AMATH 482 Homework 5

Shane Fretwell

March 16, 2021

Abstract

Here we explore the ability of Dynamic Mode Decomposition to separate the foreground from the background of videos. We see varying success depending on the size of the foreground and the steadiness of the camera, but overall, the DMD is shown to be an effective tool for the task. We will also discuss why this approach is theoretically correct as well as the necessity of relaxations of the theoretical constraints for the sake of improved image quality.

1 Introduction and Overview

We have two videos on which we will perform our Dynamic Mode Decomposition. One is a video of a ski drop filmed from a great distance, wherein the foreground is the skier and any snow disturbed by the skier. The other of a race occurring in Monte Carlo, wherein the foreground is composed of the cars rounding the bend over the finish line.

2 Theoretical Background

2.1 Low Rank Approximation

As a short summary, one of the most powerful tools for low rank approximation of a matrix A is the Singular Value Decomposition. In fact, decomposing a matrix A according to equation 1 allows us to utilize a theorem from lecture 11.

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

This theorem states that the partial sum of equation 2 to some $r < n$, where n is the number of columns in U, is the best rank r approximation of the matrix A, as measured by the minimized L-2 norm.

$$A = \sum_{k=1}^{r} \sigma_k u_k v_k^* \tag{2}$$

2.2 Dynamic Mode Decomposition

The DMD, a method for dimensionality reduction, is undergone in several steps. Each requires an understanding of a different concept from linear algebra and differential equations.

2.2.1 Linearization

The first step is to assume that the system is linear. This is to say that each measurement of the system $x \in \mathbb{R}^n$ at time t is described by the relationship in equation 3. Discretizing this for use in numeric calculations, we get that the next measurement x_{i+1} is attained by left-multiplying x_i by A. Here, the matrix A is often referred to as the Koopman operator (1). Summarized in different words, we are assuming that the measurements of our system form a Krylov Subspace when measured at regular intervals.

$$\frac{dx}{dt} = Ax \tag{3}$$

Another, somewhat more useful way of representing this relationship is shown in equation 4, where X is a matrix of *m* evenly spaced measurements of the system.

$$\mathrm{X} \in \mathbb{R}^{\mathrm{mxn}}, \qquad \mathrm{X}_1 = [\mathrm{x}_1\ \mathrm{x}_2\ \ldots\ \mathrm{x}_{\mathrm{m}-1}], \qquad \mathrm{X}_2 = [\,\mathrm{x}_2\ \mathrm{x}_3\ \ldots\ \mathrm{x}_{\mathrm{m}}], \qquad \mathrm{X}_2 = \mathrm{A}\mathrm{X}_1 \tag{4}$$

2.2.2 Pseudoinverse

To take advantage of the assumption of linearity, we will need to consider the most suitable matrix A. Using the description of the system in equation 4, we know that the best fit for the matrix A, as it minimizes the L-2 norm error, is given by right-multiplying by the pseudoinverse of X_1. This is a property of the pseudoinverse, that if used to solve an overdetermined linear system of equations, the solution will minimize the L-2 norm error (2). Note that the pseudoinverse is denoted by a dagger: $X_1^{\dagger}$, and that it is calculated easily from the SVD as $V\Sigma^{-1}U^*$ where Σ^{-1} is Σ with its diagonal elements inverted (1).

2.2.3 Similar Matrices and Their Properties

Directly computing the pseudoinverse of X_1 is wasteful, however, when X_1 is large and/or has low rank structure. So, we use a low-rank approximation given by the SVD as described in section 2.1. From here on, the matrices U, Σ, and V should be assumed to be their truncated, low-rank versions.

