# AMATH 482 Homework 5

Tsz Wai Tsui

March 17, 2021

Abstract

Given two video clips, we aim to separate the video stream to both the foreground video and a background using Dynamic Mode Decomposition (DMD). With the help of Singular Value Decomposition (SVD), we are able to separate the background and foreground object and reconstruct original frames with fewer modes.

## 1 Introduction and Overview

The first video clip shows 6 seconds of car racing, so the background contains the track and the buildings around, and the foreground contains the moving cars. The second clip shows 8 seconds of a person skiing, so the background contains the snow mountain, and the foreground contains the skiing person. We use SVD to find the number of modes enough for us to make good reconstruction of the videos. Then, we apply DMD on the dimension reduced matrix to separate the background and foreground object.

## 2 Theoretical background

Singular Value Decomposition (SVD is a way of factoring matrices:

$$\mathbf{A} = \mathbf{U\Sigma V}^*, \tag{1}$$

where matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{U} \in \mathbb{R}^{m\times m}$ 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 $\sigma_n$ on the diagonal of $\mathbf{\Sigma}$ are called the singular values of the matrix $\mathbf{A}$. The vectors $u_n$ which make up the columns of $\mathbf{U}$ are called the left singular vectors of $\mathbf{A}$. The vectors $v_n$ which make up the columns of $\mathbf{V}$ are called the right singular vectors of $\mathbf{A}$.

The aim of DMD is to take advantage of low-dimensionality in experimental data without having to rely on a given set of governing equations. To start with, we need snapshots of spatio-temporal data, which are the video frames in this came. We will define:

$$N = number\ of\ spatial\ points\ saved\ per\ unit\ time\ snapshot$$
$$M = number\ of\ snapshots\ taken.$$

The most important thing is that the time data is collected at regular spaced intervals:

$$t_{m+1} = t_m + \Delta t, \qquad m = 1, \dots, M-1, \qquad \Delta t > 0. \tag{2}$$

The snapshots are denoted

$$U(x, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \vdots \\ U(x_n, t_m) \end{bmatrix} \tag{3}$$

for each $m = 1, \dots, M$. We can use these snapshots to form columns of data matrices

$$\mathbf{X} = [U(x, t_1) \quad U(x, t_2) \quad \cdots \quad U(x, t_M)], \tag{4}$$

and

$$\mathbf{X}_{\mathrm{j}}^{\mathrm{k}} = \begin{bmatrix} U(x,t_j) & U(x,t_{j+1}) & \cdots & U(x,t_k) \end{bmatrix}. \tag{5}$$

The matrix $\mathbf{X}_{\mathrm{j}}^{\mathrm{k}}$ is just columns $j$ through $k$ of the full snapshot matrix $\mathbf{X}$.

The DMD method approximates the modes of the Koopman operator $\mathbf{A}$, which is a linear, time-independent operator such that

$$\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_{\mathbf{j}}, \tag{6}$$

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 $\Delta t$.
To construct the appropriate Koopman operator that best represents the data collected, we will consider the matrix:

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{M-1} \end{bmatrix}, \tag{7}$$

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

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{A}\mathbf{x}_1 & \mathbf{A^2}\mathbf{x}_1 & \cdots & \mathbf{A}^{M-2}\mathbf{x}_1 \end{bmatrix}. \tag{8}$$

The columns are formed by applying powers of $\mathbf{A}$ to the vector $\mathbf{x}_1$, and are said to form the basis for the Krylow subspace. Hence, we have a way of relating the first $M-1$ snapshots to $\mathbf{x}_1$ using just the Koopman matrix. We can write the above in matrix form to get

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_{\mathbf{1}}^{\mathbf{M-1}} + \mathbf{r}e_{M-1}^t, \tag{9}$$

where $e_{M-1}$ is the vector with all zeros except a 1 at the $(M-1)$st component. That is, applied $\mathbf{A}$ 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 $\mathbf{x}_M$ wasn't 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. We will see that the eigenvectors come easily from there.

