Background Subtraction in Video Streams

Jiajun Bao

Abstract

This project aims to use the Dynamic Mode Decomposition method on the video clips **skidrop.mov** and **monte_carlo.mov** to separate the video stream to both the moving objects in the foreground video and a static background.

1 Introduction and Overview

In this project, we are provided with two video clips **skidrop.mov** and **monte_carlo.mov**. The first video is a person skiing from a snowy mountain. Our goal is to use the Dynamic Mode Decomposition method to separate the moving skier and the snowy mountain background. The second video is a short clip of formula racing, and similarly, we want to separate the waving flags and two moving racing cars from the static track and audience background. In order to achieve this, we first convert the video into a data matrix for computation purposes and find the Dynamic Mode Decomposition spectrum. Then we use it to subtract background modes from the original data matrix to get the sparse data matrix representing the foreground objects and reconstruct the video into the foreground video and background video.

2 Theoretical Background

2.1 Dynamic Mode Decomposition (DMD)

Dynamic Mode Decomposition (DMD) is a data-based algorithms for dimensionality reduction and takes advantage of low-dimensionality in experimental data without having to rely on a given set of governing equations to create a low-rank approximation. In here, the DMD spectrum of frequencies can be used to subtract background modes. Specifically, assume that ω_p, where $p \in 1, 2, ..., l$, satisfies $||\omega_p|| \approx 0$, and that $||\omega_j|| \forall j \neq p$ is bounded away from zero. Thus,

$$\mathbf{X}_{DMD} = b_p \varphi_p e^{w_p t} + \sum_{j \neq p} b_j \varphi_j e^{w_j t} \tag{1}$$

where the first term represent the background video and the second term represent foreground video. Assuming that $\mathbf{X} \in \mathrm{R}^{nxm}$, then a proper DMD reconstruction should also produce $X_{DMD} \in \mathrm{R}^{nxm}$. However, each term of the DMD reconstruction is complex: $b_j \varphi_j e^{w_j t} \forall j$ though they sum to a real-valued matrix. This poses a problem when separating the DMD terms into approximate low-rank and sparse reconstructions because real-valued outputs are desired and handling the complex elements can make a significant difference in the accuracy of the results. Consider calculating the DMD's approximate low-rank reconstruction according to

$$\mathbf{X}_{DMD}^{Low-Rank} = b_p \varphi_p e^{w_p t}, \mathbf{X}_{DMD} = \mathbf{X}_{DMD}^{Low-Rank} + \mathbf{X}_{DMD}^{Sparse}, then, \mathbf{X}_{DMD}^{Sparse} = \sum_{j \neq p} b_j \varphi_j e^{w_j t}, \tag{2}$$

then the DMD's approximate sparse reconstruction can be calculated with real-valued elements only as:

$$\mathbf{X}_{DMD}^{Sparse} = \mathbf{X} - |\mathbf{X}_{DMD}^{Low-Rank}| \tag{3}$$

where $|\cdot|$ yields the modulus of each element within the matrix. However, this may result in $\mathbf{X}_{DMD}^{Sparse}$ having negative values in some of its elements, which would not make sense in terms of having negative pixel intensities. These residual negative values can be put into a n×m matrix R and then be added back into $\mathbf{X}_{DMD}^{Low-Rank}$ as follows:

$$\mathbf{X}_{DMD}^{Low-Rank} \leftarrow \mathbf{R} + |\mathbf{X}_{DMD}^{Low-Rank}|, \tag{4}$$

$$\mathbf{X}_{DMD}^{Sparse} \leftarrow \mathbf{X}_{DMD}^{Sparse} - \mathbf{R} \tag{5}$$

This way the magnitudes of the complex values from the DMD reconstruction are accounted for, while maintaining the important constraints

$$\mathbf{X} = \mathbf{X}_{DMD}^{Low-Rank} + \mathbf{X}_{DMD}^{Sparse} \tag{6}$$

so that none of the pixel intensities are below zero, and ensuring that the approximate low-rank and sparse DMD reconstructions are real-valued. This is the method we will use in this project to get the foreground video and background video.

