Background Subtraction in Video Streams

Crystal Yang

ABSTRACT

Using Dynamic Mode Decomposition (DMD), I am going to split the two videos called "ski_drop_low.mp4" and "monte_carlo_low.mp4" into foreground and background. We will be able to utilize and see the application of DMD.

INTRODUCTION

We had two videos "ski_drop_low.mp4" and "monte_carlo_low.mp4" and the first video we can see a man skiing down the hill covered with snow, and the other videos there are several racecars passing through the racetrack. In both videos, there contain objects that are always active throughout the video (the skier and the racecars), and the background of the video is the same all the time (the camera itself is not moving). Thus, we got an object moving in time, and we can capture its movement by studying the video frame by frame and these are the snapshots of the spatial-temporal data which can be used in DMD, so we can subtract the foreground and background in studying the DMD results.

BACKGROUND THEOREM

DMD (Dynamic Mode Decompositions) is a dimensionality reduction method that computes a set of mode contains periodic behaviors in time to represent the original data. This means that DMD will allow us to study how the data change with time. Thus, to start with, we need a data matrix that contains data which both evolve in space and time. Suppose we have a data matrix X, this data matrix should have a dimension of N and M where N refers to the number of spatial points stored in each time snapshot, and M represents the number of time snapshots.

We are looking for a linear mapping from one snapshot to another snapshot, and this linear mapping we are looking for is called Koopman operator. The Koopman operator A is a linear, time-dependent operator such that $x_{j+1} = Ax_j$ where j represents the snapshot time j. Thus, we can write out the Koopman operator given the matrix X_1^{M-1} where:

$$X_1^{M-1} = [x_1\, x_2 \ldots x_{M-1}] \;=\; [x_1\, Ax_1\, A^2x_1 \ldots A^{M-2}x_1]$$

$We\ can\ write\ the\ above\ matrix\ as$: $X_2^M \;=\; AX_1^{M-1} + re_{M-1}^t$, and use SVD to express $X_1^{M-1} = U\Sigma V^*$, we can then get $X_2^M \;=\; AU\Sigma V^* + re_{M-1}^t$

We want to choose A such that we can express the columns of X_2^M with linear combination of columns in U, thus we get:

$$U^*X_2^M \;=\; U^*AU\Sigma V^* + U^*re_{M-1}^t, and\ U^*r \;=\; 0$$

$$\therefore U^*AU \;= \underbrace{U^*X_2^M V\Sigma^{-1}}_{\hat{S}}$$

Matrix A and $\tilde{S}$ shared the same eigenvectors and eigenvalues since they are related by multiplying a matrix next to A, then we can get the eigenvector of $\tilde{S}$:

$$\tilde{S}y_k = \mu y_k$$

Where gives the eigenvector of A and we call this the DMD modes that describe the linear mapping between different time instance:

$$\varphi_k = Uy_k$$

We can then describe the DMD solution as: $X_{DMD} = \sum_1^K b_k \varphi_k e^{\omega t}$ where b_k is the initial amplitude of the periodic pattern of each mode and $\omega = ln(\mu)/\Delta t$. This DMD solutions that we are able to find describe the original data with sinusoidal modes we got from the eigenvector of the $\tilde{S}$ matrix, and with the eigenvalue describe the variance each mode explained, we can find the dominant objects in the video which will be the background that 'explain' the most part of the video, and the fast-moving objects.

From the eigenvalue of $\tilde{S}$, we can extract the modes that has a zero (or near zero) eigenvalue which will indicate that these modes are not changing throughout the time domain, and these are the modes that we are going to extract to split the data matrix into background video. Even though the snapshots of data could be nonlinear, using DMD could still yields us a linear mapping describing the relationship between each time snapshots!

ALGORITHM IMPLEMENTATION AND DEVELOPMENT

After loading the "ski_drop_low.mp4" and "monte_carlo_low.mp4" using `VideoReader()` in MATLAB, I read the mp4 file frame by frame and capture the image of each frame, use `reshape()` to store the image as a column vector and this is represent as one snapshot of data, and this is the data matrices we are working with.

