AMATH 482 Homework 5: Background subtraction in Video Streams

Yilin Cao

March 17, 2021

Abstract

We have two video clips that recorded with a foreground object and a background object, our mission is to separate these two objects by implementing the Dynamic Mode Decomposition (DMD) method to the video streams. We decomposed the algorithms of pixels into two complex terms that represent the background and foreground video parts. For some reason, problems raise when separating DMD terms into approximate low-rank and sparse reconstructions. By a specific method, we overcame the problems of separating DMD terms and successfully separated the videos.

I. Introduction and Overview

Think about complex systems such as high-dimensional, nonlinear dynamical systems that shows a high order of complexity in both space and time. Even though these systems are abstruse and complicated, we still having methods to solve them—representing them by low dimensional spatio-temporal coherent structure. The aim of Dynamic Mode Decomposition is to take advantages of low dimensionality in experimental data without having to rely on a given set of governing equations, it allows us to find a basis of spatial modes (not necessarily orthogonal) for which the time dynamics are just exponential functions (with potentially complex exponents). Meanwhile, it is simple and fast to execute and will not tell you how the data will evolve in time-just how they were evolved on the training set. Essentially, the DMD is a combination of spatial-dimensionality reduction techniques and Fourier transforms in time.

The DMD will return a low-rank approximate reconstruction of the original data, which will represent the background part of our video. Subtracting this part from the original data, we will get the sparse reconstruction that represent the foreground object.

II. Theoretical Background

To start up a dynamic mode decomposition, we are going to assume that we have snapshots of spatio-temporal data, which evolves in both space and time. The 'space' part can be loosely defined since we just need a collection of vectors that all evolves in time. Therefore, we define that:

$$N = number\ of\ spatial\ points\ saved\ per\ unit\ time\ snapshot$$

$$M = number\ of\ snapshots\ taken.$$

The snapshots are denoted by:

$$U(x, t_m) = \begin{matrix} U(x_1, t_m) \\ \vdots \\ U(x_n, t_m) \end{matrix}$$

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

The DMD method approximates the modes of the Koopman operator, a linear, infinite-dimensional operator. We define the Koopman operator as **A**, 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 a N-dimensional vector of the data points collect at time j. Which means applying A to a snapshot of data will advance it forward in time by Δt.

Dynamic Mode Decomposition calculation:

Firstly, we have to correctly construct an appropriate Koopman operator that best represents the data collected, consider using the matrix $X_1^{M-1} = [\, x_1, x_2\, , x_2 \dots x_{M-1}]$, which can also be written as $X_1^{M-1} = [\, x_1, Ax_1\, , A^2x_1 \dots A^{M-2}x_1]$ according to the Koopman operator. Then, we have a way of relating the first M-1 snapshots to x_1 using just Koopman operator/matrix.

Then write the previous matrix into the following form: $X_2^M = AX_1^{M-1} + re_{M-1}^T$, where e_{M-1} is the vector with all zeros except a 1 at the (M-1) st component. Namely, **A** applied to each column of X_1^{M-1}, given by x_j, maps to the corresponding column of X_2^M, but the final point x_M wasn't included in our Krylov basis, so we add in the residual vector r to account for this.

Since we can completely understand a matrix by finding their eigenvalues and eigenvectors, we are going to find A indirectly by calculating other matrices with the same eigenvalues. Firstly, we use SVD to write $X_1^{M-1} = U\Sigma V^*$. From above we can 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. Giving that $U * r = 0$, multiplying the above equation through by U^* on the left side gives: $U^*X_2^M = U^*AU\Sigma V^*$. Then, we can isolate for U^*AU by multiplying by V and then Σ^{-1} on the right to get $UAU = U^*X_2^M V\Sigma^{-1}$. Here, everything on the right side is known from input data, we use **S** to represent this part.

