AMATH 482 Homework 5

Jiatian Xie

March 17, 2021

Abstract

This paper aims to illustrate the background substraction in video streams. I am going to use the Dynamic Mode Decomposition(DMD) method on two video clips ski drop(ski) and monte carlo(monte) that containing a foreground and background object and separate the video stream to both the foreground video and a background video.

1 Introduction and Overview

In this assignment, we are given two video streams containg both foreground and background object. In order to separate them, I am planning to apply the SVD to speed up and then use Dynamic Mode Decomposition(DMD) to extract two videos.

DMD allows us to forecast our future. It is helpful to deal with low-dimensinality in experimental data without relying other equations. The basic idea of DMD is to take a "snapshot" of data in each time, and then predict the state of the next time.

2 Theoretical Background

The DMD spectrum of frequencies is used to subtract background modes. And the DMD method approximates the modes of Koopman operator. The general idea is:

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

Assume we firstly have a snapshot of the data from t = 1,2,3,... m, and another snapshot shifted by one frame in time is t = 2,3,4,...,m+1. To construct an appropriate Koopman operator, we need to first construct a matrix based on the previous setting:

$$X_1^{M-1} = [x_1 x_2 x_2 x_{M-1}]$$

where we can use x_j to represent a snapshot at time t_j. Therefore, we can write the Koopman operator as:

$$X_1^{M-1} = [x_1 A x_1 A^2 x_1 A^{M-2} x_1]$$

That is:

$$X_2^M = AX_1^{M-1} + re_{M-1}^T$$

With the concept of DMD, we can use this in order to distinguish between foreground and background. A video can be represented as:

$$X_{DMD} = b_p \phi_p e^{w_p t} + \sum b_j \phi_j e^{w_j t}$$

Where the first term $b_p \phi_p e^{w_p t}$ represents the background video and $\sum b_j \phi_j e^{w_j t}$ is the foreground video.

This poses a problems when sepratating the DMD terms into approximate low-rank and sparse reconstruction because real valued outputs are desired. Now, consider calculating the DMD's approximate low-rank reconstruction according to:

$$X_{DMD}^{Low-Rank} = b_p \phi_p e^{w_p t}$$
$$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$$

then the DMD's approximate sparse reconstruction can be found with real-valued elements as:

$$X_{DMD}^{Sparse} = X - |X_{DMD}^{Low-Rank}|$$

However, in order to eliminate the negative values in X_{DMD}^{Sparse}, I need to create a n*m matrix R to store these residual negative values and then added back into $X_{DMD}^{Low-Rank}$:

$$X_{DMD}^{Low-Rank} \leftarrow R + |X_{DMD}^{Low-Rank}|$$
$$X_{DMD}^{Sparse} \leftarrow X_{DMD}^{Sparse} - R$$

This approach helps to ensure the approximate low-rank and sparse DMD reconstructions are real-valued.

3 Algorithm Implementation and Development

Here, I would like to share my algorithm and development for dealing with monte clip. To begin with this question, we need to firstly convert two video clips to 3D matrix. The monte video is a 960*540*379 3D matrix and I reshaped it to (960*540)*379 2D matrix, where 379 is the number of frames. More importantly, I found that several frames have the same data. Thus, I deleted the repeated data frame in while loop.

Once we have the video matrix, we are able to extract X_1 and X_2. Then, we can perform an SVD on X_1 to truncate our data and get the principal modes that we need to get a low-rank construction of the video. I will use `m` to represent the number of modes that I need. Hence, I only take the first `m` columns of U, V matrices and the first `m` values of Σ matrix. Now, we can create the matrix:

$$\tilde{S} = U * X_2^M V \Sigma^{-}1$$

and find its eigenvalues and eigenvectors by using the Koopman operator $A\phi = \phi\omega$. Then, we can use it to convert to **omega** vector which contains another form of eigenvalue. Also, according to the specification, $\|\omega\| \approx 0$. See Algorithm 1

Algorithm 1: Find Omega

Based on the Koopman operator $A\phi = \phi\omega$

```
S = U'*v2*V*diag(1./diag(Sigma));
compute eigenvalues + eigenvectors [eV, D] = eig(S);
extract eigenvalues mu = diag(D);
omega = log(mu);
```

Now, I can use the initial snapshot `v1` and the pseudoinverse of Φ to find the coefficient b_k. Once we have calculated the background video matrix $X_{DMD}^{Low-rank}$, we can then subtract it from the original video matrix to get X_{DMD}^{Sparse}, which represents the foreground video matrix.

To avoid the negative values in X_{DMD}^{Sparse} as I mentioned above, I firstly found the residual matrix and add it back to $X_{DMD}^{Low-rank}$ matrix. Finally, subtract that matrix from the sparse video matrix to get our new DMD matrices.

I repeat the similar process for ski video, but use the larger dt=10 in order to get a clear video since with larger dt, the number of frames is smaller.