With "ski_drop_low.mp4" video as our data matrix X, I extract the X1 (X_1^{M-1}) and X2(X_2^M) as the lag 1 data matrices from beginning and from the end. Compute the SVD of X1 matrix using `svd(X1,'econ')`. From here, I truncated the matrix to a rank of 200, and from the SVD result we can see that the first 200 variables can explain more than 99% percent of the variance. Then I can calculate the $\tilde{S}$ matrix using the formula $\tilde{S} = U^* X_2^M V \Sigma^{-1}$, then use the `eig()` command to calculate the eigenvalue and eigenvector of it. The ω is calculated by the formula $\omega = ln(\mu)/\Delta t$. To calculate b_k, we use the first snapshot x_1, the pseudoinverse of φ and use the backslash method in MATLAB.

We will first calculate the Koopman operator of background, as mentioned in the previous sections, the modes that are still have a ω of 0, thus, we can pick the mode that has $\omega \approx 0$ then use this omega to calculate the background DMD solution of $X_{DMD}^{background} = \sum_1^K b_k \varphi_k e^{0*t}$ where K is the rank of X1.

After we get background matrix, note that $X = X_{DMD}^{background} + X_{DMD}^{foreground}$, thus, we can get the foreground video by subtracting the absolute value of the background solution from the original data matrix. This may result in some negative value in the matrix, so we will store the negative value in a

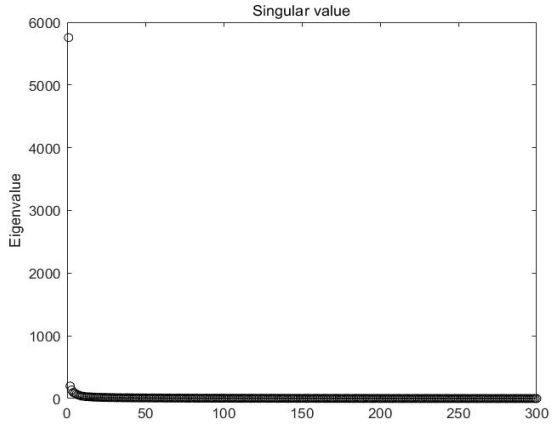
matrix R first by $R = X_{DMD}^{background} - |X_{DMD}^{background}|$, and then add this back to the foreground matrix, then we got that $X_{DMD}^{foreground} = X - |X_{DMD}^{background}| - R$.

With "Monte_Carlo_low.mp4", the procedure to construct the background video is the same, and I truncate the SVD of X1 matrix to 200 also.

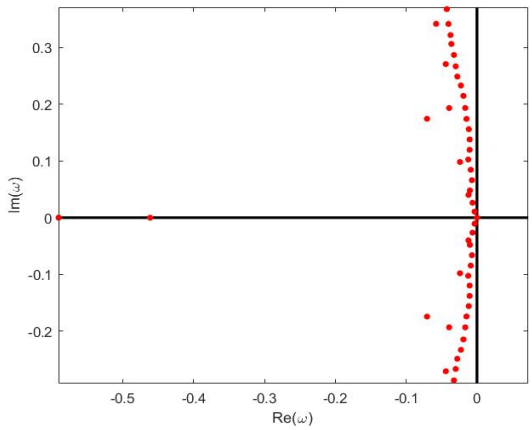
COMPUTATIONAL RESULT

"Ski_drop_low.mp4"

The eigenvalue from the SVD result and the eigenvalue of $\tilde{S}$ in the complex plane:



Eigenvalue plot of "Ski_drop_low.mp4"