From what we know about singular values, K is the rank of data matrix X_1^{M-1}, which basically tells us the number of dimensions that the data varies in. Ideally, we would like K to relatively small compared to N and M. Notice that $\tilde{S}$ and A are related by applying a matrix on one side and its inverse on the other. This means they are similar matrices, sharing the same eigenvalues. Therefore, let's y an eigenvector of $\tilde{S}$, U_y is an eigenvector of A, we have $\tilde{S}y_k = \mu_k y_k$.

Thus, giving the DMD model, eigenvectors of A:

$$\psi_k = Uy_k$$

Now we've got all we need to describe continual multiplications by A:

$$X_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t}) b$$

Where K is the rank of X_1^{M-1}, the b_k are the initial amplitude of each mode, matrix Ψ contains the eigenvectors ψ_k as its columns. To calculate the b_k, we know that at time $t = 0$, we can get x_1 from the above formula because this is our initial condition to generate other x_m. Therefore, taking $t = 0$ gives us:

$$x_1 = \Psi \mathrm{b} \Rightarrow \mathrm{b} = \Psi^{\dagger} x_1$$

Where $\Psi^{\dagger}$ is the pseudoinverse of matrix Ψ

SVD (Singular Value Decomposition):

The SVD performed in this case to understand the low rank structure of the system. i.e: suppose A is a data matrix, the SVD will decompose it as $A = U\Sigma V^*$, which can also be expressed as following form:

$$[A] = [u_1\ u_2\ \dots\ u_n] \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

In the equations above, we can see that Σ is the diagonal matrix of eigenvectors comprising the singular values of A. columns of U are orthonormal basis vector of the space, while V accounts for the projection of each vectors onto the principal components.

III. Algorithm Implementation and Development

- Clean workspace
- Load video clips and get number of frames
- Crop out the video and convert to grayscale
- Apply DMD algorithm described below to separate background and foreground video
- Compute DMD construction of background
- Subtract background from videos to get foreground
- Plot SVD modes
- Plot original, foreground, background at different frames

DMD reconstruction algorithm:

In the assignment spec, we see that we can use the DMD spectrum of frequencies to subtract background modes. Therefore, we create two data frames at initial: one from the beginning to the last negative one frame, the other from the second to the last frame, each of them represented a data snapshot. Using the first data frame for the SVD and corresponding analysis:

$$X_{DMD} = b_p \varphi_p e^{\omega_p t}(Background) + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t}(Foreground)$$

This will produce a X_{DMD} matrix such that $X_{DMD} \epsilon \mathbb{R}^{n \times m}$.

After we get this matrix, we 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

$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$, then the DMD's approximate sparse reconstruction can be solved by $X_{DMD}^{Sparse} = \sum_{j \neq p} b_j \varphi_j e^{\omega_j t}$ via $X_{DMD}^{Sparse} = X - \left| X_{DMD}^{Low-Rank} \right|$. Since we subtract the low-rank approximation from our initial data to obtain the sparse approximation reconstruction, we may obtain negative values in our sparse matrix, which would not make sense in terms of having negative pixel intensities.

To solve this problem, we put these negative values into a residual $n \times m$ matrix R and then create our low-rank reconstruction according to

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

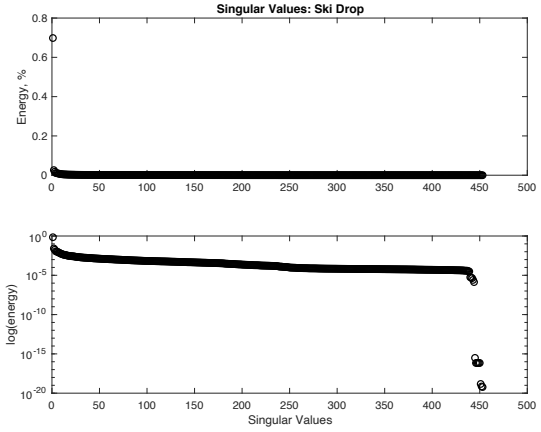
$$X_{DMD}^{Sparse} \leftarrow X_{DMD}^{Sparse} - R$$

This allows us to account for the magnitudes of complex values from the DMD reconstruction and preventing pixel intensities from being negative, while maintaining the important constrains that

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

Thus, also ensure that the approximate low-rank and sparse DMD reconstructions are real-valued.

