AMATH 482 HW 5

Biyao Li

Mar 13, 2020

Abstract

In this assignment, we are given two videos and the goal is to separate each video into high frequency, moving foreground and low frequency, static background. This can be done by applying Dynamic Mode Decomposition(DMD) which is a dimensionality reduction algorithm similar to the Singular Value Decomposition(SVD) that allow us to study the spatial dynamics of the system and make prediction to the future state of the system.

1 Introduction and Overview

Recall that a dimensionality reduction algorithm is a way to transform high dimensional data set into the low-dimensional data that contains majority of the information. In this assignment, we will use the Dynamic Mode Decomposition on the video data matrix to achieve the goal of video separation. There are 2 advantages of using DMD:

- Comparing to the Proper Orthogonal(POD) Decomposition that requires a governing equation/model, DMD doesn't need such equation for us to study the dynamic of the system.
- Unlike the POD and SVD that give us an orthogonal spatial basis without any information of how it evolves in time, the DMD gives us spatial modes that is not necessarily orthogonal where the time dynamics is a exponential equation. These temporal information is what allow us to predict the future dynamic of the system.

In the context of this assignment, since the videos don't involves with any governing equations or models and we need to know how the background and foreground modes change with time, DMD is the best algorithm that will help us to achieve our goal. In the first video, it displays 5 cars that move through the race car track so that the foreground should includes the cars that are moving while the background should includes only the track without the cars. For the second videos, it shows a person skiing down the hill so that the foreground should includes only the person who is skiing and the background should includes the hill covered by snow.

2 Theoretical Background

2.1 Singular Value Decomposition(SVD)

For any matrix $A \in C^{m\times n}$, we can apply the Singular Value Decomposition to A such that

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

Where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are unitary matrices, and $\Sigma \in \mathbb{R}^{m\times n}$ diagonal matrix. The columns of U are the left singular vectors of A and the columns of V are the right singular vectors of A. The values on the diagonal of Σ are the singular values of A and if we let σ_1 be the first diagonal value and so on, the singular values of A are always ordered from the largest to smallest such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$. Moreover, since U and V are unitary matrix, it is true that $UU^* = I$ and $VV^* = I$ where I is the identity matrix.

2.2 Dynamic Mode Decomposition(DMD)

Suppose we have snapshots of the data that evolves both in space and time where there are n snapshots and each snapshot contains m spatial points such that each snapshot is in the following form:

$$\begin{bmatrix} (z_1, t_1) \\ (z_2, t_1) \\ (z_3, n_1) \\ \vdots \\ (z_{m-1}, t_1) \\ (z_m, t_1) \end{bmatrix} \tag{2}$$

Based on this, we can define the data matrix X as:

$$X = [S(z, t_1), S(z, t_2), \ldots, S(z, t_{n-1}), S(z, t_n)] \tag{3}$$

and X_i^j as the matrix X from the i-th to the j-th snapshot. In DMD, what we want to compute is a Koopman Operator A such that if we treat the first column of X as the initial state of the dynamic system, we can reconstruct rest of the columns in X as:

$$X = [x_0, Ax_0, A^2x_0, \ldots, A^{n-2}x_0, A^{n-1}x_0] \tag{4}$$

Moreover, we can derive X_1^{n-1} and X_2^n from X as:

$$X_1^{n-1} = [x_0, Ax_0, A^2x_0, \ldots, A^{n-2}x_0] \tag{5}$$

$$X_2^n = [Ax_0, A^2x_0, \ldots, A^{n-2}x_0, A^{n-1}x_0] \tag{6}$$

such that $X_2^n = AX_1^{n-1}$. Furthermore, we can compute the SVD of X_1^{n-1} to rewrite the equation as $X_2^n = AUSV^*$. Before we move on to the computation of A, we want to reduce the size of A by projecting A onto the dominant principal components of U. This can be done by multiplying U^* on both side and then we can try to isolate A by multiplying the pseudo-inverse of S and V^*. This will lead us to the following equation:

$$\tilde{A} = U^*AU = U^*X_2^nVS^{-1} \tag{7}$$

Moreover, $\tilde{A}$ and A are similar matrices such that they have the same eigenvalues and if W is the eigenvector of $\tilde{A}$, then the eigenvector of A is equal to UW. The eigenvector UW is the DMD mode of the data matrix and we can use this to find the initial amplitude of each mode by dividing the eigenvector by the first column of X. We can also compute the frequency spectrum, ω, of the DMD by taking the nature log of the eigenvalues and dividing it by the time step.

