AMATH 482 Homework 5

Haley Riggs

March 10, 2021

Abstract

Dynamic Mode Decomposition (DMD) is a data-based algorithm that breaks the time dynamics of a system up into exponential functions, allowing for a relatively straightforward way to represent a low-rank approximation of the data. In this report, DMD is used to separate videos into their foreground and background parts, which are then added together to create full approximations of the video data.

1 Introduction and Overview

In this assignment, low-dimensional approximations of videos of cars racing on a race track and a skier skiing down a mountain were created using the DMD algorithm. The videos were separated into their foreground and background parts, which were approximated and then added to create the DMD solution. In order to execute this, the singular value decomposition (SVD) of each video data matrix was computed and the rank of each set of data was determined. This number was utilized to create low-rank approximations of the data.

2 Theoretical Background

Dynamic Mode Decomposition (DMD) is a data-based algorithm that utilizes the low-dimensionality of empirical data without requiring a set of equations that govern the phenomena. DMD finds an optimal basis for the data in which time is expressed as exponential functions, thereby expressing time as oscillations, growth, and decay. DMD is also unique because it allows for the forecasting of data.

Assuming that the data evolves in both space and time, we can define two scalars, N and M where

$$N = \text{number of spatial points per unit of time} \tag{1}$$
$$M = \text{number of time points/snapshots} \tag{2}$$

For DMD to work properly, the time data needs to be gathered at evenly spaced intervals and therefore

$$t_{m+1} = t_m + \Delta t, \qquad m = 1, \dots, M-1, \qquad \Delta t > 0 \tag{3}$$

We can then create column vectors comprised of the data snapshot information called U for m values from 1 to M.

$$U(\mathbf{x}, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \vdots \\ U(x_n, t_m) \end{bmatrix} \tag{4}$$

These column vectors can then be used as columns of data matrices:

$$\mathbf{X} = \begin{bmatrix} U(\mathbf{x}, t_1) & U(\mathbf{x}, t_2) & \dots & U(\mathbf{x}, t_M) \end{bmatrix} \qquad \mathbf{X_j^k} = \begin{bmatrix} U(\mathbf{x}, t_j) & U(\mathbf{x}, t_{j+1}) & \dots & U(\mathbf{x}, t_k) \end{bmatrix} \tag{5}$$

where $\mathbf{X_j^k}$ is a matrix whose columns are only the snapshots j through k of the matrix $\mathbf{X}$, which contains all of the snapshots.

DMD creates approximations for the Koopman operator's modes. This Koopman operator $\mathbf{A}$ is linear and time-independent, satisfying

$$\mathbf{x_{j+1}} = \mathbf{A}\mathbf{x_j} \tag{6}$$

where j represents the time the data was collected, $\mathbf{A}$ maps the data from the current time step t_j to the next time step t_{j+1}, and $\mathbf{x_j}$ is an N-dimensional vector comprised of data at time j. Essentially, multiplying a data snapshot by the matrix $\mathbf{A}$ steps the data one step forward in time. This is a form of global linearization.

Now, use the notation $\mathbf{x_j}$ to represent a data snapshot corresponding to time t_j to create the matrix

$$\mathbf{X_1^{M-1}} = \begin{bmatrix} \mathbf{x_1} & \mathbf{x_2} & \dots & \mathbf{x_{M-1}} \end{bmatrix} \tag{7}$$

Using the Koopman operator, we can then rewrite this as

$$\mathbf{X_1^{M-1}} = \begin{bmatrix} \mathbf{x_1} & \mathbf{A}\mathbf{x_1} & \dots & \mathbf{A^{M-2}}\mathbf{x_1} \end{bmatrix} \tag{8}$$

The columns of this rewritten matrix $\mathbf{X_1^{M-1}}$ form the basis for what is known as the Krylov subspace. We may then rewrite this equation into the form

$$\mathbf{X_2^M} = \mathbf{A}\mathbf{X_1^{M-1}} + \mathbf{r}e_{M-1}^T \tag{9}$$

where e_{M-1} is a vector whose entries are all zeros except the $(M-1)$st entry, which is equal to one. This equation accounts for the fact that the final point, called $\mathbf{x_M}$, was not included in the definition for $\mathbf{X_1^{M-1}}$ as noted in Equation 7.

