AMATH 482 Homework 5

Jessica Shi, 1762333

March 14, 2021

Abstract

In this homework, we apply the Dynamic Mode Decomposition (DMD) to the given 2 video clips. The given videos have both foreground and background objects. We use the DMD algorithm to separate the foreground object and background object and then do the video reconstruction.

1 Introduction and Overview

We are given 2 different video clips that both contain a foreground and background objects. Our task is to separate these objects. Here, a proper method to use is the Dynamic Mode Decomposition, also known as DMD. It allows us to rely directly on the low-dimensionality of data but not a given set of governing equations. This is a data-based approach. It helps us to find a basis of spatial modes and turn the complex problem into linear systems of differential equations. I would use this approach to separate the foreground and background objects of the given videos and then compare the result.

2 Theoretical Background

The fundamental concept used in this homework is the Dynamic Mode Decomposition (DMD).

2.1 Dynamic Mode Decomposition

First, we do the basic setup of the algorithm. Assume we have the snapshots of spatio-temporal data that involves both space and time. And the time data is collected at regularly spaced intervals. We define N to be the number of spatial points saved per unit time snapshot and M to be the number of snapshots taken. And We denote the snapshots as

$$U(\mathbf{x}, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ ... \\ U(x_n, t_m) \end{bmatrix} \tag{1}$$

for each $m = 1, ..., M$. Then, we are able to use the snapshots to form the columns of data matrices

$$\mathbf{X} = \begin{bmatrix} U(\mathbf{x}, t_1) & U(\mathbf{x}, t_2) & ... & U(\mathbf{x}, t_M) \end{bmatrix} \tag{2}$$

and

$$\mathbf{X}_j^k = \begin{bmatrix} U(\mathbf{x}, t_j) & U(\mathbf{x}, t_{j+1}) & ... & U(\mathbf{x}, t_k) \end{bmatrix} \tag{3}$$

which represents the column j through column $k$of the full matrix $\mathbf{X}$.

Then, we introduce the idea of the Koopman Operator. The Koopman Operator $\mathbf{A}$ is a linear, time-independent operator such that $\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j$ where j shows the data collection time and $\mathbf{A}$ maps the data from time t_j to t_{j+1}. This operator helps us to advance the snapshot of data forward in time by Δt.

Next, we try to construct the proper Koopman operator for the DMD algorithm. We consider the matrix

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & ... & \mathbf{x}_{M-1} \end{bmatrix} \tag{4}$$

$\mathbf{x}_j$ denotes a snapshot of the data at time t_j. According to the previous idea, we could rewrite the matrix as

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{A}\mathbf{x}_2 & \mathbf{A}^2\mathbf{x}_2 & ... & \mathbf{A}^{M-2}\mathbf{x}_1 \end{bmatrix} \tag{5}$$

by applying the powers of $\mathbf{A}$ to the vector $\mathbf{x}_1$. Therefore, here we have a way to relate the first $M-1$ snapshots to $\mathbf{x}_1$ using the Koopman operator. We write it as the matrix form:

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T \tag{6}$$

Here, since we don't include the final point $\mathbf{x}_M$, we add in the residual vector $\mathbf{r}$ to account for it.

We then try to find $\mathbf{A}$ by finding another matrices with same eigenvalues. First, we use SVD to get $\mathbf{X}_1^{M-1} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* + \mathbf{r}e_{M-1}^T$. Then we could also get $\mathbf{X}_2^M = \mathbf{A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* + \mathbf{r}e_{M-1}^T$. According to this, we want to choose $\mathbf{A}$ to ensure that $\mathbf{X}_2^M$ can be written as linear combinations of columns of $\mathbf{U}$, which is the same as to require that they can be written as linear combinations of POD modes. So, $\mathbf{r}$ should be orthogonal to the POD basis and we get $\mathbf{U*r} = 0$.

Thus, we could get

$$\mathbf{U*X}_2^M = \mathbf{U*A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{7}$$

By multiplying by $\mathbf{V}$ and Σ^{-1}, we could get $\mathbf{U*A}\mathbf{U} = \mathbf{U*X}_2^M\mathbf{V}\Sigma^{-1}$.

