Amath 482 Homework 5 Report
Background Subtraction in Video Streams

Colin Kong

March 17th, 2021

Abstract: This assignment started with using the Dynamic Mode Decomposition method on the video clips ski_drop.mov and monte_carlo.mov containing a foreground and background object and separate the video stream to both the foreground video and a background.

Section I: Introduction and Overview

Introduction: The experiments are an attempt to illustrate the theory of Dynamic Mode Decomposition (DMD). To avoid the limited memory issue for the computer, I used the ski_drop_low.mp4 and monte_carlo_low.mp4 as the original videos and create the foreground and background based on them.

Overview:
I started with loading the videos by the function Video Reader, and then cut them into a perfect size so that it is easier for us to get the frames. Then I reshape the original video into a gray scale video and apply with DMD method. Later I applied the DMD method into the data from the original video and change the data into foreground and background form. Finally, I printed out the frames and put them in subplots.

Section II: Theoretical Background

DMD (Dynamic Mode Decomposition):

The setup:
We are going to assume that we have snapshots of spatio-temporal data. That is, data that evolves in both space and time. The 'space' part can be loosely defined since we really just need a collection of vectors that all evolve in time. We will define:

N= number of spatial points saved per unit time snapshot

M= number of snapshots taken.

The most import thing is that the time data is collected at regularly spaced intervals:

$$t_{m+1} = t_m + \Delta t, \quad m = 1, \dots, M-1, \quad \Delta t > 0$$

The snapshots are denoted:

$$U(x, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \dots \\ U(x_n, t_m) \end{bmatrix}$$

for each m = 1, …, M. We can use these snapshots to form columns of data matrices

$$X = [U(x, t_1)\ U(x, t_2) \dots U(x, t_M)]$$

and

$$X_j^k = \left[U(x,t_j)\; U(x,t_{j+1}) \dots\; U(x,t_k)\right]$$

The matrix X_j^k is just columns j through k of the full snapshot matrix X.

The Koopman Operator:
The DMD method approximates the modes of the Koopman operator. Here's the general idea. The Koopman operator A is a linear, time-independent operator such that

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

where the j indicates the specific data collection time and A is the linear operator that maps the data from time t_j to t_{j+1}. The vector x_j is an N-dimensional vector of the data points collect at time j. That is, applying A to a snapshot of data will advance it forward in time by Δt. We are asking for a linear mapping from one timestep to the next, even though the dynamics of the system are likely nonlinear.
To construct the appropriate Koopman operator that best represents the data collected, we will consider the matrix

$$x_1^{M-1} = [x_1\;\; x_2\;\; x_2\;\; \cdots x_{M-1}],$$

where we use the shorthand x_j to denote a snapshot of the data at time t_j The Koopman operator allows us to rewrite this as

$$x_1^{M-1} = [x_1\;\; Ax_1\;\; A^2x_1\;\; \cdots A^{M-2}x_1].$$

The columns are formed by applying powers of A to the vector x_1, and are said to form the basis for the Krylov subspace. Hence, we have (in theory at least) a way of relating the first M−1 snapshots to x_1 using just the Koopman operator/matrix. We can write the above in matrix form to get

$$X_2^M = Ax_1^{M-1} + re_{M-1}^T,$$

where e_{M-1} is the vector with all zeros except a 1 at the (M−1) st component. That is, A applied to each column of x_1^{M-1} , given by x_j, maps to the corresponding column of X_2^M, given by x_{j+1}, but the final point x_M wasn't included in our Krylov basis, so we add in the residual(or error) vector r to account for this. Remember that A is unknown, and it is our goal to find it. We should also remember that matrices are completely understood by their eigenvalues and eigenvectors, so we are going to circumvent finding A directly by finding other matrices with the same eigenvalues. We will see that the eigenvectors come easily from there. First, let's use the SVD to write x_1^{M-1}=UΣV∗. Then, from above we get

$$X_2^M = AU\Sigma V^* + re_{M-1}^T$$

We are going to choose A in such a way that the columns in X_2^M can be written as linear combinations of the columns of U. This is the same as requiring that they can be written as linear combinations of the POD modes. Hence, the residual vector r must be orthogonal to the POD basis, giving that U^{*}r = 0. Multiplying the above equation through by U^{*} on the left gives:

$$U^{*}X_2^M = U^{*}AU\Sigma V^{*}.$$

Then, we can isolate for $U^{*}AU$ by multiplying by V and Σ^{-1} then on the right to get

$$U^{*}AU = U^{*}X_2^M \mathrm{V}\,\Sigma^{-1}$$

SVD (Singular Value Decomposition):

