AMATH 482 Homework 5

Sandy Yang

March 18, 2020

Abstract

In this assignment, we are focusing on the dynamic mode decomposition (DMD) algorithm and its use in subtracting video streams into individual streams of foreground videos and background videos. Stop motion videos of cars racing and skier moving down the mountain were created to test the algorithm. The method successfully subtracted the background from the test videos.

1 Introduction and Overview

The Dynamic Mode Decomposition (DMD) is a very useful tool when it comes to the analysis of nonlinear systems and the dynamics that make them. The basic behind DMD is to take a "snapshot" of data in time, x_k, in order to predict the future state of the system x_{k+1}. We can express this in the linear form $Ax_k = x_{k+1}$. In this assignment, we will use the basic concepts of the Dynamic Mode Decomposition in order to separate the background from the foreground of several different videos.

2 Theoretical Background

2.0.1 Singular Value Decomposition

Similar to the previous assignment, solving this problem relies heavily on the processes of Singular Value Decomposition/Principal Component Analysis. To recall, the reduced SVD is written mathematically as:

$$A = U\Sigma V^T \tag{1}$$

where U and V^T are unitary matrices that contain information on rotation and Σ is a diagonal matrix describing the stretch in each direction. The columns of U are the left singular vectors of A and the columns of V are called the right singular vectors of A. Each non-zero value in the Σ matrix is a singular value of A and the total number of non-zero singular values is the rank of A.

2.0.2 Dynamic Mode Decomposition (DMD)

We start off by taking a snapshot of the given video, from $t = 1, 2, 3...m$, then another snapshot of the data shifted by one step in time, $t = 2, 3, 4...m + 1$. Then, we can form a Krylov space, which is given by

$$X_1^{m-1} = [x_1 A x_1 A^2 x_1 \dots A^{m-2} x_1]$$

here, A is known as a Koopman operator (which estimates the possible nonlinear dynamics of a system), and the relationship between the two snapshots taken is given by the following equation

$$X_2^m = X_1^{m-1} S + r e_{m-1}^*$$

where k_j's are the coefficients of the Krylov space basis vectors and S is the following $n \times (m-1)$ matrix.

$$S = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 & k_1 \\ 1 & 0 & \dots & 0 & 0 & k_2 \\ 0 & 1 & \dots & 0 & 0 & k_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & k_{m-3} \\ 0 & 0 & \dots & 1 & 0 & k_{m-2} \\ 0 & 0 & \dots & 0 & 1 & k_{m-1} \end{bmatrix}$$

We will use these later in finding the Fourier modes and frequencies necessary to reconstruct our video. With the concept of the DMD in mind, we can use this in order to distinguish between a video's foreground and background. In the DMD, a video can be represented by two video sequences:

$$X_{DMD} = \beta_p \phi_p e^{\omega_p t} + \sum_{j \neq p} \beta_j \phi_j e^{\omega_j t}$$

Where p belongs to $[1, 2, \dots, l]$ (l is the number of low rank modes), ϕ is the eigenvector of the Koopman operator A, and ω represents the Fourier modes. The foreground of the video is represented by the left term:

$$X_{DMD}^{Low-rank} = \beta_p \phi_p e^{\omega_p t}$$

On the other hand, the background of the video is represented by:

$$X_{DMD}^{Sparse} = \sum_{j \neq p} \beta_j \phi_j e^{\omega_j t}$$

If we wish to find DMD, we can use the following equation to do so:

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

Where $|\dot{|}$ represents the modulus of each element within the matrix. The reason why the absolute value of low-rank representation of X is because that $\beta_p \phi_p e^{\omega_p t}$ yields a matrix with complex values. We only want real-valued outputs for the analysis of the results of our system, and getting rid of complex values can make a big difference in accuracy.
But this may result in negative values in some elements for X_{DMD}^{Sparse}, and negative pixel intensities do not make sense. So these residual negative values can be put back into X_{DMD}^{Sparse} as follows:

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

Now the magnitudes of the complex values for the DMD reconstruction are accounted for while maintaining the important constraints that:

$$X = X_{DMD}^{Low-rank} + X_{DMD}^{Sparse}$$

so that there are no pixel intensities below zero, and gives us the approximate low-rank and sparse DMD construction of the video.

3 Algorithm Implementation and Development

This part includes the algorithm implementation and development for this assignment.

- Import video clips. Get height, weight, and number of frames. Reshape the video in to $[height * weight \times framesnumber]$ matrix so we can further process it.
- Convert video to gray scale.