Remembering the definition of the SVD, we can write $\mathbf{X_1^{M-1}}$ as the product of matrices $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}^*$. Plugging this into Equation 9, we obtain

$$\mathbf{X_2^M} = \mathbf{A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* + \mathbf{r}e_{M-1}^T \tag{10}$$

We choose $\mathbf{A}$ to allow the columns of $\mathbf{X_2^M}$ to be written as linear combinations of $\mathbf{U}$'s columns or, in other words, linear combinations of the POD modes. This implies that $\mathbf{U} * \mathbf{r} = 0$, since $\mathbf{r}$ needs to be orthogonal to the POD basis. Using this knowledge, we can multiply Equation 10 by $\mathbf{U}^*$ on the left to obtain

$$\mathbf{U}^*\mathbf{X_2^M} = \mathbf{U}^*\mathbf{A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{11}$$

Multiplying by $\mathbf{V}$ and $\mathbf{\Sigma^{-1}}$ on the right yields

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X_2^M}\mathbf{V}\mathbf{\Sigma^{-1}} =: \tilde{\mathbf{S}} \tag{12}$$

Note that we are assuming that $\mathbf{\Sigma}$ is invertible. $\mathbf{\Sigma}$ can be made invertible by using a low-rank approximation of the data, taking off unnecessary singular values and making $\mathbf{\Sigma}$ square. We are thus assuming

$$\mathbf{U} \in \mathbb{C}^{N \times K}, \qquad \mathbf{\Sigma} \in \mathbb{R}^{K \times K}, \qquad \mathbf{V} \in \mathbb{C}^{M-1 \times K} \tag{13}$$

Noting that $\mathbf{A}$ and $\tilde{\mathbf{S}}$ are similar, we know that they share the same eigenvalues. Additionally, if $\mathbf{y}$ is an eigenvector of $\tilde{\mathbf{S}}$, we know that $\mathbf{U}\mathbf{y}$ an eigenvector of $\mathbf{A}$. We can represent the eigenvalue equation of $\tilde{\mathbf{S}}$ as

$$\tilde{\mathbf{S}}\mathbf{y_k} = \mu_k \mathbf{y_k} \tag{14}$$

and thus we know that the eigenvectors of $\mathbf{A}$, called DMD modes, are given by

$$\phi_k = \mathbf{U}\mathbf{y_k} \tag{15}$$

We can then find the DMD solution by creating an eigenvalue expansion, given by

$$\mathbf{x}_{\mathrm{DMD}}(t) = \sum_{k=1}^{K} b_k \phi_k \mathrm{e}^{\omega_k t} = \Phi \mathrm{diag}(\mathrm{e}^{\omega_k t})\mathbf{b} \tag{16}$$

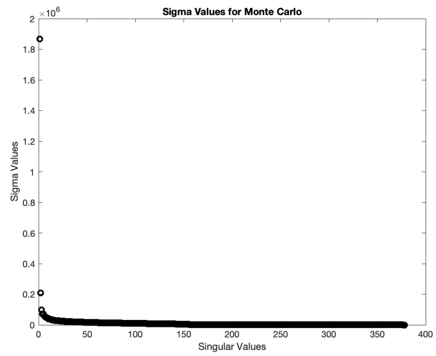

Figure 1: Here is a graph of the sigma values for the Monte Carlo video data.

where K is the rank of $\mathbf{X_1^{M-1}}$, b_k are the initial amplitudes, and Φ is a matrix with the eigenvectors ϕ_k as its columns. In order to put our DMD solution, as we did here, we express time dynamics as exponential functions, so

$$\omega_k = \ln(\mu_k)/\Delta t \tag{17}$$

We can then compute the b_k values by considering that Equation 16 evaluated at $t = 0$ is equal to $\mathbf{x_1}$, which is then equal to the matrix $\mathbf{\Phi}$ times $\mathbf{b}$. In equation form, if we let $\mathbf{\Phi}^{\ddagger}$ be the pseudoinverse of $\mathbf{\Phi}$, this is

$$\mathbf{x_1} = \mathbf{\Phi b} \implies \mathbf{b} = \mathbf{\Phi}^{\ddagger}\mathbf{x_1} \tag{18}$$