The reduced SVD is not the standard definition of the SVD used in the literature. What is typically done is to construct a matrix U from $\widehat{U}$ by adding an addition m−n column that are orthonormal to the already existing set $\widehat{U}$. In this way, the matrixUbecomes a squarem×mmatrix, and in order to make the decomposition work, and additional m−n row of zeros is also added to the matrix $\widehat{\sum}$, resulting in the matrix Σ. This leads to the singular value decomposition of the matrix A: A = UΣV^{*},where U ∈ $\mathbb{R}^{m*n}$ and V ∈ $\mathbb{R}^{m*n}$ are unitary matrices, and Σ ∈ $\mathbb{R}^{m*n}$ is diagonal. Geometrically, the SVD is doing the following. Multiplying by V^{*} on the left rotates the hypersphere to align the v_n vectors with the axes. Then we multiply on the left by a diagonal matrix which stretches in each direction. Then we multiply on the left by U to rotate the hyper ellipse to the proper orientation.

To compute SVD, we have:

$$\begin{aligned} A{*}A &= (U\Sigma V^{*}){*}(U\Sigma V^{*}) \\ &= V\Sigma U{*}U\Sigma V^{*} \\ &= V\Sigma^2 V^{*}. \end{aligned}$$

Multiplying on the right by V gives

$$A{*}AV = V\Sigma^2,$$

so that the columns of V are eigenvectors of A∗A with eigenvalues given by the square of the singular values. Similarly,

$$AA^{*} = U\Sigma^2 U^{*}.$$

and so, multiplying on the right by U gives the eigenvalue problem

$$AA^{*} * U = U\Sigma^2$$

which can be used to solve for U.

Section III: Algorithm implementation and Development

The basic algorithm using in this project is DMD reconstruction:

Assuming that $X \in R^{n\times m}$, then a proper DMD reconstruction should also produce $X_{DMD} \in RR^{n\times m}$. However, each term of the DMD reconstruction is complex:$b_j\varphi_j exp\ (\omega jt) \in C^{n\times m}\forall j$, though they sum to a real-valued matrix. Consider calculating the DMD's approximate low-rank reconstruction according to

$$X_{DMD}^{Low-Rank} = b_p\varphi_p e^{\omega_p t}.$$

Since it should be true that

$$\mathrm{X} = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse},$$

then the DMD's approximate sparse reconstruction,

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

can be calculated with real-valued elements only as follows. . .

$$X_{DMD}^{Sparse} = X - \left| X_{DMD}^{Low-Rank} \right|,$$

where $|\cdot|$ yields the modulus of each element within the matrix. However, this may result in X_{DMD}^{Sparse} having negative values in some of its elements, which would not make sense in terms of having negative pixel intensities. These residual negative values can be put into a n × m matrix R and then be added back into $X_{DMD}^{Low-Rank}$ as follows:

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

$$X_{DMD}^{Sparse} \leftarrow X_{DMD}^{Sparse} - R$$

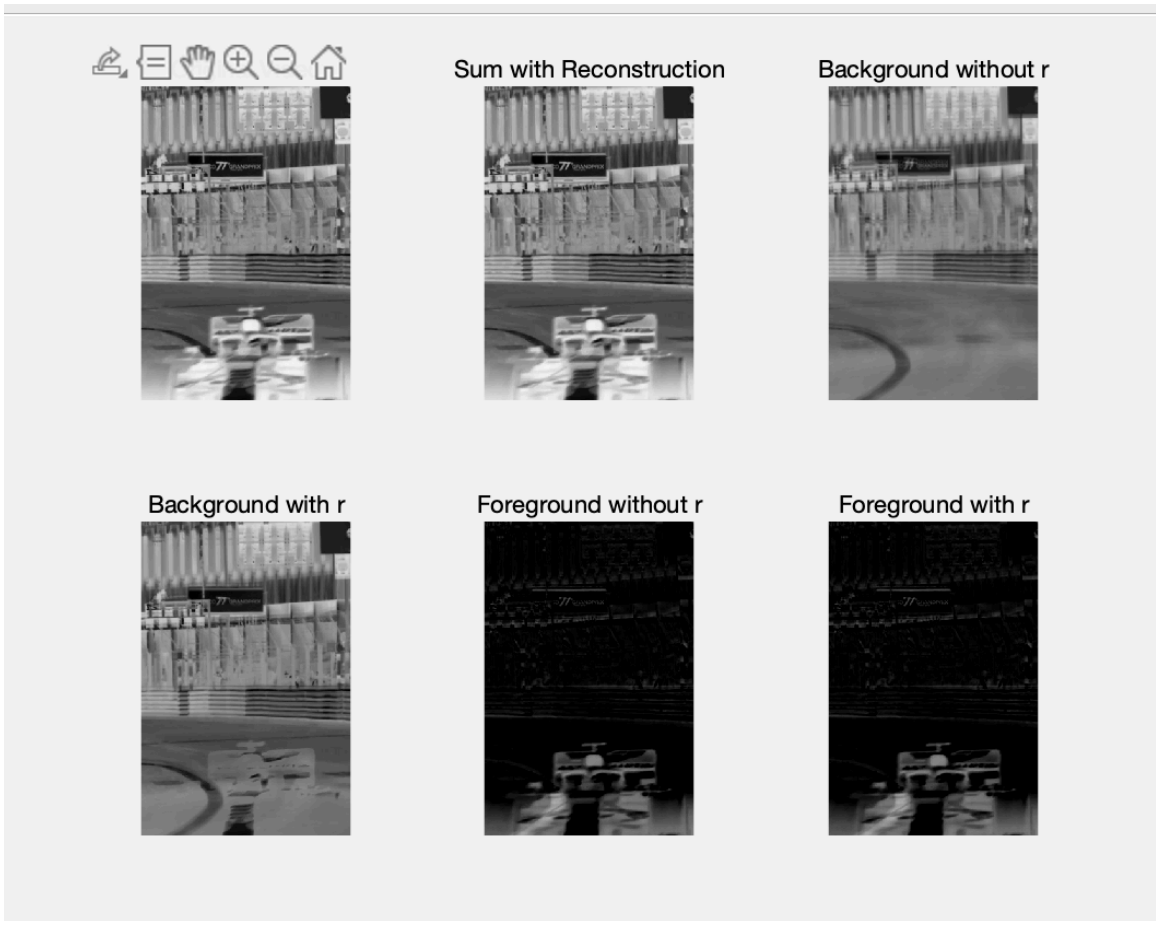
This way the magnitudes of the complex values from the DMD reconstruction are accounted for, while maintaining the important constraints that

$$\mathrm{X} = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse},$$

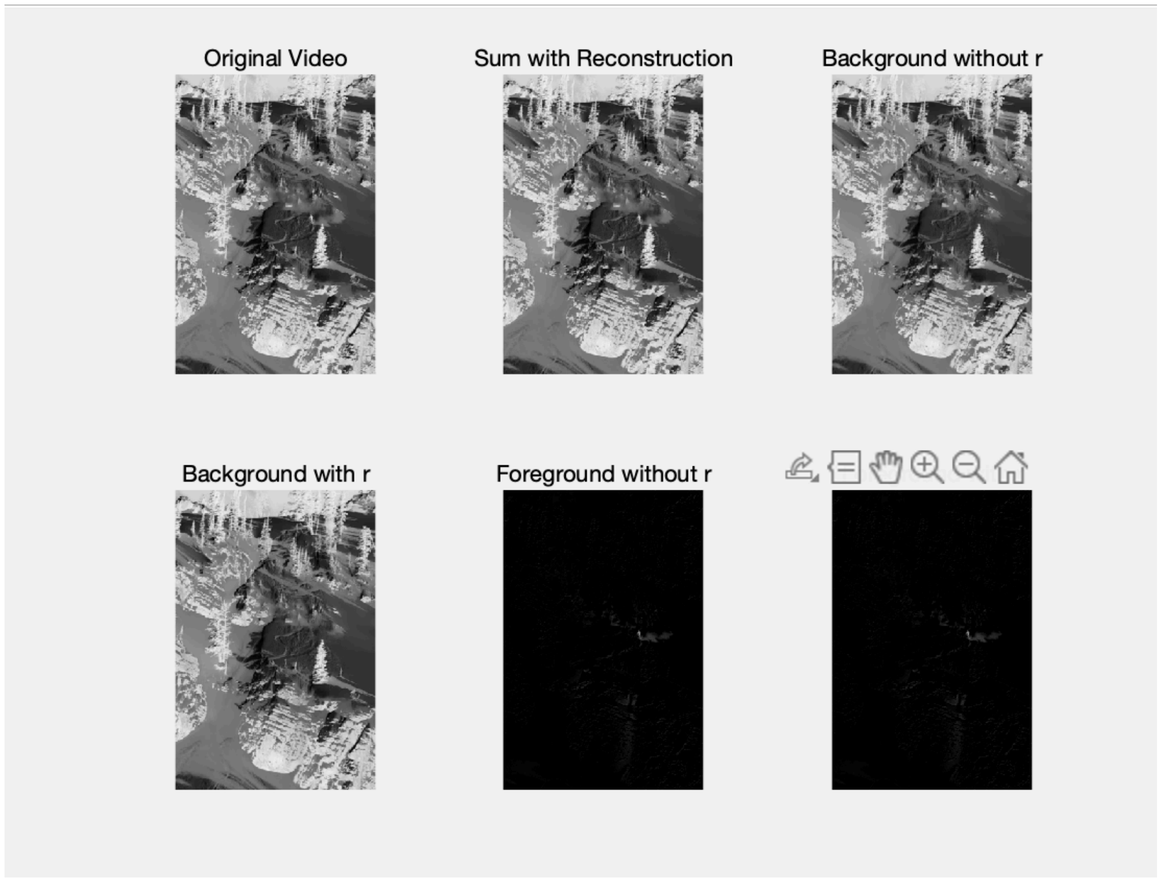
so that none of the pixel intensities are below zero and ensuring that the approximate low-rank and sparse DMD reconstructions are real-valued.