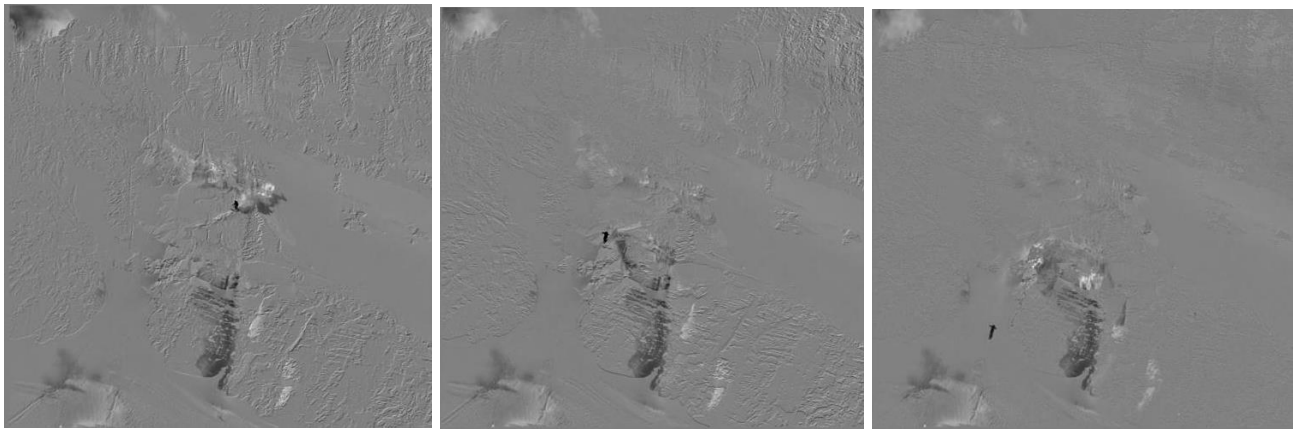


Eigenvalue of $\tilde{S}$

We can see that there is one mode that sit at the origin, and this mode is the mode I used to construct the background.

The first row are three screen shots from the original video at time = 1.88s, 3.79s, 6s; The middle row is the three frames of the foreground at these four times, and the last row is the four frames shows the background at these three times, note that for the foreground, to make it visible, I added some even number to the foreground matrix so that the skier is visible.

Screenshots of the original video

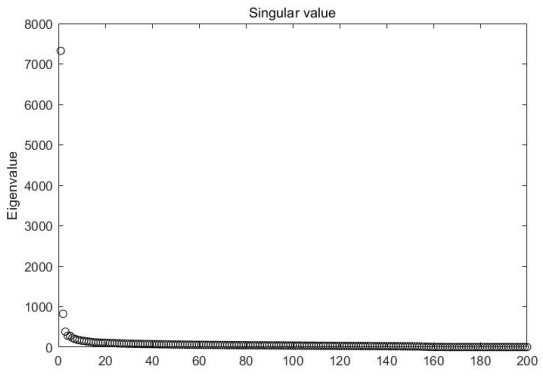


Screenshots of the foreground video

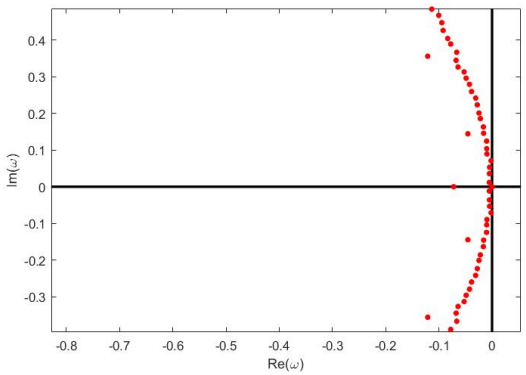
Screenshots of the background video

“Monte_Carlo_low.mp4”

First look at the eigenvalue from the SVD result and the eigenvalue of $\tilde{S}$:



Eigenvalue of SVD



Eigenvalue of $\tilde{S}$

Now let’s look at the screenshots from background video and foreground video that we reconstructed using DMD, and compare them with the original video. The first row contains three screenshots from

original, and the middle row contains 3 from the foreground, the bottom row contains 3 from the background. (The 3 time snapshots are: t = 1s, t = 2s, and t =5.2s)

Screenshots of the original video

Screenshots of the foreground video

Screenshots of the background video

CONCLUSION

From the result we can see that by using DMD method, we successfully split the video into background video and foreground video. This is an important application, by recognizing the modes that are constant throughout the time, we will be able to make significant subtraction from the data matrix by using DMD.

