AMATH 482 Homework 5

Fiona Fu
March 2020

Abstract

Use the Dynamic Mode Decomposition (DMD) method on two video clips and separate the video stream to both a foreground video and a background.

1 Introduction and Overview

We mainly use the Dynamic Mode Decomposition (DMD) spectrum of frequencies to subtract background modes, and algorithms were about:

1.1 Ski-Drop Clip

1.1.1 SVD (Singular Value Decomposition) analysis

After creating the DMD matrices, we can use svd to find the **U, Σ, V** matrices and reconstruct them by identifying the appropriate rank on the singular value spectrum.

1.1.2 DMD (Dynamic Mode Decomposition) analysis

Creating a low-rank (based on the analysis of SVD) approximation of the linear mapping that best iterates through the snapshots of the data.

1.2 Monte-Carlo Clip

1.2.1 Same method as above

2 Theoretical Background

In this assignment, the main algorithm we use is the **Dynamic Mode Decomposition (DMD),** a data-driven method, to take advantage of low-dimensionality in experimental data. We start with the snapshot as our sample data that evolves in both space at time, and what we need is a collection of vectors with dimension $\boldsymbol{N} \times \boldsymbol{M}$ where N is number of spatial points and M is number of snapshots taken (time points). We made the data snapshots to be evenly spaced in time by Δt, and the snapshots was used to create the columns of data matrix denoted as:

$$\boldsymbol{X_j^k} = \left[\boldsymbol{U}\left(\boldsymbol{x}, \boldsymbol{t_j}\right) \dots \boldsymbol{U}(\boldsymbol{x}, \boldsymbol{t_k})\right] \tag{1}$$

The DMD method approximates the modes of the Koopman operator A, a linear, time-independent operator such that:

$$\boldsymbol{x_{j+1}} = \boldsymbol{A}\boldsymbol{x_j} \tag{2}$$

Then we can construct the operator as:

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

(3)

For which we can relate the first M-1 snapshots to x1 by the Koopman operator and rewrite as:

$$\boldsymbol{X_2^M = AX_1^{M-1} + re_{M-1}^T} \tag{4}$$

Then, let's use **Singular Value Decomposition (SVD)** to plug in $\boldsymbol{X_1^{M-1}} = \mathbf{U\Sigma V^*}$
That A applied to each column of $\boldsymbol{X_1^{M-1}}$, given by $\boldsymbol{x_j}$, maps to the corresponding column of $\boldsymbol{X_2^M}$, given by $\boldsymbol{x_{j+1}}$, but the final point wasn't included in Krylow basis, so we add in the residual vector $\boldsymbol{r}$ to account for this, which is to form:

$$\boldsymbol{X_2^M} = \mathbf{AU\Sigma V} * + \boldsymbol{re_{M-1}^T} \tag{5}$$

where $\boldsymbol{re_{M-1}^T}$ is orthogonal to columns of **U** (**U*r = 0**), we can finally get the tilde over the S to be:

$$\boldsymbol{\sim S = U^* AU = U^* X_2^M V \Sigma^{-1}} \tag{6}$$

Let's write the eigenvector/eigenvalue of ~S by assume:

$$\boldsymbol{\tilde{s}_{y_k} = \mu_k y_k} \tag{7}$$

where we can find the eigenvectors of A, also called the DMD modes by:

$$\boldsymbol{\psi_k = U y_k} \tag{8}$$

Finally, we can expand in our eigenbasis with the continuous equation:

$$\boldsymbol{x_{DMD}(t) = \sum_{k=1}^{k} b_k \psi_k e^{\omega_k t} = \Psi\, diag(e^{\omega_k t}) b} \tag{9}$$

