 Introduction and Overview

In this project we have several videos. Each video contains of static background and non-static foreground. We will apply Dynamic Mode Decomposition for each of the video. That will allow us to create a low rank deconstruction, representing the background video stream. Then by subtracting this background video stream from our original video we will hope to filter the foreground video stream.

2 Theoretical Background

Imagine we have a dynamic system, which behaviour we want to examine. We want to be able to forecast its behaviour in the future so that we could possibly control it. The Dynamic Mode Decomposition (DMD) allows us to do exactly that. In it we take a dynamic system that can be described by some function $x(t)$ and find a matrix A, such that $A = \frac{dx}{dt} = f(x, t, \mu)$. Since in real world most of the times it is impossible to find of exact solution, we will approximate A by minimizing the L2 norm of the difference between our predicted value and actual readings. That can be written as the following: min:$||x_{t+1} - Ax_t||_2$. Note that this solution is only approximation: $Ax_t \approx x_{t+1}$.

In our project in order to find matrix A we have to take several steps. We will first load the data from a video. Then we will transform it into a matrix of doubles representing densities in gray scale of each pixel. Then we will split that matrix on matrix X_1, which includes first $n-1$ frames, and matrix X_2, which includes last $n-1$ frames. Since our matrix A should be an approximation of $AX_1 = X_2$, we will multiply the X_2 and the pseudo-inverse of X_1 to find the matrix A. This process can be written as the following: $A = X_2 X_1^+$.

Since it is more efficient computationally to compute and use a rank-reduced matrix $\tilde{A}$, we will find a low rank approximation of A. Then, we can represent a rank-reduced approximation of our original data as the following function: $f(t) = \mathbf{\Phi} \exp(\mathbf{\Omega} t)\boldsymbol{b}$, where $\mathbf{\Phi}$

is the matrix representing eigenvectors of $\tilde{A}$, $\mathbf{\Omega}$ is the matrix representing eigenvalues of $\tilde{A}$, and $\mathbf{b}$ is the matrix of DMD mode amplitudes.

Now, let us switch gears toward our actual project - using DMD in background subtraction. Let us start with defining what background is. In this paper we will address background as a set of all thing in the video that remain stationary. That allows us to state that the Fourier mode corresponding to the background video stream will be the smallest mode. Let us call it w_p. Since $||w_p|| \approx 0$, we can now separate our data on the low-ranked background stream: $x_b(t) = b_p\phi_p \exp(w_p t)$, and the foreground video stream: $x_f(t) = \sum_{k \neq p} b_k\phi_k \exp(w_k t)$. Note that if we will sum the background and the foreground we will get our rank-reduced approximation of our original data back. Note that now our foreground matrix can possibly have negative elements. For that reason we have to find a residual matrix R that will contain all of these negative values. This will alter our separated matrices in the following way: $x_b = |x_b| + R$ and $x_f = x_f - R$.

In real world, however, if one will get a clear separation without removing R from the background video stream and adding it to the foreground video stream matrix, one will just take the absolute value of both matrices. Note that is done only for the reason that having negative pixel intensities does not make any sense. Also note that by taking an absolute values of the matrices we are loosing the integrity of the DMD. In other words, we will not be able to reconstruct our original matrix by simply adding background video stream and foreground video stream together.

3 Algorithm Implementation

We begin with reading our video file. We use VideoReader and read functions for that purpose. Then we transform our raw frames by first moving them from rgb space to gray scale space, then converting all values to double so that we could manipulate the data, then resizing them by the factor of 0.25, and lastly reshaping each frame from a matrix representation into a vector. Now, when we have our data ready we can start with DMD.

Let us start by dividing our prepared data into X_1 and X_2:

```
X1 = videoGrayScale(1:end-1, :)';
X2 = videoGrayScale(2:end, :)';
```

Now we will perform SVD on the X_1. Then by plotting variances of top forty singular values we will see how many of them we can use in our rank reduction later on.

```
[U2,Sigma2,V2] = svd(X1, 'econ');
plot(diag(Sigma2(1:40, 1:40))/sum(diag(Sigma2)), 'o');
```

Now we will perform a rank reduction and using rank-reduced model we will compute $\tilde{A}$.

```
r=10;
```

```
U=U2(:,1:r);
Sigma=Sigma2(1:r,1:r);
V=V2(:,1:r);

Atilde = U'*X2*V/Sigma;
```

Now we will find DMD Modes and DMD Spectrum. Then we will find the DMD Solution.

