AMATH582 Hw5: Background Subtraction in Video Streams

Xiangyu Gao

Mar. 17, 2021

Abstract

In this report, we analyzed two video clips with the dynamic mode decomposition method. After finding the low-rank frequency that represents the static background, we successfully separate the video stream to both the foreground video and a background.

1 Introduction and Overview

Exploiting low-dimensionality in complex systems has already been demonstrated to be an effective method for either computationally or theoretically reducing a given system to a more tractable form. The data-based algorithms can be constructed to understand, mimic and control the complex system. Dynamic Mode Decomposition (DMD) is a relatively new data-based algorithm that requires no underlying governing equations, rather snapshots of experimental measurements are used to predict and control a given system [1].The methodology presented here uses data alone as the source for informing us of the state and dimensionality of the system.

Using the Dynamic Mode Decomposition method on the video clips ski drop.mov and monte carlo.mov containing a foreground and background object, we can separate the video stream to both the foreground video and a background.

The DMD spectrum of frequencies can be used to subtract background modes. Specifically, assume that ω_p, where $p \in \{1, 2, \ldots, \ell\}$, satisfies $\|\omega_p\| \approx 0$, and that $\|\omega_j\| \, \forall j \neq p$ is bounded away from zero. We can separate the DMD terms into approximate low-rank and sparse reconstructions, which are given by:

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low-Rank}} = b_p \varphi_p e^{\omega_p \mathrm{t}}$$

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}} = \sum_{j \neq p} b_j \varphi_j e^{\omega_j \mathbf{t}}$$

2 Theoretical Background

2.1 Dynamic Mode Decomposition (DMD)

To construct the appropriate Koopman operator that best represents the data collected, the matrix $\mathbf{X}_1^{M-1}$ is considered:

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{M-1} \end{bmatrix}$$

and the matrix $\mathbf{X}_2^M$ is considered:

$$\mathbf{X}_2^M = \begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \cdots & \mathbf{x}_M \end{bmatrix}$$

The SVD of $\mathbf{X}_1^{M-1}$ is computed:

$$\mathbf{X}_1^{M-1} = \mathbf{U}\Sigma\mathbf{V}^*$$

where $*$ denotes the conjugate transpose, $\mathbf{U} \in \mathbb{C}^{N \times K}, \Sigma \in \mathbb{C}^{K \times K}$ and $\mathbf{V} \in \mathbb{C}^{M-1 \times K}$. Here K is the reduced SVD's approximation to the rank of $\mathbf{X}_1^{M-1}$ If the data matrix is full rank and the data has no

suitable low-dimensional structure, then the DMD method fails immediately. However, if the data matrix can be approximated by a low-rank matrix, then DMD can take advantage of this low dimensional structure to project a future state of the system. Thus once again, the SVD plays the critical role in the methodology.

The key idea now is the observation that the eigenvalues of $\mathbf{S}$ approximate some of the eigenvalues of the unknown Koopman operator A, making the DMD method similar to the Arnoldi algorithm and its approximations to the Ritz eigenvalues. Schmid showed that rather than computing the matrix $\mathbf{S}$ directly, we can instead compute the lower-rank matrix

$$\tilde{\mathbf{S}} = \mathbf{U}^* \mathbf{X}_2^M \mathbf{V} \Sigma^{-1}$$

which is related to $\mathbf{S}$ via a similarity transformation. Recall that the matrices $\mathbf{U}, \mathbf{\Sigma}$ and $\mathbf{V}$ arise from the SVD reduction of $\mathbf{X}_1^{M-1}$.

Consider then the eigenvalue problem associated with $\tilde{\mathbf{S}}$:

$$\tilde{\mathbf{S}}\mathbf{y}_k = \mu_k \mathbf{y}_k \quad k = 1, 2, \cdots, K$$

where K is the rank of the approximation we are choosing to make. The eigenvalues μ_k capture the time dynamics of the discrete Koopman map $\mathbf{A}$ as a Δt step is taken forward in time. These eigenvalues and eigenvectors can be related back to the similarity transformed original eigenvalues and eigenvectors of $\mathbf{S}$ in order to construct the DMD modes:

$$\psi_k = \mathbf{U}\mathbf{y}_k$$

With the low-rank approximations of both the eigenvalues and eigenvectors in hand, the projected future solution can be constructed for all time in the future. By first rewriting for convenience $\omega_k = \ln(\mu_k)/\Delta t$ (recall that the Koopman operator time dynamics is linear), then the approximate solution at all future times, $\mathbf{x}_{DMD}(t)$, is given by