2.2 The Singular Value Decomposition

The Singular Value Decomposition (SVD) is a very powerful tool that is widely used in data analysis to produce low-rank approximations of a data set, which is described as following:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{7}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are unitary matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is diagonal. The values σ_n on the diagonal of Σ are the singular values of the matrix A. The vectors u_n which makes up the columns of U are the left singular vectors of A. The vectors v_n which make up the columns of V are the right singular vectors of A. The SVD tells us how to keep the most amount of information while keeping the fewest number of dimension of the data. The large singular values help us pick out which directions are most important.

3 Algorithm Implementation and Development

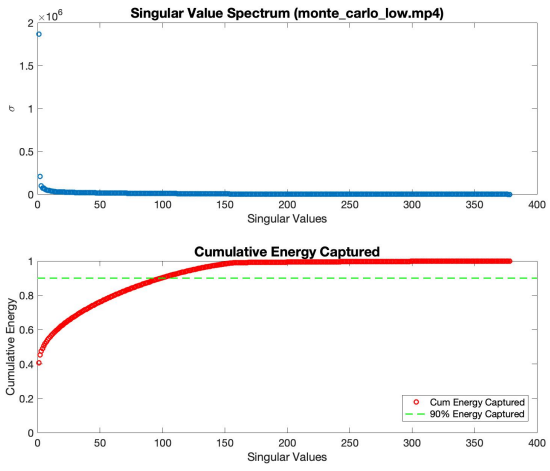
- Read from **monte_carlo_low.mp4** using functions *VideoReader* and *read* and convert to grey scale for computation later. Then reshape each image into a column vector and each column of the data matrix is a different image, then convert the data matrix to double for SVD analysis.
- For data matrix v1_mat, construct the two submatrices X1(from first to the second to the last frame) and X2(second to the last frame), and compute the SVD of X1.(*X1 = v1_mat(:,1:end-1); X2 = v1_mat(:,2:end)*)
- Plot singular value spectrum and the cumulative energy capture, where x coordinate is the singular values and y coordinate is the energy captured by each singular value and cumulative energy captured, then add the reference line representing 90%, and find numbers of modes needed(r=99) for capturing 90% energy.
- Truncate the u, s, and v matrices from the svd to contain only 1:r of its original row or column values, then use the rank-r approximation to compute $\widetilde{S}$ ($\widetilde{S}$ = *U_r1'*X2*V_r1*diag(1./diag(S_r1))*) and find its eigenvalues and eigenvectors. Plot the DMD eigenvalues(omega) to check if the real parts of omega are negative and verify the ideal number of modes to use. Then use the first frame to create DMD solution, which represent as X_dmd_low.
- Find the sparse reconstruction using the data matrix X1 and the low-rank reconstruction X_dmd_low, and compute R, which are negative values in some elements of sparse matrix
- Construct the foreground and background videos and plot the video images at chosen frame for reference, also plot the low-dimension reconstruction at chosen frame to see the effectiveness of reconstruction and compare it with the same frame from the original video clips in gray scale.
- Repeat the above steps for the vide **ski_drop_low.mp4** and the determine that we use r=57 for this case. The only additional step is that when plotting the foreground picture, due to the video is taken in a large scale with small moving target and hard to separate visually from the background, we also plot at a different color scale using *imagesc* to better examination the result of out separation.

