AMATH 482 Homework 5: Video Background Subtraction

Leqi Wang

March 17, 2020

Abstract

In this assignment, we use Dynamic Mode Decomposition(DMD) to separate the background and foreground object of two videos.

1 Introduction and Overview

Dynamic Mode Decomposition is a dynamical process to separate governing equations and discover low rank structures in a multi-scale system[1]. DMD allows us to forecast the behavior of certain data set without depending on the governing equations. We have two videos of skiing and Monaco Grand Prix, using the method of DMD, we are going to separate the background and the moving object in the foreground according to specific properties in the DMD process.

2 Theoretical Background

2.1 Koopman Operator

Koopman Operator A is a linear and time independent operator making

$$x_{j+1} = Ax_j \tag{1}$$

where j indicates the time data x is collected, so that the non-linear systems could get a best approximation by mapping previous snapshots to next snapshots with span of Δt. Note: Koopman operator is not limited to an area of space or time, it is characterized as a "global" operator.

2.2 Dynamic Mode Decomposition (DMD)

From a video clip, we have different snapshots, given these snapshots, we organize pixels in a data matrix X and X' to be current matrix and future matrix, with each column expressing the x by y pixels matrix: X = $\left[\vec{x_1} \quad \vec{x_2} \quad ... \quad \vec{x_m - 1}\right]$, X' = $\left[\vec{x_2} \quad \vec{x_2} \quad ... \quad \vec{x_m}\right]$ where m is the length of the video. DMD tries to find the Koopman operator A so that we get the best approximation of:

$$X' = AX \tag{2}$$

that's

$$A = X' \times X^T \tag{3}$$

But since A is huge to compute, DMD approximate the leading eigen decomposition without compute the actual $X2 \times X1^T$. Once we have the eigen decomposition, the eigen vectors are in columns of the data matrix, they are the spatial modes (interpreted same as each col of X), which corresponds to a coherent pure tone of waves/exponential decay/growth, or combination of these.

Step1 − SVD : Since in most high dimensional complex systems like disease, there's a highly dominant repetitive pattern even if there's million degrees of freedom. Therefore use SVD to extract the most dominant

pattern by truncate useful ranks of A.

$$\begin{aligned} X &= U\Sigma V^*, \\ X' &= AX, \\ X' &= A \times U\Sigma V^* \end{aligned} \tag{4}$$

Step2 Compute the projection of A onto the singular vectors U:

$$A = X'X^T = X'V\Sigma^{-1}U^* \tag{5}$$

Now, take $\widetilde{A} = U^*AU$, $\widetilde{A}$ and A has the same eigen values. Therefore:

$$\widetilde{A}W = W\Lambda \tag{6}$$

and

$$\Phi = X'V\Sigma^{-1}W \tag{7}$$

The columns of Φ should be the eigen spatial DMD modes. After expansion the eigen basis, we'll have

$$X_{DMD}(t) = \sum_{k=1}^{K} b_k\phi_k exp(\omega_k)t \tag{8}$$

with (x_1 being random initial value)

$$b = \Phi^T x_1 \tag{9}$$

3 Algorithm Implementation and Development

I first truncate the video and set certain time span (Δt) to include the frames I want. I converted these frames to gray scale and stuck them into data matrix X, each column of X represent a frame. m is length of the video and n is size of each frame. Then I chose my data matrix X1 (current time) to be from first frame to m - 1 frame, X2 to be from 2nd frame to m'th frame. I calculate the SVD of X1 (figure1 and 2) and decide the rank I need to keep, then truncate USV matrix. I used following equation (X' as X2)

$$\begin{aligned} \widetilde{A} &= U * X'V\Sigma^{-1} \\ [W, D] &= eig(\widetilde{A}) \end{aligned} \tag{10}$$

to find eigen basis, then I get the DMD modes by equation(8). According to the equation given in spec:

$$X_{DMD} = b_p\phi_p e^{\omega_p t} + \sum_{j!=p} b_j\phi_j e^{\omega_j t} \tag{11}$$

