AMATH 482 Homework 5

Tommy Dong

March 16, 2021

Abstract

This world is full of high-dimensional data sets, but many of them use tons of of different observable variables to describe something that when viewed the right way turns out to be low-dimensional. We really care about dimensionality reduction. In this project, we will focus on one of the data-based algorithms called Dynamic Mode Decomposition.

1 Introduction and Overview

Dynamic Mode Decomposition(DMD) can take the advantage of low-dimensionality in experimental data without having to rely on a given set of governing equations. DMD can assist of solving data or system that varied with time. In particular for this project, we utilized its property to decompose videos into two parts: Background video and Foreground video.

1.1 Data Description

Two videos are provided for this project. One is from Monte-Carlo Tournament, and the other is a ski drop clips. Our goal is to developed a Dynamic Mode Decomposition Algorithm in order to separate background and foreground of the videos.

1.2 Goal

1.2.1 Develop Dynamic Mode Decomposition(DMD) Algorithm

Initially, we took the compressed Monte-Carlo clip to develop our Dynamic Mode Decomposition Algorithm. And we compared the DMD reconstruction of the data to the original data to testify its fidelity.

1.2.2 Background and Foreground Separation on one video

After we had a convincing DMD algorithm, we went on to distinct the background and foreground of the video by construct low-rank DMD by our DMD algorithm, which is the background data of the video. Then we subtracted the entire data with background data, what was left is the sparse DMD, which is the foreground of the video. We determined the mode we use for DMD, and reconstructed the video separately with background and foreground to see if they did a decent job.

1.2.3 Background and Foreground separation for all video clips

When we eventually verified the effectiveness of our DMD algorithm, we took all rest of the videos in, changed the modes by the situation of the video, and then separated them into backgrounds and foregrounds. Fianlly, we reported our result.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) decompose some matrix A into unitary matrices U and V and a diagonal matrix Σ.

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

Σ is the singular value matrix A, U are the left singular vectors of A and V is the right singular vectors of A.

2.2 Dynamic Mode Decomposition(DMD)

To begin with, assume we use frames of the video we have to form columns of data matrices X.

$$X = [U(x,t_1)U(x,t_2)...U(x,t_M)]X_j^k = [U(x,t_j)U(x,t_{j+1})...U(x,t_k)] \tag{2}$$

where M is the number of frames the video have. X_j^k is just columns j through k of the matrix X.

DMD approximates the modes of the Koopman operator.We define Koopman operator A as a linear, time-independent operator such that

$$x_{j+1} = Ax_j \tag{3}$$

A maps the data from time t_j to t_{j+1}. What A general do is to advance a frame of data in time by Δt. We perform a linear mapping from one timestep to the next, regardless of the fact that the system is whether linear or nonlinear.

Then we will perform the process of Dynamic Mode Decomposition. We first consider a matrix

$$X_1^{M-1} = [x_1x_2...x_{M-1}] \tag{4}$$

$$X_1^{M-1} = [x_1Ax_1...A^M - 2x_1] \tag{5}$$

The first equation can be expressed with the help of Koopman operator in the form of the second equation. Then we can get a new matrix

$$X_2^M = AX_1^M - 1 + re_{M-1}^T \tag{6}$$

e_{M-1}^T is the vector with all zeros except 1 at the (M-1)st component. Our goal is to retrieve A. And we can achieve this by performing several steps as below

$$X_1^{M-1} = U\Sigma V^* \tag{7}$$

$$X_2^M = AU\Sigma V^* + re_{M-1}^T \tag{8}$$

$$U^*X_2^M = U^*AU\Sigma V^* \tag{9}$$

$$U^*AU = U^*X_2^MV\Sigma^{-1} \tag{10}$$

We let $\tilde{S} = U^*X_2^MV\Sigma^{-1}$. Notice everything on the right-hand side is known. Also, $\tilde{S}$ are related to A. This implies they are similar, and similar matrices have lots of properties in common, having the same eigenvalues would be one of them. So assume y is the eigenvector of $\tilde{S}$, then Uy is the eigenvector of A. We called the eigenvector of A the DMD modes,

$$\phi_k = Uy_k \tag{11}$$

With DMD modes, we can finally construct the DMD of the original data. We use $\omega_k = ln(\mu_k/\Delta t$ as our time dynamics.