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k(0)\psi_k(\mathbf{x})\exp(\omega t) = \Psi\,\mathrm{diag}(\exp(\omega t)\mathbf{b}$$

where $b_k(0)$ is the initial amplitude of each mode, Ψ is the matrix whose columns are the eigenvectors $\psi_k, \mathrm{diag}(\omega t)$ is a diagonal matrix whose entries are the eigenvalues $\exp(\omega_k t)$, and $\mathbf{b}$ is a vector of the coefficients b_k

It only remains to compute the initial coefficient values $b_k(0)$. If we consider the initial snapshot (x_1) at time zero, let's say, then gives $\mathrm{x}_1 = \Psi\mathrm{b}$ This generically is not a square matrix so that its solution

$$\mathbf{b} = \Psi^+ \mathbf{x}_1$$

can be found using a pseudo-inverse. Indeed, Ψ^+ denotes the Moore-Penrose pseudo-inverse that can be accessed in MATLAB via the pinv command. As already discussed in the compressive sensing section, the pseudo-inverse is equivalent to finding the best solution b the in the least-squares (best fit) sense. This is equivalent to how DMD modes were derived originally.

3 Algorithm Implementation and Development

We introduce the algorithm implementation details below and attach corresponding MATLAB codes in Appendix B.

First, we read the movie data frame by frame, reshape each frame into a vector and concatenate all vectors along columns. The new matrix we obtain is X. The *dtyte* of X is transformed from *uint8* to *single*. Then we construct matrix X_1 taken from the first to penult column vector of X, and construct matrix X_2 taken from the second to last column vector of X.

The matrix X_1 and X_2 are used for the DMD analysis above. In particular, we initialize the rank used in DMD with 3, and only pick one of them as the low rank corresponds to the background. The determination method is to check if the norm of ω approximates zero.

With the picked low rank p, we can reconstruct corresponding DMD mode $X_{DMD}^{Low-Rank} = b_p \varphi_p e^{\omega_p t}$. Then we substrata the low-rank background mode from the original signal X to obtain the foreground video X_{DMD}^{Sparse}. We set all positive values of X_{DMD}^{Sparse} to zero with matrix R, and then add R to $X_{DMD}^{Low-Rank}$ to maintain the constraints that the sum of $X_{DMD}^{Low-Rank}$ and X_{DMD}^{Sparse} is equal to X.

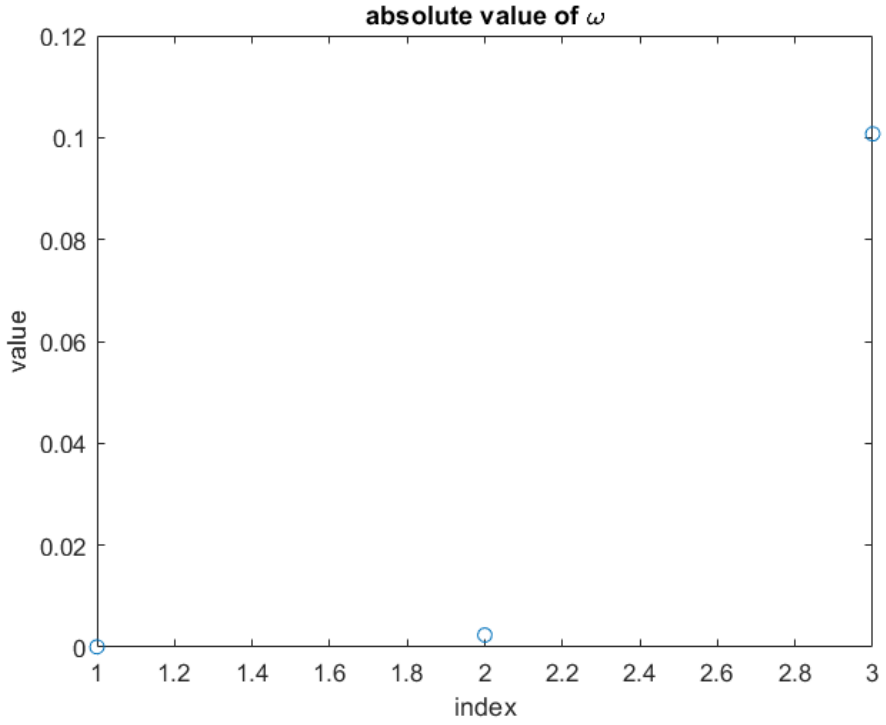

Figure 1: The absolute value of calculated ω for DMD analyzing **ski drop** movie with rank 3.

4 Computational Results

For the **ski drop** movie, we plot the calculated absolute value of ω in Fig. 1 and find the first frequency has the smallest value. Hence, we assume ω_1 to represent the background static component.

For the 100th frame, we reconstruct the background and foreground image, which are shown in Fig. 2(b), (c) respectively. In the background image, it is obvious that the skier has been removed compared to the original image (Fig. 2(c)).