$b_p\phi_p e^{\omega_p t}$ indicates the data matrix for the background, where for all p, the norm of ω_p is approximately zero because the dominant basis for background should not be influenced by time changing, in a graph, every basis looks like a straight line. Thus I can separate the eigen basis and eigen values for background. Afterwards, I tried to subtract the low rank matrix directly from the data matrix, but sometimes I get negative pixel values. So I constructed a matrix R to be the residual negative values by command $R = X_{fg}. * (X_fg < 0)$. I made the sparse matrix = so that the result in the foreground is more visualizable.

4 Computational Results

4.1 PartI: SVD of X1

I used rank = 20 to truncate the SVD results of X1 for both cases because first 10 rank already contains major information, but I still kept additional 10(figure 1).

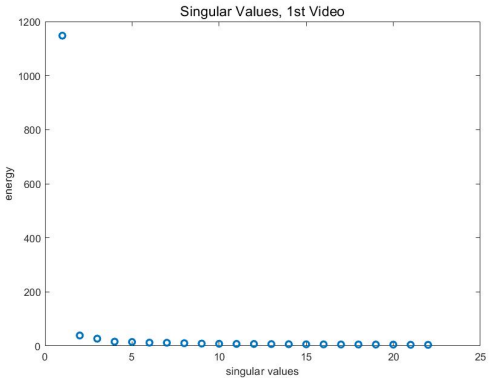

Figure 1: Singular Values of Ski Video

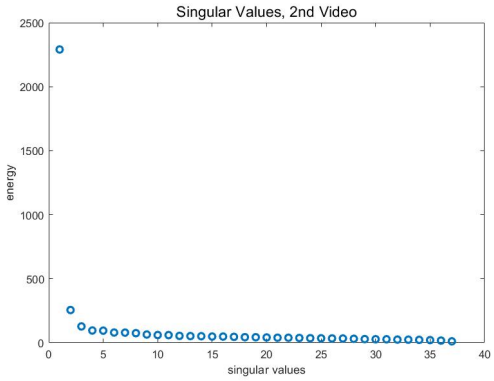

Figure 2: Singular Values of $\text{Monte}_C arlo$

4.2 Part II DMD Subtraction

Figure 3 is the result of the first video background subtraction. The left shows background at frame number equals 100, 200, and 420; right column shows foreground frame at frame number equals 100, 200, and 420.

Figure 4 is the result of the second video background subtraction. The left shows background at frame number equals 90, 200, and 380; right column shows foreground frame at frame number equals 90, 200, and 380.

5 Summary and Conclusions

DMD is like the mix of SVD and FFT, based on the former we construct linear regression model so that we know the principal components, and the later tells what the dominant dynamics is with respect to time. It can be used to subtract background in a video, after which we can manipulate the data matrix and get a foreground.

References

[1] Steve Brunton. *Dynamic Mode Decomposition (Overview)*. Youtube. 2018. URL: `https://www.youtube.com/watch?v=sQvrK8AGCAo&t=1s`.

Appendix A MATLAB Functions

- Loading section: `VideoReader(filename)`: read in movie files, I used its property NumFrames as the length of row of data matrix.
- `im2double(rgb2gray(read(v,i)))`: convert each frame into grayscale and store them as columns of data X.
- `reshape(frames, x * y,1)`: reshape the data from x by y to x * y by 1, vice versa.
- `svd(X1, 'econ')`: calculate the reduced singular value decomposition.
- `[W, D] = eig(A`$_t ilde$`)` : $find the eigen decomposition for A_t ilde (also for A)$.`diag()` : $find the diagonal of a matrix$.
- `setdiff(1:r, bg)`: find the difference between two arrays. I used it to calculate j's for equation 11.
- `imwrite`: write the result to a jpg file.
- `imshow`: show the result of subtracted background and foreground, used for testing.

