# Background Subtraction in Video Streams

Yumin Yin

March 17, 2021

**Abstract**

We have two video clips that contain moving foreground objects and static background objects. Our goal is to apply the Dynamic Mode Decomposition (DMD) method on these two given video streams and separate them to both foreground and background videos.

# 1 Introduction and Overview

To apply the DMD method on two given video streams, we first create the evenly spaced time vector, the data matrix and its submatrices. We then compute the SVD of the submatrix $\mathbf{X}_1^{M-1}$ and apply the DMD standardly. After that, We calculate the DMD's approximate low-rank reconstruction, which is our background video, by using the $\psi$ and coefficient $b$ corresponding the $\omega$ that is close to zero. Then the DMD's approximate sparse reconstruction, the foreground video, can be calculated as the data matrix $\mathbf{X}$ subtracting the absolute value of the DMD's approximate low-rank reconstruction matrix. Lastly, we rescale the foreground matrix to the range from 0 to 1 while keeping all the information we have obtained to address the issue that negative intesiteis do not make sense.

# 2 Theoretical Background

## 2.1 The Koopman Operator

The Koopman operator $\mathbf{A}$ is a linear, time-independent operator such that, the Eq. (1):

$$\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j, \tag{1}$$

where the $j$ indicates the specific data collection time and $\mathbf{A}$ is the linear operator that maps the data from time $t_j$ to $t_{j+1}$. The vector $\mathbf{x}_j$ is an $N$-dimensional vector of the data points collect at time $j$. That is, applying $\mathbf{A}$ to a snapshot of data will advance it forward in time by $\Delta t$. This is crazy! We are asking for a linear mapping from timestep to the next, even though the dynamics of the system are likely nonlinear.

## 2.2 Dynamic Mode Decomposition

To construct the appropriate Koopman operator that best represents the data collected, we will consider the matrix, the Eq. (2)

$$\mathbf{X}_1^{M-1} = [\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3\ ...\ \mathbf{x}_{M-1}], \tag{2}$$

where we use the shorthand $\mathbf{x}_j$ to denote a snapshot of the data at time $t_j$. The Koopman operator allows us to rewrite this as the Eq. (3)

$$\mathbf{X}_1^{M-1} = [\mathbf{x}_1\ \mathbf{A}\mathbf{x}_1\ \mathbf{A}\mathbf{x}_1\ ...\ \mathbf{A}\mathbf{x}_1]. \tag{3}$$

The columns are formed by applying power of $\mathbf{A}$ to the vector $\mathbf{x}_1$, and are said to form the basis for the Krylov subspace. Hence, we have a way of relating the $M$ - 1 snapshots to $\mathbf{x}_1$ using just the Koopman operator/matrix. We can write the above in matrix form to get the Eq (4):

$$\mathbf{X}_2^{M} = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T, \tag{4}$$

where $e_{M-1}$ is the vector with all zeros except a 1 at the $(M-1)$st component. That is, $A$ applied to each column of $\mathbf{X}_1^{M-1}$, given by $\mathbf{x}_j$, maps to the corresponding column of $\mathbf{X}_2^{M}$, given by $\mathbf{x}_{j+1}$, but the final point $\mathbf{x}_M$ was not included in our Krylov basis, so we add in the *residual* (or error) vector $\mathbf{r}$ to account for this. Remember that $\mathbf{A}$ is unknown and it is our goal to find it. We should also remember that matrices are completely understood by their eigenvalues and eigenvectors, so we are going to circumvent finding $\mathbf{A}$ directly by finding another matrices with the same eigenvalues. First, let's use the SVD to write the Eq. (5):

$$\mathbf{X}_1^{M-1} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*. \tag{5}$$

Then, from above we get the Eq. (6)

$$\mathbf{X}_2^{M} = \mathbf{A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* + \mathbf{r}e_{M-1}^T, \tag{6}$$