First, we use the SVD to write $\mathbf{X}_1^{M-1} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$. Then, from above we get

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* + \mathbf{r}e_{M-1}^t. \tag{10}$$

We are going to choose $\mathbf{A}$ in such a way that the columns in $\mathbf{X}_2^M$ can be written as linear combinations of the columns of $\mathbf{U}$. The residual vector $\mathbf{r}$ must be orthogonal to $\mathbf{U}$, giving that $\mathbf{U}^*\mathbf{r} = \mathbf{0}$. Multiplying the above equation through by $\mathbf{U}^*$ on the left gives:

$$\mathbf{U}^*\mathbf{X}_2^M = \mathbf{U}^*\mathbf{A}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*. \tag{11}$$

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

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\mathbf{\Sigma^{-1}}. \tag{12}$$

Since everything on the right side is known from the input data, we can refer it as $\tilde{\mathbf{S}}$ to keep with the convention in the literature.

The goal a lot of the time with these data-driven method is dimensionality reduction. Therefore, we don't have to use the full matrix $\mathbf{\Sigma}$, we can use $\mathbf{\Sigma}$ with just the first $K \geq 1$ nonzero singular values on the diagonal. From what we know about singular values, $K$ is the rank of the data matrix $\mathbf{X}_1^{M-1}$, which essentially tells us the number of dimensions that the data varies in. This means that $\mathbf{U} \in \mathbb{C}^{N\times K}$, $\mathbf{\Sigma} \in \mathbb{R}^{K\times K}$, $\mathbf{V} \in \mathbb{C}^{M-1\times K}$.

Notice that $\tilde{\mathbf{S}}$ and $\mathbf{A}$are related by applying a matrix on one side and its inverse on the other. This means they are similar. Similar matrices have share a lot of properties, including that they have the same eigenvalues. Furthermore, if $\mathbf{y}$ is an eigenvector of $\tilde{\mathbf{S}}$, then $\mathbf{U}y$ is an eigenvector of $\mathbf{A}$. So, let's write the eigenvector/eigenvalue pairs of $\tilde{\mathbf{S}}$ as

$$\tilde{\mathbf{S}}y_k = \mu_k y_k, \tag{13}$$

thus giving the eigenvectors of $\mathbf{A}$, called the DMD modes, by

$$\psi_k = \mathbf{U}y_k. \tag{14}$$

Now, we can expand in our eigenbasis to get

$$\mathrm{x}_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k \, e^{\omega_k t} = \psi \mathrm{diag}(e^{\omega_k t})\mathbf{b}. \tag{15}$$

where $K$ is the rank of $\mathbf{X}_1^{M-1}$. The $b_k$ are the initial amplitude of each mode, the matrix $\psi$ contains the eigenvectors $\psi_k$ as its columns, and $\omega_k = \ln(\mu_k)\,/\Delta t$.

At last, we just take the linear dynamics governed by $\mathbf{A}$, compute the eigenvalues and eigenvectors, and then write the dynamics in terms of a linear combination of exponential growth/decay/oscillation in the direction of each eigenvector. 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

$$\mathrm{x}_1 = \psi \mathrm{b} \rightarrow \mathrm{b} = \psi^{\dagger} \mathrm{x}_1, \tag{16}$$

where $\psi^{\dagger}$ is the pseudoinverse of the matrix $\psi$.

The DMD spectrum of frequencies can be used to subtract background modes. Specifically, assume that $\omega_p$, where$p \in \{1, 2, \ldots, l\}$, satisfies $\|\omega_p\| \approx 0$, and that $\|\omega_j\| \; \forall \; j \neq p$ is bounded away from zero. Thus,

$$\mathbf{X}_{\mathrm{DMD}} = b_p \varphi_p e^{\omega_p t} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t}. \tag{17}$$

Assuming that $\mathbf{X} \in \mathbb{R}^{n \times m}$, then a proper DMD reconstruction should also produce $\mathbf{X}_{\mathrm{DMD}} \in \mathbb{R}^{n \times m}$. However, each term of the DMD reconstruction is complex: $b_j \varphi_j \exp(\omega_j \mathbf{t}) \in \mathbb{C}^{n \times m} \forall \, j$, 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 knowing how to handle 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