- Perform SVD on X. Using the low rank approximation to get rank r. For this assignment, I tried the first rank and the rank of the 90 percent energy (r = 99) for the low rank approximation. Then, we take the first r columns of our U and V matrices and the first r singular value of our matrix.
- Doing a eigen-decomposition on a a matrix similar to S and compute the Fourier modes. We call this matrix `Stilde`.
- Compute the eigen decomposition of `Stilde` and compute the background video
- Subtract the background video from the original video to obtain the foreground video of moving objects
- Looping through the background and foreground video to check the result.

4 Computational Results

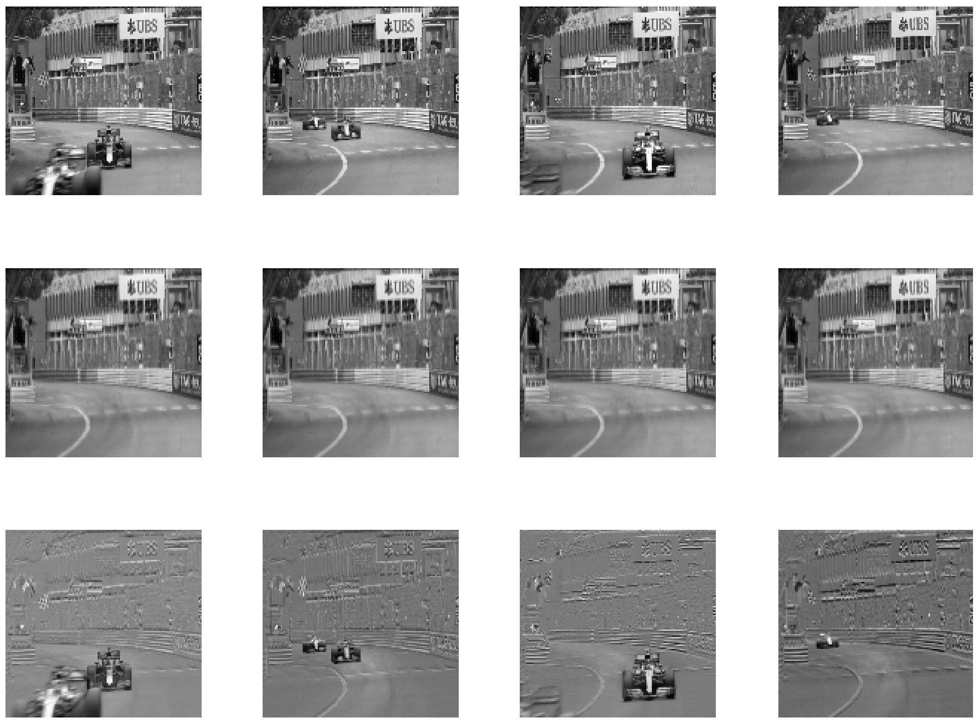

Figure 1: Result of DMD with monte-carlo-low.mp4

For the `monte-carlo-low.mp4`, the separation works pretty well. The cars in the foreground is correctly captured moving in the foreground video. One thing to note is that the result using with rank = 1 and rank = 99 looks pretty similar.

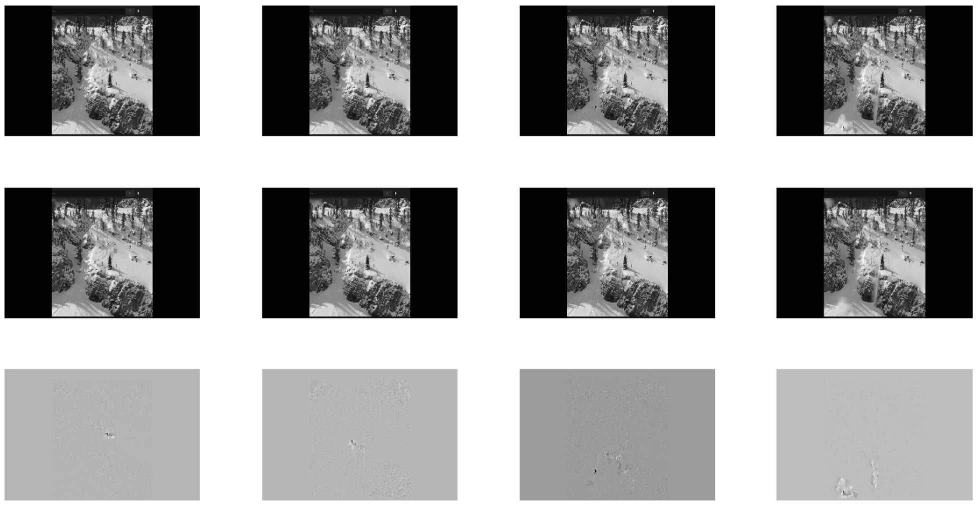

Figure 2: Result of DMD with ski-drop-low.mp4

For `the ski-drop-low.mp4`, the separation doesn't look that obvious since the moving target is very small in the original video. Also, it doesn't perfectly black out the snow in the original video. However, if we make a graph bigger, we can see the skier is moving. DMD in this case still works pretty well.