The way how we compute the b_k is to initialize snapshot $\boldsymbol{x_1}$ and the pseudoinverse of Ψ, which is:

$$\boldsymbol{b = \Psi^{\dagger} x_1} \tag{10}$$

Considering separating the DMD into approximate low-rank and sparse reconstructions that matrix X with dimension n*m equals to the sum of low-rank DMD and sparse DMD:

$$\boldsymbol{X_{DMD}^{Low-rank} = b_P \varphi_P e^{\omega_p t}} \tag{11}$$

When the sparse DMD may have negative values, we need to put the residual negative values into a $n \times m$ matrix R and add back into low-rank DMD as:

$$\boldsymbol{X_{DMD}^{Low-Rank} \Leftarrow R + \left|X_{DMD}^{Low-Rank}\right|} \tag{12}$$

$$\boldsymbol{X_{DMD}^{Sparse} \Leftarrow X_{DMD}^{Sparse} - R} \tag{13}$$

3 Algorithm Implementation and Development

At first, we need to read the file and get to know its size and number of frames, which we stored as ‘slices’.

```
v = VideoReader('monte_carlo_low.mp4');
slices = v.NumberOfFrames;
```

Then, we run over each frame and turn the video to gray that would be easy for us to analyze later, and we reshape it to columns to use later.

```
while hasFrame(v)
   img = rgb2gray(readFrame(v));
   img = reshape(img, [540*960], 1);
   Snapshot = [Snapshot img];
end
Snapshot = double(Snapshot);
t2 = linspace(0,v.CurrentTime,slices+1);
t = t2(1:slices);
dt = t(2) - t(1);
```

After initializing the time, we need to create the DMD matrices by two X as stated in equation (3) and (4). We denote $\boldsymbol{X_1^{M-1}}$ as X1 and $\boldsymbol{X_2^{M}}$ as X2.

```
% Create DMD Matrices
X1 = Snapshot(:,1:end-1);
X2 = Snapshot(:,2:end);
```

Followed by the step, we need to take SVD of X1 and plot the singular value spectrum to find the appropriate rank.

```
% SVD of X1
[U, Sigma, V] = svd(X1, 'econ');
figure(1)
plot(diag(Sigma)/sum(diag(Sigma)),'m*')
```

Reconstruct U, S, V by the rank and then compute ~S by the formula in equation (6).
rank = 2;

```
U = U(:, 1:rank);
Sigma = Sigma(1:rank, 1:rank);
V = V(:, 1:rank);
% Computation of ~S
S = U'*X2*V*diag(1./diag(Sigma));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract DMD eigenvalues
omega = log(mu)/dt;
Phi = U*eV;
```

Then we create DMD solution starting with finding coefficient: using pseudoinverse and the initial condition:

```
% Create DMD Solution
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
modes = zeros(length(y0),length(t));
```

```
for iter = 1:length(t)
  modes(:,iter) = y0.*exp(omega*t(iter));
end
X_LR_dmd = Phi*modes;
```

Because we are using a combination of low-rank DMD and sparse DMD, we follow the formula as described in the prompt to find out R and then X-sparse:

```
% X_SP_dmd = Snapshot - abs(X_LR_dmd);
X_SP = Snapshot - X_LR_dmd;
R = X_SP .* (X_SP<0);
X_SP_dmd = X_SP - R;
```

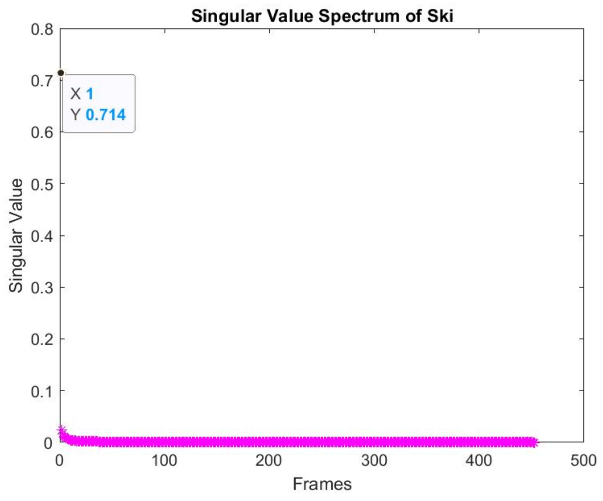
Finally, we can plot each frames, or snapshots, and compare the original, background, and foreground video.

```
for img = 1:slices
   subplot(1, 3, 1)
   original = uint8(Snapshot(:,img));
   imshow(reshape(original, 540, 960))
   title('Original Video')
…
end
```

