AMATH 482 HW5: Dynamic Mode Decomposition on Video Clips

Tiffany Tang
Department of Applied Mathematics
University of Washington
leqitang@uw.edu

Abstract

In this paper, we look at two video clips that contain a foreground object each. Singular value decomposition (SVD) and dynamic mode decomposition (DMD) are implemented to separate the video stream to a foreground video and background video. The foreground video was captured by the DMD's approximate sparse reconstruction, and the background video was captured by the DMD's approximate low-rank reconstruction.

1 Introduction and Overview

Two video clips with moving objects are analyzed with the singular value decomposition (SVD) and dynamic mode decomposition (DMD). One video contains a man ski down a snow mountain, the other depicts the scenario of racing cars. The SVD is used for reducing the dimension of the videos by finding the highest modes. The DMD algorithm as a data-driven method then creates an approximate of the linear mapping that best iterate through the video data. The low-rank reconstruction of DMD could present the background of each video. With the subtraction of these low-rank reconstruction, we acquire the moving object in the foreground of the video clips.

2 Theoretical Background

Dynamic mode decomposition is a numerical procedure for extracting dynamical features from flow data. For a data X that have M columns of snapshot of the flow field, we consider a matrix

$$X_1^{M-1} = [x_1 \ x_2 \ x_3 \ \dots \ x_{M-1}]$$

Where x_j denotes a snapshot of the data at time t_j. Utilizing the modes of the Koopman operator, where we have

$$x_{j+1} = Ax_j$$

We rewrite the matrix as

$$X_1^{M-1} = [x_1 \ Ax_1 \ A^2x_1 \ \dots \ A^{M-2}x_1]$$

In this way, we relate the first M-1 snapshots to x_1 using the Koopman operator. Then we write the matrix

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

Where e_{M-1} is the vector with all eros except a 1 at the (M-1)st component. R is the residual vector that accounts for behaviors that cannot be described completely by A.

By SVD, we have $X_1^{M-1} = U\Sigma V^*$. Then

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

Since A is chosen to satisfy that the columns in X_2^M are the linear combinations of the columns of U, the residual vector r must be orthogonal to the basis, giving U*r=0. So we multiplying the above equation through by U* to get:

$$U^* X_2^M = U^* AU\Sigma V^*$$

Then, we multiplying by V and Σ^{-1} on the right to get:

$$U^* AU = U^* X_2^M V\Sigma^{-1} = \tilde{S}$$

$\tilde{S}$ and A are similar matrixes, so they have the same eigenvalues:

$$\tilde{S} y_k = \mu_k y_k$$

This gives the eigenvectors of A, called the DMD modes, by

$$\psi_k = U y_k$$

We expand in our eigenbasis to get

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t}) b \,, \omega_k = \ln(\mu_k)/\Delta t$$

b_k are the initial amplitude of each mode and the matrix Ψ contains the eigenvectors as its columns. We calculate the initial coefficient value b_k by considering initial snapshot x1 at time t1 = 0. We find the pseudo inverse to get our b: $b = \Psi^{\dagger} x_1$

Then we can use the DMD spectrum of frequencies to subtract background modes. We assume that ω_p, where $p \in \{1, 2, \dots, l\}$, satisfies $\|\omega_p\| \approx 0, and\ that\ \|\omega_j\| \forall j \neq p$ is bounded away from zero, giving us the following:

$$X_{DMD} = \underbrace{b_p \varphi_p e^{\omega_p t}}_{\text{Background Video}} + \underbrace{\sum_{j \neq p} b_j \varphi_j e^{\omega_j t}}_{\text{Foreground Video}}$$