We are going to choose $\mathbf{A}$ in such a way that the columns in $\mathbf{X}_2^{M}$ can be written as linear combinatinos of the columns of $\mathbf{U}$. This is the same as requiring that they can be written as linear combinations of the POD modes. Hence, the residual vector $\mathbf{r}$ must be orthogonal to the POD basis, giving the Eq. (7):

$$\mathbf{U}^*\mathbf{r} = 0. \tag{7}$$

Multiplying the above equation through by $\mathbf{U}^*$ on the left gives the Eq. (8):

$$\mathbf{U}^*\mathbf{X}_2^M = \mathbf{U}^*\mathbf{A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*. \tag{8}$$

Then, we can isolate for $\mathbf{U}^*\mathbf{A}\mathbf{U}$ by multiplying by $\mathbf{V}$ and the $\Sigma^{-1}$ on the right to get the Eq. (9)

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \underbrace{\mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\Sigma^{-1}}_{\tilde{\mathbf{S}}}. \tag{9}$$

Note that $\tilde{\mathbf{S}}$ and $\mathbf{A}$ are relate by applying matrix on one side and its inverse on the other. This means they are similar! They have the same eigenvalues! Furthermore, if $\mathbf{y}$ is eigenvector of $\tilde{\mathbf{S}}$, the $\mathbf{U}_y$ is an eigenvector of $\mathbf{A}$. So, let's write the eigenvector/eigenvalue pairs of $\tilde{\mathbf{S}}$ as the Eq. (10):

$$\tilde{\mathbf{S}}\mathbf{y}_k = \mu_k\mathbf{y}_k, \tag{10}$$

where K is the rank of $\mathbf{X}_1^{M-1}$. This equation gives the eigenvector of $\mathbf{A}$, called the **DMD modes**, by the Eq. (11):

$$\psi_k = \mathbf{U}\mathbf{y}_k, \tag{11}$$

Now we have got all we need to describe continual multiplications by $\mathbf{A}$! We can just expand in our eigenbasis to get the Eq. (12): **DMD modes**, by the Eq. (11):

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k\psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t})\mathbf{b} \tag{12}$$

We know that at time t=0, in the above formula we have to get $\mathbf{x}_1$ because this was our initial condition to generate the other $\mathbf{x}_m$. Therefore, taking t=0 in the above gives the Eq. (13):

$$mathbfx_1 = \Psi\mathbf{b} \Longrightarrow \mathbf{b} = \Psi^{-1}\mathbf{x}_1, \tag{13}$$

where $\Psi^{-1}$ is the pseudoinverse of the matrix $\Psi$.

## 2.3 Background Video and Foreground Video

Assume that $\omega_p$ satisfies $||\omega_p|| \approx 0$ and that $||\omega_j|| \neq 0$. This gives the Eq. (14)

$$\mathbf{X}_{DMD} = \underbrace{b_p\psi_p e^{\omega_p t}}_{\text{Background Video}} + \underbrace{\sum_{j\neq p}^{K} b_j\psi_j e^{\omega_j t}}_{\text{Foreground Video}} \tag{14}$$

Consider calculating the DMD's approximate low-rank reconstruction according to the Eq. (15):

$$\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} = b_p\psi_p e^{\omega_p t} \tag{15}$$

Since the Eq. (16) should be true:

$$\mathbf{X} = \mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} + \mathbf{X}_{\text{DMD}}^{\text{Sparse}}, \tag{16}$$

then the DMD's approximate sparse reconstruction, the Eq. (17):

$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \sum_{j\neq p}^{K} b_j\psi_j e^{\omega_j t}, \tag{17}$$

can be calculated as the Eq. (18):

$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \mathbf{X} - \mathbf{X}_{\text{DMD}}^{\text{Low-Rank}}, \tag{18}$$

## 3 Algorithm Implementation and Development

### 3.1 Preparing the data

Our time vector $t$, from beginning to end of the video, is evenly spaced in time by a fixed $\Delta t$, which is the number of seconds per video frame. For each frame in the video, we convert it to the grayscale image and the values in the image to double precision. We reshape each frame-representing matrix into column vector, add them together to create our data matrix $\mathbf{X}$. From this data matrix, we construct the two submatrices $\mathbf{X}_1^{M-1}$, snapshots of the data from the first frame to the second-last frame, and $\mathbf{X}_2^M$, snapshots of the data from the second frame to the last frame.