2.3 Reconstructing the video matrix

Recall that the goal of this assignment is to separate the video into low frequency background and high frequency foreground. To compute the data matrix that contains only the background pixels, we need to find the value and the index of ω that is close to zero. This will allows us to find the DMD modes that is corresponding to zero frequency and we can use these modes to construct each column of the data matrix with the following equation:

$$X(t) = \sum_{k=0}^{K} b_k \phi_k e^{\omega_k t} \tag{8}$$

where ϕ_k is the zero-frequency modes and b_k is the initial amplitude that is corresponding to each mode. Moreover, it is true that $X = X_{DMD}^{Background} + X_{DMD}^{Foreground}$ such that we can compute the foreground of each frame with $X - |X_{DMD}^{Background}|$. A problem with this subtraction is that it will cause foreground matrix to contains negative values which is not possible since pixels can't be negative. Hence, we need to define a R matrix which contains all of the negative values in the foreground matrix. Then we need to perform the following operations:

$$X_{DMD}^{Background} = |X_{DMD}^{Background}| + R \tag{9}$$

$$X_{DMD}^{Foreground} = X_{DMD}^{Foreground} - R \tag{10}$$

By doing these operations, we will be able to visualize the original video if we add the background and foreground matrix together. Furthermore, if we iterating through each column of the foreground and background matrix, we will be able to visualize the foreground and the background of the video separately which is what we want to achieve.

3 Algorithm Implementation and Development

3.1 Processing the video

In this assignment, we are given two mp4 video file where one contains a clip of race cars driving and another contains a clip of skiing down the hill. Before applying the Dynamic Mode Decomposition, we first need to load each file and turn them into a data matrix similar to the one in equation 3. First, we need to load the mp4 file into MATLAB which can be done by using the **VideoReader** function. This part of the code in appendix B is on line 4 for the race car clip and on line 51 for the skiing clip.

The **VideoRead** function will return us a VideoReader object which can be iterated through by using a while loop with a conditional statement regarding the existence of another frame to iterate using the **hasFrame** function. Inside the while loop, we will use the **readFrame** and **rgb2gray** function to turn each frame into a matrix that represents a gray-scale image. Then we will use the **reshape** function to turn this matrix into a column vector and these vectors will form the data matrix X. Since we will need to go through this process twice, I put this part of the implementation into a MATLAB function called **frame2Matrix** which takes in a VideoReader object and return us the data matrix X, time of the video, time step, and height and the width of each frame.

3.2 DMD

Following the procedure that was described in section 2.2, we first need to compute the matrices X_1^{n-1} and X_2^n which will be used to compute the Eigenvalue and Eigenvector of the Koopman Operator A. After we applied the SVD on X_1^{n-1}, we need to compute the low-rank approximation of the data in order to reduce the dimension of the data. Another reason behind this low-rank approximation is that it will get rid of all the zero singular values such that we won't run into problem when we multiply Σ^{-1} in equation 7 of section 2.2. In this assignment, the low-rank approximation will go up to r such that the first r singular values contains 90% of the energy in the system.

Then we will compute the eigenvalue and eigenvector of A using the reduced U, Σ, V matrices and we will also compute the frequencies using the eigenvalues and the time step returned from the **frame2Matrix** function. Moreover, we can plot the absolute value of the frequencies so that we can visualize the frequency that is close to 0. After we computed the index of the zero frequency, we will use index to find the corresponding eigenvector and the initial amplitude of the eigenvector. Then, if we follow the summation in equation 8, we will be able to compute the data matrix of the background and we can also compute the data matrix of the foreground by following the latter part of the section 2.3. Again, since we are going to compute the DMD for both video, I formulated this part of the code into another MATLAB function called **DMD** that takes in the data matrix X, length of the video, time step, and a threshold value such that every frequency below the threshold is considered to be the background frequency. This function will return the foreground and background matrices.