Assuming that $X \in \mathbb{R}^{n*m}$, then a $X_{DMD} \in \mathbb{R}^{n*m}$. Then we calculate the DMD's approximate low-rank reconstruction according to

$$X_{DMD}^{Low-Rank} = b_p \varphi_p e^{\omega_p t}$$

Since that

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

Then the DMD's approximate sparse reconstruction,

$$X_{DMD}^{Sparse} = \sum_{j \neq p} b_j \varphi_j e^{\omega_j t}$$

Can be calculated with real-valued elements as follows

$$X_{DMD}^{Sparse} = X - \left| X_{DMD}^{Low-Rank} \right|$$

3 Algorithm Implementation and Development

We used `VideoReader()` to load the video clips and `read()` to read it. We have a dt by 1/Framerate of the loaded video, and a time vector by spacing from 0 to the length of the video in intervals of dt. We used `rgb2gray()` and `reshape()` to convert the video into grayscale and get the data matrix with `double` values for the videos. Each column of the matrix represented a grayscale image at a specific time. We separate the data matrix into X1 and X2, where X1 represented the snapshots of data from frame1 to the last -1 frame, while the second matrix represented the data from frame 2 to the last frame.

We took the singular value decomposition of X1 using `svd(X1)`. Singular value spectrum was plotted by acquiring the diagonal value of S diagonal matrix. The first two largest singular values were selected as the singular value for low-rank approximation. We then truncated u, s, and v matrices from svd to contain only first two rows and columns. We created the $\tilde{S}$ matrix which equals to `U_r'*X2*V_r*/Sigma_r.` We compute dthe eigenvalues and eigenvectors of $\tilde{S}$ with `eig()` command. Then we extracted the eigenvalues by `eig()` and acquired the continuous-time eigenvalues `omega` by taking log of eigenvalues and divided by the dt time intervals. Eigenvalues `omega` were plotted in a complex plane. We created our DMD modes as `Phi` using `X2*V_r/Sigma_r*eV_r`.

Then we calculated the DMD solution. We first calculated b value and set an empty `zeros` matrix of size `(2, length(t))`. We create a time dynamic matrix, which we have `b.*exp(omega*t(i))` at each time point i. We then multiplied `Phi` to get low-rank DMD approximation.

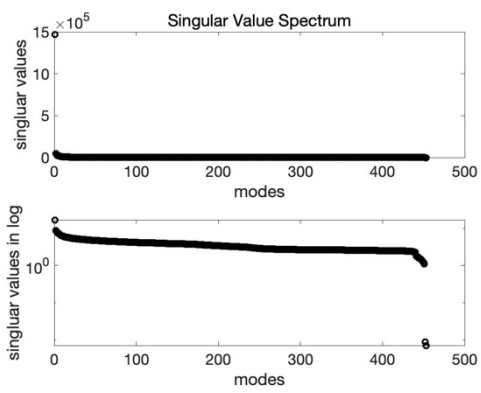
We subtracted the low-rank DMD approximation from the original data matrix X1 to get the sparse matrix approximations. R matrix which account for the residual negative values of sparse matrix is created. We subtracted R from the sparse matrix to get the filtered sparse matrix approximation and added R to the low-rank DMD matrix to get more accurate low-rank DMD approximation.

Lastly, we plotted the low-rank matrix, sparse matrix, low-rank with R matrix, and sparse without R matrix by picking a frame from the matrix. We reshaped our matrices and converted double into uint8 data type. Then we used `imshow()` to visualize them.

4 Computational Results

First for the video of a man skiing, we computed the singular value for X1. The first one mode has the largest value: $1.5*10^6$. We kept two singular value to calculate the low-rank approximation of DMD matrix. The two omega values are 0.0008 and -0.7047. Since the skiing man is wearing black outfit, the foreground video could not recognize the movement of the man. When we subtracted the residual negative vector R twice, we can see a write dot which is the skiing man in the frame 200.

In order to recognize the man clearer, we normalized the sparse matrix by rescaling all the data into 0 to 1. In this way, the skiing man are clearly showed at the center of the gray background.



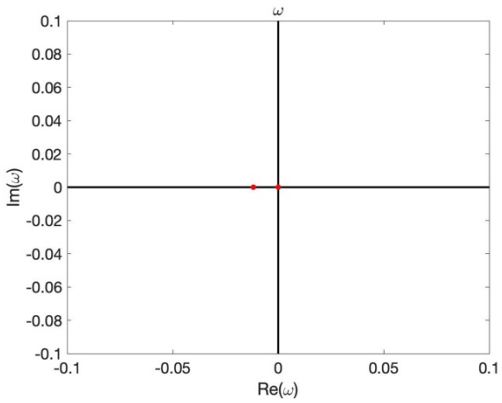


Figure 1(left): the singular value spectrum of the ski Video data.
Figure 2(right): the omega(continuous-value eigenvalues) of low-rank approximation for the ski Video in the complex plane.