Similar matrices are matrices related by an arbitrary invertible matrix. In our case, we will consider the first part of equation 5, wherein the arbitrary invertible matrix is U, from the SVD of X_1. $\tilde{A}$ is similar to A because of this relationship through U (1).

$$\tilde{A} = U^*AU, \qquad A = X_2X_1^{\dagger} = X_2V\Sigma^{-1}U^*, \qquad \tilde{A} = U^*X_2V\Sigma^{-1} \tag{5}$$

The eigenvalues of similar matrices are exactly the same, and the eigenvectors are related by the invertible matrix (2). Specifically, left-multiplying the eigenvectors of $\tilde{A}$ by the matrix U will give the eigenvectors of A. So, we can calculate the eigenvalues and eigenvectors of the (often orders of magnitude smaller) matrix $\tilde{A}$ and transform them into those of A.

2.2.4 Solving the Linear Differential System

The solution to problems of the form given by equation 3 are given by equation 6, where λ_j is the j^{th} eigenvalue, ϕ_j is the j^{th} eigenfunction, and b_j is the j^{th} constant as determined by initial conditions (2). In our case, the eigenfunctions are the eigenvectors of A, as calculated via the methods section 2.2.3. Each term of the sum in equation 6 is called a mode of the DMD.

$$x = \sum_{j=1}^{n} b_j * \phi_j * e^{\lambda_j t} \tag{6}$$

Note that this brings us back into continuous time, which means we will have to replace the eigenvalues λ_j with ω_j according to equation 7, where dt is the sampling interval of the original matrix X (2).

$$\omega_j = log(\lambda_j)/\mathrm{dt} \tag{7}$$

2.3 Identifying the Background Mode(s)

Note that the modes with $\omega_j \cong 0$ will become constant in time. These are the modes, should they exist, that will be describing the unchanging background of the data in X. This is the reason that DMD can be used to separate the foreground from the background of a video.

3 Algorithm Implementation and Development

The process for separating the background from the foreground of a video with DMD is almost exactly the same from video to video. The only difference is in the rank of the low-rank approximation chosen in the calculation of the pseudoinverse. Thus, the following algorithms are described generally and were applied to both videos without modification.

3.1 Obtaining the Data Matrices

For DMD to work, each column of the data matrix X must be a separate instance of the system. That is to say, we will have to reshape the video frames each into their own column of X. Before doing so, however, it is necessary for ease of computation to convert these frames into grayscale. The computation of many of the objects related to DMD requires large amounts of memory, and so the space saved by reducing the size of these matrices by a factor of three is a welcome shortcut.

3.2 Approximating X with Low Rank

Once we decompose X_1 into its SVD matrices, we can plot its singular values and determine a reasonable rank *r* from equation 2. A partial sum of equation 2 up to this *r* gives the best *r*-rank approximation of X_1.

3.3 Obtaining the Eigenfunctions

Here we apply sections 2.2.2 and 2.2.3 to get the eigenfunctions of the video, illustrated in algorithm 1.

Algorithm 1: Obtaining Eigenfunctions

```
Ã = U'*X2*V*diag(1./sigma);
[eV, D] = eig(Ã); % compute eigenvalues + eigenvectors of Ã
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV; % convert to eigenvectors of A
```

3.4 Solve Linear Differential System

Setting t to zero and using the built-in MATLAB backslash for solving the resulting linear system gives us the final element of the solution to the differential system, seen in algorithm 2.

Algorithm 2: Solve Initial Conditions and Generate DMD Modes

```
omega = log(mu)/dt;
y0 = Phi\X1; % backslash uses pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
   u_modes(:,iter) = y0.*exp(omega*t(iter));
end
```

3.5 Identify and Separate Background Modes

Subtracting the background from the foreground yields negative values of pixel intensity, which is nonsensical. Thus, we must add the negative remainders back into the foreground to remove negative values, while preserving relative pixel values. Here, we accomplish this by subtracting the minimum value pixel intensity from every pixel in each foreground frame. This forces the minimum value to zero, and while there is gray added to the image, this is preferable to losing the inter-pixel details that one would lose by simply setting all negative pixels to zero. Ideally, we would also be able to add this minimum value to the background to preserve the relationship between the background and foreground wherein their sum is exactly equal to X. However, this decreases image quality drastically, and should be avoided. Also in the interest of image quality, the values of each of the background and foreground are restricted to the range of the original video