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low-Rank}} = b_p \varphi_p e^{\omega_p t}. \tag{18}$$

Since it should be true that

$$\mathbf{X} = \mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low-Rank}} + \mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}}, \tag{19}$$

Then the DMD's approximate sparse reconstruction,

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}} = \sum_{j \neq p} b_j \varphi_j e^{\omega_j t}, \tag{20}$$

can be calculated with read-valued elements only as follows

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}} = \mathbf{X} - \left|\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low-Rank}}\right|, \tag{21}$$

where $|\cdot|$ yields the modulus of each element within the matrix. However, this may results in $\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}}$ having negative values can be put into a $n \times m$ matrix $\mathbf{R}$ and then be added back into $\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low-Rank}}$ as follows:

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low-Rank}} \leftarrow \mathbf{R} + \left|\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low-Rank}}\right| \tag{22}$$

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}} \leftarrow \mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}} - \mathbf{R} \tag{23}$$

This way the magnitudes of the complex values from the DMD reconstruction are accounted for, while maintaining the important constraints of equation 19 so that none of the pixel intensities are below zero, and ensuring that the approximate low-rank and sparse DMD reconstructions are real-valued.

# 3 Algorithm Implementation and development

For two videos, we performed exact same steps. First, we use `VideoReader` to load in the video and define the number of frames, dt and t. Then, we construct the data matrix $\mathbf{X}$ stated in equation 4 and convert it from RBG to grayscale. From the data matrix $\mathbf{X}$, we construct the two submatrices $\mathbf{X}_1^{M-1}$ and $\mathbf{X}_2^{M}$. After that, we compute the SVD of $\mathbf{X}_1^{M-1}$. To determine the number of modes we need, we plot the energy captured against each singular value. We create the matrix $\tilde{\mathbf{S}} = \mathbf{U}^{*}\mathbf{X}_2^{M}\mathbf{V}\boldsymbol{\Sigma}^{-1}$ with the number of modes we found and find its eigenvalues and eigenvectors. Note that we use the eigenvalues stored in `mu` to convert to the `omega` vector, which are just the eigenvalues in another form. The reason for the different form is that we want to make the DMD solution look more like a solution to a continuous time ODE. Therefore, the Phi vector contains the eigenvectors of the matrix $\mathbf{A}$, which we obtain by multiplying the eigenvectors of $\tilde{\mathbf{S}}$ against $\mathbf{U}$. In a for loop, we use the initial snapshot $\mathbf{x}_1$ and the pseudoinverse of $\psi$ to find the coefficients of $b_k$. At last, we compute the low rank and sparse DMD matrix, which represent the background and foreground object respectively. The addition of two reconstructs the original matrix with fewer modes.