In simple terms, we can summarize the DMD algorithm as follows:

1. Sample data at N spatial locations M times, using a fixed Δt value to increment time. Create a matrix $\mathbf{X}$ containing these data snapshots.
2. Create submatrices $\mathbf{X_1^{M-1}}$ and $\mathbf{X_2^M}$ from $\mathbf{X}$.
3. Calculate the SVD of $\mathbf{X_1^{M-1}}$.
4. Compute matrix $\mathbf{tildeS} = \mathbf{U}^*\mathbf{X_2^M}\mathbf{V}\mathbf{\Sigma^{-1}}$ and calculate its eigenvalues and eigenvectors.
5. Calculate b_k using initial snapshot $\mathbf{x_1}$ (or really, anything) and the pseudoinverse of $\mathbf{\Phi}$.
6. Use the DMD modes and their projection to the initial conditions, as well as the calculated time dynamics, to create the solution at any future time.

While in some cases, the DMD algorithm creates an excellent approximation of future conditions, it'd not always perfect. It should be noted that eigenvalues ω_k with positive real part eventually "blow up," making the solution wildly inaccurate over long periods of time.

3 Algorithm Implementation and Development

In this assignment, data from a video of racing cars and from a video of a skier skiing down a mountain was collected and then ran through the DMD algorithm. The data was separated into parts pertaining to the foreground and background of the images. These two parts were then added together to create the full DMD solution.

First, the video data was loaded in using the function `VideoReader()` and then passed into `read()` to create a four-dimensional tensor for each video. The dimensions of the tensor represented height by width by color by time, which was the number of frames.

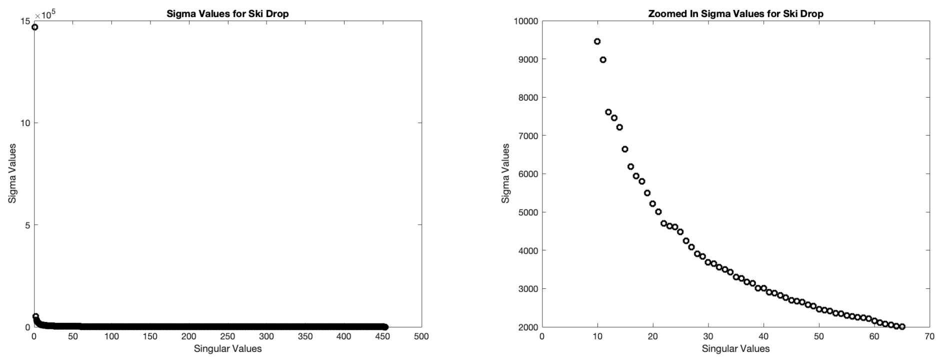

Figure 2: Here are two graphs of the sigma values for the ski drop video data. The graph on the left shows the entire spectrum of singular values and the graph on the right shows a zoomed-in view.

Then, the data matrices were created. The height, width, and number of frames for each matrix were saved, as well as an array of zeros whose dimensions were the product of the discovered height times width by the number of frames. A for loop was implemented, taking the data for each frame, converting it to grayscale and then to a double, and finally reshaping each into a column vector. These column vectors replaced the zeros of the data matrix, becoming its columns. This process was executed for each video's data. Afterwards, the time vectors, t, were defined to run from one to the number of frames per video in steps of one.

First working with the Monte Carlo data, the DMD matrices were formulated. Matrix $\mathbf{X}$ was set to be equal to the Monte Carlo data matrix, $\mathbf{X1}$ contained all of the columns except the last from $\mathbf{X}$ and $\mathbf{X2}$ contained all of the columns of $\mathbf{X}$ except the first. The SVD of $\mathbf{X1}$ was then calculated and the resulting singular values were plotted. Using a visual test, the rank of the data was determined to be approximately 25. Subsets of the $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{\Sigma}$ were taken (keeping only the first 25 columns from each) and, using these, the $\tilde{\mathbf{S}}$ matrix was created.

Next, the eigenvalues and eigenvectors of $\tilde{\mathbf{S}}$ were calculated. From this, the ω_k values were discovered and the $\mathbf{\Phi}$ matrix was created. At this point, a plot was made of the ω_k values. From the plot, it can be deduced that roughly one or two ω_k values have real part that is approximately equal to zero. These ω_k values correspond to the background of the video.