$$x_{DMD}(t) = \sum_{k=1} Kb_k\phi_k e^{\omega_k t} = \Phi diag(e^{\omega_k t})b \tag{12}$$

This x_{DMD} is also a reconstruction of the data. And since we knew time is 0 at initial frame, we can calculate coefficient b_k with t = 0

$$x_1 = \Phi b \tag{13}$$

$$b = \Phi^* x_k \tag{14}$$

Φ^* is the pseudoinverse of the matrix Φ, which consists of eigenvector ϕ_k as its columns.

3 Algorithm Implementation and Development

This project only relied on Dynamic Mode Decomposition. The general procedure is to transform data into columns of data matrices. Then we subtract submatrices X_1^{M-1} and X_2^M from X. Then we implement SVD decomposition on X_1^{M-1} in order to gather all the information needed to construct $\tilde{S} = U^* X_2^M V \Sigma^{-1}$. Then we utilized the initial frame x_1 and the pseudoinverse of Φ to find the coefficient b_k.

Particularly for this project, we required more steps to separate background and foreground. To begin with,

$$X_{DMD} = b_p \phi_p e^{\omega_p t)} + \sum_{j \neq p} b_j \phi_j e^{\omega_j t)} \tag{15}$$

$$X_{DMD}^{Low-Rank} = b_p \phi_p e^{\omega_p t)} \tag{16}$$

$$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse} \tag{17}$$

The $X_{DMD}^{Low-Rank}$ is the data we extracted from the entire data as background, for $||\omega_p|| \approx 0$. And by subtracted the modulus of $X_{DMD}^{Low-Rank}$ from X, we can have X_{DMD}^{Sparse}, which is our foreground of the video. There might exists some data being negative, which did not make sense. We take those negative values out from X_{DMD}^{Sparse} and return them to $X_{DMD}^{Low-Rank}$. After all these procedures finished, we get ourselves the data matrices for background and foreground. With the algorithm, we experimented with it to retrieve the appropriate mode of DMD used for background.

4 Computational Results

Since the two original video clips always crash when doing DMD, we use low versions instead for computation.

4.1 Monte-Carlo

After running DMD algorithm on Monte-Carlo video, we set the threshold as $||\omega_p|| < 0.001$, which only used 1 modes. When we increased to use 3 DMD modes, the foreground only has a fluctuated paths on the path of the race car, but without any movements of race cars. For 1 mode, though the background accidentally included two static images of race car(which I guess can be improved with the original clip for more details), foreground did captured moving objects like race cars and the waving flags on the up left corner. And for background, moving objects still appeared, but in some level transparent or looked similar to the background. The plot of eigenvalues is also included, as we use the threshold to filter those modes whose real component in ω is less than 0.0001, the values that almost lined up on the imaginary axis.

Figure 1: The left is background and the right is foreground, both are created from the 200th frame.