4 Computational Results

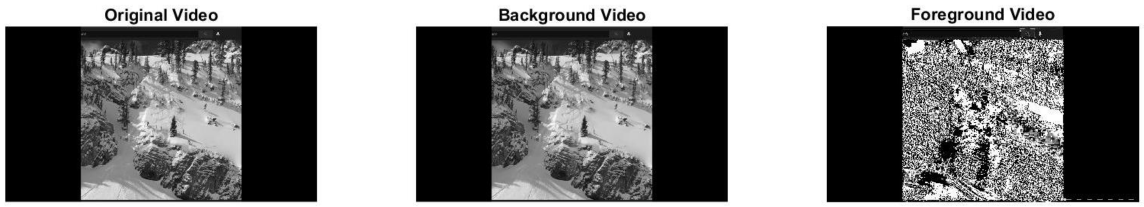
4.1: Ski-Drop Clip

After the data snapshots are evenly spaced in a fixed time t with dimension m*n, we construct the two submatrices and compute the SVD on $\boldsymbol{X_1^{M-1}}$. Then we are able to plot the singular values spectrum (Figure 1) for this data set, and we can approximate rank r = 1. In other words, we are able to get a relatively good image for the images by using just r = 1 columns.



(Figure 1)

Then, we are using DMD algorithm to this video clip of skiing to separate the foreground and background with results shown in Figure 2.

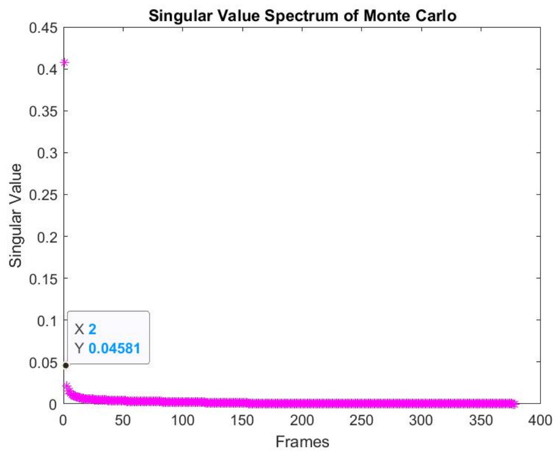


(Figure 2)

In this case, the background video extracted was pretty clear when the overall image didn’t have so much motions within or any shaking of camera. The background video looks like the same with the original video except there’s a small, tiny skier changing position each frame in the original one. The foreground video looks so messy, and we can barely see the people moving around. The motion is so tiny compared to the overall, big picture with still trees, so it’s acceptable to have an image like that.

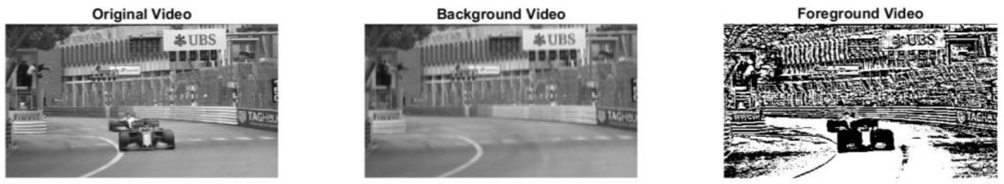
4.2: Monte-Carlo Clip

Having the same idea as the last clip, we set up all of the time, space, and DMD matrices before taking the SVD of X1 that contains the image from the first to end – 1. By using the SVD function, we can plot the singular value spectrum for this clip (Figure 3) and conclude that we should take a rank r = 2.



(Figure 3)

We also create a DMD solution for this clip with separated foreground and background video in Figure 4.



(Figure 4)

Compared to this clip with ski drop's clip, this one with car racing has a clearer foreground image with the moving car that is relatively much bigger than the skier in the forest. The background video is really clear with almost no messy 'noises'. But the foreground video has many features that's actually in the background, although the moving car at front is easy to identify.