### 3.2 Computing the SVD and applying the DMD

We compute the SVD of $\mathbf{X}_1^{M-1}$ and plot the singular value spectrum to see how many modes are necessary for reconstruction. After that, we truncate our U, S, V matrices to the rank-$r$ to compute $\tilde{\mathbf{S}}$, and then we are able to find $\omega$ and $\psi$. Furthermore, we use the inital snapshot $\mathbf{x}_1$ and the pesudoinverse of $\psi$ to find the the coefficients $b_k$.

### 3.3 Reconstruction

We calculate the DMD's approximate low-rank reconstruction, which is our background video, by using the $\psi$ and coefficient $b$ corresponding the $\omega$ that is close to zero. Then the DMD's approximate sparse reconstruction, the foreground video, can be calculated as the data matrix $\mathbf{X}$ subtracting the absolute value of the DMD's approximate low-rank reconstruction matrix. There might be negative values, which would not make sense in terms of having negative pixel intensities. To handle this problem, we re-scale our foreground matrix to the range from 0 to 1 while keeping all the information we have obtained.

## 4 Computational Results

### 4.1 Monte Carlo

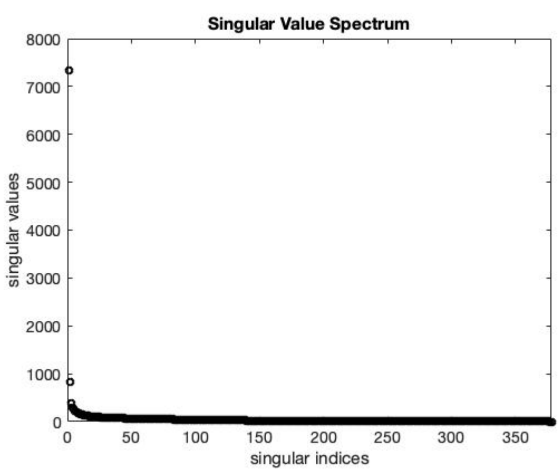


Figure 1: Singular value spectrum

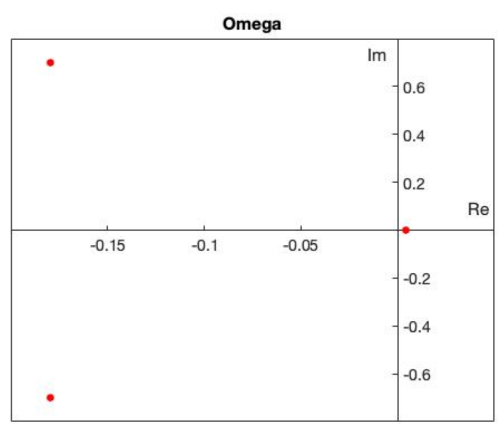


Figure 2: Omega values

We can tell from the Fig. (1) that 3 modes are necessary for reconstruction, and the Fig. (2) shows there is one omega value that is close to zero. This is our background mode and the rest are the foreground modes. The Fig (3), (4) and (5) are the sample frames of original, background and foreground videos respectively.