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

mu=diag(D);

b = Phi\X1(:,1);
for iter = 1:timeSize
    timeDynamics(:,iter) = (b.*exp(omega*t(iter)));
end
X-dmd = Phi*timeDynamics;
```

The only part left is finding the sparse matrix and if needed removing the residual.

```
X-sparse = videoGrayScale' - abs(X-dmd);
Residual = X-sparse .* (X-sparse < 0);
```

Now we can use computed matrices to asses the effectiveness of DMD. We will do so by plotting original video, background video, and foreground video for three separate frames.

4 Computational Results

4.1 Test Video One

This video is a recording of highway traffic. Note that it has little to no noise - shaking, thus we will treat it as our ideal case. The results of SVD can be seeing at the Figure 1.

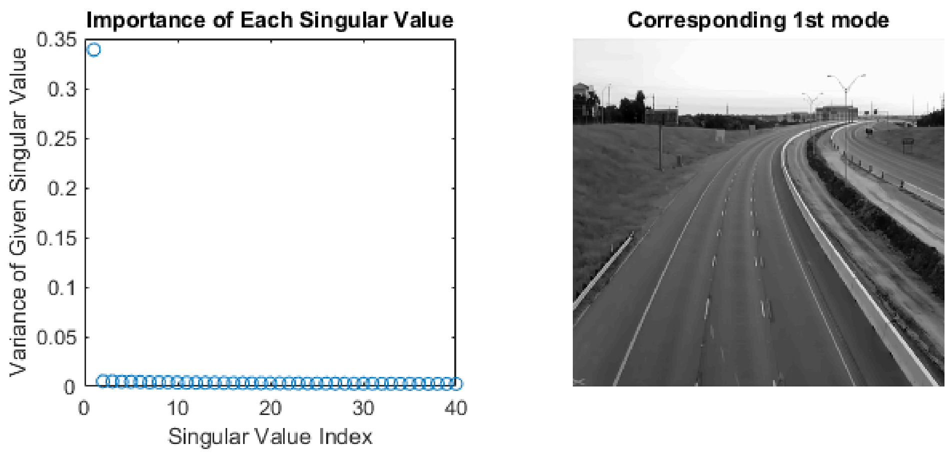

Figure 1: Finding the number of nodes to use in DMD.

Note that we have a single dominating node that represents our background. Since only one node is dominant we will reduce our system to $r = 1$ and then apply DMD. Now let us plot (Figure 2) the eigenvalue of our rank-reduced $\tilde{A}$ to see whether our rank-reduced $\tilde{A}$ is a stable or unstable approximation of the system.

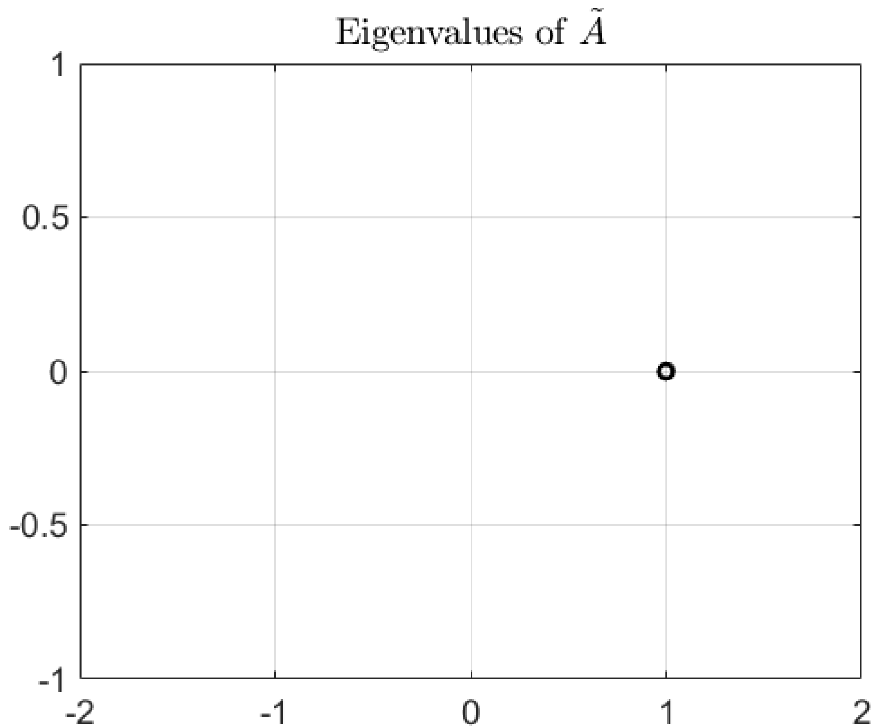

Figure 2: Plot of one eigenvalue of rank-reduced $\tilde{A}$.

The results of DMD can be seeing at the Figure 3.