At this point, the indices of the ω_ks whose absolute values were close to zero were found. These indices were used to create a subset of the vector storing the ω_k values and of the $\mathbf{\Phi}$ matrix. Then, the pseudoinverse of the subset of $\mathbf{\Phi}$ matrix was calculated to find the initial conditions b_k.

It was then time to compute the DMD solution. A for loop filled the columns of a matrix of zeros with the element-wise product of the obtained initial conditions and the exponential of the product of the ω_k values and the value at the current index of the time vector. This matrix was then multiplied by the subset of the $\mathbf{\Phi}$ matrix on the left. This computed the DMD solution relating to the background.

In order to create the DMD solution for the foreground, the absolute value of the background DMD solution was subtracted from the data matrix $\mathbf{X}$, creating a foreground DMD solution matrix. Then, the indices of any negative values in this matrix were located and stored in a residual matrix. The final DMD solution for the background was created by adding the residual matrix to the absolute value of the previously derived DMD solution matrix for the background. The final foreground DMD solution was equal to the difference between the previously derived DMD solution for the foreground and the residual matrix. The full DMD solution was the sum of these two solutions.

The columns of the resulting full solution matrix were reshaped back into matrices of size height by width of the Monte Carlo data, transformed using `uint8()`, and displayed to create a video. Then, a figure was created relating the original video and DMD results at different frames. This entire process was repeated for the ski drop video. All of the code is included in Appendix B.

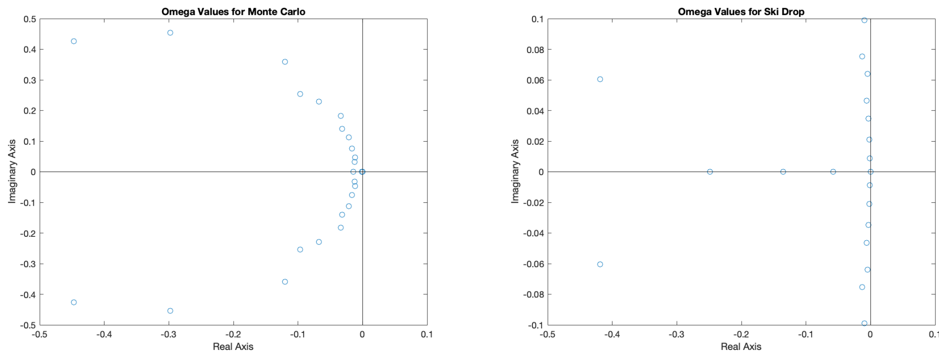

Figure 3: Here are two plots of the ω_k values for the Monte Carlo and the ski drop videos, respectively.

4 Computational Results

Before implementing the DMD algorithm on the two videos, low-rank approximations of the data matrices were computed. In order to do this, the singular values of each matrix were examined. In Figure 1, there is a visual depiction of the singular values of the data matrix for the Monte Carlo data. From this graph, it can be deduced that the rank appears to be about 25. Similarly, in Figure 2, the values of sigma for the ski drop video are depicted. It is difficult to confidently deduce the approximate rank using only the graph on the left, though it appears to be around 20. Looking to the graph on the right, which is a zoomed-in view of the data, a more apparent drop-off in value around singular value 20 confirms that this is a decent approximation for the rank.

Figure 3 shows the ω_k values for the two videos. As labeled, the left figure corresponds to the Monte Carlo data and the video on the right is for the ski drop video. Although it may be easier to see for the ski drop video, for each case, approximately one ω_k value was within the threshold for zero on the real axis. These ω_k values corresponded to the background of the videos.

After completing the DMD algorithm implementation, figures comparing frames from the original video and the DMD solution were created. Figure 4 shows the Monte Carlo video data and Figure 5 shows the results for the ski drop video. Amazingly, there appears to be no obvious difference between the original videos and full DMD solution.

Now we consider the background and foreground solutions. For the Monte Carlo video, shown in Figure 4, perhaps because they were in the first frame, the two race cars in the back were thought to be part of the background of the entire video by the DMD algorithm. As a result, the foreground images work to essentially cancel out the images of these cars, shown as the dominant white spots in the foreground images. When added to the background images, the cars vanish.