Here, we use $\tilde{\mathbf{S}}$ to represent the right hand side. And we know that it is similar to $\mathbf{A}$ and thus have same eigenvalues. Also, $\tilde{\mathbf{S}}\mathbf{y}_k = \mu_k\mathbf{y}_k$. Therefore, we could get the eigenvectors of $\mathbf{A}$, $\phi_k = \mathbf{U}\mathbf{y}_k$, which are called the DMD modes.

We then could expand the eigenbasis to get

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k\varphi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t})\mathbf{b} \tag{8}$$

The b_k are the initial amplitude of each mode, and the matrix Ψ contains the eigenvectors as its columns. Since we write the time dynamics as exponential functions, we could get $\omega_k = ln(\mu_k)/\Delta t$. Since we just write the dynamics in terms of a linear combination of exponential growth/decay/oscillation in the direction of each eigenvector, by taking $t = 0$ we could get

$$\mathbf{x}_1 = \Psi\mathbf{b} \tag{9}$$

$$\mathbf{b} = \Psi\,\dagger\,\mathbf{x}_1 \tag{10}$$

where $\Psi\dagger$ is the pseudoinverse of Ψ.

2.2 Video Reconstruction

Now that we understand the DMD algorithm, we could use it to do video reconstruction. The DMD spectrum of frequencies can be use to subtract the background modes in the video clips.

Assume that ω_p with $p \in \{1, 2, ..., l\}$ and $\|\omega_p\| \approx 0$. Also, $\|\omega_p\|$ $\forall j \neq p$ is bounded away from 0. Thus, we could get

$$\mathbf{X}_{DMD} = b_p \varphi_p e^{\omega_p t} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t} \tag{11}$$

where the first element represents the background video and the second element is the foreground video. Although each term is complex, we get a matrix $\mathbf{X}_{DMD}$ that has the same dimension as the original $\mathbf{X}$.

After calculating $\mathbf{X}_{DMD}^{Low-Rank} = b_p \varphi_p e^{\omega_p t}$ and we know that $\mathbf{X} = \mathbf{X}_{DMD}^{Low-Rank} + \mathbf{X}_{DMD}^{Sparse}$, then we get $\mathbf{X}_{DMD}^{Sparse} = \sum_{j \neq p} b_j \varphi_j e^{\omega_j t}$ and we could calculate it with real-valued elements as

$$\mathbf{X}_{DMD}^{Sparse} = \mathbf{X} - |\mathbf{X}_{DMD}^{Low-Rank}| \tag{12}$$

But this might make $\mathbf{X}_{DMD}^{Sparse}$ have negative values in some elements. But it does not make sense to have negative pixel intensities. To deal with this situation, we put the residual negative values into a $n \times m$ matrix $\mathbf{R}$ and add it back into $\mathbf{X}_{DMD}^{Low-Rank}$ as follows:

$$\begin{aligned} \mathbf{X}_{DMD}^{Low-Rank} &\leftarrow \mathbf{R} + |\mathbf{X}_{DMD}^{Low-Rank}| \\ \mathbf{X}_{DMD}^{Sparse} &\leftarrow \mathbf{X}_{DMD}^{Sparse} - \mathbf{R} \end{aligned}$$

This way while we keep the constraint that $\mathbf{X} = \mathbf{X}_{DMD}^{Low-Rank} + \mathbf{X}_{DMD}^{Sparse}$, we could also ensure that the approximate low-rank and sparse DMD reconstructions are all real values.

3 Algorithm Implementation and Development

First, I load the given video clip into MATLAB. I choose the proper Δt and time vector and then loop over all the video frames, convert the image matrix into gray scale and then get the double values. In order to save computation time, I resize the images and reshape them into column vectors. After the loop is done, I get the matrix `data` with each column as a gray scale image.