Figure 3: original video


Figure 4: background video


Figure 5: foreground video

### 4.2 Ski Drop

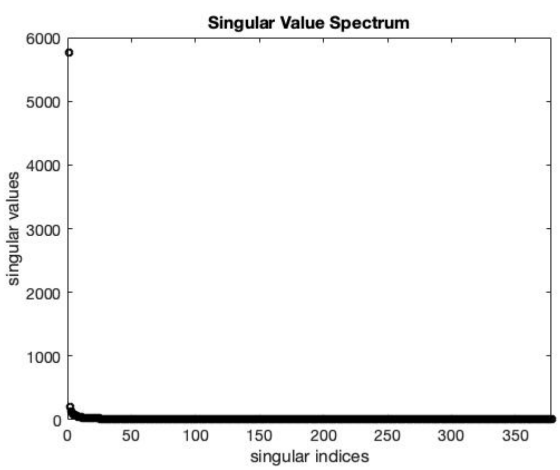


Figure 6: Singular value spectrum

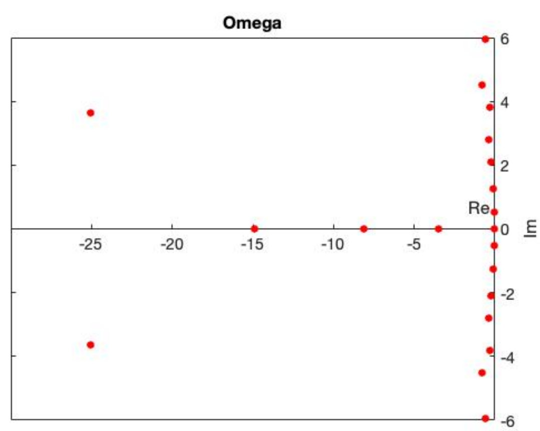


Figure 7: Omega values

We can tell from the Fig. (6) that 20 modes are necessary for reconstruction, and the Fig. (7) shows there is one omega value that is close to zero. This is our background mode and the rest are the foreground modes. The Fig (8), (9) and (10) are the sample frames of original, background and foreground videos respectively.

## 5 Summary and Conclusions

We have succeeded in applying the Dynamic Mode Decomposition (DMD) method on these two given video streams and separating them to both foreground and background videos. The DMD's approximate low-rank

Figure 8: original video

Figure 9: background video

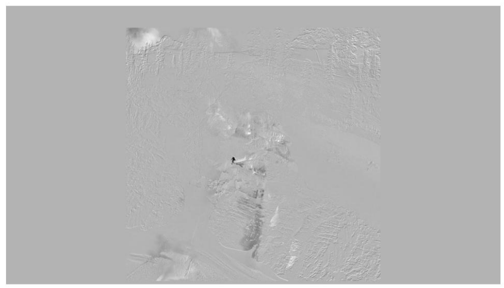

Figure 10: foreground video

reconstruction is calculated using the $\psi$ and coefficient $b$ corresponding the $\omega$ that is close to zero, and this is our background video. Subtracting the DMD's approximate low-rank reconstruction matrix from the original data matrix, we obtain the DMD's approximate sparse reconstruction, which is our foreground video.

## Appendix A MATLAB Functions

- `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.
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I.
- `I2 = im2double(I)` converts the image `I` to double precision.
- `B = reshape(A,sz1,...,szN)` reshapes `A` into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension.
- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of m-by-n matrix `A`.
- `x = diag(A)` returns a column vector of the main diagonal elements of `A`.
- `[V,D] = eig(A)` returns diagonal matrix `D` of eigenvalues and matrix `V` whose columns are the corresponding right eigenvectors, so that `A*V = V*D`.
- `L = length(X)` returns the length of the largest array dimension in `X`.
- `find (X == Y)` returns the indices where `X = Y`.
- `plot(X,Y)` creates a 2-D line plot of the data in `Y` versus the corresponding values in `X`.
- `Y = abs(X)` returns the absolute value of each element in array X.
- `M = max(A)` returns the maximum elements of an array.
- `M = min(A)` returns the minimum elements of an array.
- `imshow(I)` displays the grayscale image `I` in a figure.
- `X = real(Z)` returns the real part of each element in array `Z`.
- `X = imag(Z)` returns the imaginary part of each element in array `Z`.
- `Y = log(X)` returns the natural logarithm of each element in array `X`.
- `Y = exp(X)` returns the exponential for each element in array `X`. For complex elements, it returns the complex exponential.