4 Computational Results

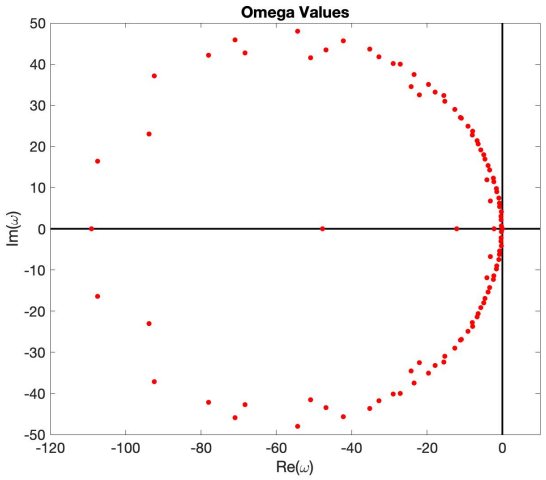
4.1 Video Clip1: monte_carlo_low.mp4

Based on the above algorithms in section 3, please refer to Appendix B for detailed codes in Matlab. Figure 1(a) top part shows the singular values of the data matrix X1, which is obtained from the the video clip data matrix v1_max, (*X1 = v1_mat(:,1:end-1)*). The cumulative energy captured is shown in the Figure 1(a) bottom. We decide to use 90% energy as the threshold for determining the numbers of modes, r = 99 is used for dimension reduction and low-rank approximation. Figure 1(b) top shows all the omega values(*omega = log(mu)/v1_dt*) that is log of eigenvalues of $\widetilde{S}$ divided by dt and Figure 1(c) shows the zoom-in version of omega values near the origin, which shows all the the real parts of omega are negative.

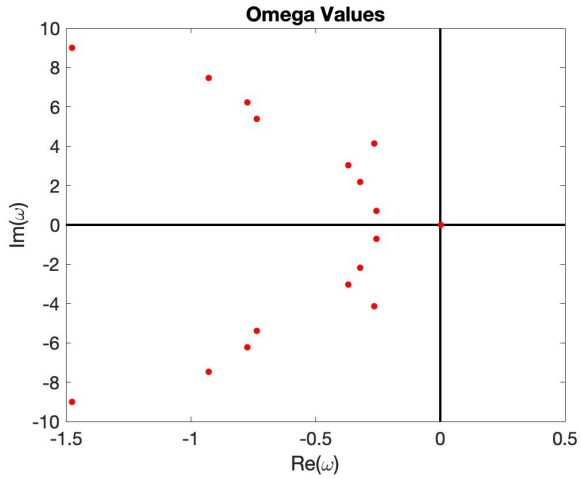
Figure 2 is a set of the image reconstruction and comparison based on the above steps in section 3, and these images are extracted from the original, foreground, background, and reconstructed video at a chosen frame for demonstration(we choose frame = 30 here, but could be any frame). Another set of comparisons at different time frames is included in Appendix A. Figure 2(a) is a snapshot of the original video in grayscale. Figure 2(b) is a foreground snapshot from the foreground video. As we see, the waving racing flag and two moving formula cars are well captured by the foreground video based on DMD's sparse reconstructions. Figure 2(c) is the background snapshot from the background video, and it shows the static background, such as the racing track and fence, based on DMD's low-rank reconstructions. Figure 2(d) is the DMD reconstruction.



(a) Singular Values and Energy Captured



(b) All Omega Values



(c) Zoom-in Omega Values

Figure 1: Video-monte_carlo_low: Singular and Omega Values

(a) Snapshot from Original Video

(b) Foreground Snapshot

(c) Background Snapshot

(d) Reconstruction Snapshot

Figure 2: Video-monte_carlo_low Images (frame=30)

4.2 Video Clip1: ski_drop_low.mp4

Based on the above algorithms in section 3, please refer to Appendix B for detailed codes in Matlab. Figure 3(a) top part shows the singular values of the data matrix X1. Similarly to the first video, X1 is obtained from the the second video clip data matrix v1_max, (*X1 = v1_mat(:,1:end-1)*). The cumulative energy captured is shown in the Figure 1(a) bottom. We decide to use 90% energy as the threshold for determining the numbers of modes, r = 57 is used for dimension reduction and low-rank approximation.