Next, I start to do the DMD algorithm. I split the data into 2 matrices $\mathbf{X}_1$ and $\mathbf{X}_2$ and then do the SVD to $\mathbf{X}_1$. I then plot the singular values and energies to determine the proper rank to use. After deciding the rank, I truncate the resulting `U`, `S`, `V` matrices to do low-rank approximations. And I do the eigenvalue decomposition of the new matrix `A_tilde` and create the DMD modes. We could now find the eigenvalues and be able to get the `omega` by taking log of the eigenvalues and then divide by `dt`.

Then, I start to do the video analysis and reconstruction. I set the initial condition to be the first column of $\mathbf{X}_1$ and then solve for `b`. By looping over the time, I create a time dynamics matrix and get the low-rank DMD approximation. Since $\mathbf{X}_1$ is the sum of low-rank and sparse matrix, we could get the sparse reconstruction approximation. I then create the residual matrix `R` of the sparse matrix. In order to get better approximation, I add `R` to the absolute value of the low rank matrix and subtract `R` from the sparse matrix.

Finally, I pick a frame(column) and plot the reconstruction results from the above algorithm so that we could compare it with the original video. I repeat the whole process for the second video clip.

4 Computational Results

4.1 Video 1: Monte Carlo

This is a video about a car racing game. In the video, there are several cars passed through the camera and the angle of the camera is fixed. The background object is the venue and the track while the foreground object are the cars.

From Figure 1, we could know that the first mode captured almost all the energy and there is a big gap between the first and second mode. Here, I choose the rank to be 2 in order to have a better reconstruction

result since there is also a small gap between the second mode and the third mode. The second mode also captured some energy.

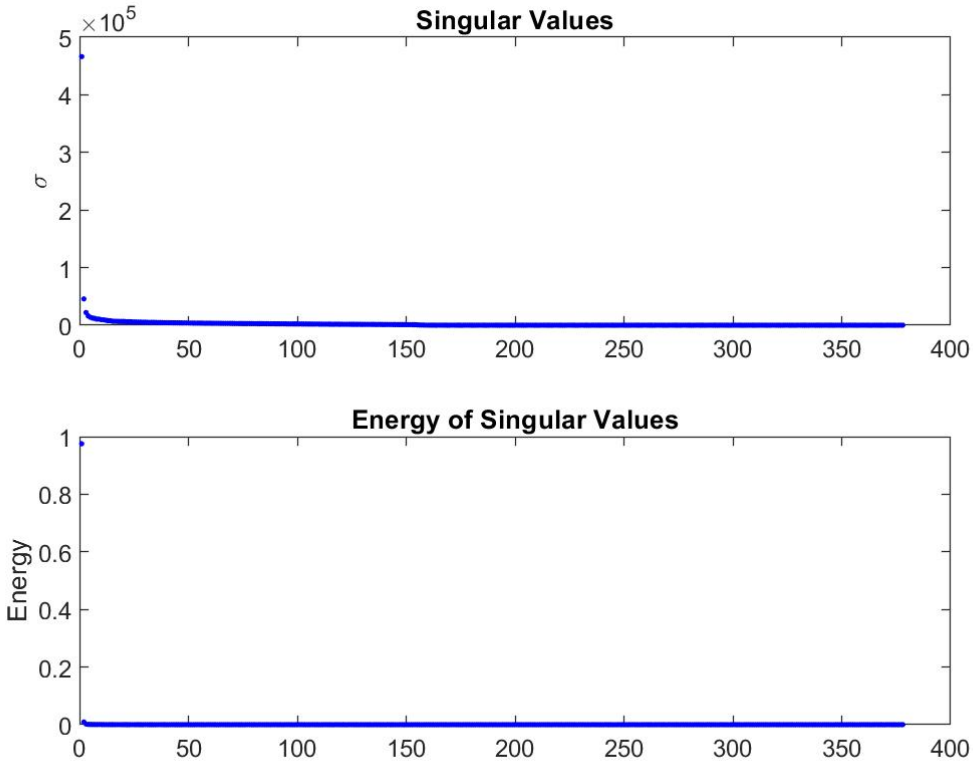

Figure 1: Energy and Singular Values for Video 1

Figure 2 is the reconstruction result for this video 1. I choose the frame 100 since in this frame we could see a car in the video, which is the foreground object. From Figure 2 we could see that the sparse reconstruction with R matrix and without R matrix are similar. In both plot, the car is relatively clear to see than the background object. For the low-rank reconstruction, we could see that without the R matrix, the DMD algorithm does a good work, we could hardly see the car in the plot while the background is clear to see. But adding the R matrix to the low-rank reconstruction make it worse since the car is present again. Overall, the reconstruction works well since the final total reconstruction looks very similar to the original frame 100 in the video clip.