3 Summary and Conclusions

When the moving object in the foreground is bigger, occupied a relatively large pixels of each frames, it's easier to identify and separate by using the DMD methods, just as the example of the car compared to the skier. Once the camera is not shaking, we had a bigger chance to get a really nice and clean background video extracted by using the DMD.

As a data-driven method, DMD is useful and applicable for any data snapshots without a governing equation. We can apply the algorithm just follow the steps of:

1. Create DMD matrices X1, X2;
2. Compute SVD of X1;
3. Create matrix S and compute its eigenvalues and eigenvectors
4. Find coefficients by using the initial condition and pseudoinverse.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[2] MathWork. https://www.mathworks.com/help/matlab/ref/videoreader.readframe.html

[3] Jason J. Bramburger, Lecture Notes for AMATH482, Winter, 2021

[4] Prof. Bramburger, Slack, assignment-5

Appendix A MATLAB Functions

- B = reshape(A,sz) reshapes A using the size vector, sz, to define size(B).
- I = rgb2gray(RGB) converts the truecolor image RGB to the grayscale image I.
- video = readFrame(v) reads the next available video frame from the file associated with v.

[U,S,V] = svd(A,'econ') produces an economy-size decomposition of m-by-n matrix A:

- m > n — Only the first n columns of U are computed, and S is n-by-n.
- m = n — svd(A,'econ') is equivalent to svd(A).
- m < n — Only the first m columns of V are computed, and S is m-by-m

Appendix B MATLAB Code

(Partial Code for clip of monte_carlo_low, similar code for ski_drop_low)

```
%%
v = VideoReader('monte_carlo_low.mp4');
slices = v.NumberOfFrames;
%%
Snapshot = [];
% The matrix snapshot has a dimension m*n where m is pixel of image and
% n is number of images
while hasFrame(v)
   img = rgb2gray(readFrame(v));
   img = reshape(img, [540*960], 1);
   Snapshot = [Snapshot img];
end
Snapshot = double(Snapshot);
t2 = linspace(0,v.CurrentTime,slices+1);
t = t2(1:slices);
dt = t(2) - t(1);
%%
% Create DMD Matrices
X1 = Snapshot(:,1:end-1);
X2 = Snapshot(:,2:end);
% SVD of X1
[U, Sigma, V] = svd(X1, 'econ');
figure(1)
plot(diag(Sigma)/sum(diag(Sigma)),'m*')
xlabel('Frames')
ylabel('Singular Value')
title('Singular Value Spectrum of Monte Carlo')
%%
% We got the rank is one, then we rebuild the U, S, V
rank = 2;
U = U(:, 1:rank);
Sigma = Sigma(1:rank, 1:rank);
V = V(:, 1:rank);
% Computation of ~S
S = U'*X2*V*diag(1./diag(Sigma));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract DMD eigenvalues
```

```
omega = log(mu)/dt;
Phi = U*eV;
%%
% Create DMD Solution
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
modes = zeros(length(y0),length(t));
for iter = 1:length(t)
  modes(:,iter) = y0.*exp(omega*t(iter));
end
X_LR_dmd = Phi*modes;

%%
% % DMD Solution
% waterfall(x,1:slices,abs(Phi')), colormap([0 0 0])
% xlabel('x')
% ylabel('modes')
% zlabel('|u|')
% title('DMD Modes')
% set(gca,'FontSize',16)
%%
% X_SP_dmd = Snapshot - abs(X_LR_dmd);
X_SP = Snapshot - X_LR_dmd;
% X_SP = X_SP + 0.000002;
R = X_SP .* (X_SP<0);
X_SP_dmd = X_SP - R;

%%
figure(2)
for img = 1:slices
   subplot(1, 3, 1)
   original = uint8(Snapshot(:,img));
   imshow(reshape(original, 540, 960))
   title('Original Video')
   subplot(1, 3, 2)
   back = uint8(X_LR_dmd(:,img));
   imshow(reshape(back, 540, 960))
   title('Background Video')
   subplot(1, 3, 3)
   fore = real(X_SP_dmd(:,img));
   imshow(reshape(fore, 540, 960))
   title('Foreground Video')
   drawnow
end
```

Listing: Excerpt Code from MATLAB