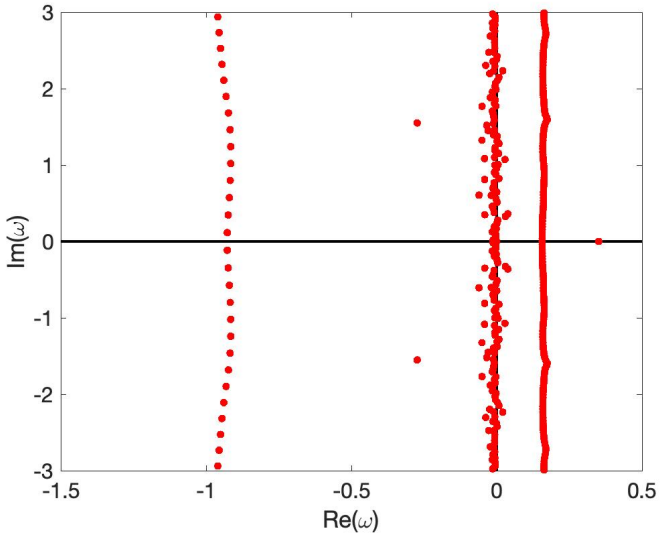

Figure 2: Eigenvalues Plot for Monte-Carlo(ω)

4.2 Ski Drop

For Ski Drop, the moving object this time is a single man skiing on a slope. This is harder to detect than the big race car from Monte-Carlo. Though we used the same threshold that $||\omega_p|| < 0.001$, but this time 33 DMD modes was used, which is a significant increase compared to the previous one. For more real component of ω values are close to 0, makes them almost lied on the imaginary axis. Form the reconstructed video we could see the skier's movement clearly in foreground(The frame of the picture made it hard to detect, but the movement is obvious in the video), and skier disappeared in some frames of the background.

Figure 3: The left is background and the right is foreground, both are created from the 200th frame.

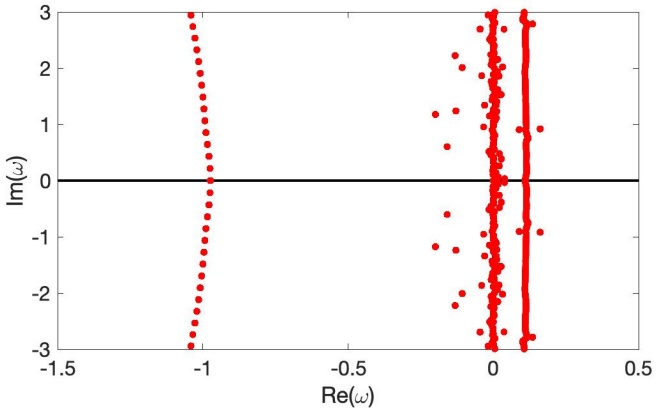

Figure 4: Eigenvalues Plot for Ski Drop(ω)

5 Summary and Conclusions

The achievement for this project is that we successfully took the advantage of Dynamic Mode Decomposition(DMD) to do dimensionality reductions and separate low-rank and sparse portion of videos to create backgrounds and foregrounds for the video. DMD did not require us to come up with any existing model or equations. Instead, it efficiently utilized the information we known for reconstruct the data. We haven't encountered any future prediction problem, but DMD can be also used to do future prediction, which is different from SVD, PCA, POD where these algorithm can only work on existing information.

In reality, this implementation have its potentials. It can be used to reduce the size of the data we need to store. For example, we don't need to keep track of static background all along, with the help of DMD algorithm, we can separate background and foreground, then only store one frame of background to reduce unnecessary data and save space for moving objects like racing cars, waving flags, or skiers. With this kind of techniques, we can perform our algorithm faster and more efficient, like one of our guest speakers mentioned in her seminar. Use them properly, maybe the runtime can reduced from weeks to only a couple of hours.

Appendix A MATLAB Functions

- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of m-by-n matrix `A`.
- `[V,D] = eig(A,B)` returns diagonal matrix `D` of generalized eigenvalues and full matrix `V` whose columns are the corresponding right eigenvectors, so that `A*V = B*V*D`.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector `v` on the main diagonal.
- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `video = read(v)` reads all video frames from the file associated with `v`.
- `y = linspace(x1,x2,n)` generates `n` points. The spacing between the points is `(x2-x1)/(n-1)`.