Notice that we have a clear separation of the background video stream and a foreground video stream with just one SVD mode.

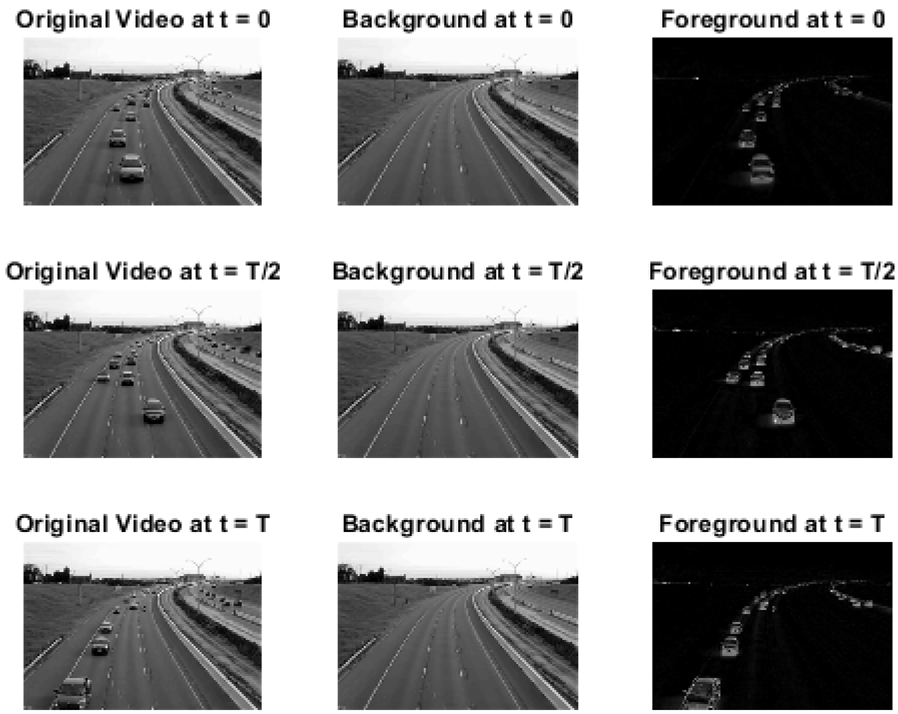

Figure 3: Results of DMD, where T stands for last time frame.

4.2 Test Video Two

This video is another recording of highway traffic. Note that it has slight noise represented by shaking of the camera. The results of SVD can be seeing at the Figure 4.

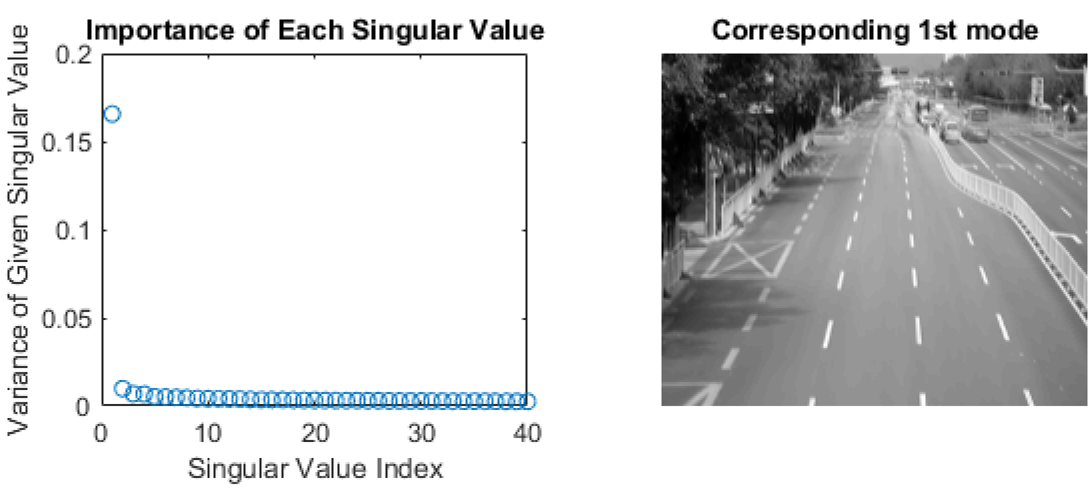

Figure 4: Finding the number of nodes to use in DMD.

Just by looking at the plot of singular values, we can see that our system can be reduced to $r = 5$. Now let us plot (Figure 5) the eigenvalues of our rank-reduced $\tilde{A}$ to see whether our rank-reduced $\tilde{A}$ is a stable or unstable approximation of the system.

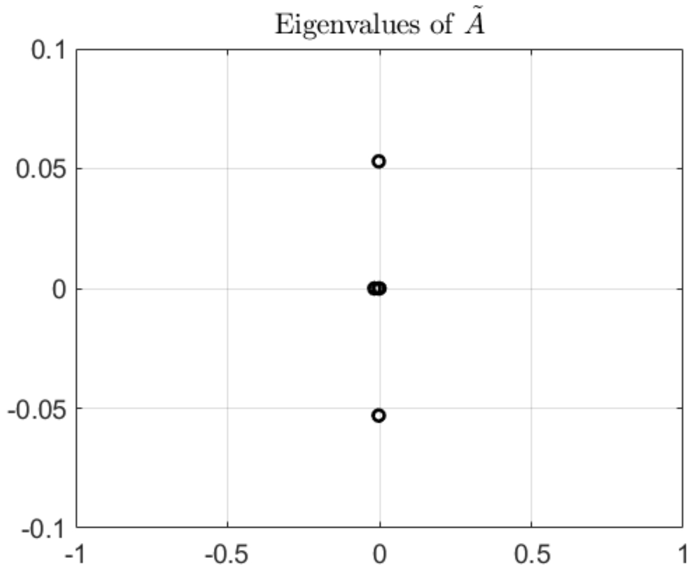

Figure 5: Plot of five eigenvalues of rank-reduced $\tilde{A}$.

The results of DMD can be seeing at the Figure 6.

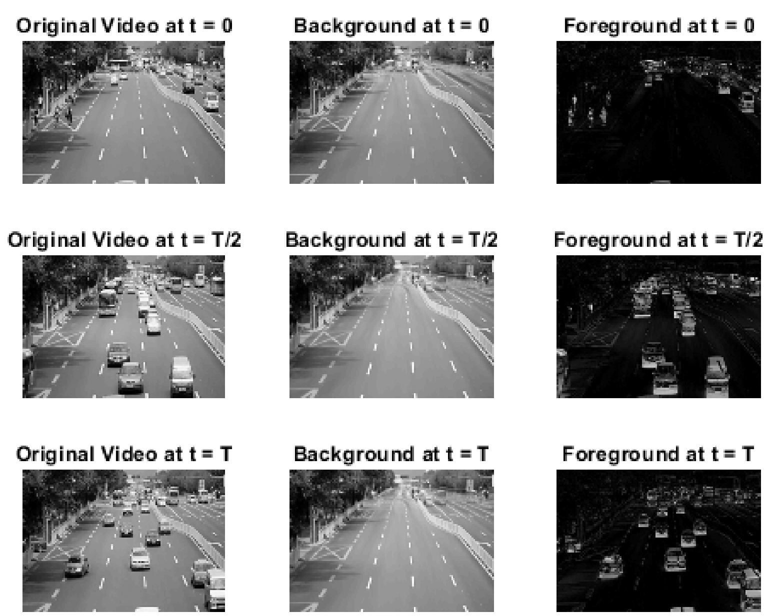

Figure 6: Results of DMD, where T stands for last time frame.

Notice that although we can see a pretty well separation of the background video stream and a foreground video stream, due to the fact that we had some noise in our data we can see slight "shadow" of the background in our foreground video stream (white lines).

5 Summary and Conclusions

From both of the examples we can see that DMD is good tool to have in your stash. In both cases we were able to separate background video stream. The quality of separation was however quite different. We related this to the fact whether we had a noise (shaking of the camera) or not. This result shows a very important point that is applicable both to PCA and DMD: both algorithms perform bad on systems in which noise is present. That comes from a fact that both PCA and DMD are using SVD. And as we learned from previous projects SVD does not perform very well on data sets with noise.

Another thing worth mentioning is that we did not use residuals in our computations. That decision came from the fact that our separation was already quite well, however, when we would subtract residuals from our foreground matrix and add them to the background one, our results decreased dramatically. Such result could be explained by the following reasoning. Whenever we were adding residuals to the rank-reduced matrix that was representing background, in some sense we were adding a part of foreground back to the background video stream.

Appendices

A MATLAB commands

VideoReader(): Used to read the video.
read(): Used to read each frame of the video.
reshape(): Used to reshapes a matrix to new dimensions.
rgb2gray(): Used to move image from rgb space to grayscale space.

svd(): Used to perform the SVD
pcolor(): Used to create a plot in pseudo color
zeros(): Used to create empty matricies.
eig(): Used to perform eigenvalue decomposition.

B MATLAB code