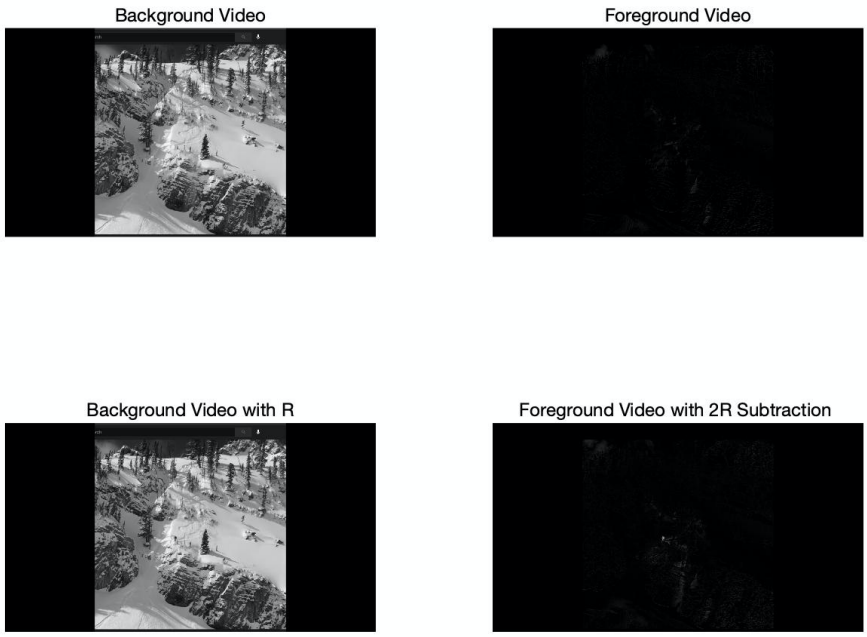


Figure3: Left top: frame 200 of the background video. Right top: frame 200 of the foreground video. Left bottom: frame 200 of the background video with R. Right bottom: frame 200 of the foreground video with 2R subtraction.

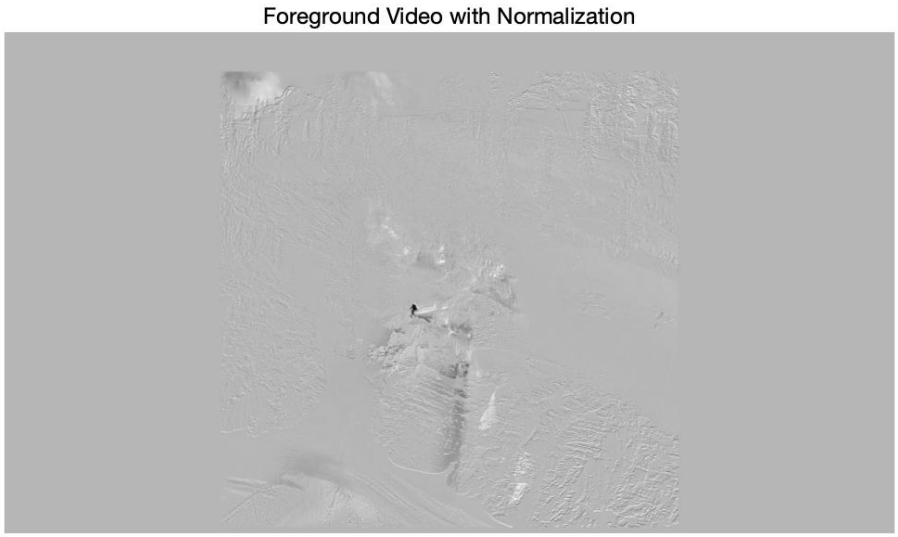


Figure 4: Frame 200 of the foreground video with normalization.