Section IV: Computational Results

1. **Frames of "monte_carlo_low.mp4" with 2 modes:**



2. **Frames of “ski_low.mp4” with 2 modes:**



Section V: Summary and Conclusion

In this project, for each video, I came out with six frames representing original, reconstruction, Background with/without r and Foreground with/without r. This is based on the perfect use of the DMD sparse and low-rank reconstructions. We could find out that the frame of foreground video is entirely dark, and the frame of Background video is extremely vague.

Appendix A: MATLAB Functions Used

1. **svd(): $[U, S, V]$ = svd(A)** -- where $[U, S, V]$ corresponds to $U, \Sigma\ and\ V$ respectively. This command in Matlab calculates the SVD.
2. **rgb2gray()** -- converts the true color image RGB to the grayscale image.
3. **[V,D] = eig(A)** – returns diagonal matrix D of eigenvalues and matrix W whose columns are the corresponding right eigenvectors, so that A*V = V*D.
4. **reshape (A, [M,N])** – reshapes A into a given matrix.
5. **Y = uint8** – converts the values in X to type uint8
6. **v = VideoReader(filename)** -- creates object v to read video data from the file named filename.

Appendix B: MATLAB Codes

For video 1:

```
clear all; close all; clc

v1 = VideoReader("monte_carlo_low.mp4");
vid1_dt = 1/v1.Framerate;
vid1_t = 0:1:v1.Duration;
vid1Frames = read(v1);
[height1, width1, RGB1, numFrames1] = size(vid1Frames);

v2 = VideoReader("ski_drop_low.mp4");
vid2dt = 1/v2.Framerate;
vid2_t = 0:1:v2.Duration;
vid2Frames = read(v2);
[height2, width2, RGB2, numFrames2] = size(vid2Frames);

for i = 1:numFrames1
    X = vid1Frames(:,:,:,i);
%    imshow(X);drawnow;
end

numRows = 500-49;
numCols = 600-299;
gray_vid1 = zeros(numRows, numCols, numFrames1);

for j=1:numFrames1
    gimage = rgb2gray(vid1Frames(50:500,300:600,:,j));
    gray_vid1(:,:,j) = abs(255-gimage);
    %imshow(abs(255-gimage));drawnow
end

X = reshape(gray_vid1, numRows*numCols, numFrames1);
height = numRows;
width = numCols;

X1 = X(:,1:end-1);
X2 = X(:,2:end);

[U,S,V] = svd(X1,'econ');
r = 2;
U_r = U(:, 1:r);
```

```matlab
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
A_tilde = U_r' * X2 * V_r / S_r;
[W_r,D] = eig(A_tilde);
Phi = X2 * V_r / S_r * W_r;

lambda = diag(D);
omega = log(lambda)/vid1_dt;

%thresh = 0.001;
%bg = find(abs(omega) < thresh);
%omega_bg = omega(bg);
%plot(real(omega_bg),imag(omega_bg),'rx');

x1 =X1(:, 1);
b = Phi \ x1;

mm1 = size(X1,2);
time_dynamics = zeros(r,mm1);
t = (0:mm1-1)*vid1_dt;
for iter = 1:mm1
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end
X_bg = Phi * time_dynamics;

for k = 1:numFrames1 - 1
    frame = reshape(X_bg(:,k),height,width);
    frame = uint8(real(frame));
    figure(2);
    %imshow(frame); drawnow
end

X_fg = X1 - abs(X_bg);
ind = find(X_fg < 0);
X_rec = X_fg+X_bg;
X_bgr = X_bg;
X_bgr(ind) = X_bg(ind) + X_fg(ind);
X_fgr = X_fg;
X_fgr(ind) = 0;

for i = 1:numFrames1 - 1
```

```
    frame = reshape(X_fg(:,i),height,width);
    frame = uint8(real(frame));
    figure(3);
    %imshow(frame); drawnow
end

figure(4)
subplot(2,3,1)
frame = reshape(X1(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Original Video");
subplot(2,3,2)
frame = reshape(X_rec(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Sum with Reconstruction");
subplot(2,3,3)
frame = reshape(X_bg(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Background without r");
subplot(2,3,4)
frame = reshape(X_bgr(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Background with r");
subplot(2,3,5)
frame = reshape(X_fg(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Foreground without r");
subplot(2,3,6)
frame = reshape(X_fgr(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Foreground with r");
```