# 4 Computational Results

## 4.1 Car racing video

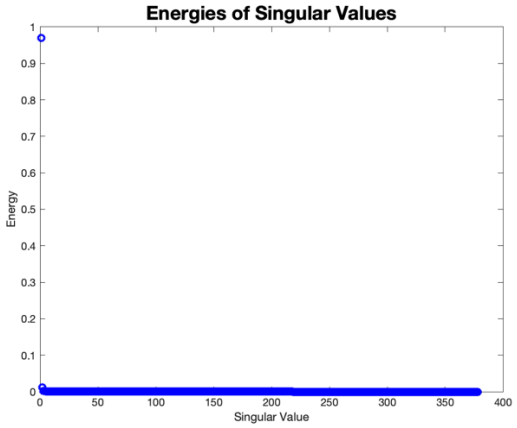


Fig. 1: Energies captured by each singular value

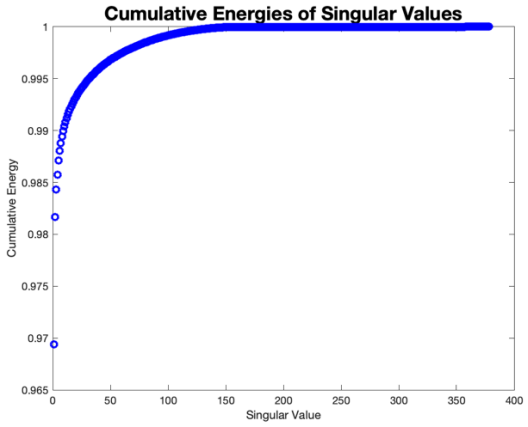


Fig. 2: Cumulative energies capture by singular values

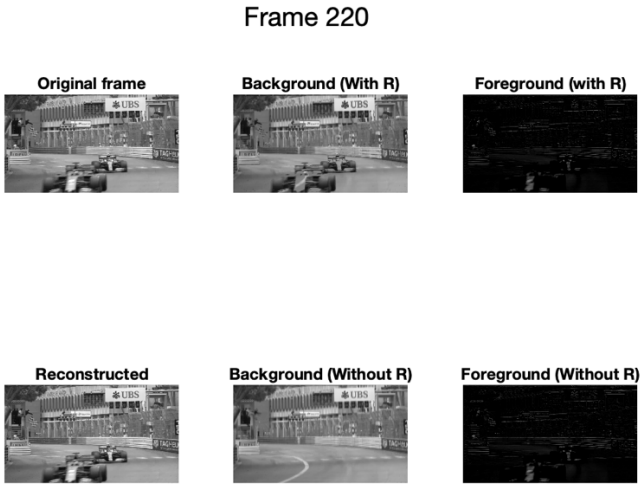


Fig. 3: Original frame and reconstructed backgrounds, foregrounds, and the sum of them

As shown in Fig. 1 and 2, the first singular value contains nearly 97% of the total energy. To optimize the reconstruction, we decide to do rank 2 approximation, which is still a good reduction from 379 modes.

As shown in Fig. 3, we reconstructed the original frame very well. With the addition of R to Xsparse, we can still see the two cars in the picture, but they are dimmer compared to the original frame. It is because it removes all the bright parts, such as reflections on cars and background, from the original frame. When we only show `abs(u_dmd)` in the subplot titled, "Background (Without R), it becomes blurrier, but it successfully removes the foreground object and only leaves the background. Since DMD mistreats all bright spots in the frame to be foreground objects, the subplots of foreground only show a mix of background and foreground objects instead of only the cars. For foreground, having and without R do not seem to have a big difference. DMD performs better on foreground removal.

### 4.2 Ski Video

As shown in Fig. 5 and 6, the first singular value contains over 99% of the total energy. To optimize the reconstruction, we decide to do rank 2 approximation, which is still a good reduction from 454 modes.

As shown in Fig. 7, we reconstructed the original frame very well. With the addition of R to Xsparse, we can still see the person skiing at the lower left corner of the picture, but some of the snow splash is removed. When we only show `abs(u_dmd)` in the subplot titled, "Background (Without R), it successfully removes the foreground object and only leaves the background. For foreground, having and without R do not seem to have a big difference. The subplots of foreground only contains the snow splash. DMD performs better on foreground removal.