4.2 Video 2: Ski Drop

This video is about a person skiing in the snow. The difference between this video and the last one is that the foreground object is relatively small, if we consider the person to be the foreground object. Thus, it might be hard to separate the person from the background object. Similarly, we first do the SVD to the video data matrix and find that the first mode almost captured all the energy in this case. Therefore, here I choose the rank to be 1. The result is shown in Figure 3.

Figure 4 is the reconstruction result for this video 2. I choose the frame 114 since in this frame we could see that the person is in the center of the video. In the sparse reconstruction, we could hardly see a proper shape for the foreground object since the person is small comparing to the background. There are only some white blurry mark which might be the shadow of the snow ground. For the low-rank reconstruction, due to the low resolution of the video, it might be hard to see from the plot at first. But if we zoom in, in the

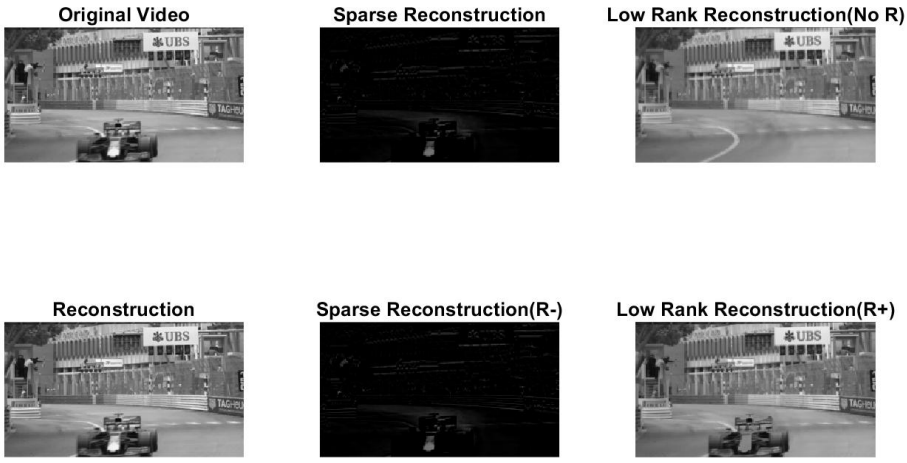

Figure 2: Energy and Singular Values for Video 1

low-rank reconstruction without R matrix, we could hardly see the person. On the contrary, in the low-rank reconstruction with R matrix, we could see the person. So here the DMD also did a good work. Overall, the reconstruction works well since the final total reconstruction looks very similar to the original frame 114 in the video clip. A minor flaw is that the low resolution of the video makes it hard to distinguish using our eyes.

5 Summary and Conclusions

To conclude, the Dynamic Mode Decomposition (DMD) is a proper algorithm to use for separating foreground and background object and do the video reconstruction under most situations. If the foreground object is easy to distinguish, then the process would work well. The background and foreground objects can be properly separated and the final total reconstruction is very close to the original video, as shown in both of our video clips. For the videos with a low resolution, it might be hard for us to distinguish using bare eyes, but the DMD algorithm still works.

Appendix A MATLAB Functions

- `v = VideoReader(filename)` creates object `v` to read video data from the file named filename.
- `B = reshape(A,sz)` reshapes `A` using the size vector, `sz`, to define `size(B)`.
- `x = diag(A)` returns a column vector of the main diagonal elements of `A`.
- `plot(X,Y)` creates a 2-D line plot of the data in `Y` versus the corresponding values in `X`.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix `A`, such that `A = U*S*V'`.
- `RGB = frame2im(F)` returns the truecolor (RGB) image from the single movie frame `F`.
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image `I`.
- `B = imresize(A,scale)` returns image `B` that is scale times the size of `A`. The input image `A` can be a grayscale, RGB, or binary image.