In Figure 5, it can be observed that almost all of the image is considered to be the background. This is likely because only a small amount of the video moves at any given time, namely the skier and on occasion, some snow. These elements are instead considered to be part of the foreground. Examining Frame 370, in both the original video and in the DMD solution, a stream of snow is seen to be falling towards the center of the frame. This snow is thought to be part of the foreground by the DMD algorithm, so it does not appear in the corresponding background image. Instead, it is added in by the foreground image.

5 Summary and Conclusions

In this assignment, videos of cars racing and a skier skiing were loaded into MATLAB. The DMD algorithm was implemented, creating a low-rank approximation of the videos. In order to do this, the videos were separated into their foreground and background parts through the examination of the eigenvalues ω_k resulting from the algorithm. These solutions were added together to create the full approximation. As evidenced in this report, the DMD algorithm can have exceptional results. To the naked eye, no difference may be observed between the DMD solutions and the original videos.

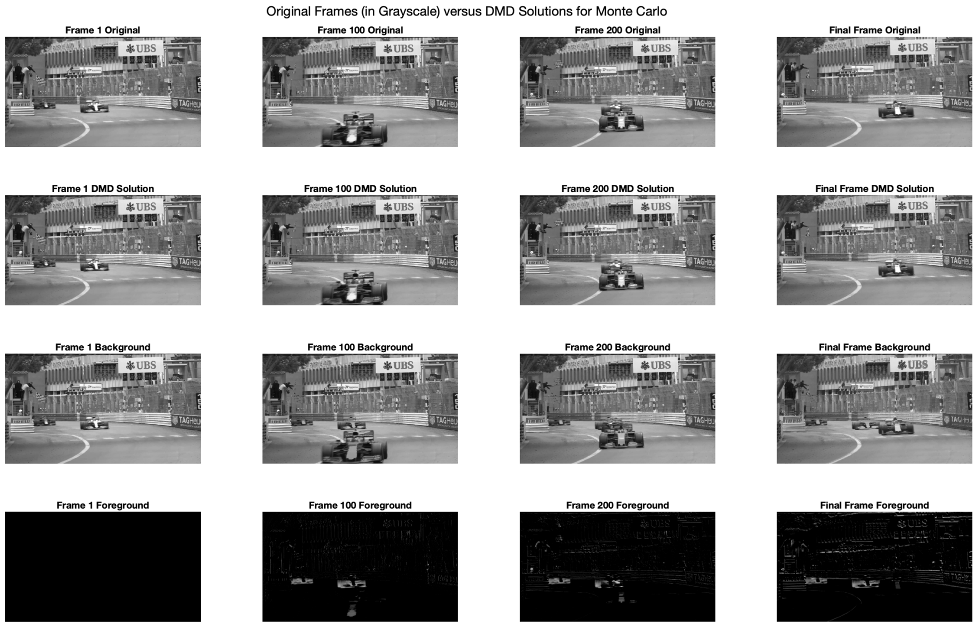

Figure 4: Here is a depiction of the DMD results compared to the original Monte Carlo video.