Algorithm 3: Separation

```
[~,i] = min(omega); % Get index of minimum omega and its corresponding mode
back = real(Phi(:,i)*u_modes(i,:));
fore = X - back;
% Force all positive pixel values
R = min(fore);
fore = fore - R;
% Force all pixel values within the video's original range
M = max(fore);
XM = max(X);
if (M > 255)
   back = back / M * XM;
   fore = fore / M * XM;
end
```

4 Computational Results

4.1 Singular Value Spectrum and Reconstruction

Figure 1 shows the singular value spectrum of X_1 in the case of each video. For the Monte Carlo video, a rank 30 approximation gives results indistinguishable from the full rank reconstruction, and the same is true for the rank 25 approximation of the ski drop video. The Monte Carlo video likely requires more SVD modes because of the relative size of the foreground; the motion of the video occurs in approximately one fifth of the pixels of Monte Carlo, and that figure is closer to one twenty-fifth in the case of the ski drop video.

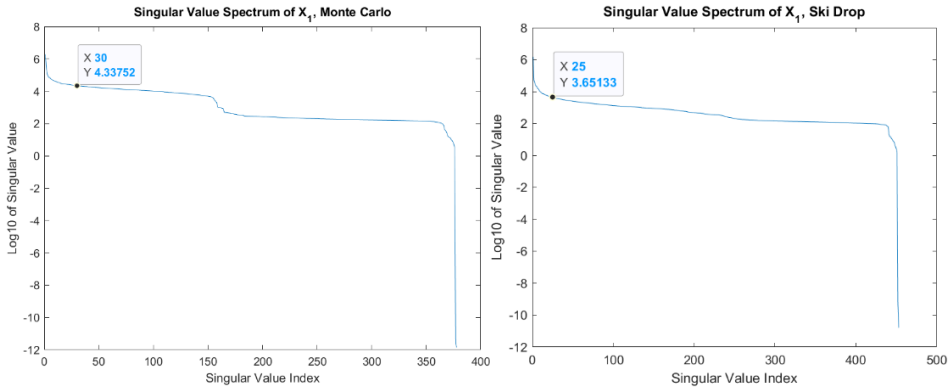

Figure 1: Spectral decomposition of X_1 matrices

4.2 Values of ω and Their Meaning

In figure 2, we have the values of ω from each video's DMD plotted in the complex plane. The long-term behavior of a DMD solution varies drastically depending on these values. Modes with an omega that has a positive real component will tend towards infinity at an exponential rate, and a negative real component will tend towards zero. Across both videos, there is only one ω with a positive real component, and it is that of the background in Monte Carlo. For the time frame of the videos, however, this value does not inflate its corresponding mode to infinity because its magnitude is so low. Both videos have one background mode (with $\omega_j \cong 0$) each. This makes sense, because it is only necessary to have one mode that remains constant to capture all of the constant behavior of the videos.