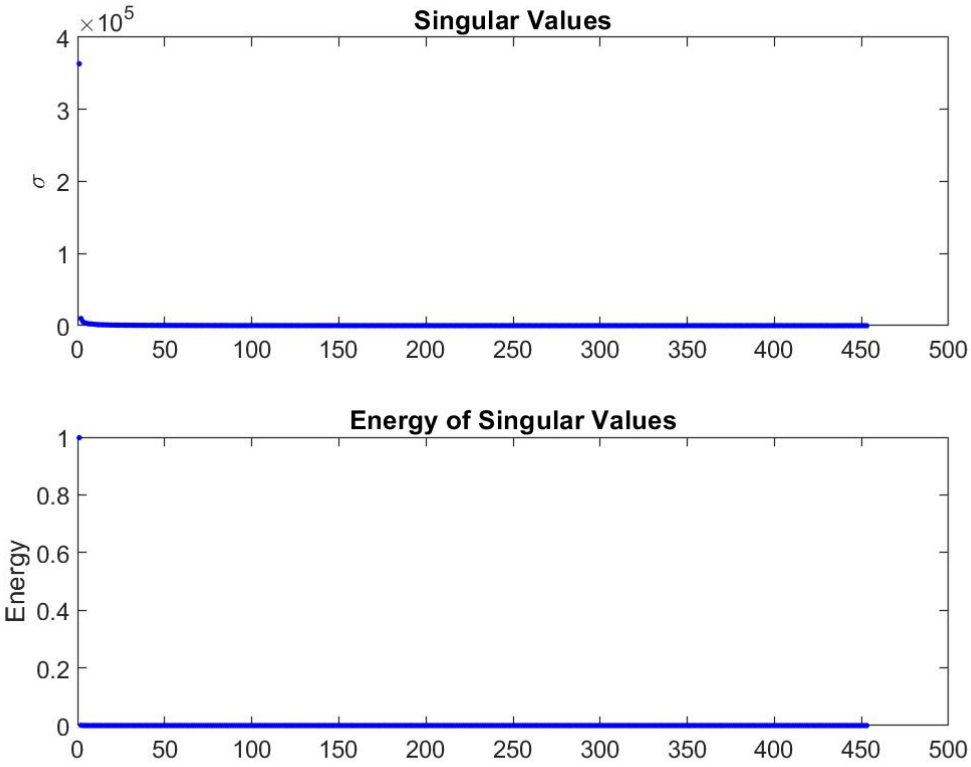

Figure 3: Energy and Singular Values for Video 2

- `[V,D] = eig(A)` returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that `A*V = V*D`.

Appendix B MATLAB Code

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


%% load the video and reshape ---------------------------------------------
% change the video file name here
% vid1 = VideoReader('monte_carlo_low.mp4');
vid1 = VideoReader('ski_drop_low.mp4');
dt = 1 / vid1.Framerate;
t = 0:dt:vid1.Duration;
vidFrames = read(vid1);
numFrames = get(vid1, 'numberOfFrames');

for i = 1:numFrames
    mov(i).cdata = vidFrames(:, :, :, i);
    mov(i).colormap = [];
end