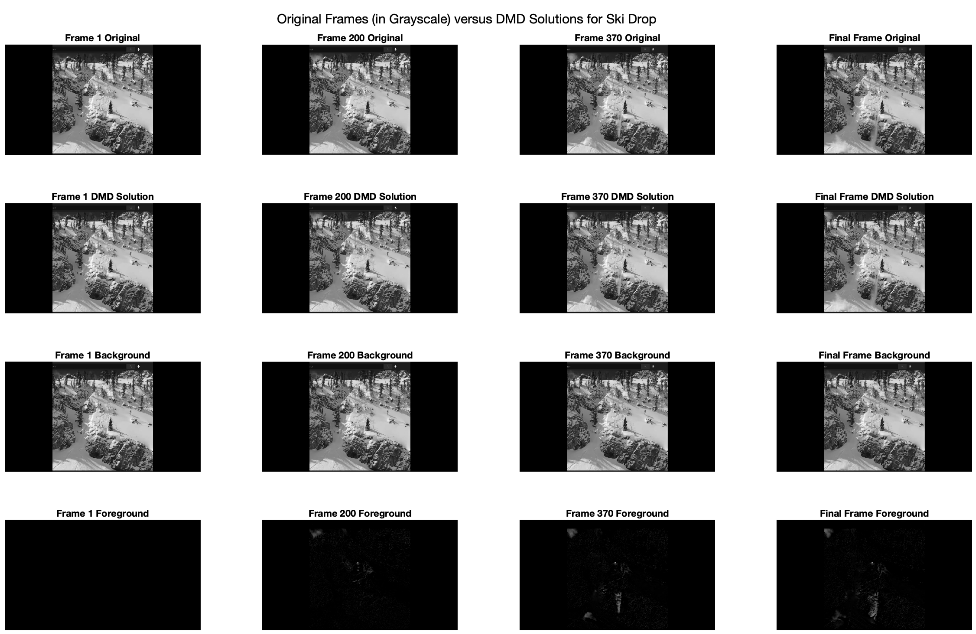

Figure 5: Here is a depiction of the DMD results compared to the original ski drop video.

Appendix A MATLAB Functions

- `video = read(v)` accepts a `VideoReader` object and outputs an associated tensor of video data.
- `v = VideoReader(filename)` read the data from `filename` and creates a corresponding `VideoReader` object.