Appendix B MATLAB Code

Add your MATLAB code here. This section will not be included in your page limit of six pages.

```
%% Clean workspace
clear all; close all; clc
format shortG
```

```
%%
MCL = VideoReader('monte_carlo_low.mp4');
mcl_frames = read(MCL);
SDL = VideoReader('ski_drop_low.mp4');
sdl_frames = read(SDL);
numFrames_mcl = size(mcl_frames,4);
numFrames_sdl = size(sdl_frames,4);
t_mcl = linspace(0,MCL.Duration,numFrames_mcl); dt_mcl = t_mcl(2) - t_mcl(1);
t_sdl = linspace(0,SDL.Duration,numFrames_sdl); dt_sdl = t_sdl(2) - t_sdl(1);

%%
M_mcl = zeros(size(mcl_frames,1)*size(mcl_frames,2),numFrames_mcl);
M_sdl = zeros(size(sdl_frames,1)*size(sdl_frames,2),numFrames_sdl);
for i = 1:numFrames_mcl
    M_mcl(:,i) = reshape(im2double(rgb2gray(mcl_frames(:,:,:,i))),[size(
        mcl_frames,1)*size(mcl_frames,2),1]);
end
for i = 1:numFrames_sdl
    M_sdl(:,i) = reshape(im2double(rgb2gray(sdl_frames(:,:,:,i))),[size(
        sdl_frames,1)*size(sdl_frames,2),1]);
end

%% Monte-Carlo
M_mcl_1 = M_mcl(:,1:end-1);
M_mcl_2 = M_mcl(:,2:end);
[U_mcl,Sigma_mcl,V_mcl] = svd(M_mcl_1,'econ');
S_mcl = U_mcl'*M_mcl_2*V_mcl*diag(1./diag(Sigma_mcl));
[eV_mcl,D_mcl] = eig(S_mcl);
mu_mcl = diag(D_mcl);
omega_mcl = log(mu_mcl)/dt_mcl;
Phi_mcl = U_mcl*eV_mcl;
y0_mcl = Phi_mcl\M_mcl_1(:,1);

mode_mcl = 0;
mode_mcl_i = zeros(length(omega_mcl),1);
for i = 1:length(omega_mcl)
    if abs(real(omega_mcl(i))) < 0.01
        mode_mcl = mode_mcl+1;
        mode_mcl_i(mode_mcl) = i;
    end
end
mode_mcl_i = mode_mcl_i(1:mode_mcl);

y0_mcl_low = y0_mcl(mode_mcl_i);
omega_mcl_low = omega_mcl(mode_mcl_i);
Phi_mcl_low = Phi_mcl(:,mode_mcl_i);

u_modes_mcl_low = zeros(length(y0_mcl_low),numFrames_mcl);
for iter = 1:numFrames_mcl
    u_modes_mcl_low(:,iter) = y0_mcl_low.*exp(omega_mcl_low*t_mcl(iter));
end
u_dmd_mcl_low = Phi_mcl_low*u_modes_mcl_low;

u_dmd_mcl_sparse = M_mcl - abs(u_dmd_mcl_low);
```

```
u_mcl_R = zeros(size(u_dmd_mcl_sparse,1),size(u_dmd_mcl_sparse,2));
for i = 1:size(u_dmd_mcl_sparse,1)
    for j = 1:size(u_dmd_mcl_sparse,2)
        if u_dmd_mcl_sparse(i,j) < 0
            u_mcl_R(i,j) = u_dmd_mcl_sparse(i,j);
        end
    end
end
u_dmd_mcl_low_final = u_mcl_R+abs(u_dmd_mcl_low);
u_dmd_mcl_sparse_final = u_dmd_mcl_sparse-u_mcl_R;

R_mcl_b = zeros(size(mcl_frames,1),size(mcl_frames,2),numFrames_mcl);
R_mcl_f = zeros(size(mcl_frames,1),size(mcl_frames,2),numFrames_mcl);