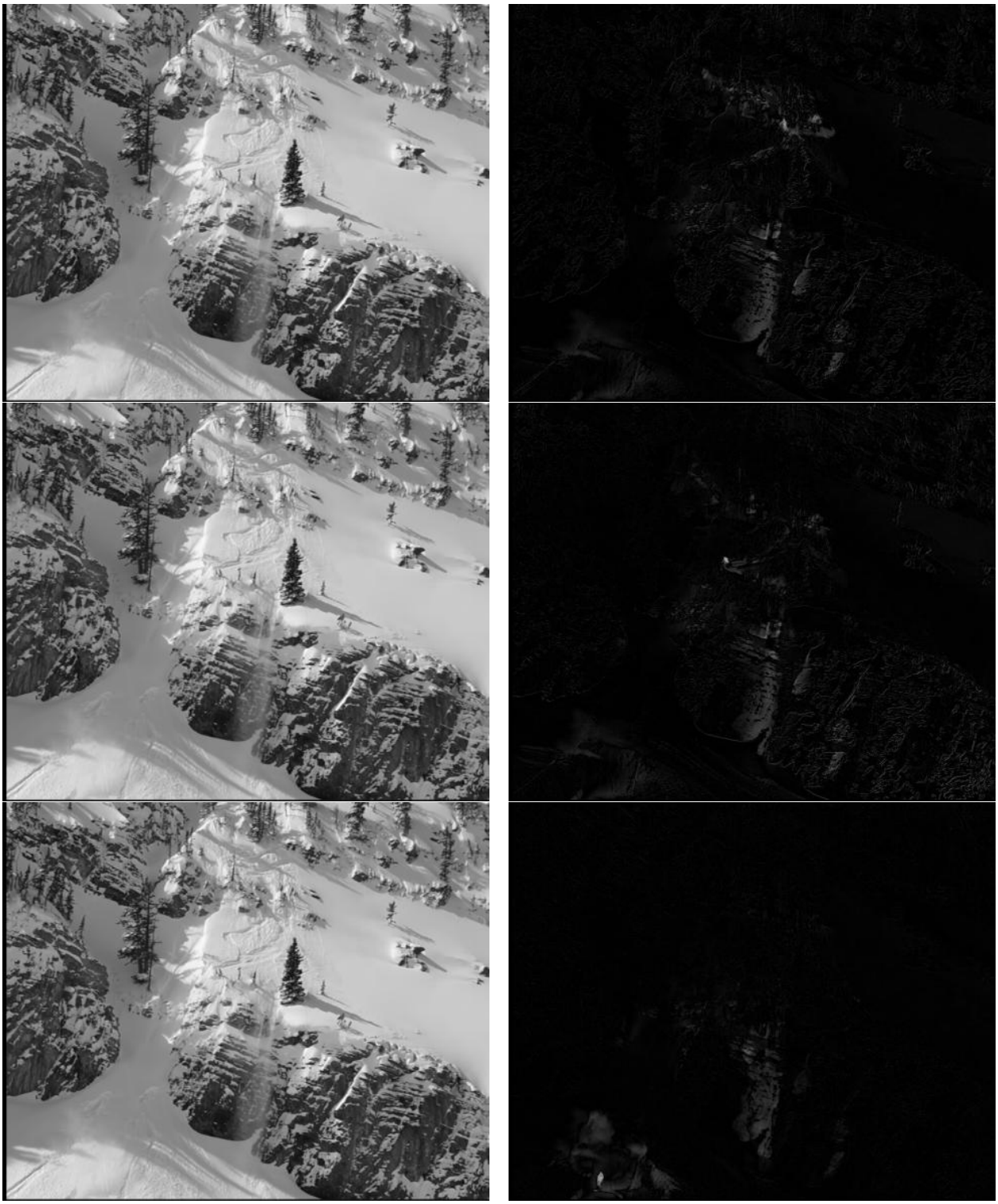

Figure 3: Left: Background at frame 100, 200 and 420; Right: Foreground at frame 100, 200 and 420