3.3 Visualizing the background and foreground frames

After we have computed the foreground and background data matrices, we will iterate through each column of the matrices to visualize each frame. This can be done with a for loop in which we transform each column into a 540 by 960 gray scale image and we can visualize each matrix using the **pcolor** function. Note that we also need to use the **flipud** function on the matrix of each frame because the way I computed the data matrices will flip the frame upside down so that this function will help to us adjust each frame in the right orientation. This part of the code in appendix B is from line 9 to 28 for the race car clip and line 55 to 72 for the skiing clip.

We can also analysis the separation by comparing a particular frame of the background and foreground. The code in appendix B from line 29 to 38 and 75 and 83 will allow us to perform such comparison.

Moreover, if we add the background and foreground matrix together, we should get the matrix for the original frames since we computed the foreground matrix by subtracting the background matrix from the original matrix. From line 40 to 47 and line 85 to 92, the code will produce 4 images that will allow us to compare the differences between the frame of the original matrix and the background plus foreground matrix for both clips.

4 Computational Results

4.1 Monte carlo clip

The first clip of this assignment is the Monte carlo clip that include several race cars moving along the track so what we should expect is the race cars in the foreground while the static track and everything else are in the background. First if we apply the DMD onto the data matrix of this clip, we will get two plots that is shown in figure 1. On the left, it is a plot for the amount of energy contained in each singular values of X_1^{n-1}

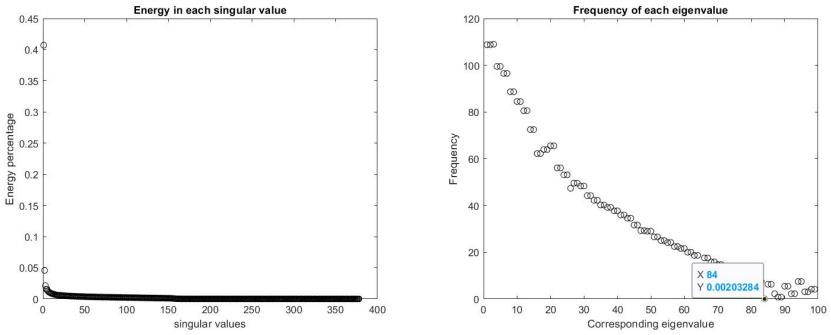

Figure 1: The energy plot on the left and frequency plot on the right

and from this, we can see that the first singular values has the majority of the energy. On the line 13 of the **DMD** function in appendix B, it calculates the number of singular values that captures 90% of the energy and for this clip, this number is equal to 99. On the right, it is a plot for the frequencies of each eigenvalue and recall that we are trying to find the eigenvalue that corresponding to zero frequency. From the plot, we can see that there is one frequency that is equal to 0.002 and this is the frequency that is closest to zero. Moreover, by finding its corresponding eigenvector, we can compute the data matrix of the background with the formula in equation 8.

After we completed applying DMD to the data matrix, we will have the matrices for each frame of the backgrounds and foregrounds. With these matrices, we can iterating through each of them to visualize the foreground and background video. Moreover, we can evaluate the quality of the DMD separation of the background and foreground by comparing one particular frame. In figure 2, it contains the background, foreground for the 40th frame. From the figure on the right, we can clearly see the two moving cars and the flag that is held by the men standing on the left of the track. Beside the moving parts, everything else is pretty much black which make sense since in a sparse matrix, must of the entries are zero which is black on gray scale. On the left, we have the background of the 40th frame and we can see that some parts of the car is still visible. Recall from equation 9, we added the negative values R to the background matrix and if we plot the R matrix of the 40-th frame, we can see that the parts of the car that is showing up in the background are caused by these negative values. However, in order to ensure the foreground matrix contains only positive values, we must add these negative value to the background matrix. Besides the parts of the car that is in the background, everything else in the background of the original video is showing up so we can conclude that the DMD was able to separate the original frame into background and foreground.