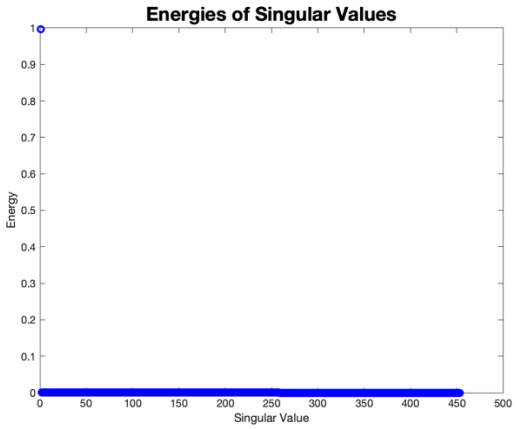


Fig. 5: Energies captured by each singular value

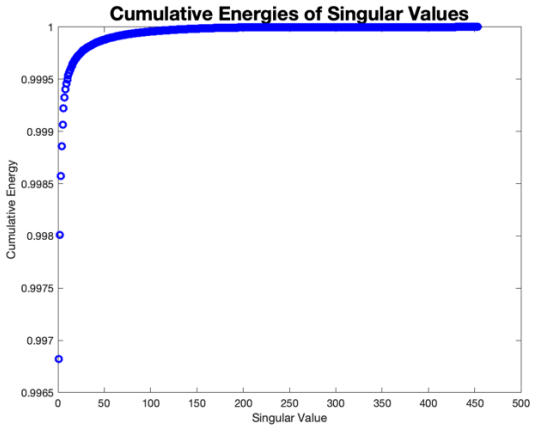


Fig. 6: Cumulative energies capture by singular values

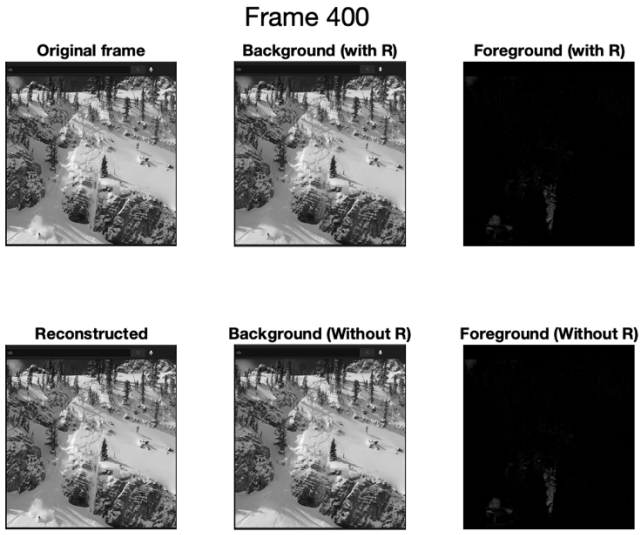


Fig. 7: Original frame and reconstructed backgrounds, foregrounds, and the sum of them

# 5 Summary and Conclusions

In conclusion, DMD successfully identifies the moving objects (cars, person skiing and snow splash) and removes them when we only want the background. However, the matrix R affects the results of separation between the background and the foreground. When we add R to the low rank matrix, it includes the foreground object as the background. On the other hand, R does not affect much on the background removal. Although the performance of DMD on background removal is not as well as that on foreground removal, DMD is still useful to separate the background and the foreground in general.

# Appendix A. MATLAB functions

- `X = zeros(sz1,...,szN)` returns an `sz1`-by-...-by-`szN` array of zeros where `sz1`, ... , `szN` indicate the size of each dimension. For example, `zeros(2,3)` returns a 2-by-3 matrix.
- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of m-by-n matrix `A`:
  - `m > n` — Only the first n columns of `U` are computed, and `S` is n-by-n.
  - `m = n` — `svd(A,'econ')` is equivalent to `svd(A)`.
  - `m < n` — Only the first m columns of `V` are computed, and `S` is m-by-m.
  - The economy-size decomposition removes extra rows or columns of zeros from the diagonal matrix of singular values, `S`, along with the columns in either `U` or `V` that multiply those zeros in the expression `A = U*S*V'`. Removing these zeros and columns can improve execution time and reduce storage requirements without compromising the accuracy of the decomposition.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector `v` on the main diagonal.
- `[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`.
- `B = reshape(A,sz)` reshapes `A` using the size vector, `sz`, to define `size(B)`. For example, `reshape(A,[2,3])` reshapes `A` into a 2-by-3 matrix. `sz` must contain at least 2 elements, and `prod(sz)` must be the same as `numel(A)`.
- `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.
- `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`. The `rgb2gray` function converts `RGB` images to grayscale by eliminating the hue and saturation information while retaining the luminance.
- `double(s)` converts the symbolic value `s` to double precision. Converting symbolic values to double precision is useful when a MATLAB® function does not accept symbolic values.