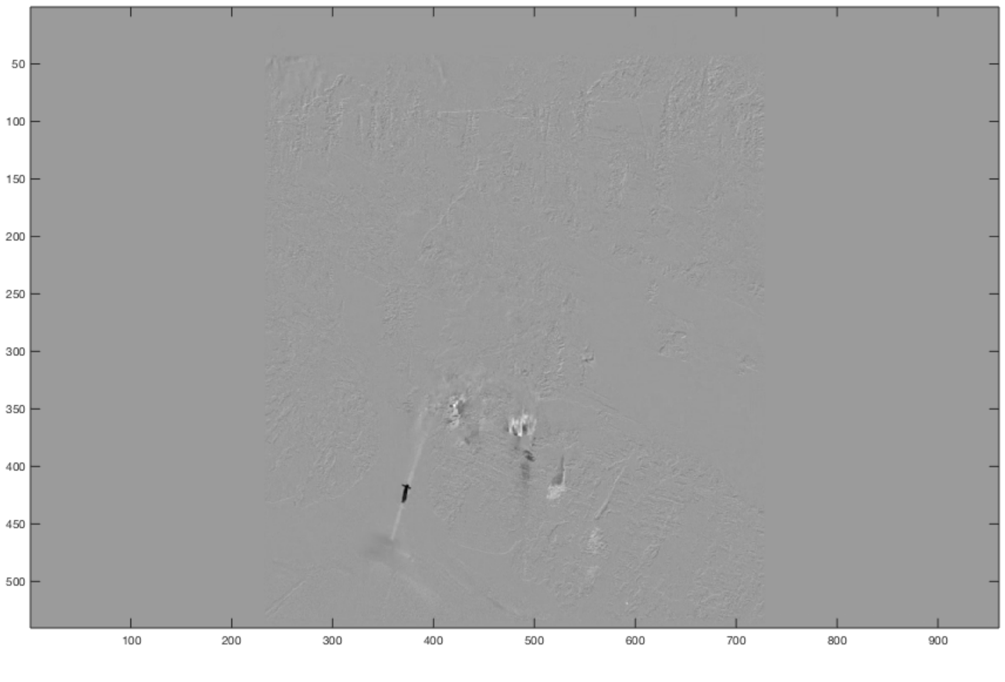

Figure 3: ski-drop-low.mp4, forground image at frame 300

5 Summary and Conclusions

In this assignment, we see that the videos can be separated into a background and a foreground by using DMD, which applies principal component analysis and Singular value decomposition in order to have a low-rank approximation of the original stream of data from which to distinguish from. We also see that the DMD works on all the cases!

Appendix A MATLAB Functions

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

- `reshape(A,sz)`: Reshapes the array A to the size "sz". We used this to resize our image matrices into a vector.
- `size(X)`: Returns the dimensions of the matrix "X". We used this to calculate size for audio.
- `[u,s,v] = svd(X)`: Returns the U, S, and V matrices corresponded with the singular value decomposition of "X". We used this for build the PCA space for our training data.