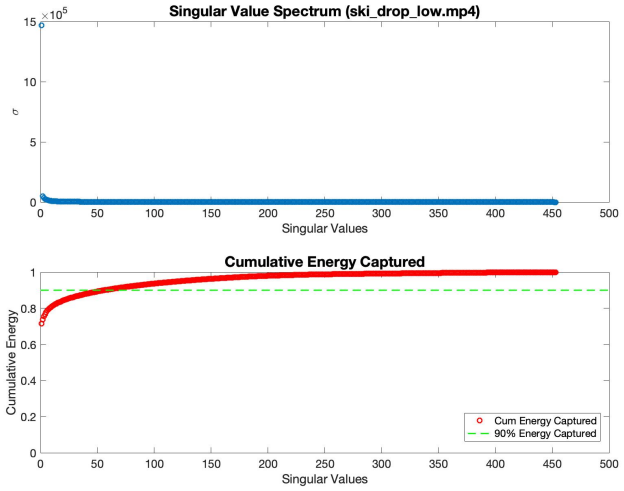
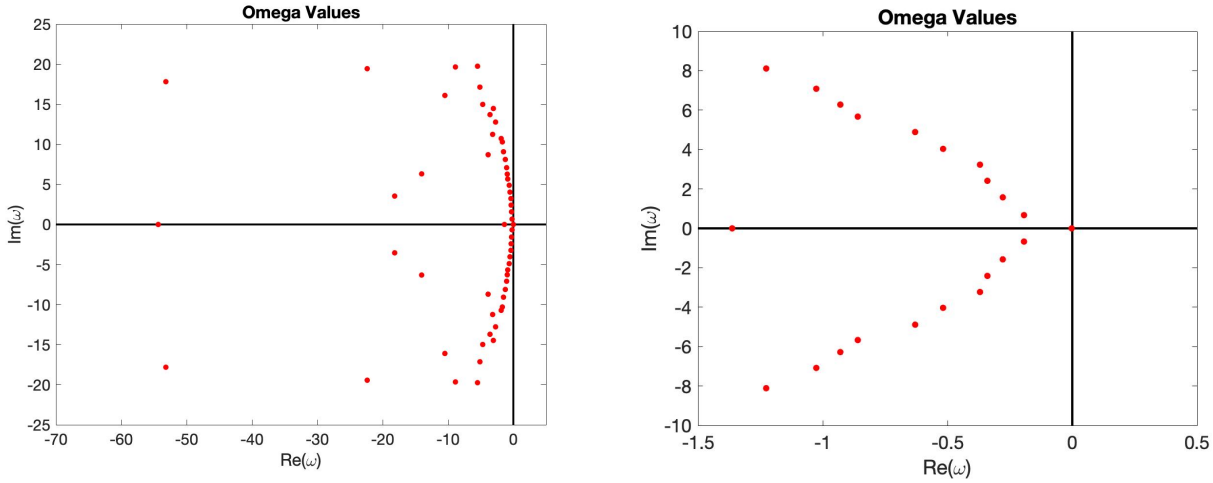