data = [];
for j = 1:numFrames
    X = frame2im(mov(j));
    X = double(rgb2gray(imresize(X, 0.25)));
```

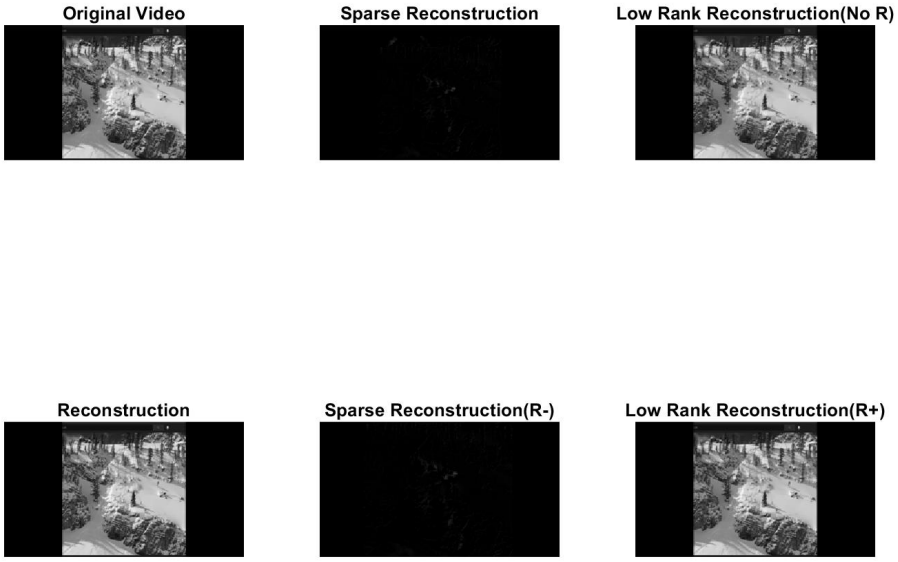

Figure 4: Energy and Singular Values for Video 2

```
    s = size(X);
    a = s(1);
    b = s(2);
    data = [data, reshape(X, [a*b,1])];

end

%% DMD - Dynamic Mode Decomposition ------------------------------------
X1 = data(:, 1:end-1);
X2 = data(:, 2:end);

[U, S, V] = svd(X1, 'econ');
sig = diag(S);

% plot the energy and singular value
figure(1)
subplot(2, 1, 1)
plot(sig, 'b.', 'Linewidth', 0.5);
title("Singular Values");
ylabel("\sigma");

subplot(2, 1, 2)
plot(sig.^2 / sum(sig.^2), 'b.', 'Linewidth', 0.5)
title("Energy of Singular Values");
ylabel('Energy')

%% low-rank approximation ----------------------------------------------
rank = 1;

U_2 = U(:, 1:rank);
```

```
S_2 = S(1:rank, 1:rank);
V_2 = V(:, 1:rank);

% low-rank dynamics
Sigma = U_2' * X2 * V_2 * diag(1./diag(S_2));
[eV, D] = eig(Sigma);
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U_2*eV;

%% DMD reconstruction ------------------------------------------------------
y0 = Phi\X1(:,1);

u_modes = zeros(length(y0), length(t)-1);
for iter = 1:length(t)-1
    u_modes(:, iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;


% create sparse part
X_sparse = X1 - abs(u_dmd);
R = X_sparse.*(X_sparse < 0);

X_back = R + abs(u_dmd);
X_fore = X_sparse - R;

X_reconstruct = X_fore + X_back;


%% Comparison ------------------------------------------------------------
v = reshape(X1, [a, b, length(t)-1]);
v1 = reshape(X_sparse, [a, b, length(t)-1]);
v2 = reshape(u_dmd, [a, b, length(t)-1]);
v3 = reshape(X_back, [a, b, length(t)-1]);
v4 = reshape(X_fore, [a, b, length(t)-1]);
v5 = reshape(X_reconstruct, [a, b, length(t)-1]);


figure(2)
% num_frame = 100;
num_frame = 114;
subplot(2,3,1)
imshow(uint8(v(:,:,num_frame)))
title("Original Video");

subplot(2,3,2)
imshow(uint8(v1(:,:,num_frame)))
title("Sparse Reconstruction");

subplot(2,3,3)
imshow(uint8(v2(:,:,num_frame)))
title("Low Rank Reconstruction(No R)");

subplot(2,3,4)
imshow(uint8(v5(:,:,num_frame)))
title("Reconstruction");

subplot(2,3,5)
imshow(uint8(v4(:,:,num_frame)))
title("Sparse Reconstruction(R-)");

subplot(2,3,6)
imshow(uint8(v3(:,:,num_frame)))
title("Low Rank Reconstruction(R+)");
```