Moreover, from figure 3, we can see the original 40th frame and the 40th frame from adding the background and foreground matrix together. With no surprise, there is no difference between the two frames.

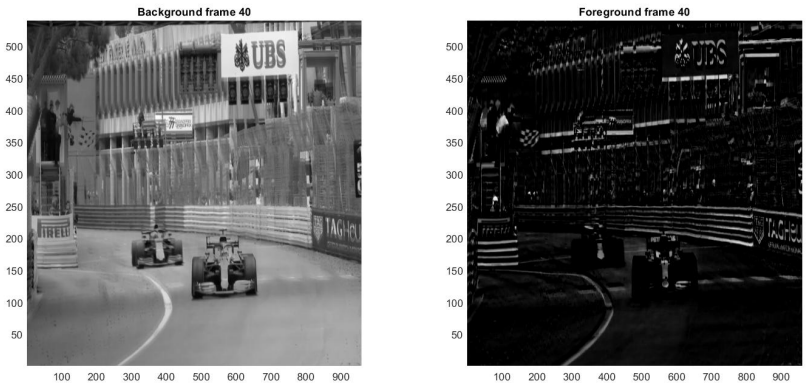

Figure 2: Background, foreground of 40th frame

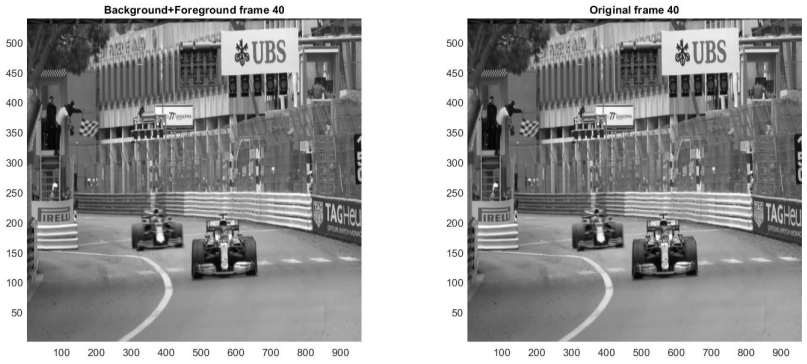

Figure 3: Original and foreground+background of 40th frame

4.2 Skiing clip

Following the same process as the first clip, we will first get two plots in figure 4. From the plot on the right, we can see that the first singular value contains most of the energy and from the DMD function, the smallest rank that contains 90% of the energy is 57. From the plot on the right, we can see that the frequency that is closest to zero is around 0.002 and using the corresponding eigenvector, we can compute the background matrix. With the background and foreground matrices, we can plot the background and foreground of a particular frame to check out the quality of the separation. In figure 5, we have the foreground of the 410-th frame and since in the original video, the person that is skiing is wearing a dark tone clothes so it is really hard to spot the person in the foreground. However, at the end of the video, the person jumped off the hill and caused the snow to splash when he reached to ground. This action of the snow is captured by the foreground at the bottom left which implies that the foreground is well separated from the original frame. On the left, we can see that the background is also well separated from the original frame.

Unfortunately, I wasn't able to fit another image in the 6 page limit so I couldn't include the figure for comparing the 410-th frame of the original matrix and the foreground plus background matrix. However, from the compareOriginal2.jpg in the GitHub repo of this assignment, we can see that there is no difference between the two frames which is what we expected.

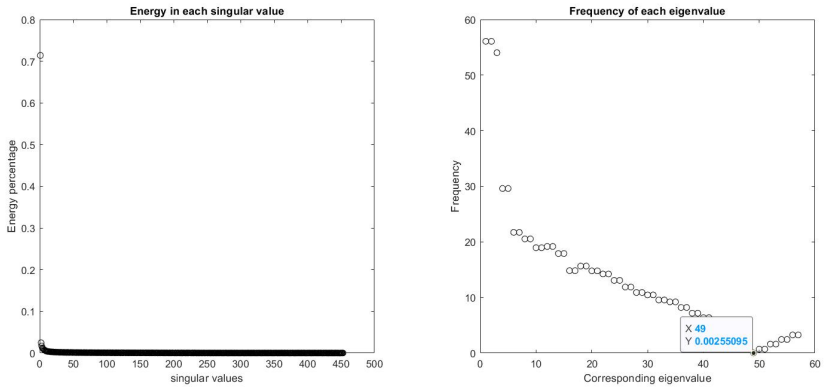

Figure 4: Energy contained in each singular value on the left and frequency of each eigenvalue on the right