Appendix B MATLAB Code

```
clear; close all; clc

%% Load data
vMC = VideoReader('monte_carlo_low.mp4');
vS = VideoReader('ski_drop_low.mp4');

videoMC = read(vMC);
videoS = read(vS);
% video is height by width by color by time/numFrames

%% Create data matrices
[hMC,wMC,numFramesMC] = size(videoMC,1,2,4);
dataMatMC = zeros(hMC*wMC,numFramesMC);
for j = 1:numFramesMC
    frame = double(rgb2gray(videoMC(:,:,:,j))); % format to double
    frame = reshape(frame,[],1); % reshape to column vector
    dataMatMC(:,j) = frame;
end

[hS,wS,numFramesS] = size(videoS,1,2,4);
dataMatS = zeros(hS*wS,numFramesS);
for j = 1:numFramesS
    frame = double(rgb2gray(videoS(:,:,:,j))); % format to double
    frame = reshape(frame,[],1); % reshape to column vector
    dataMatS(:,j) = frame;
end

% Define time vectors
dt = 1;
tMC = 1:dt:numFramesMC;
tS = 1:dt:numFramesS;

%% Create DMD matrices for Monte Carlo video and compute SVD of X1
XMC = dataMatMC;
X1MC = XMC(:,1:end-1);
X2MC = XMC(:,2:end);
[UMC, SigmaMC, VMC] = svd(X1MC,'econ');

%% Create plot of sigma values
plot(diag(SigmaMC),'ko','Linewidth',2)
title('Sigma Values for Monte Carlo')
xlabel('Singular Values')
ylabel('Sigma Values')

%% Calculate S tilde matrix
```

```
rankMC = 25; % visually deduced
UMCr = UMC(:,1:rankMC);
VMCr = VMC(:,1:rankMC);
SigmaMCr = SigmaMC(:,1:rankMC);
SMC = UMCr'*X2MC*VMCr*diag(1./diag(SigmaMCr));

%% Calculate eigenvalues and eigenvectors
[eVMC, DMC] = eig(SMC);
muMC = diag(DMC);
omegaMC = log(muMC)/dt;
phiMC = UMCr*eVMC;

%% Create plot of omega values
figure
plot(real(omegaMC),imag(omegaMC),'o')
hold on
plot([0 0], [-0.5 0.5], 'k')
plot([-0.5 0.1], [0 0], 'k')
title('Omega Values for Monte Carlo')
xlabel('Real Axis')
ylabel('Imaginary Axis')

%% Separate foreground and background omega values
thresh = 0.001;
idxOmMC = find(abs(omegaMC) < thresh); % find omegas close to zero in abs val
% Create subset of omega and phi vectors
omegaSubMC = omegaMC(idxOmMC);
phiSubMC = phiMC(idxOmMC);

%% Create DMD Solution
% Find background DMD Solution
y0MC = phiSubMC\X1MC(:,1); % pseudoinverse to obtain initial conditions

uModesMC = zeros(length(y0MC),numFramesMC);
for k = 1:numFramesMC
    uModesMC(:,k) = y0MC.*exp(omegaSubMC*tMC(k));
end
uDMDbackMC = phiSubMC*uModesMC;

% Find foreground DMD solution
uDMDforeMC = XMC-abs(uDMDbackMC);
idxResMC = find(uDMDforeMC < 0);
resMC = zeros(size(uDMDforeMC));
resMC(idxResMC) = uDMDforeMC(idxResMC);
uDMDnewBackMC = resMC+abs(uDMDbackMC);
uDMDnewForeMC = uDMDforeMC-resMC;

% Create full DMD solution
uDMDMC = uDMDnewBackMC + uDMDnewForeMC;

%% Create video of results for Monte Carlo
figure
for j = 1:numFramesMC
    reshaped = reshape(uDMDMC(:,j), [hMC,wMC]);
```

```
    imshow(uint8(reshaped))
    drawnow
end

%% Create figure comparing frames from original video and DMD results
figure

subplot(4,4,1)
reshaped = reshape(dataMatMC(:,1), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 1 Original')

subplot(4,4,2)
reshaped = reshape(dataMatMC(:,100), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 100 Original')

subplot(4,4,3)
reshaped = reshape(dataMatMC(:,200), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 200 Original')

subplot(4,4,4)
reshaped = reshape(dataMatMC(:,numFramesMC), [hMC,wMC]);
imshow(uint8(reshaped))
title('Final Frame Original')

subplot(4,4,5)
reshaped = reshape(uDMDMC(:,1), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 1 DMD Solution')

subplot(4,4,6)
reshaped = reshape(uDMDMC(:,100), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 100 DMD Solution')

subplot(4,4,7)
reshaped = reshape(uDMDMC(:,200), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 200 DMD Solution')

subplot(4,4,8)
reshaped = reshape(uDMDMC(:,numFramesMC), [hMC,wMC]);
imshow(uint8(reshaped))
title('Final Frame DMD Solution')

subplot(4,4,9)
reshaped = reshape(uDMDnewBackMC(:,1), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 1 Background')

subplot(4,4,10)
reshaped = reshape(uDMDnewBackMC(:,100), [hMC,wMC]);
```

```
imshow(uint8(reshaped))
title('Frame 100 Background')

subplot(4,4,11)
reshaped = reshape(uDMDnewBackMC(:,200), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 200 Background')

subplot(4,4,12)
reshaped = reshape(uDMDnewBackMC(:,numFramesMC), [hMC,wMC]);
imshow(uint8(reshaped))
title('Final Frame Background')

subplot(4,4,13)
reshaped = reshape(uDMDnewForeMC(:,1), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 1 Foreground')

subplot(4,4,14)
reshaped = reshape(uDMDnewForeMC(:,100), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 100 Foreground')

subplot(4,4,15)