Figure 3: Video ski_drop_low: Singular Values and Energy Captured

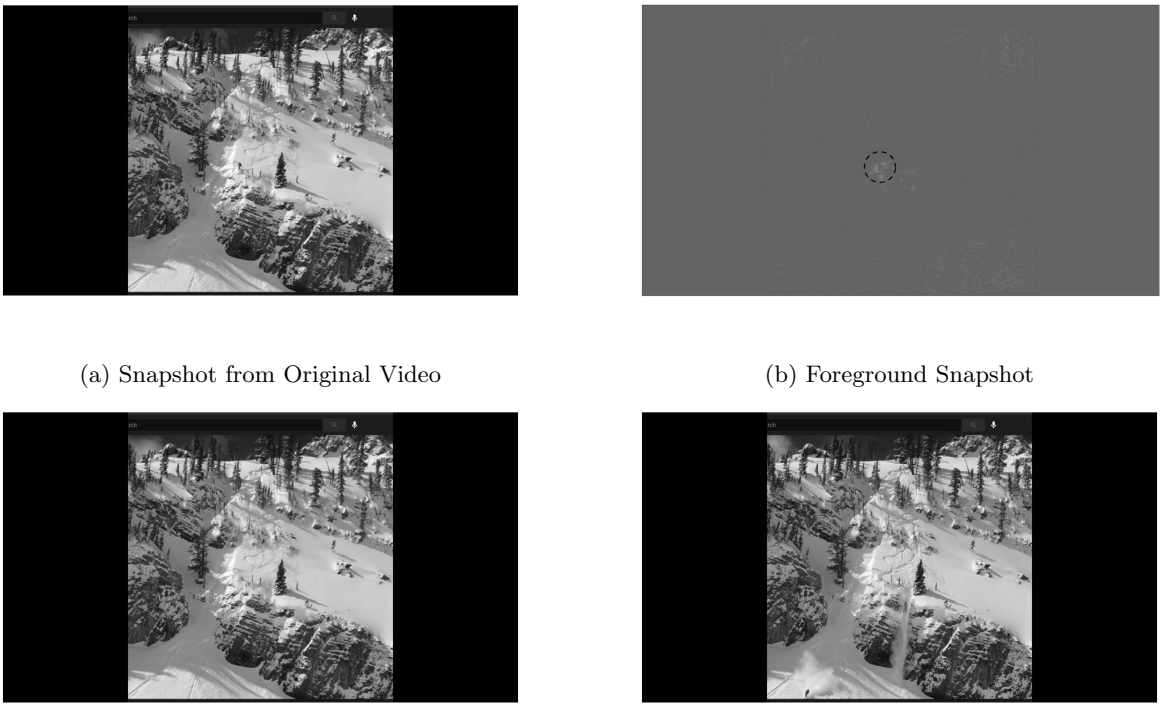


(a) All Omega Values

(b) Zoom-in Omega Values

Figure 4: Video ski_drop_low: Omega values

For the video ski_drop_low, Figure 4(a) top shows all the omega values(*omega = log(mu)/v1_dt*) that is log of eigenvalues of $\widetilde{S}$ ($\widetilde{S}$=*U_r1'*X2*V_r1*diag(1./diag(S_r1))*, r1 is rank-r approximation) divided by dt, and Figure 4(b) shows the zoom-in version of omega values near the origin, which shows all the the real parts of omega are negative.



(a) Snapshot from Original Video

(b) Foreground Snapshot

(c) Background Snapshot

(d) Reconstruction Snapshot

Figure 5: Video ski_drop_low: Images (frame=200)