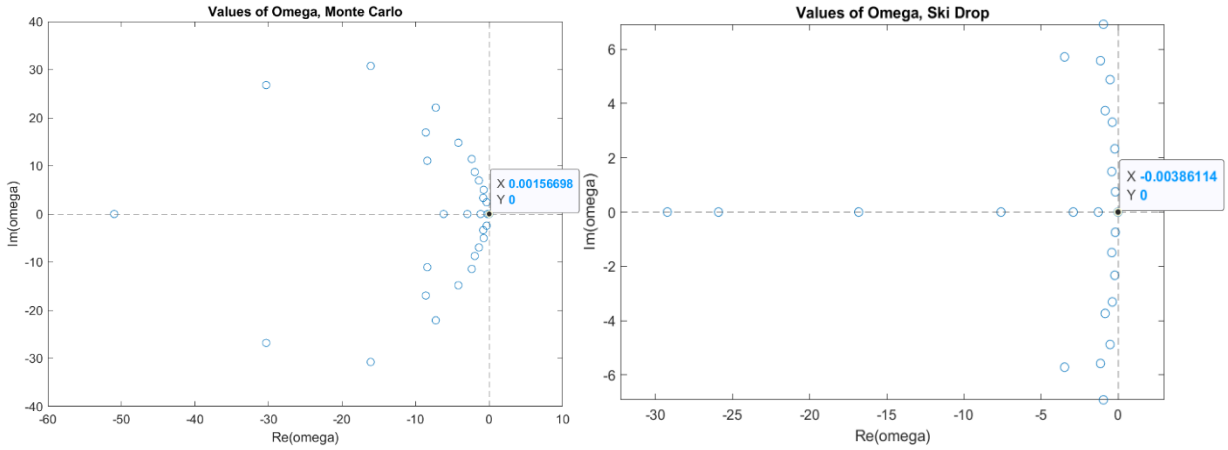

Figure 2: Spectrum of continuous eigenvalues of Koopman operators

4.3 Separated Images

In figures 3 and 4, we have the separated background and foreground images as compared to the original grayscale images. More examples of foreground extraction are found in Appendix B. There are some artifacts in the extracted foregrounds that look like edge detection, and these are the result of camera shake. These artifacts in the Monte Carlo video are far more pronounced, likely due to the vibrations transferred to the camera by the cars themselves. Upon close inspection, the cars in the Monte Carlo video are slightly transparent post foreground extraction, which can be seen in Appendix B.

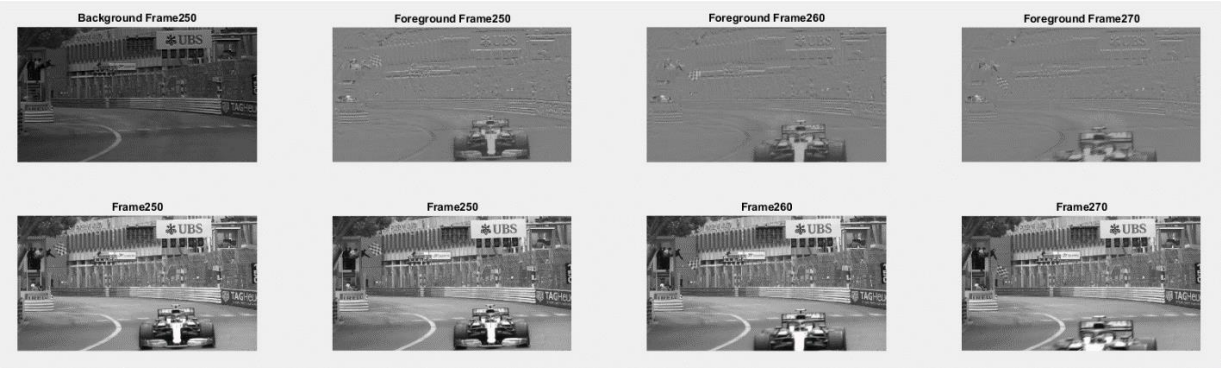

Figure 3: Background and foreground separation of Monte Carlo video frames, with original gray scale for reference.