Appendix B MATLAB Code

```
clear all; close all; clc;
```

Figure 4: Left: Background at frame 90, 200 and 380; Right: Foreground at frame 90, 200 and 380

```
2
3  %%
4  % filename = 'ski_drop_low.mp4';
5  filename = 'monte_carlo_low.mp4';
6  v = VideoReader(filename);
7  numFrames = v.NumFrames; % time(s)
8  m = 1;
9  dt = 10;
10 % prediction = [230 140 484 395]; % crop ski video;
11 for i = 1:dt:numFrames % construct X
12     frames = im2double(rgb2gray(read(v,i))); % read(v,i) to read every frames
13 %     frames = imcrop(frames, prediction);
14     [x, y] = size(frames);
15     frames = frames(1: x, 1: y);
16     frames = reshape(frames, x * y,1);
17     X(:, m) = frames;
18     m = m + 1;
19 end
```

```
m = m - 1;
n = x * y;
X1 = X(:, 1:end - 1); % data matrix
X2 = X(:, 2:end);
[U, S, V] = svd(X1, 'econ');

figure(1);
plot(diag(S), 'o', 'Linewidth', 2);
title("Singular Values, 2nd Video", 'FontSize', 14);
xlabel('singular values');
ylabel('energy');

r = 20; % keep ranks
U_trunc = U(:, 1:r); S_trunc = S(1:r, 1:r); V_trunc = V(:, 1:r);
A_tilde = U_trunc' * X2 * V_trunc / S_trunc;
[W, D] = eig(A_tilde); %W = basis;
Phi = X2 * V_trunc / S_trunc * W; % DMD eigen vectors

lambda = diag(D);
omega = log(lambda) / dt;

%%
bg = find(abs(omega) < 1e-2); % calculate p for background
fg = setdiff(1:r, bg); % foreground(j != p)

omega_bg = omega(bg); % background
Phi_bg = Phi(:,bg);
omega_fg = omega(fg); % foreground
Phi_fg = Phi(:,fg);

%% background
x1 = X(:, 1); % use first frame as initial condition
bp = transpose(Phi_bg) * x1;
% reconstruct background
X_bg = zeros(3, m);
for i = 1:m
    t = i * dt; % which frame
    X_bg(:, i) = bp .* exp(omega_bg .* t);
end
X_bg = Phi_bg * X_bg;
X_bg_reshape = reshape(X_bg, x, y, m);

%% foreground
X_fg = X - abs(X_bg);

% isolation of foreground by R
R = X_fg .* (X_fg < 0);
X_fg = X_fg - R - R;
X_fg_reshape = reshape(X_fg, x, y, m);

%% gen pics for monte_carlo
imwrite(X_bg_reshape(:, :, 9), 'car_bg1.jpg');
imwrite(X_bg_reshape(:, :, 20), 'car_bg2.jpg');
imwrite(X_bg_reshape(:, :, 38), 'car_bg3.jpg');
```

```
74 imwrite(X_fg_reshape(:, :, 9), 'car_fg1.jpg');
75 imwrite(X_fg_reshape(:, :, 20), 'car_fg2.jpg');
76 imwrite(X_fg_reshape(:, :, 38), 'car_fg3.jpg');
77
78 %% gen pics for ski
79 imwrite(X_bg_reshape(:, :, 10), 'ski_bg1.jpg');
80 imwrite(X_bg_reshape(:, :, 20), 'ski_bg2.jpg');
81 imwrite(X_bg_reshape(:, :, 42), 'ski_bg3.jpg');
82 imwrite(X_fg_reshape(:, :, 10), 'ski_fg1.jpg');
83 imwrite(X_fg_reshape(:, :, 20), 'ski_fg2.jpg');
84 imwrite(X_fg_reshape(:, :, 42), 'ski_fg3.jpg');
85 %%
86 for i = 1:m
87     fg_t = X_fg_reshape(:, :, i); % test frame background @ t
88     imshow(fg_t)
89 end
```