Appendix. A MATLAB functions used and brief implementation explanation

`Rbg2gray()`:takes an true color RBG image and returnsa grayscale image.

`svd(X, 'econ')`:takes an matrix X, returns a reduced Singular Value Decomposition of the m-by-nmatrix X, that is, the returned U only contained n columns, S is n-by-n, and only the first m columns of V are computed. This reduced form of SVD is sufficient for our explanation.

`Reshape(X,[],n)`: this function reshape the data X into an n-column vector, and the row number is calculated by MATLAB for a most suitable one.

Appendix. B MATLAB Code

```
clear all; close all; clc
% Read in the video
v1 = VideoReader("ski_drop_low.mp4");
%% "ski_drop_low.mp4" video
j = 1;
while hasFrame(v1)
    I = readFrame(v1);
    I = rgb2gray(I);
    X(:,j) = reshape(I,[],1);
    j = j + 1;
end
X = im2double(X);
dt = v1.duration/(j-1);
t = linspace(0,v1.duration,size(X,2));
%%
X1 = X(:,1:end-1);
X2 = X(:,2:end);


[U, Sigma, V] = svd(X1,'econ');
rank = 1:300;
U = U(:,rank);
Sigma = Sigma(rank,rank);
V = V(:,rank);

S = U'*X2*V*diag(1./diag(Sigma)); % S tilda
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV; % eigenvector
%%
omega_abs = abs(omega);
zero_ind = find(omega_abs < 0.5);
Phi_bg = Phi(:,zero_ind);

%%
subplot(1,2,1)
plot(diag(Sigma),'ko')
title("Singular value")
ylabel('Eigenvalue')

subplot(1,2,2)
% make axis lines
line = -15:15;

plot(zeros(length(line),1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(length(line),1),'k','Linewidth',2) % real axis
plot(real(omega)*dt,imag(omega)*dt,'r.','Markersize',15)
```

```
xlabel('Re(\omega)')
ylabel('Im(\omega)')
%% background
y0 = Phi_bg\X1(:,1);

u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
   u_modes(:,iter) = y0.*exp(omega(zero_ind)*t(iter));
end
u_background = Phi_bg*u_modes;

%% foreground
u_foreground = X - abs(u_background) + 0.5; % to make contrast, add some number to
raise the tone of 'background'

%% looking at the background
im = reshape(u_foreground(:,61),[540,960]);
imshow(im)
%% looking at the foreground
im = reshape(X(:,300),[540,960]); % t = 0s
imshow(im); drawnow;


%% "monte_carlo_low.mp4" video
clear all; close all;clc
v2 = VideoReader("monte_carlo_low.mp4");
j = 1;
while hasFrame(v2)
    I = readFrame(v2);
    I = rgb2gray(I);
    X(:,j) = reshape(I,[],1);
    j = j + 1;
end
X = im2double(X);
dt = v2.duration/(j-1);
t = linspace(0,v2.duration,size(X,2));
%%
X1 = X(:,1:end-1);
X2 = X(:,2:end);

[U, Sigma, V] = svd(X1,'econ');
rank = 1:300;
U = U(:,rank);
Sigma = Sigma(rank,rank);
V = V(:,rank);

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

```
%%
omega_abs = abs(omega);
zero_ind = find(omega_abs < 0.5);
Phi_bg = Phi(:,zero_ind);
%%
y0 = Phi_bg\X1(:,1); % pseudoinverse to get initial conditions

u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
   u_modes(:,iter) = y0.*exp(omega(zero_ind)*t(iter));
end
u_background = Phi_bg*u_modes;

%% foreground
u_foreground = (X - abs(u_background));

%% looking at the background
im = reshape(X(:,320),[540,960]);
imshow(im)
%% looking at the foreground
im = reshape(u_background(:,320),[540,960]); % t = 0s
imshow(im); drawnow;
```