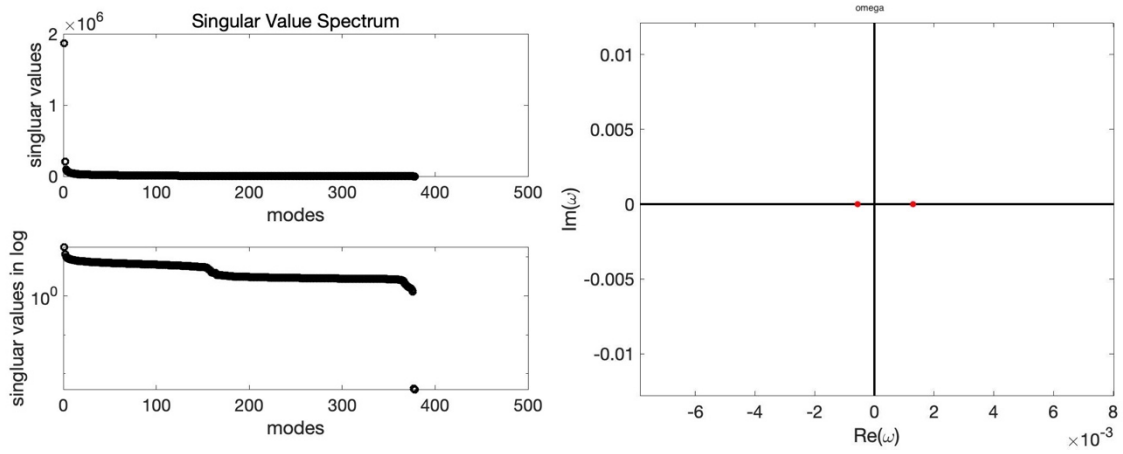


Figure 5(left): the singular value spectrum of the car racing Video data
Figure 6(right): the omega (continuous-value eigenvalues) of low-rank approximation for the car racing Video in the complex plane.

For the car racing video, we have two cars racing together. The first and second singular values of the matrix X1 are $1.9*10^6$and 0.2101. We had the first two singular values for calculating the low-rank approximation. The real and complex values of two omega are also showed in figure 6. These omega values are -0.0337 and 0.0778 without the imaginary component. The top part of figure 7 shows the background video and foreground video after DMD. When we add R matrix to background, the left bottom plot shows a worse presentation of background, which contains the car. However, with R matrix subtraction, the foreground did slightly better of visualizing the car in the foreground.

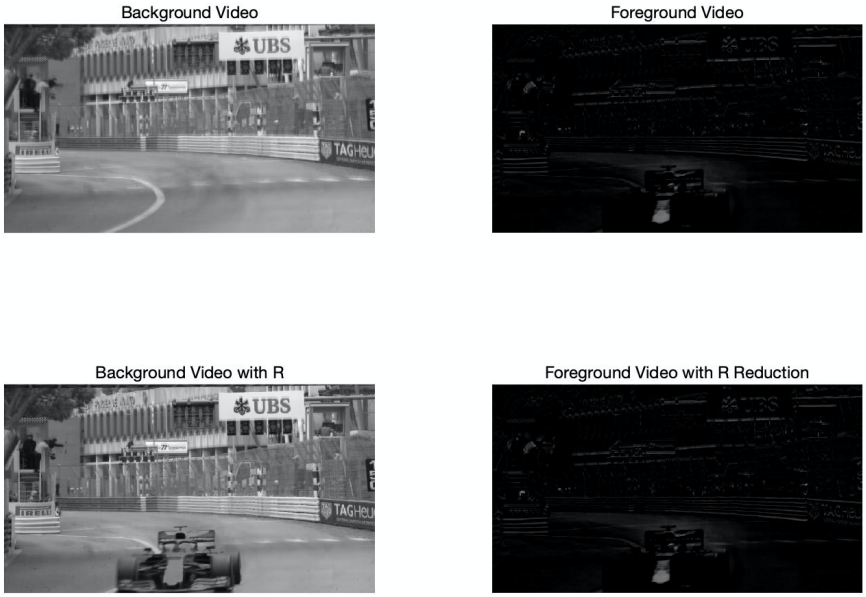


Figure7: Left top: frame 100 of the background video. Right top: frame 100 of the foreground video. Left bottom: frame 100 of the background video with R. Right bottom: frame 100 of the foreground video with R subtraction.