## Appendix B MATLAB Code

```
clear all; close all; clc

monte_carlo = VideoReader('monte_carlo.mp4');
monte_carlo_frames = read(monte_carlo);

%%
dt = 1 / monte_carlo.FrameRate; % number of seconds per video frame
t = 0:dt:monte_carlo.Duration;
nFrames = monte_carlo.NumFrames;

for i = 1:nFrames
    I = rgb2gray(monte_carlo_frames(:,:,:,i));
    I = im2double(I);
    X(:,i) = reshape(I,[],1);;
end
```

```
%% Create DMD matrices

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

%% SVD of X1

[U, S, V] = svd(X1,'econ');

%% Plot the singular value spectrum

plot(diag(S),'ko','Linewidth',2)
title('Singular Value Spectrum')
set(gca,'Fontsize', 14, 'Xlim', [0 378])
xlabel('singular indices')
ylabel('singular values')

%% Apply DMD

r = 3;
U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);

Stilde = U_r' * X2 * V_r / S_r;
[eV, D] = eig(Stilde); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;

plot(real(omega), imag(omega), 'ro', 'MarkerFaceColor', 'r')
title('Omega')
set(gca, 'Fontsize', 14 , 'XAxisLocation', 'origin', 'YAxisLocation', 'origin')
xlabel('Re')
ylabel('Im')

%%

Phi = U_r * eV;
b = Phi \ X1(:,1);
ind = find(abs(omega) < 0.01);

%% reconstruction

for iter = 1:length(t)
    U_modes(:, iter) = b .* exp(omega * t(iter));
end
U_dmd = Phi(:,ind) * U_modes(ind,:);

video_background = U_dmd;
video_foreground = X - abs(U_dmd);
video_foreground = (video_foreground - min(video_foreground)) ./ (max(video_foreground) - min(video_foreground));
```

```
%% sample original video

sample_video_original = reshape(X(:, 20), 540, 960);
imshow(sample_video_original);

%% sample background

sample_video_background = reshape(video_background(:,20), 540, 960);
imshow((sample_video_background));
%% sample foreground

sample_video_foreground = reshape(video_foreground(:,20), 540, 960);
imshow((sample_video_foreground));
```

```
clear all; close all; clc

ski_drop = VideoReader('ski_drop.mp4');
ski_drop_frames = read(ski_drop);

%%

dt = 1 / ski_drop.FrameRate; % number of seconds per video frame
t = 0:dt:ski_drop.Duration;
nFrames = ski_drop.NumFrames;

for i = 1:nFrames
    I = rgb2gray(ski_drop_frames(:,:,:,i));
    I = im2double(I);
   X(:,i) = reshape(I,[],1);;
end

%% Create DMD matrices

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

%% SVD of X1

[U, S, V] = svd(X1,'econ');

%% Plot the singular value spectrum

plot(diag(S),'ko','Linewidth',2)
title('Singular Value Spectrum')
set(gca,'Fontsize', 14, 'Xlim', [0 378])
xlabel('singular indices')
ylabel('singular values')

%% Apply DMD

r = 20;
U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
```

```
V_r = V(:, 1:r);

Stilde = U_r' * X2 * V_r / S_r;
[eV, D] = eig(Stilde); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;

plot(real(omega), imag(omega), 'ro', 'MarkerFaceColor', 'r')
title('Omega')
set(gca, 'Fontsize', 14 , 'XAxisLocation', 'origin', 'YAxisLocation', 'origin')
xlabel('Re')
ylabel('Im')

%%

Phi = U_r * eV;
b = Phi \ X1(:,1);
ind = find(abs(omega) < 0.001);

%% reconstruction

U_modes = [];
for iter = 1:length(t)
    U_modes(:, iter) = b .* exp(omega * t(iter));
end
U_dmd = Phi(:,14) * U_modes(14,:);

%%
video_background = U_dmd;
video_foreground = X - abs(U_dmd);
video_foreground = (video_foreground - min(video_foreground)) ./ (max(video_foreground) - min(video_foreground));

%% sample original video

sample_video_original = reshape(X(:, 200), 540, 960);
imshow(sample_video_original);

%% sample background

sample_video_background = reshape(video_background(:,200), 540, 960);
imshow((sample_video_background));
%% sample foreground

sample_video_foreground = reshape(video_foreground(:,200), 540, 960);
imshow(sample_video_foreground);
```