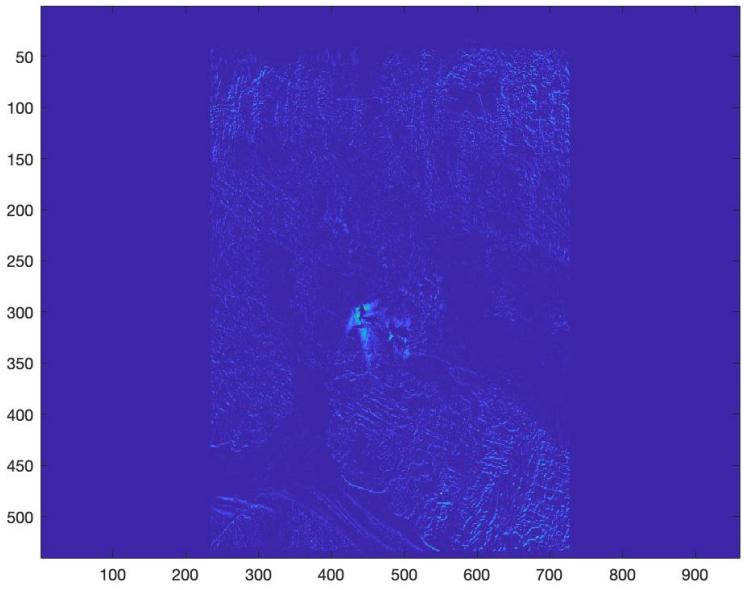


Figure 6: Foreground Snapshot with Different Color Scale

Similar to the first video processing, figure 5 is a set of the image reconstruction comparison based on the above steps in section 3. And these images are extracted from each original, foreground, background, and reconstructed video at a chosen frame for demonstration (we choose frame = 200 here, but could be any frame). Another set of comparisons at different time frames is included in Appendix A. Figure 5(a) is the snapshot of the original video in grayscale. Figure 5(b) is the foreground snapshot from the foreground video. As we see, the moving skier is captured by the foreground video based on DMD's sparse reconstructions. Since the moving object is relatively small and hard to see visually on grayscale, figure 6 shows the foreground moving object in a different color scale. Figure 5(c) is the background snapshot from the background video, and it shows the static background, such as snowy mountains and trees, based on DMD's low-rank reconstructions. Figure 5(d) is the DMD reconstruction.

5 Summary and Conclusions

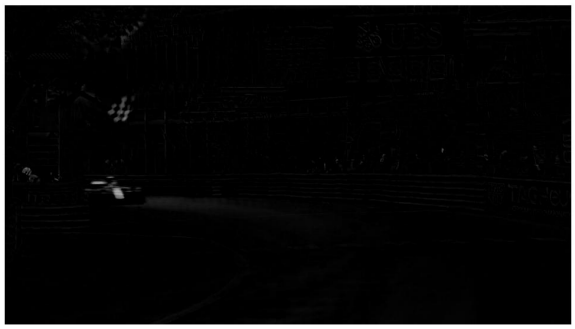
In this problem, for each sample video clip, we first convert the video images to a data matrix representing each frame. We perform the SVD analysis to know how much energy each node contained. We find the DMD's approximate low-rank reconstruction and DMD's approximate sparse reconstruction to reconstruct the background and foreground video. As we can see, the Dynamic Mode Decomposition method does a great job in terms of differentiating the moving objects in a video from the static background. Besides, since both of the cameras are very stable in the two videos, we don't have much noise in the two samples. If the camera is shaking intensely to introduce a lot of noise, the method may not perform as well as expected in the stable camera.

Appendix A MATLAB Functions + Extra Demo Figures

- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I.
- `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'.
- `[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.
- `Y = uint8(X)` converts the values in X to type uint8. Values outside the range [0,28-1] map to the nearest endpoint.
- `imshow(I)` displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.
- `imagesc(C)` displays the data in array C as an image that uses the full range of colors in the colormap. Each element of C specifies the color for one pixel of the image. The resulting image is an m-by-n grid of pixels where m is the number of rows and n is the number of columns in C. The row and column indices of the elements determine the centers of the corresponding pixels.

Extra Demonstration Figures:

(a) Snapshot from Original Video



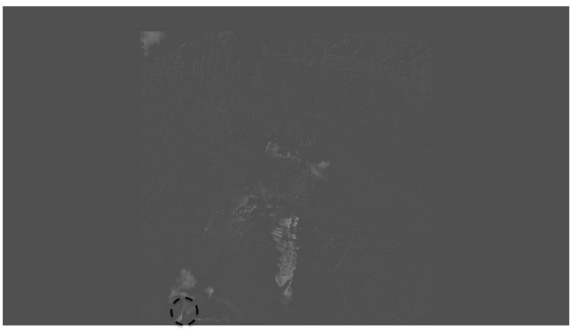
(b) Foreground Snapshot

(c) Background Snapshot

(d) Reconstruction Snapshot

Figure 7: Video-monte_carlo_low Images (frame=310)

(a) Snapshot from Original Video



(b) Foreground Snapshot

(c) Background Snapshot

(d) Reconstruction Snapshot

Figure 8: Video ski_drop_low: Images (frame=430)