# Appendix B. MATLAB codes

```
% Clean workspace
clear all; close all; clc
vid1 = VideoReader('monte_carlo_low.mp4');
numFrames = vid1.NumFrames;
dt = 1/vid1.Framerate;
t = 0:dt:vid1.Duration;
video = read(vid1);
%%
for k = 1:numFrames
    Xdata = video(:,:,:,k);
    gray = rgb2gray(Xdata);
    Xdata = reshape(gray, [], 1);
    X(:,k) = double(Xdata);
end
%%
X1 = X(:,1:end-1);
X2 = X(:,2:end);
[U,Sigma,V] = svd(X1, 'econ');
sig = diag(Sigma);
%%
figure(1)
plot(sig.^2/sum(sig.^2),'bo','Linewidth',2);
ylabel('Energy')
xlabel('Singular Value')
title('Energies of Singular Values','Fontsize', 18)
print('Figure 1','-dpng');
figure(2)
plot(cumsum(sig.^2/sum(sig.^2)),'bo','Linewidth',2);
ylabel('Cumulative Energy')
xlabel('Singular Value')
title('Cumulative Energies of Singular Values','Fontsize', 18)
print('Figure 2','-dpng');
%% mode = 2
S = U(:,1:2)'*X2*V(:,1:2)*diag(1./diag(Sigma(1:2,1:2)));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U(:,1:2)*eV;
%%
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(t)-1);
for iter = 1:length(t)-1
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;
%%
Xsparse = X1 - abs(u_dmd);
R = Xsparse.*(Xsparse<0);
Xbg = R + abs(u_dmd); % background
Xfg = Xsparse - R; % foreground
X_recon = Xbg + Xfg;

%%
figure(3)
subplot(2,3,1)
p1 = reshape(X1, [540, 960, length(t)-1]);
```

```
imshow(uint8(p1(:,:,100)));
title('Original frame');
subplot(2,3,4)
p2 = reshape(X_recon, [540, 960, length(t)-1]);
imshow(uint8(p2(:,:,100)));
title('Reconstructed');
subplot(2,3,2)
p3 = reshape(Xbg, [540, 960, length(t)-1]);
imshow(uint8(p3(:,:,100)));
title('Background (With R)');
subplot(2,3,3)
p4 = reshape(Xsparse, [540, 960, length(t)-1]);
imshow(uint8(p4(:,:,100)));
title('Foreground (with R)');
subplot(2,3,6)
p5 = reshape(Xfg, [540, 960, length(t)-1]);
imshow(uint8(p5(:,:,100)));
title('Foreground (Without R)');
subplot(2,3,5)
p6 = reshape(abs(u_dmd), [540, 960, length(t)-1]);
imshow(uint8(p6(:,:,100)));
title('Background (Without R)');
sgtitle('Frame 100', 'Fontsize', 18);
print('Figure 3','-dpng');
%%
figure(4)
subplot(2,3,1)
p1 = reshape(X1, [540, 960, length(t)-1]);
imshow(uint8(p1(:,:,220)));
title('Original frame');
subplot(2,3,4)
p2 = reshape(X_recon, [540, 960, length(t)-1]);
imshow(uint8(p2(:,:,220)));
title('Reconstructed');
subplot(2,3,2)
p3 = reshape(Xbg, [540, 960, length(t)-1]);
imshow(uint8(p3(:,:,220)));
title('Background (With R)');
subplot(2,3,3)
p4 = reshape(Xsparse, [540, 960, length(t)-1]);
imshow(uint8(p4(:,:,220)));
title('Foreground (with R)');
subplot(2,3,6)
p5 = reshape(Xfg, [540, 960, length(t)-1]);
imshow(uint8(p5(:,:,220)));
title('Foreground (Without R)');
subplot(2,3,5)
p6 = reshape(abs(u_dmd), [540, 960, length(t)-1]);
imshow(uint8(p6(:,:,220)));
title('Background (Without R)');
sgtitle('Frame 100', 'Fontsize', 18);
print('Figure 4','-dpng');

%% % Clean workspace
clear all; close all; clc
vid1 = VideoReader('ski_drop_low.mp4');
numFrames = vid1.NumFrames;
dt = 1/vid1.Framerate;
```

```
t = 0:dt:vid1.Duration;
video = read(vid1);