for i = 1:numFrames_mcl
   R_mcl_b(:,:,i) = reshape(u_dmd_mcl_low_final(:,i), [size(mcl_frames,1),size
       (mcl_frames,2)]);
   R_mcl_f(:,:,i) = reshape(u_dmd_mcl_sparse_final(:,i), [size(mcl_frames,1),
       size(mcl_frames,2)]);
end

%%
figure(1)
imshow(mat2gray(R_mcl_b(:,:,200)));
figure(2)
imshow(mat2gray(R_mcl_f(:,:,200)));

%%
line = -189:189;
plot(zeros(numFrames_mcl,1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(numFrames_mcl,1),'k','Linewidth',2) % real axis
plot(real(omega_mcl)*dt_mcl,imag(omega_mcl)*dt_mcl,'r.','Markersize',15)
xlabel('Re(\omega)')
ylabel('Im(\omega)')
set(gca,'FontSize',16,'Xlim',[-1.5  0.5],'Ylim',[-3  3])
%%
implay(R_mcl_b);
implay(R_mcl_f);

%% Ski Drop
M_sdl_1 = M_sdl(:,1:end-1);
M_sdl_2 = M_sdl(:,2:end);
[U_sdl,Sigma_sdl,V_sdl] = svd(M_sdl_1,'econ');
S_sdl = U_sdl'*M_sdl_2*V_sdl*diag(1./diag(Sigma_sdl));
[eV_sdl,D_sdl] = eig(S_sdl);
mu_sdl = diag(D_sdl);
omega_sdl = log(mu_sdl)/dt_sdl;
Phi_sdl = U_sdl*eV_sdl;
y0_sdl = Phi_sdl\M_sdl_1(:,1);

mode_sdl = 0;
mode_sdl_i = zeros(length(omega_sdl),1);
for i = 1:length(omega_sdl)
```

```
    if abs(real(omega_sdl(i))) < 1e-3
        mode_sdl = mode_sdl+1;
        mode_sdl_i(mode_sdl) = i;
    end
end
mode_sdl_i = mode_sdl_i(1:mode_sdl);

y0_sdl_low = y0_sdl(mode_sdl_i);
omega_sdl_low = omega_sdl(mode_sdl_i);
Phi_sdl_low = Phi_sdl(:,mode_sdl_i);

u_modes_sdl_low = zeros(length(y0_sdl_low),numFrames_sdl);
for iter = 1:numFrames_sdl
   u_modes_sdl_low(:,iter) = y0_sdl_low.*exp(omega_sdl_low*t_sdl(iter));
end
u_dmd_sdl_low = Phi_sdl_low*u_modes_sdl_low;

%
u_dmd_sdl_sparse = M_sdl - abs(u_dmd_sdl_low);
u_sdl_R = zeros(size(u_dmd_sdl_sparse,1),size(u_dmd_sdl_sparse,2));
for i = 1:size(u_dmd_sdl_sparse,1)
    for j = 1:size(u_dmd_sdl_sparse,2)
        if u_dmd_sdl_sparse(i,j) < 0
            u_sdl_R(i,j) = u_dmd_sdl_sparse(i,j);
        end
    end
end
u_dmd_sdl_low_final = u_sdl_R+abs(u_dmd_sdl_low);
u_dmd_sdl_sparse_final = u_dmd_sdl_sparse-u_sdl_R;

%
R_sdl_b = zeros(size(sdl_frames,1),size(sdl_frames,2),numFrames_sdl);
R_sdl_f = zeros(size(sdl_frames,1),size(sdl_frames,2),numFrames_sdl);

for i = 1:numFrames_sdl
   R_sdl_b(:,:,i) = reshape(u_dmd_sdl_low_final(:,i), [size(sdl_frames,1),size
      (sdl_frames,2)]);
   R_sdl_f(:,:,i) = reshape(u_dmd_sdl_sparse_final(:,i), [size(sdl_frames,1),
      size(sdl_frames,2)]);
end
```