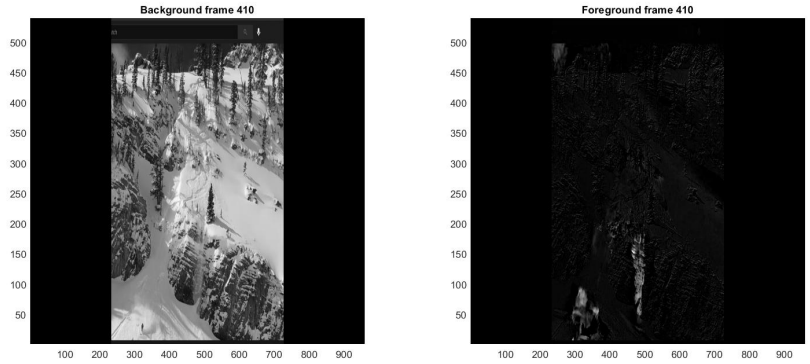

Figure 5: Background of 410-th frame on the left and foreground on the right

5 Summary and Conclusions

In this assignment, we used the Dynamic Mode Decomposition to separate the background and the foreground in two videos. Based on the results, we can conclude that DMD is able to separate the background and the foreground at a good quality. Besides not able to use the high resolution videos, this is indeed a fun and interesting assignment and I wish to learn more about DMD in the future.

Appendix A MATLAB Functions

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

- `B = reshape(A,sz)` reshapes A using the size vector, sz, to define size(B).
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X.
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector v on the main diagonal.
- `n = numel(A)` returns the number of elements, n, in array A, equivalent to prod(size(A)).

- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `B = flipud(A)` returns A with its rows flipped in the up-down direction (that is, about a horizontal axis).
- `pcolor(C)` creates a pseudocolor plot using the values in matrix C.