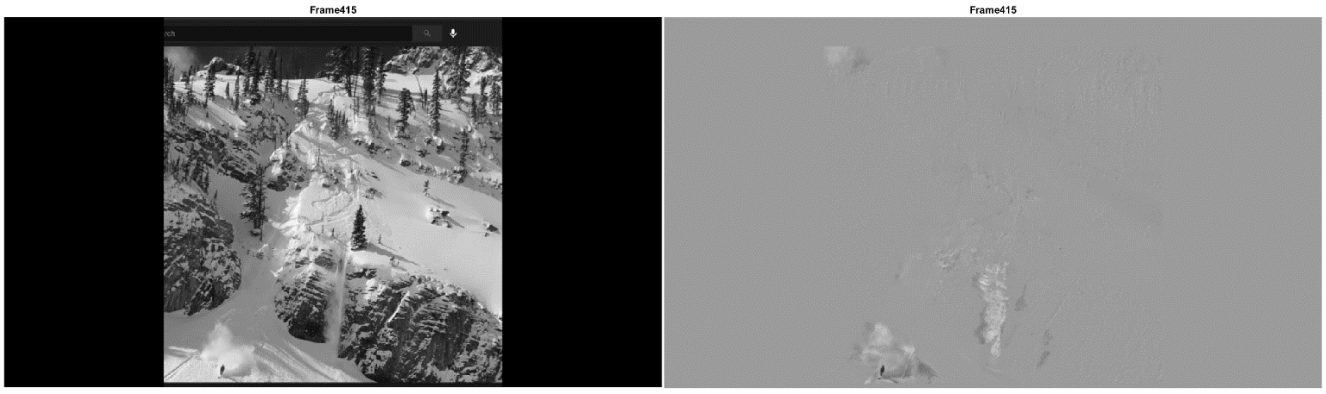

Figure 4: Original gray scale and extracted foreground of Ski Drop video frame. The still mountain is edited out, and only elements in the video that are in motion remain.

5 Summary and Conclusions

Having processed two different videos in the same manner, we are able to compare the results and hypothesize about possible influential factors on the level of success one can expect in using DMD to extract the foreground from the background of a video. Intuitively, a smaller foreground as compared to the total size of the individual video frames seems to be beneficial for the separation of the foreground. This is likely because more of the background is 'visible' to the process of DMD because it is not being covered up by the moving foreground. Most importantly, however, it seems that videos that contain small amounts of camera shake, even imperceptible by eye, will have artifacts reflecting this shaking in the extracted foreground. Most videos are not taken from a steady tripod or similar device and will contain camera shake that makes this method for foreground extraction largely useless. As a method for vibration detection, however, DMD should be a very useful tool. More investigation is required, but industries that require highly controlled environments and/or immediate detection of anomalous movement could make use of the processes undergone here in order to highlight otherwise imperceptible movement.

Finally, note that videos that contain only negligible camera shake, such as the Ski Drop video, do indeed lend themselves well to this method of foreground extraction. Also, the background of this video shows little to no sign of once having contained a skier. Overall, these analyses show the usefulness of the DMD for several video processing applications, not exclusively our original goal of foreground extraction.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[2] Nathan Kutz. (2018, April 14). *Dynamic Mode Decomposition (Theory)* [Video]. YouTube. https://www.youtube.com/watch?v=bYfGVQ1Sg98

Appendix A MATLAB Functions

- [eV, D] = eig($\tilde{A}$); computes the eigenvalues and eigenvectors of the matrix $\tilde{A}$, and returns them in two matrices; the first, eV, has as its columns the eigenvectors of $\tilde{A}$, and the second, D, has the corresponding eigenvalues on its diagonal, and zeros elsewhere.

- R = real(A); returns the real component(s) of the complex matrix or scalar A.

- [U, S, V] = svd(A); returns the matrices U, Σ, and V from the SVD decomposition of A.