5 Summary and Conclusions

We performed the SVD and DMD algorithm to decomposed two video clips. Then we split a low-rank and sparse approximation to separate the background and foreground video from video clips. The separation was performed successfully in most parts. The DMD algorithm is a useful tool to decompose not only videos, but also images and signal data. For the efficiency of subtracting residual matrix R, it makes the object in the foreground have better recognition, but the addition of R makes the object appear in the background videos. Further investigation on the calculation and optimalization of the residual matrix R could be researched.

Appendix A. MATLAB Functions

1. [U, S, V] = svd(A) performs a singular value decomposition of matrix A.
2. D = diag(v) returns a square diagonal matrix with the element of vector v on the main diagonal.
3. e = eig(A) returns a column vector containing the eigenvalues of square matrix A.
4. I = rgb2gray(RGB) converts the truecolor image RGB to the grayscale image I.
5. v = VideoReader(filename) creates object v to read video data from the file named filename.
6. Y = imag(Z) returns the imaginary part of each element in array Z.
7. X = real(Z) returns the real part of each element in array Z.

Appendix B. Matlab Code

```
clear all; close all;clc;
%% ski_drop
vid1 = VideoReader('ski_drop_low.mp4');
dt = 1/vid1.Framerate;
t = 0:dt:vid1.Duration;
vidFrames = read(vid1);
numFrames = get(vid1, 'NumFrames');

for i = 1:numFrames
    v_mat2 = rgb2gray(vidFrames(:, :, :, i));
    v_reshape = reshape(v_mat2, [], 1);
    X(:, i) = double(v_reshape);
end

%% DMD
X1 = X(:, 1:end-1);
X2 = X(:, 2:end);

% SVD of X1 and Computation of ~S

[U, Sigma, V] = svd(X1,'econ');
r = 2;
U_r = U(:, 1:r);
Sigma_r = Sigma(1:r, 1:r);
V_r = V(:, 1:r);
S_r = U_r'*X2*V_r*diag(1./diag(Sigma_r));
[eV_r, D] = eig(S_r); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt; % continuous-time eval
Phi = X2*V_r/Sigma_r*eV_r;

%% singular value
figure(1)
subplot(2,1,1)
plot(diag(Sigma),'ko','Linewidth',2)
title('Singular Value Spectrum')
set(gca,'Fontsize',16,'Xlim',[0 500])
xlabel('modes')
ylabel('singluar values')
subplot(2,1,2)
semilogy(diag(Sigma),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 500])
```

```
xlabel('modes')
ylabel('singluar values in log')
%%  Plotting Eigenvalues (omega)
figure(2)
line = -10:10;

plot(zeros(length(line),1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(length(line),1),'k','Linewidth',2) % real axis
plot(real(omega)*dt,imag(omega)*dt,'r.','Markersize',15)
xlabel('Re(\omega)')
ylabel('Im(\omega)')
title('\omega')
set(gca,'FontSize',16,'Xlim',[-0.1 0.1],'Ylim',[-0.1 0.1])
%% DMD solution
b = Phi\X1(:,1);
u_modes = zeros(r, length(t));
t = (0:length(t)-1)*dt;
for i = 1:length(t)
    u_modes(:, i) = b.*exp(omega*t(i));
end
Xdmd = Phi * u_modes;
%% sparse DMD
Xsparse = X1- abs(Xdmd);

R = Xsparse.* (Xsparse<0);
X_lr = R+abs(Xdmd); % low dimension DMD with R
% makeit look better on imshow
X_sp = Xsparse-R-R; % Xsparse without R

X_sp1 = (Xsparse-min(Xsparse))./(max(Xsparse)- min(Xsparse));