For the **monte carlo** movie, we plot the calculated absolute value of ω in Fig. 3 and find the first frequency has the smallest value. Hence, we assume ω_1 to represent the background static component.

For the 20th frame, we reconstruct the background and foreground image, which are shown in Fig. 4(b), (c) respectively. In the background image, it is obvious that the most of the parts of race cars has been removed compared to the original image (Fig. 4(c)).

5 Summary and Conclusions

In this report, we analyzed two video clips with the dynamic mode decomposition method. After finding the low-rank frequency that represents the static background, we successfully separate the video stream to both the foreground video and a background.

References

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

Figure 2: DMD Analysis for frame 100 of **ski drop** movie: (a) The foreground image; (b) The background image; (c) The original image.

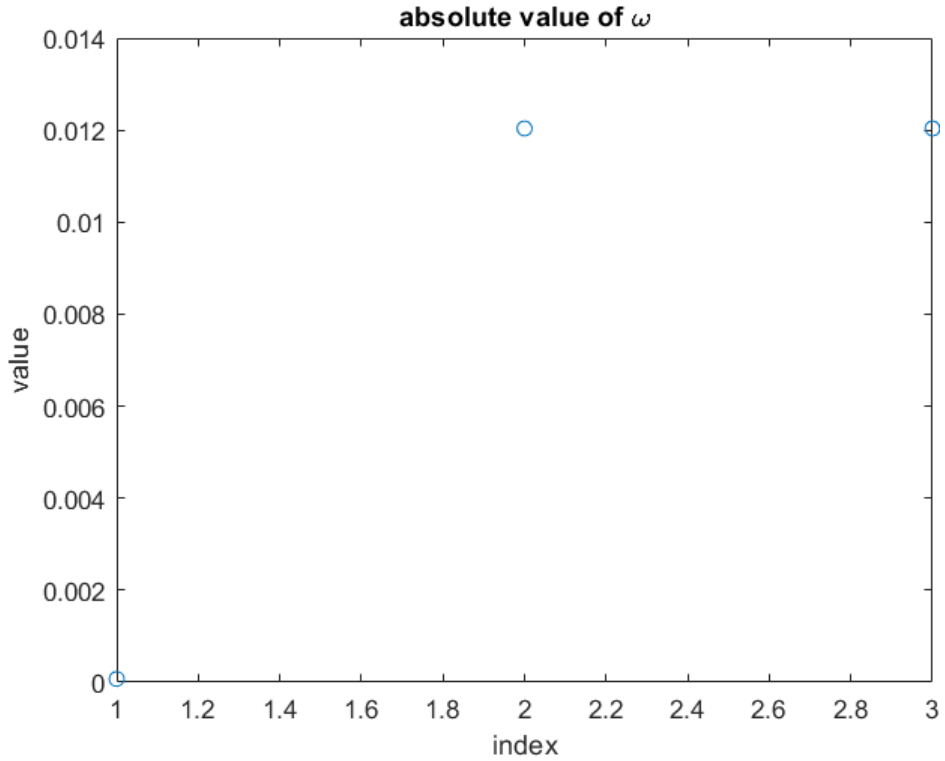

Figure 3: The absolute value of calculated ω for DMD analyzing **monte carlo** movie with rank 3.