IV. Computational Results



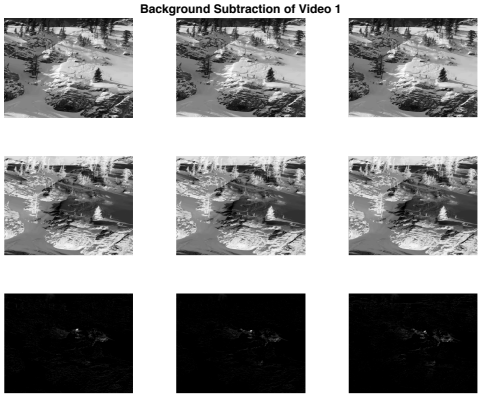


Figure 1. Singular value for "ski_drop_low.mp4"

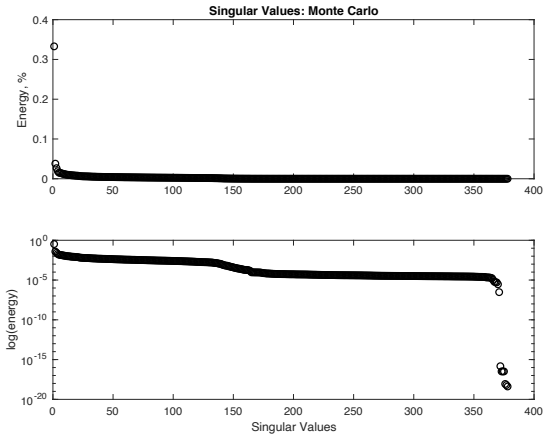


Figure 2. Background Subtraction of "ski_drop_low.mp4"

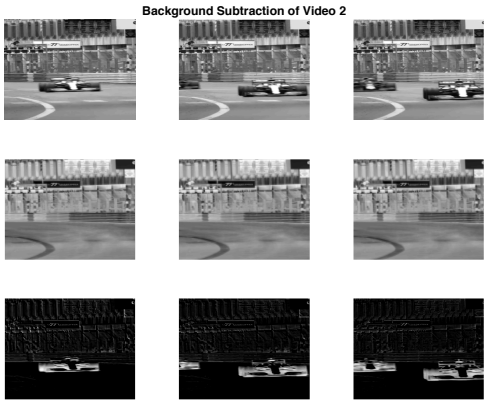


Figure 3. Singular value for "monte_carlo_low.mp4"

Figure 4. Background Subtraction of "monte_carlo_low.mp4"

Summary and Conclusions

In this project, I successively built a DMD algorithm to separate background and foreground video streams from the two video clips. I have created a low-rank and sparse approximation of this data which represented the foreground and background of the object. Notice that, the resting part in the original video clip goes to background and the moving objects become foreground in our reconstruction.

Appendix A. MATLAB functions

abs(X): Returns the absolute value of every element in X. We used this when calculating our DMD's approximate sparse reconstruction.

diag(X): Returns a column vector of the diagonal values of a matrix. We used this to generate our variance numbers from SVD.

double(X): Converts elements of X into type double. We used this to convert uint8 values into type double for analysis.

imshow(X): Displays the matrix X as an image. We used this to compare frames the original video to the DMD's sparse and low-rank reconstruction frames.

reshape(X, sz): Reshapes the array or matrix X to the size sz. We used this to resize our image matrices from video frames into column vectors at each timepoint for data storage. We also used this to convert these column vectors back to a matrix, to view the image.

rgb2gray(): rgb2gray converts RGB values to grayscale values by forming a weighted sum of the R, G, and B components.

size(X): Returns the dimensions of the matrix X. we used this to calculate the size of our matrix in our DMD algorithm.

[u,s,v] = svd(X): returns the U, S, and V matrices corresponded with the singular value decomposition of X. we used this in our initial steps of the DMD algorithm.

uint8(X): converts elements of X into type uint8. We used this to convert our double values to uint8 to view them using imshow.

videoReader(mov.mp4): creates a videoReader object, with information such as frame rate and duration. We used this to load in our video files for analysis and create our dt and t vectors.