%% display

v_back = uint8(reshape(Xdmd, 540, 960, 453));
v_fore = uint8(reshape(Xsparse, 540, 960, 453));
v_back_r = uint8(reshape(X_lr, 540, 960, 453));
v_fore_rreduction = uint8(reshape(X_sp, 540, 960, 453));
v_fore_nor = uint8(reshape(X_sp1, 540, 960, 453));
%% figure
figure(3)
subplot(2, 2, 1),
imshow(v_back(:, :, 200))

title('Background Video')
```

```
subplot(2, 2, 2),
imshow(v_fore(:, :, 200))
title('Foreground Video')
subplot(2, 2, 3),
imshow(v_back_r(:, :, 200))
title('Background Video with R')
subplot(2, 2, 4),
imshow(v_fore_rreduction(:, :, 200))
title('Foreground Video with 2R Subtraction ')
%% visualize normalized sparse matrix
figure (4)
sample = reshape(X_sp1(:, 200), 540, 960);
imshow(sample)
title('Foreground Video with Normalization')

clear all; close all;clc;
%% monte_carlo
vid1 = VideoReader('monte_carlo_low.mp4');
dt = 1/vid1.Framerate;
t = 0:dt:vid1.Duration;
vidFrames = read(vid1);
numFrames = get(vid1, 'NumFrames');
%

for i = 1:numFrames
    v_mat2 = rgb2gray(vidFrames(:, :, :, i));
    v_reshape = reshape(v_mat2, [], 1);
    X(:, i) = double(v_reshape);
end
%v_new = uint8(reshape(v_mat, 540, 960, 454));

%% DMD

X1 = X(:, 1:end-1);
X2 = X(:, 2:end);

%% SVD of X1 and Computation of ~S

[U, Sigma, V] = svd(X1,'econ');
r = 2;
U_r = U(:, 1:r);
Sigma_r = Sigma(1:r, 1:r);
V_r = V(:, 1:r);
S_r = U_r'*X2*V_r*diag(1./diag(Sigma_r));
```

```
[eV_r, D] = eig(S_r); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt; % continuous-time eval
Phi = X2*V_r/Sigma_r*eV_r;

%% singular value
figure(1)
subplot(2,1,1)
plot(diag(Sigma),'ko','Linewidth',2)
title('Singular Value Spectrum')
set(gca,'Fontsize',16,'Xlim',[0 500])
xlabel('modes')
ylabel('singluar values')
subplot(2,1,2)
semilogy(diag(Sigma),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 500])
xlabel('modes')
ylabel('singluar values in log')
%%  Plotting Eigenvalues (omega)
figure(1)
line = -1:1;

plot(zeros(length(line),1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(length(line),1),'k','Linewidth',2) % real axis
plot(real(omega)*dt,imag(omega)*dt,'r.','Markersize',15)
xlabel('Re(\omega)')
ylabel('Im(\omega)')
title('\omega', 'Fontsize', 24)
set(gca,'FontSize',16,'Xlim',[-0.1 0.1],'Ylim',[-0.1 0.1])
%% DMD solution
b = Phi\X1(:,1);
m = size(X1, 2);
u_modes = zeros(r,m);
t = (0:m-1)*dt;
for i = 1:m
    u_modes(:, i) = b.*exp(omega*t(i));
end
Xdmd = Phi * u_modes;
%% sparse and nonsparse DMD
Xsparse = X1- abs(Xdmd);

R = Xsparse.* (Xsparse<0);
X_lr = R+abs(Xdmd); % low dimension DMD with R
```

```
X_sp = Xsparse-R; % Xsparse without R
X_reconstruct = X_lr +X_sp;

%% display
back = uint8(reshape(Xdmd(:, 100), 540, 960));
fore = uint8(reshape(Xsparse(:, 100), 540, 960));
back_r = uint8(reshape(X_lr(:, 100), 540, 960));
fore_rr = uint8(reshape(X_sp(:, 100), 540, 960));
figure(3)
subplot(2, 2, 1),
imshow(back)
title('Background Video')
subplot(2,2,2),
imshow(fore)
title('Foreground Video')
subplot(2,2,3)
imshow(back_r)
title('Background Video with R')
subplot(2,2,4)
imshow(fore_rr)
title('Foreground Video with R Reduction')
```