Appendix B MATLAB Code

```
clear; close all; clc
% Create a VideoReader object for the first example movie file
v = VideoReader('monte_carlo_low.mp4');
v1_frames = read(v);
v1_total_frames = v.NumFrames;  % find total number of frames
% implay('monte_carlo_low.mp4')
v1 = rgb2gray(v1_frames(:,:,:,30));
imshow(v1)

%% Reshape the given data
for i = 1:v1_total_frames
    v1 = rgb2gray(v1_frames(:,:,:,i));  % convert to grey scale
    % imshow(v1)
    v1 = reshape(v1, [], 1);
    % converted to double precision for mathematical processing
    v1_mat(:,i) = double(v1);
end
```

```
%% DMD
X1 = v1_mat(:,1:end-1);
X2 = v1_mat(:,2:end);
[U1,Simga1,V1] = svd(X1, 'econ');

% Plot singular value spectrum
sig1 = diag(Simga1);
figure(1)
subplot(2,1,1)
plot(sig1,'o','Linewidth',1.5)
set(gca,'Fontsize',14)
title("Singular Value Spectrum (monte\_carlo\_low.mp4)","FontSize", 18);
xlabel("Singular Values"); ylabel("\sigma");
subplot(2,1,2)
plot(cumsum(sig1/sum(sig1)),'ro','Linewidth',1.5)
set(gca,'Fontsize',14, 'ylim',[0 1])
title("Cumulative Energy Captured", "FontSize", 18);
xlabel("Singular Values"); ylabel("Cumulative Energy");
hold on
plot([0 400], [0.9 0.9], '--g', 'Linewidth',1.5)
legend('Cum Energy Captured','90% Energy Captured')

%% DMD low-rank reconstructions
r = 99;
v1_total_time = v.Duration;
v1_dt = v1_total_time/v1_total_frames;
x1_t = (0:v1_total_frames-2)*v1_dt;

% Computation of ~S
U_r1 = U1(:, 1:r); % low-rank approximattion (rank-r)
S_r1 = Simga1(1:r, 1:r);
V_r1 = V1(:, 1:r);
S = U_r1' * X2 * V_r1 * diag(1./diag(S_r1));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/v1_dt;
Phi = U_r1*eV;

% Create DMD Solution
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(x1_t));
for iter = 1:length(x1_t)
   u_modes(:,iter) = y0.*exp(omega*x1_t(iter));
end
X_dmd_low = Phi*u_modes;
```

```
%% Plotting Eigenvalues (omega)
% make axis lines
line = -130:50;
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),imag(omega),'r.','Markersize',15)
xlabel('Re(\omega)')
ylabel('Im(\omega)')
title('Omega Values')
set(gca,'FontSize',16,'Xlim',[-120 10],'Ylim',[-50 50])

%%  Sparse reconstruction
X_sparse = X1 - abs(X_dmd_low);
check_neg = (X_sparse <0); % create logical index
R = X_sparse.*check_neg;
X_background = R + abs(X_dmd_low);
X_foreground = X_sparse - R;
X_reconstruct = X_background + X_foreground;

%% plot background
window_height = v.Height;
window_width = v.Width;
frame = length(x1_t);
X_dmd_low = reshape(X_dmd_low, [window_height, window_width, frame]);
X_dmd_low = uint8(X_dmd_low);
% for i = 1:frame
%     imshow(X_dmd_low(:,:,i))
% end
imshow(X_dmd_low(:,:,30))

%% plot foreground
foreground = reshape(X_foreground, [window_height, window_width, frame]);
foreground = uint8(foreground);
% for i = 1:frame
%     imshow(foreground(:,:,i))
% end
imshow(foreground(:,:,310))

%% plot reconstruction
reconstruct = reshape(X_reconstruct, [window_height, window_width, frame]);
reconstruct = uint8(reconstruct);
% for i = 1:100
%     imshow(reconstruct(:,:,i))
% end
imshow(reconstruct(:,:,310))
```

```
%% Second Vide Processing
clear; close all; clc
% Create a VideoReader object for the second example movie file
v = VideoReader('ski_drop_low.mp4');