```
%% HW5 - Background Subtraction in Video Streams
clear all; close all; clc

disp("Getting Film")
video = VideoReader('mov7.mp4');

disp("Processing It")
v = read(video);
disp("Done")
% save('mov1.mat', 'v', '-mat')

%% Processing Video
clc

timeSize = size(v, 4);

xSize = round(size(v, 1)/4);

ySize = round(size(v, 2)/4);

videoGrayScale = zeros(timeSize, xSize*ySize);

dt = 1;
t = 0:dt:timeSize;

disp("Working On Video Matrix")
for i = 1:timeSize
```

```
    videoGrayScale(i, :) = reshape(imresize(double(rgb2gray(v
        (:,:,:,i))), [xSize, ySize]), 1, xSize*ySize);
end
disp("Done Working On Video Matrix")

%%  Preparing Data for SVD
clc

disp("Preparing Matrix X1")
X1 = videoGrayScale(1:end-1, :)';
disp("Preparing Matrix X2")
X2 = videoGrayScale(2:end, :)';
disp("Done Preparing Matricies")

%% SVD
clc

tic
disp("Preforming SVD")
[U2,Sigma2,V2] = svd(X1, 'econ');
disp("Done with SVD")
toc

%% Finding Number of Modes Needed
clc

figure()
subplot(1,2,1)
plot(diag(Sigma2(1:40, 1:40))/sum(diag(Sigma2)), 'o');
xlabel("Singular Value Index")
ylabel("Variance of Given Singular Value")
title("Importance of Each Singular Value")
subplot(1,2,2)
to_show = U2(:,1);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Corresponding 1st mode')
axis off


%% Post-SVD
clc

% Rank Reduction
r=1;
```

```
U=U2(:,1:r);
Sigma=Sigma2(1:r,1:r);
V=V2(:,1:r);

% A Tilde and DMD Modes
Atilde = U'*X2*V/Sigma;
[W,D] = eig(Atilde);
Phi=X2*V/Sigma*W;

% DMD Spectrum
mu=diag(D);
omega=log(mu)/dt;

omegToUse = min(abs(omega)); % Smallest Mode

disp("Done with post SVD")

%% Plot Omegas

figure()
plot(omega,'ko','Linewidth',[2]), grid on, axis([-2 2 -1 1]), set(
    gca,'Fontsize',[14])
title({'Eigenvalues of $\tilde{A}$'},'Interpreter','latex')

%% The DMD Solution
clc

b = Phi\X1(:,1);
time_dynamics = zeros(r,timeSize);
for iter = 1:timeSize
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end
X_dmd = Phi*time_dynamics;

disp("Done with finding DMD solution")

%% Finding sparse with residual
clc

X_sparse = videoGrayScale' - abs(X_dmd);
disp("Done with finding sparse with residual")

%% Finding residual
clc

```

```
Residual = X_sparse .* (X_sparse < 0);
disp("Done finding residual")

%% Removing residual
clc

X_dmd = abs(X_dmd);
X_sparse = abs(X_sparse);
disp("Done removing residual")

%% Comparison plot
clc

figure()
subplot(3,3,1)
to_show = videoGrayScale(1,:);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Original Video at t = 0')
axis off
subplot(3,3,2)
to_show = X_dmd(:,1);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Background at t = 0')
axis off
subplot(3,3,3)
to_show = X_sparse(:,1);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Foreground at t = 0')
axis off
subplot(3,3,4)
to_show = videoGrayScale(round(timeSize/2),:);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Original Video at t = T/2')
axis off
subplot(3,3,5)
to_show = X_dmd(:,round(timeSize/2));
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Background at t = T/2')
axis off
subplot(3,3,6)
```

```
to_show = X_sparse(:,round(timeSize/2));
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Foreground at t = T/2')
axis off
subplot(3,3,7)
to_show = videoGrayScale(timeSize,:);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Original Video at t = T')
axis off
subplot(3,3,8)
to_show = X_dmd(:,timeSize);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Background at t = T')
axis off
subplot(3,3,9)
to_show = X_sparse(:,timeSize);
to_show = reshape(to_show,[xSize, ySize]);
pcolor(flipud(abs(to_show))), shading interp, colormap(gray);
title('Foreground at t = T')
axis off
```