Appendix B. MATLAB codes

```
%% Amath482 HW5 Code

% Clean workspace
clear all; close all; clc

%% Set up

% Load Video 1
vid1 = VideoReader("ski_drop_low.mp4"); % video file name can be replaced
vidFrames = read(vid1);
[height, width, RGB, numFrames] = size(vidFrames);

% Watch Movie
for i=1:numFrames
    X = vidFrames(:,:,:,i);
    % imshow(X); drawnow
end

% Crop out edges and convert to grayscale
numRows = 500-49;
numCols = 600-299;
gray_vid = zeros(numRows,numCols,numFrames);

for j=1:numFrames
    gimage = rgb2gray(vidFrames(50:500,300:600,:,j));
    gray_vid(:,:,j) = abs(255-gimage);
    % imshow(abs(255-gimage)); drawnow
end

%% Set up DMD

X = reshape(gray_vid, numRows*numCols, numFrames);

height = numRows;
width = numCols;

X1 = X(:,1:end-1);
X2 = X(:,2:end);
r = 2;
dt = 1/ vid1.Framerate;

% DMD
[U, S, V] = svd(X1,'econ');
r = min(r, size(U,2));
U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
Atilde = U_r' * X2 * V_r / S_r; % low-rank dynamics
[W_r , D] = eig(Atilde);
Phi = X2 * V_r / S_r * W_r; % DMD modes
lambda = diag(D); % discrete -time eigenvalues
omega = log(lambda)/dt; % continuous-time eigenvalues
```

```
% Compute DMD mode amplitudes
x1 =X1(:, 1);
b = Phi\x1;

% DMD reconstruction
mm1 = size(X1, 2); % mm1 = m - 1
time_dynamics = zeros(r, mm1);
t = (0:mm1 - 1)*dt; % time vector
for iter = 1:mm1
    time_dynamics(:, iter) = (b.*exp(omega*t(iter)));
end
Xdmd = Phi * time_dynamics;

%% Separate background and foreground
og = vidFrames(50:500,300:600,:,:);
bg = uint8(Xdmd);
fg = uint8(X(:,1:numFrames-1) - Xdmd);

% reshaped videos
for i=1:numFrames-1
    bg_frame(:,:,:,i) = reshape(bg(:,i),height,width);
end

for j=1:numFrames-1
    fg_frame(:,:,:,j) = reshape(fg(:,j),height,width);
end

%% plot SVD modes
sig = diag(S)/sum(diag(S));
figure(1);
subplot(2,1,1), plot(sig,'ko','Linewidth',[1.1])
title('Singular Values: Ski Drop')
ylabel('Energy, %')
subplot(2,1,2), semilogy(sig,'ko','Linewidth',[1.1])
ylabel('log(energy)')
xlabel('Singular Values')

%% plot original, background, and foreground at frames 2, 20, and 30
figure(2);
subplot(3,3,1), imagesc(rgb2gray(og(:,:,:,2))), colormap(gray), axis off
subplot(3,3,2), imagesc(rgb2gray(og(:,:,:,20))), colormap(gray), axis off
title('Background Subtraction of Video 1')
subplot(3,3,3), imagesc(rgb2gray(og(:,:,:,30))), colormap(gray), axis off

subplot(3,3,4), imagesc(bg_frame(:,:,:,2)), colormap(gray), axis off
subplot(3,3,5), imagesc(bg_frame(:,:,:,20)), colormap(gray), axis off
subplot(3,3,6), imagesc(bg_frame(:,:,:,30)), colormap(gray), axis off

subplot(3,3,7), imagesc(fg_frame(:,:,:,2)), colormap(gray), axis off
subplot(3,3,8), imagesc(fg_frame(:,:,:,20)), colormap(gray), axis off
subplot(3,3,9), imagesc(fg_frame(:,:,:,30)), colormap(gray), axis off
```

Same coding implementation for “monte_carlo_low.mp4”