v1_frames = read(v);
v1_total_frames = v.NumFrames;  % find total number of frames
% implay('monte_carlo_low.mp4')
v1 = rgb2gray(v1_frames(:,:,:,430));
%imshow(v1)

%% reshape the data
for i = 1:v1_total_frames
    v1 = rgb2gray(v1_frames(:,:,:,i));  % convert to grey scale
    % imshow(v1)
    v1 = reshape(v1, [], 1);
    % converted to double precision for mathematical processing
    v1_mat(:,i) = double(v1);
end

%% DMD
X1 = v1_mat(:,1:end-1);
X2 = v1_mat(:,2:end);
[U1,Simga1,V1] = svd(X1, 'econ');

% Plot singular value spectrum
sig1 = diag(Simga1);
figure(1)
subplot(2,1,1)
plot(sig1,'o','Linewidth',1.5)
set(gca,'Fontsize',14)
title("Singular Value Spectrum (ski\_drop\_low.mp4)","FontSize", 18);
xlabel("Singular Values"); ylabel("\sigma");
subplot(2,1,2)
plot(cumsum(sig1/sum(sig1)),'ro','Linewidth',1.5)
set(gca,'Fontsize',14, 'ylim',[0 1])
title("Cumulative Energy Captured", "FontSize", 18);
xlabel("Singular Values"); ylabel("Cumulative Energy");
hold on
plot([0 500], [0.9 0.9], '--g', 'Linewidth',1.5)
legend('Cum Energy Captured','90% Energy Captured')

%% DMD low-rank reconstructions
r = 57;
v1_total_time = v.Duration;
v1_dt = v1_total_time/v1_total_frames;
x1_t = (0:v1_total_frames-2)*v1_dt;
```

```
% Computation of ~S
U_r1 = U1(:, 1:r); % low-rank approximattion (rank-r)
S_r1 = Simga1(1:r, 1:r);
V_r1 = V1(:, 1:r);
S = U_r1' * X2 * V_r1 * diag(1./diag(S_r1));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/v1_dt;
Phi = U_r1*eV;

% Create DMD Solution
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(x1_t));
for iter = 1:length(x1_t)
   u_modes(:,iter) = y0.*exp(omega*x1_t(iter));
end
X_dmd_low = Phi*u_modes;

%%  Sparse reconstruction
X_sparse = X1 - abs(X_dmd_low);
check_neg = (X_sparse <0); % create logical index
R = X_sparse.*check_neg;
X_background = R + abs(X_dmd_low);
X_foreground = X_sparse - R;
X_foreground_more_grey = X_sparse - R + 100.*ones(518400,453);
X_reconstruct = X_background + X_foreground;

%% Plotting Eigenvalues (omega)
% make axis lines
line = -80:30;
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),imag(omega),'r.','Markersize',15)
xlabel('Re(\omega)')
ylabel('Im(\omega)')
title('Omega Values')
set(gca,'FontSize',16,'Xlim',[-70 5],'Ylim',[-25 25])

%% plot background
window_height = v.Height;
window_width = v.Width;
frame = length(x1_t);
X_dmd_low = reshape(X_dmd_low, [window_height, window_width, frame]);
X_dmd_low = uint8(X_dmd_low);
% for i = 1:100
%     imshow(X_dmd_low(:,:,i))
```

```
% end
imshow(X_dmd_low(:,:,430))

%% plot foreground
%plot original color scale
%foreground = reshape(X_foreground_more_grey, [window_height, window_width, frame]);
%foreground = uint8(foreground);
%imshow(foreground(:,:,430))

% plot using different color scale uses the full range of the colormap
foreground1 = reshape(X_sparse, [window_height, window_width, frame]);
foreground1 = uint8(foreground1);
% for i = 1:150
%     imshow(foreground(:,:,i))
% end
imagesc(foreground(:,:,200))
%% plot reconstruction
reconstruct = reshape(X_reconstruct, [window_height, window_width, frame]);
reconstruct = uint8(reconstruct);
% for i = 1:100
%     imshow(reconstruct(:,:,i))
% end
imshow(reconstruct(:,:,430))
```