reshaped = reshape(uDMDnewForeMC(:,200), [hMC,wMC]);
imshow(uint8(reshaped))
title('Frame 200 Foreground')

subplot(4,4,16)
reshaped = reshape(uDMDnewForeMC(:,numFramesMC), [hMC,wMC]);
imshow(uint8(reshaped))
title('Final Frame Foreground')

sgtitle('Original Frames (in Grayscale) versus DMD Solutions for Monte Carlo')

%% Create DMD matrices for ski drop video and compute SVD of X1
XS = dataMatS;
X1S = XS(:,1:end-1);
X2S = XS(:,2:end);
[US, SigmaS, VS] = svd(X1S,'econ');

%% Create plot of sigma values
figure
plot(diag(SigmaS),'ko','Linewidth',2)
title('Sigma Values for Ski Drop')
xlabel('Singular Values')
ylabel('Sigma Values')

%% Create zoomed-in plot of sigma values
figure
plot(diag(SigmaS),'ko','Linewidth',2)
title('Zoomed In Sigma Values for Ski Drop')
xlabel('Singular Values')
ylabel('Sigma Values')
```

```
ylim([2000 10000])

%% Calculate S tilde matrix
rankS = 20; % visually deduced
USr = US(:,1:rankS);
VSr = VS(:,1:rankS);
SigmaSr = SigmaS(:,1:rankS);
SS = USr'*X2S*VSr*diag(1./diag(SigmaSr));

%% Calculate eigenvalues and eigenvectors
[eVS, DS] = eig(SS);
muS = diag(DS);
omegaS = log(muS)/dt;
phiS = USr*eVS;

%% Create plot of omega values
figure
plot(real(omegaS),imag(omegaS),'o')
hold on
plot([0 0], [-0.1 0.1], 'k')
plot([-0.5 0.1], [0 0], 'k')
title('Omega Values for Ski Drop')
xlabel('Real Axis')
ylabel('Imaginary Axis')

%% Separate foreground and background omega values
thresh = 0.001;
idxOmS = find(abs(omegaS) < thresh); % find omegas close to zero in abs val
% Create subset of omega and phi vectors
omegaSubS = omegaS(idxOmS);
phiSubS = phiS(idxOmS);

%% Create DMD Solution
% Find background DMD Solution
y0S = phiSubS\X1S(:,1); % pseudoinverse to obtain initial conditions

uModesS = zeros(length(y0S),numFramesS);
for k = 1:numFramesS
   uModesS(:,k) = y0S.*exp(omegaSubS*tS(k));
end
uDMDbackS = phiSubS*uModesS;

% Find foreground DMD solution
uDMDforeS = XS-abs(uDMDbackS);
idxResS = find(uDMDforeS < 0);
resS = zeros(size(uDMDforeS));
resS(idxResS) = uDMDforeS(idxResS);
uDMDnewBackS = resS+abs(uDMDbackS);
uDMDnewForeS = uDMDforeS-resS;

% Create full DMD solution
uDMDS = uDMDnewBackS + uDMDnewForeS;

%% Create video of results for ski drop
```

```
figure
for j = 1:numFramesS
    reshaped = reshape(uDMDS(:,j), [hS,wS]);
    imshow(uint8(reshaped))
    drawnow
end

%% Create figure comparing frames from original video and DMD results
figure

subplot(4,4,1)
reshaped = reshape(dataMatS(:,1), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 1 Original')

subplot(4,4,2)
reshaped = reshape(dataMatS(:,200), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 200 Original')

subplot(4,4,3)
reshaped = reshape(dataMatS(:,370), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 370 Original')

subplot(4,4,4)
reshaped = reshape(dataMatS(:,numFramesS), [hS,wS]);
imshow(uint8(reshaped))
title('Final Frame Original')

subplot(4,4,5)
reshaped = reshape(uDMDS(:,1), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 1 DMD Solution')

subplot(4,4,6)
reshaped = reshape(uDMDS(:,200), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 200 DMD Solution')

subplot(4,4,7)
reshaped = reshape(uDMDS(:,370), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 370 DMD Solution')

subplot(4,4,8)
reshaped = reshape(uDMDS(:,numFramesS), [hS,wS]);
imshow(uint8(reshaped))
title('Final Frame DMD Solution')

subplot(4,4,9)
reshaped = reshape(uDMDnewBackS(:,1), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 1 Background')
```

```
subplot(4,4,10)
reshaped = reshape(uDMDnewBackS(:,200), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 200 Background')

subplot(4,4,11)
reshaped = reshape(uDMDnewBackS(:,370), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 370 Background')

subplot(4,4,12)
reshaped = reshape(uDMDnewBackS(:,numFramesS), [hS,wS]);
imshow(uint8(reshaped))
title('Final Frame Background')

subplot(4,4,13)
reshaped = reshape(uDMDnewForeS(:,1), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 1 Foreground')

subplot(4,4,14)
reshaped = reshape(uDMDnewForeS(:,200), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 200 Foreground')

subplot(4,4,15)
reshaped = reshape(uDMDnewForeS(:,370), [hS,wS]);
imshow(uint8(reshaped))
title('Frame 370 Foreground')

subplot(4,4,16)
reshaped = reshape(uDMDnewForeS(:,numFramesS), [hS,wS]);
imshow(uint8(reshaped))
title('Final Frame Foreground')

sgtitle('Original Frames (in Grayscale) versus DMD Solutions for Ski Drop')
```

Listing 1: This is the code used for implementing the DMD algorithm on the two videos and creating the associated figures.