Appendix B Additional Figures

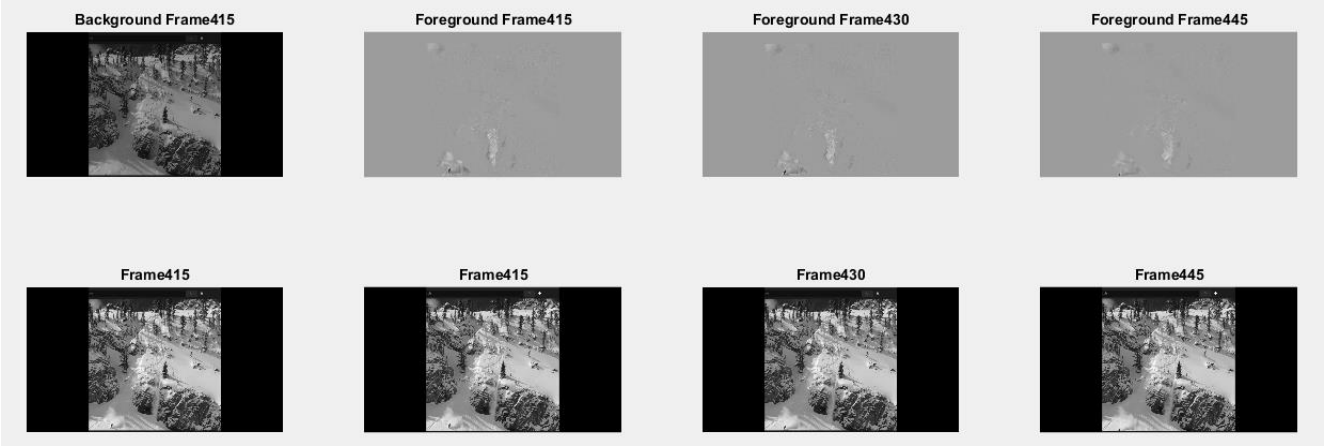

Figure B.1: Background and foreground separation of Ski Drop video frames, with original gray scale for reference.

Figure B.2: The extracted foreground of the 70th frame of Monte Carlo. This frame shows the worst of the camera shake artifacts and transparency of the cars themselves.

Figure B.3: The extracted background of the Ski Drop video.

Appendix C MATLAB Code

Monte_carlo.m

```
%%
%{
03-13-2021
Shane Fretwell
AMATH 482 Assignment 5, Foreground Extraction
%}
%%
clear; close all;

%% Read in video data
rdr =
VideoReader('C:/Users/sear/Documents/AMATH482_data/Ass5/monte_carlo_low.mp4');
N = rdr.NumFrames;
frames = read(rdr,[1 N]);
dt = rdr.Duration / rdr.NumFrames;
t = (0:N-1)*dt;

%% Convert to Grayscale
for i=1:length(t)
    frames(:,:,1,i) = rgb2gray(frames(:,:,:,i));
end
frames = reshape(frames(:,:,1,:),540,960,length(t));

%% Reshape to be one frame per column
X = double(reshape(frames, [], N));

%% The DMD matrices
% To save space, we will not store X1 and X2, but note the following:
```

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

%% SVD of X1 and Computation of ~S
[U, Sigma, V] = svd(X(:,1:end-1),'econ');
sigma = diag(Sigma);
clear Sigma

%% Plot singular value spectrum
figure(1)
plot(log10(sigma))
title('Singular Value Spectrum of X_1, Monte Carlo')
xlabel('Singular Value Index')
ylabel('Log10 of Singular Value')

%% Truncate Rank
r = 30;
U = U(:,1:r); sigma = sigma(1:r); V = V(:,1:r);

%% Obtaining Eigenfunctions
S = U'*X(:,2:end)*V*diag(1./sigma);
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV;

%% Solve Initial Conditions and Generate DMD Modes
%y0 = Phi\X1(:,1);
y0 = Phi\X(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end

%% Plot omega
figure(2)
plot(omega, 'o')
hold on
xline(0, '--'); yline(0, '--')
hold off
title('Values of Omega, Monte Carlo')
xlabel('Re(omega)'); ylabel('Im(omega)')

%% Separate Background and Foreground
[~,i] = min(omega);
back = real(Phi(:,i)*u_modes(i,:));
fore = X - back;
% Force all positive pixel values
R = min(fore, [], 'all');
fore = fore - R;
% Force all pixel values within the video's original range
M = max(fore, [], 'all');
XM = max(X, [], 'all');
if (M > 255)
    back = back / M * XM;
    fore = fore / M * XM;
```

```
end
back = reshape(uint8(abs(back)), 540, 960, []);
fore = reshape(uint8(abs(fore)), 540, 960, []);