%%
for k = 1:numFrames
    Xdata = video(:,:,:,k);
    gray = rgb2gray(Xdata);
    Xdata = reshape(gray, [], 1);
    X(:,k) = double(Xdata);
end
%%
X1 = X(:,1:end-1);
X2 = X(:,2:end);
[U,Sigma,V] = svd(X1, 'econ');
sig = diag(Sigma);
%%
figure(5)
plot(sig.^2/sum(sig.^2),'bo','Linewidth',2);
ylabel('Energy')
xlabel('Singular Value')
title('Energies of Singular Values','Fontsize', 18)
print('Figure 5','-dpng');
figure(6)
plot(cumsum(sig.^2/sum(sig.^2)),'bo','Linewidth',2);
ylabel('Cumulative Energy')
xlabel('Singular Value')
title('Cumulative Energies of Singular Values','Fontsize', 18)
print('Figure 6','-dpng');
%% mode = 2
S = U(:,1:2)'*X2*V(:,1:2)*diag(1./diag(Sigma(1:2,1:2)));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U(:,1:2)*eV;
%%
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(t)-1);
for iter = 1:length(t)-1
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;
%%
Xsparse = X1 - abs(u_dmd);
R = Xsparse.*(Xsparse<0);
Xbg = R + abs(u_dmd); % background
Xfg = Xsparse - R; % foreground
X_recon = Xbg + Xfg;
%%
figure(7)
subplot(2,3,1)
p1 = reshape(X1, [540, 960, length(t)-1]);
I1 = uint8(p1(:,:,100));
imshow(imcrop(I1,[230 0 500 540]));
title('Original frame');
subplot(2,3,4)
p4 = reshape(X_recon, [540, 960, length(t)-1]);
I4 = uint8(p4(:,:,100));
imshow(imcrop(I4,[230 0 500 540]));
title('Reconstructed');
```

```
subplot(2,3,2)
p2 = reshape(Xbg, [540, 960, length(t)-1]);
I2 = uint8(p2(:,:,100));
imshow(imcrop(I2,[230 0 500 540]));
title('Background (with R)');
subplot(2,3,6)
p3 = reshape(Xfg, [540, 960, length(t)-1]);
I3 = uint8(p3(:,:,100));
imshow(imcrop(I3,[230 0 500 540]));
title('Foreground (Without R)');
subplot(2,3,5)
p5 = reshape(abs(u_dmd), [540, 960, length(t)-1]);
I5 = uint8(p5(:,:,100));
imshow(imcrop(I5,[230 0 500 540]));
title('Background (Without R)');
subplot(2,3,3)
p6 = reshape(Xsparse, [540, 960, length(t)-1]);
I6 = uint8(p6(:,:,100));
imshow(imcrop(I6,[230 0 500 540]));
title('Foreground (with R)');
sgtitle('Frame 100', 'Fontsize', 18);
print('Figure 7','-dpng');

%%
figure(8)
subplot(2,3,1)
p1 = reshape(X1, [540, 960, length(t)-1]);
I1 = uint8(p1(:,:,400));
imshow(imcrop(I1,[230 0 500 540]));
title('Original frame');
subplot(2,3,4)
p4 = reshape(X_recon, [540, 960, length(t)-1]);
I4 = uint8(p4(:,:,400));
imshow(imcrop(I4,[230 0 500 540]));
title('Reconstructed');
subplot(2,3,2)
p2 = reshape(Xbg, [540, 960, length(t)-1]);
I2 = uint8(p2(:,:,400));
imshow(imcrop(I2,[230 0 500 540]));
title('Background (with R)');
subplot(2,3,6)
p3 = reshape(Xfg, [540, 960, length(t)-1]);
I3 = uint8(p3(:,:,400));
imshow(imcrop(I3,[230 0 500 540]));
title('Foreground (Without R)');
subplot(2,3,5)
p5 = reshape(abs(u_dmd), [540, 960, length(t)-1]);
I5 = uint8(p5(:,:,400));
imshow(imcrop(I5,[230 0 500 540]));
title('Background (Without R)');
subplot(2,3,3)
p6 = reshape(Xsparse, [540, 960, length(t)-1]);
I6 = uint8(p6(:,:,400));
imshow(imcrop(I6,[230 0 500 540]));
title('Foreground (with R)');
sgtitle('Frame 400', 'Fontsize', 18);
print('Figure 8','-dpng');
```