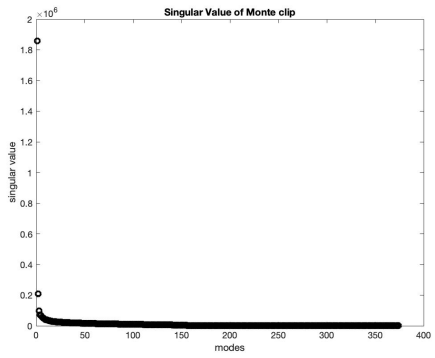

Figure 1: Here is a plot showing the singular values I got from the svd in the monte carlo clip. I can see that the number of dominant modes from this graph.

4 Computational Results

For the first video, monte carlo, which recorded a short period of racing around the corner. I firstly calculated the singular value of v1 and we can see 1, the first two value is much larger than others, therefore, I only use two modes in this case and get the following result of background video and foreground video. See Figure 2. In my view, the separation here works mostly prefect. I am able to see in the background video snapshots, the car is mostly filter out of the video. And the motions of the cars were clear in the foreground video snapshots.

For the second video, ski drop, which recorded a skier's movement. The singular value I found is 3. From this plot, I decide to use 2 modes to separate the video. The background and foreground video I got is good. See Figure 4, it is easy for me to distinguish the skier and other movement of skier from the snapshots. For example, the last column represents the three snapshots, in the second row, we cannot see the skier and the light spot in the third row is the skier. Comparing to my result from the first video, this one is better maybe because of the better choice of t and dt.

5 Summary and Conclusions

With the study of this assignment, I know that a video can be separated into a background video and a foreground video. In order to do this separation, I used a Dynamic Mode Decomposition(DMD). Through DMD, I am able to get a low-rank approximation of the original data of video. However, the results I got are very good for me. Although they are still not prefect since DMD depends on the distinctive components, that is, the singular values. According to my calculation, I found that there is still some distinctions that are hard to separate background and foreground because of the lack of contrast. Also, depending on my observation of the original video, I feel like there is some noise when recording like movement and stillness of the features.

I enjoy doing this DMD method for this assignment since it talked how to separate the video stream to both the foreground video and background video. Although my result is not good, I believe there are several ways to improve it including removing the noise or finding the sensitive dependencies in the background movement.

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that `A = U*S*V'`.

Figure 2: Here are the snapshots of the original video, the background, and the foreground at frames 30,60,90 in the monte carlo video. The first row is the original video, the second row is background video and the last row is the foreground video.

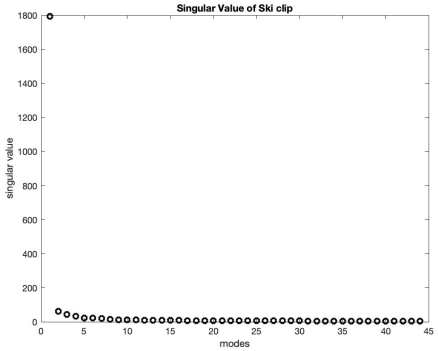

Figure 3: Here is a plot showing the singular values I got from the svd in ski drop clip. I can see that the number of dominant modes from this graph.

Figure 4: Here are the snapshots of the original video, the background, and the foreground at frames 10,20,30 in the ski clip. The first row is the original video, the second row is background video and the last row is the foreground video.

- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X.
- `rgb2gray` converts the truecolor image RGB to the grayscale image.
- `B = reshape(A,sz)` reshapes A using the size vector, sz, to define size(B).

Appendix B MATLAB Code of separation of Monte Carlo.mp4

```
v = [];
vreader = VideoReader('monte_carlo_low.mp4');
numFrame = vreader.NumFrames;
frame = readFrame(vreader);
frame = rgb2gray(frame);
frame = reshape(frame,[],1);
v = frame;
while hasFrame(vreader)
    frame = readFrame(vreader);
    frame = rgb2gray(frame);
    frame = reshape(frame,[],1);
    if sum(abs(frame - v(:,end))) ~= 0
        v = [v,frame];
    end
end
v = double(v);
v1 = v(:,1:end-1);
v2 = v(:,2:end);
[U1, Sigma1, V1] = svd(v1,'econ');

m = 2;
Sm = Sigma1(1:m,1:m);
Um = U1(:,1:m);
Vm = V1(:,1:m);
S1 = Um'*v2*Vm*diag(1./diag(Sm));
[eV1, D1] = eig(S1); % compute eigenvalues + eigenvectors
mu1 = diag(D1); % extract eigenvalues
omega1 = log(mu1);
Phi1 = v2*Vm/Sm*eV1;
y0 = Phi1\v1(:,1); % pseudoinverse to get initial conditions
u_modes1 = zeros(m,size(v1,2));
for i = 1:size(v1,2)
   u_modes1(:,i) = y0.*exp(omega1*i);
end
u_dmd1 = Phi1*u_modes1;
u_sparse1 = abs(v1 - u_dmd1);

residual_matrix1 = u_sparse1 .* (u_sparse1 < 0);
u_dmd1 = residual_matrix1 + abs(u_dmd1);
u_sparse1 = u_sparse1 - residual_matrix1;
```