Appendix B MATLAB Code

```
clear all;
close all;
clc;

%% Video 1: ski drop
video = [];
v = VideoReader('ski_drop_low.mp4');
while hasFrame(v)
    frame = readFrame(v);
    frame = rgb2gray(frame);
    frame = reshape (frame, [], 1);
    video = [video, frame];
end
%%
n = v.numFrames;
h = v.height;
w = v.width;
video = reshape(video, [h*w,n]);
video = double(video);

v1 = video(:,1:end-1);
v2 = video(:,2:end);
[U, Sigma, V] = svd(v1, 'econ');

%% low rank approximation
energy = 0;
total = sum(diag(Sigma));
% how much energy we want our modes to capture.
% 75% does alright; 90% does very good.
threshold = 0.9;
r = 0;
while energy < threshold
    r = r + 1;
    energy = energy + Sigma(r,r)/total;
end

%% DMD

%low rank approximation
r=1;
Sr = Sigma(1:r, 1:r);
Ur = U(:, 1:r);
Vr = V(:, 1:r);
Stilde = Ur'*v2*Vr*diag(1./diag(Sr));
[eV, D] = eig(Stilde);
mu = diag(D);
omega = log(mu);
Phi = v2*Vr/Sr*eV;

y0 = Phi\video(:,1);
```

```matlab
v_modes = zeros(r,length(v1(1,:)));
for i = 1:length(v1(1,:))
    v_modes(:,i) = (y0.*exp(omega*i));
end
v_dmd = Phi*v_modes;
v_dmd = abs(v_dmd);
v_sparse = v1 - v_dmd;

%%
%residual_matrix = v_sparse.*(v_sparse < 0);
%v_dmd = residual_matrix + abs(v_dmd);
%v_sparse = v_sparse - residual_matrix;
%uvid = reshape(v_dmd, [540, 960, 453]);
%%
figure(1)

for i = 1:12
    subplot(3,4,i)
    vidtype = floor((i-1)/4);
    timeframe = mod(i,4);
    if timeframe == 0
        timeframe = 4;
    end
    if vidtype == 0
        temp = video(:,timeframe*100);
    elseif vidtype == 1
        temp = v_dmd(:,timeframe*100); %background
    elseif vidtype == 2
        temp = v_sparse(:,timeframe*100); %forground
    end
    temp = reshape(temp, h, w);
    imagesc(temp);
    colormap(gray);
    axis off;
end

%% play vids

for i = 1:n-1
    subplot(3,1,1)
    og = reshape(video(:,i), h, w);
    imagesc(og);

    subplot(3,1,2)
    bg = reshape(v_dmd(:,i), h, w);
    imagesc(bg);

    subplot(3,1,3)
    fg = reshape(v_sparse(:,i), h, w); %forground
    imagesc(fg);

    colormap(gray);
    axis off;
    pause(.0000001)
end
```

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

%% Video 1: monte carlo
video = [];
v = VideoReader('monte_carlo_low.mp4');
while hasFrame(v)
    frame = readFrame(v);
    frame = rgb2gray(frame);
    frame = reshape (frame, [], 1);
    video = [video, frame];
```

```
end
%%
n = v.numFrames;
h = v.height;
w = v.width;
video = reshape(video, [h*w,n]);
video = double(video);

v1 = video(:,1:end-1);
v2 = video(:,2:end);
[U, Sigma, V] = svd(v1, 'econ');

%% low rank approximation
energy = 0;
total = sum(diag(Sigma));
% how much energy we want our modes to capture.
% 75% does alright; 90% does very good.
threshold = 0.9;
r = 0;
while energy < threshold
    r = r + 1;
    energy = energy + Sigma(r,r)/total;
end
%% DMD

%low rank approximation
r=1;
Sr = Sigma(1:r, 1:r);
Ur = U(:, 1:r);
Vr = V(:, 1:r);
Stilde = Ur'*v2*Vr*diag(1./diag(Sr));
[eV, D] = eig(Stilde);
mu = diag(D);
omega = log(mu);
Phi = v2*Vr/Sr*eV;
y0 = Phi\video(:,1);
v_modes = zeros(r,length(v1(1,:)));
for i = 1:length(v1(1,:))
    v_modes(:,i) = (y0.*exp(omega*i));
end

%%
v_dmd = Phi*v_modes;
v_dmd = abs(v_dmd);
v_sparse = v1 - v_dmd;

%residual_matrix = v_sparse.*(v_sparse < 0);
%v_dmd = abs(v_dmd) + residual_matrix;
%v_sparse = v_sparse - residual_matrix;
%uvid = reshape(v_dmd, [h, w, n-1]);
%%
figure(1)

for i = 1:12
    subplot(3,4,i)
    vidtype = floor((i-1)/4);
    timeframe = mod(i,4);
    if timeframe == 0
        timeframe = 4;
    end
    if vidtype == 0
        temp = video(:,timeframe*80);
    elseif vidtype == 1
        temp = v_dmd(:,timeframe*80); %background
    elseif vidtype == 2
        temp = v_sparse(:,timeframe*80); %forground
    end
    temp = reshape(temp, h, w);
    imagesc(temp);
    colormap(gray);
    axis off;
end
%% play vids

for i = 1:n-1
    subplot(3,1,1)
    og = reshape(video(:,i), h, w);
    imagesc(og);

    subplot(3,1,2)
    bg = reshape(v_dmd(:,i), h, w);
    imagesc(bg);

    subplot(3,1,3)
    fg = reshape(v_sparse(:,i), h, w); %forground
    imagesc(fg);

    colormap(gray);
    axis off;
    pause(.0000001)
end
```