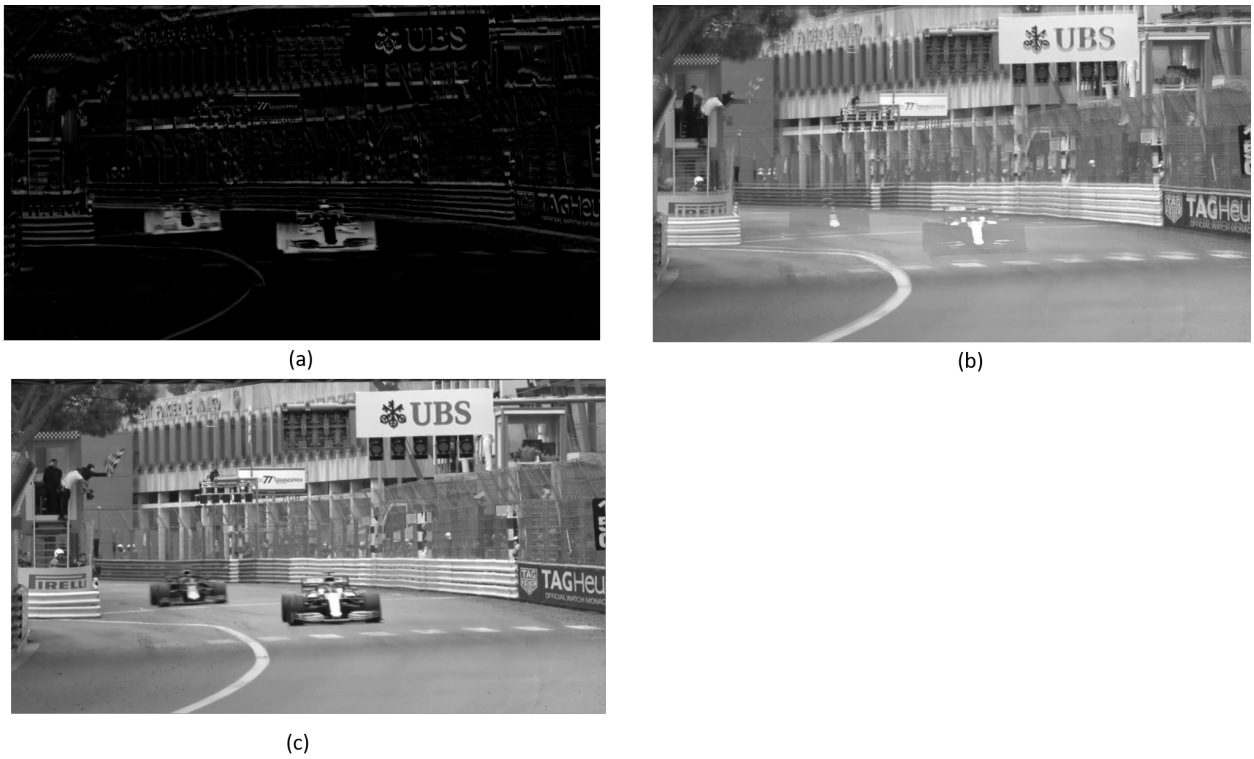

Figure 4: DMD Analysis for frame 20 of **monte carlo** movie: (a) The foreground image; (b) The background image; (c) The original image.

Appendix A MATLAB Functions

The important MATLAB functions and thier implementation explanations are listed here:

- `OBJ = VideoReader(FILENAME)` can read in video data from a multimedia file. FILENAME is a character vector or string scalar specifying the name of a multimedia file. There are no restrictions on file extensions. By default, MATLAB looks for the file FILENAME on the MATLAB path.

Appendix B MATLAB Code

The MATLAB codes are shown here. We also stored corresponding codes on Github under the repository (https://github.com/Xiangyu-Gao/AMATH-582-Computational-method-for-data-analysis/tree/ main/Hw5).

```
clc
clear all
close all

%% read movie file
% a = VideoReader('monte_carlo_low.mp4');
a = VideoReader('ski_drop_low.mp4');

X = [];
for img = 1:a.NumFrames
    filename=strcat('frame',num2str(img),'.jpg');
    b = rgb2gray(read(a, img));
    % reshape each frame and concatenate them
    X = [X, reshape(b, [540*960, 1])];

end
X = single(X);

dt = 1;
t = [1:size(X, 2)];
%% DMD
X1 = X(:,1:end-1); X2 = X(:,2:end);
[U2,Sigma2,V2] = svd(X1, 'econ');

r = 3;
U = U2(:,1:r);
Sigma = Sigma2(1:r,1:r);
V=V2(:,1:r);

Atilde = U'*X2*V/Sigma;
[W,D] = eig(Atilde);
Phi = X2*V/Sigma*W;

mu = diag(D);
omega = log(mu)/dt;

figure(1)
plot(abs(omega), 'o')
title('absolute value of \omega')
xlabel('index')
ylabel('value')


u0 = X(:, 1);
y0 = Phi\u0;  % pseudo-inverse initial conditions

u_modes = zeros(r,length(t));  % DMD reconstruction for every time point
for iter = 1:length(t)
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi(:, 1)*u_modes(1, :);   % DMD resconstruction with all modes
```

Listing 1: MATLAB codes for reading data and DMD.

```
%% Make real-valued reconstruction
X_lowrank = abs(u_dmd);
X_sparse = X - X_lowrank;

% Set all positives of sparse to zero
X_sparse_new = min(X_sparse, zeros(size(X_sparse)));
R = X_sparse - X_sparse_new;
X_lowrank_new = X_lowrank + R;
X_reconstruct = X_lowrank_new + X_sparse_new;

%% show the foreground and background\
% show movies
% for i = 1:length(t)
%     image = abs(X_sparse_new(:, i));
%     imshow(uint8(reshape(image, [540, 960])))
% end

show_image_index = 100;

% foreground
figure(2)
image = abs(X_sparse_new(:, show_image_index));
imshow(uint8(reshape(image, [540, 960])))

% background
figure(3)
image = abs(X_lowrank_new(:, show_image_index));
imshow(uint8(reshape(image, [540, 960])))
```

Listing 2: MATLAB codes for separation of background and foreground.