For video 2:

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

```
v2 = VideoReader("ski_drop_low.mp4");
vid2_dt = 1/v2.Framerate;
vid2_t = 0:1:v2.Duration;
vid2Frames = read(v2);
[height2, width2, RGB2, numFrames2] = size(vid2Frames);

for i = 1:numFrames2
    X = vid2Frames(:,:,:,i);
%    imshow(X);drawnow;
end

numRows = 500-49;
numCols = 600-299;
gray_vid2 = zeros(numRows, numCols, numFrames2);

for j=1:numFrames2
    gimage = rgb2gray(vid2Frames(50:500,300:600,:,j));
    gray_vid2(:,:,j) = abs(255-gimage);
    %imshow(abs(255-gimage));drawnow
end

X = reshape(gray_vid2, numRows*numCols, numFrames2);
height = numRows;
width = numCols;

X1 = X(:,1:end-1);
X2 = X(:,2:end);

[U,S,V] = svd(X1,'econ');
r = 2;
U_r = U(:, 1:r);
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
A_tilde = U_r' * X2 * V_r / S_r;
[W_r,D] = eig(A_tilde);
Phi = X2 * V_r / S_r * W_r;

lambda = diag(D);
omega = log(lambda)/vid2_dt;

%thresh = 0.001;
```

```matlab
%bg = find(abs(omega) < thresh);
%omega_bg = omega(bg);
%plot(real(omega_bg),imag(omega_bg),'rx');

x1 =X1(:, 1);
b = Phi \ x1;

mm1 = size(X1,2);
time_dynamics = zeros(r,mm1);
t = (0:mm1-1)*vid2_dt;
for iter = 1:mm1
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end
X_bg = Phi * time_dynamics;

for k = 1:numFrames2 - 1
    frame = reshape(X_bg(:,k),height,width);
    frame = uint8(real(frame));
    figure(2);
    %imshow(frame); drawnow
end

X_fg = X1 - abs(X_bg);
ind = find(X_fg < 0);
X_rec = X_bg+X_fg;
X_bgr = X_bg;
X_bgr(ind) = X_bg(ind) + X_fg(ind);
X_fgr = X_fg;
X_fgr(ind) = 0;

for i = 1:numFrames2 - 1
    frame = reshape(X_fg(:,i),height,width);
    frame = uint8(real(frame));
    figure(3);
    %imshow(frame); drawnow
end

figure(4)
subplot(2,3,1)
frame = reshape(X1(:,100),height,width);
frame = uint8(real(frame));
```

```
imshow(reshape(frame,[height,width]));
title("Original Video");
subplot(2,3,2)
frame = reshape(X_rec(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Sum with Reconstruction");
subplot(2,3,3)
frame = reshape(X_bg(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Background without r");
subplot(2,3,4)
frame = reshape(X_bgr(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Background with r");
subplot(2,3,5)
frame = reshape(X_fg(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Foreground without r");
subplot(2,3,6)
frame = reshape(X_fgr(:,100),height,width);
frame = uint8(real(frame));
imshow(reshape(frame,[height,width]));
title("Foreground with r");
```