Appendix B MATLAB Code

The MATLAB code for this assignment including the two MATLAB functions that I created.

```
%%
clear ; close all; clc
%% Load the movies
monte = VideoReader('monte_carlo_low.mp4');
%% Creating matrix frame by frame
[frame1,t1,dt1,h1,w1] = frame2Matrix(monte);
%% DMD of monte
[background, foreground] = DMD(frame1,t1,dt1,0.1);
numFrames1 = size(frame1,2);
%% Foreground construction
for i = 1:numFrames1
    currFrame = reshape(foreground(:,i),h1,w1);
    pcolor(flipud(currFrame)), colormap(gray), shading interp
    drawnow
end
%% Background construction
for i = 1:numFrames1
    currFrame = reshape(background(:,i),h1,w1);
    pcolor(flipud(currFrame)), colormap(gray), shading interp
    drawnow
end
%% Original video construction
total = background + foreground;
for i = 1:numFrames1
    currFrame = reshape(total(:,i),h1,w1);
    pcolor(flipud(currFrame)), colormap(gray), shading interp
    drawnow
end
%% Plot one frame
frame = 40;
subplot(1,2,1)
backFrame = reshape(background(:,frame),h1,w1);
pcolor(flipud(backFrame)), colormap(gray), shading interp
title('Background frame 40')
subplot(1,2,2)
foreFrame = reshape(foreground(:,frame),h1,w1);
pcolor(flipud(foreFrame)), colormap(gray), shading interp
title('Foreground frame 40')
%% Compare original
subplot(1,2,1)
totalFrame = reshape(total(:,frame),h1,w1);
pcolor(flipud(totalFrame)), colormap(gray), shading interp
title('Background+Foreground frame 40')
```

```
subplot(1,2,2)
OriginalFrame = reshape(frame1(:,frame),h1,w1);
pcolor(flipud(OriginalFrame)), colormap(gray), shading interp
title('Original frame 40')
%%
clear; close all; clc
%% DMD for ski
ski = VideoReader('ski_drop_low.mp4');
[frame2,t2,dt2,h2,w2] = frame2Matrix(ski);
[background2, foreground2] = DMD(frame2,t2,dt2,0.1);
%% Foreground construction
numFrames1 = size(frame2,2);
for i = 1:numFrames1
    currFrame = reshape(abs(foreground2(:,i)),h2,w2);
    pcolor(flipud(currFrame)), colormap(gray), shading interp
    drawnow
end
%% Background construction
for i = 1:numFrames1
    currFrame = reshape(abs(background2(:,i)),h2,w2);
    pcolor(flipud(currFrame)), colormap(gray), shading interp
    drawnow
end
%% Original video construction
total = background2 + foreground2;
for i = 1:numFrames1
    currFrame = reshape(abs(total(:,i)),h2,w2);
    pcolor(flipud(currFrame)), colormap(gray), shading interp
    drawnow
end
%% Plot one frame
frame = 410;
subplot(1,2,1)
backFrame = reshape(background2(:,frame),h2,w2);
pcolor(flipud(backFrame)), colormap(gray), shading interp
title('Background frame 410')
subplot(1,2,2)
foreFrame = reshape(foreground2(:,frame),h2,w2);
pcolor(flipud(foreFrame)), colormap(gray), shading interp
title('Foreground frame 410')
%% Compare original
subplot(1,2,1)
totalFrame = reshape(total(:,frame),h2,w2);
pcolor(flipud(totalFrame)), colormap(gray), shading interp
title('Background+Foreground frame 410')
subplot(1,2,2)
OriginalFrame = reshape(frame2(:,frame),h2,w2);
pcolor(flipud(OriginalFrame)), colormap(gray), shading interp
title('Original frame 410')

function [matrix,t,dt,height,width] = frame2Matrix(video)
    frames = video.NumFrames;
    height = video.Height;
    width = video.Width;
```

```
    matrix = zeros(height*width, frames);
    idx = 1;
    dt = 1/video.FrameRate;
    t = 0:dt:video.Duration;
    while hasFrame(video)
        currFrame = readFrame(video);
        frame2gray = rgb2gray(currFrame);
        matrix(:,idx) = reshape(frame2gray,height*width,1);
        idx = idx + 1;
    end
end

function [low_rank,sparse] = DMD(frameMatrix,t,dt,background)
    % Compute X and X' where AX = X'
    X = frameMatrix(:,1:end-1);
    Xprime = frameMatrix(:,2:end);
    % Find A by computing its eigenvalues and eigenvectors
    [U,S,V] = svd(X, 'econ');
    % find none zero singular value
    subplot(1,2,1)
    plot(diag(S)/sum(diag(S)), 'ko')
    title('Energy in each singular value')
    xlabel('singular values')
    ylabel('Energy percentage')
    rank = find(cumsum(diag(S)./sum(diag(S))) > 0.9,1);
    rank
    U = U(:,1:rank);
    S = S(1:rank,1:rank);
    V = V(:,1:rank);
    % Computing Eignevalues and vectors
    A = U'*Xprime*V/S;
    [ev, D] = eig(A);
    eigenValues = diag(D);
    phi = Xprime*V/S*ev;
    omega = log(eigenValues)/dt;
    subplot(1,2,2)
    plot(abs(omega),'ko')
    title('Frequency of each eigenvalue')
    xlabel('Corresponding eigenvalue')
    ylabel('Frequency')
%   Compute the background
    idx = find(abs(omega) < background);
    omega_back = omega(idx);
    phi_back = phi(:,idx);
    b_back = phi_back \ (X(:,1));
    low_rank = zeros(numel(omega_back), length(t));
    for tt = 1:length(t)
        low_rank(:, tt) = b_back .* exp(omega_back .* t(tt));
    end
    low_rank = phi_back * low_rank;
    % Compute the foreground
    sparse = frameMatrix - abs(low_rank);
    R = sparse .* (sparse < 0);
    low_rank = R + abs(low_rank) ;
```

```
    sparse = sparse - R;
end
```