%% Show the full length video, either the extracted foreground, background, or
original
figure(3)
for i=1:length(t)
    imshow(fore(:,:,i))
    %imshow(back(:,:,i))
    %imshow(frames(:,:,i))
    title(strcat('Frame ', num2str(i)))
end

%% Show several original frames and separated foregrounds side by side
figure(4)
framenums = [250 260 270];
width = length(framenums)+1;

subplot(2, width, 1)
imshow(back(:,:,framenums(1)))
title(strcat('Background Frame ', num2str(framenums(1))))

subplot(2, width, width+1)
imshow(frames(:,:,framenums(1)))
title(strcat('Frame ', num2str(framenums(1))))

for i=1:length(framenums)
    subplot(2, width, i+1)
    imshow(fore(:,:,framenums(i)))
    title(strcat('Foreground Frame ', num2str(framenums(i))))

    subplot(2, width, width+i+1)
    imshow(frames(:,:,framenums(i)))
    title(strcat('Frame ', num2str(framenums(i))))
end
```

ski_drop.m

```
%%
%{
03-13-2021
Shane Fretwell
AMATH 482 Assignment 5, Foreground Extraction
%}
%%
clear; close all;

%% Read in video data
rdr = VideoReader('C:/Users/sear/Documents/AMATH482_data/Ass5/ski_drop_low.mp4');
N = rdr.NumFrames;
frames = read(rdr,[1 N]);
dt = rdr.Duration / rdr.NumFrames;
t = (0:N-1)*dt;

%% Convert to Grayscale
```

```
for i=1:length(t)
    frames(:,:,1,i) = rgb2gray(frames(:,:,:,i));
end
frames = reshape(frames(:,:,1,:),540,960,length(t));

%% Reshape to be one frame per column
X = double(reshape(frames, [], N));

%% The DMD matrices
% To save space, we will not store X1 and X2, but note the following:
% X1 = X(:,1:end-1); X2 = X(:,2:end);

%% SVD of X1 and Computation of ~S
[U, Sigma, V] = svd(X(:,1:end-1),'econ');
sigma = diag(Sigma);
clear Sigma

%% Plot singular value spectrum
figure(1)
plot(log10(sigma))
title('Singular Value Spectrum of X_1, Ski Drop')
xlabel('Singular Value Index')
ylabel('Log10 of Singular Value')

%% Truncate Rank
r = 25;
U = U(:,1:r); sigma = sigma(1:r); V = V(:,1:r);

%% Obtaining Eigenfunctions
S = U'*X(:,2:end)*V*diag(1./sigma);
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV;

%% Solve Initial Conditions and Generate DMD Modes
%y0 = Phi\X1(:,1);
y0 = Phi\X(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end

%% Plot omega
figure(2)
plot(omega, 'o')
axis([min(real(omega))-3 3 -inf inf])
hold on
xline(0, '--'); yline(0, '--')
hold off
title('Values of Omega, Ski Drop')
xlabel('Re(omega)'); ylabel('Im(omega)')

%% Separate Background and Foreground
[~,i] = min(omega);
```

```
back = real(Phi(:,i)*u_modes(i,:));
fore = X - back;
% Force all positive pixel values
R = min(fore, [], 'all');
fore = fore - R;
% Force all pixel values between 0 and 255
M = max(fore, [], 'all');
XM = max(X, [], 'all');
if (M > 255)
    back = back / M * XM;
    fore = fore / M * XM;
end
back = reshape(uint8(abs(back)), 540, 960, []);
fore = reshape(uint8(abs(fore)), 540, 960, []);

%% Show the full length video, either the extracted foreground, background, or
original
figure(3)
for i=1:length(t)
    %imshow(fore(:,:,i))
    imshow(back(:,:,i))
    %imshow(frames(:,:,i))
    title(strcat('Frame ', num2str(i)))
end

%% Show several original frames and separated foregrounds side by side
figure(4)
delta = 15;
start = 400;
framenums = start+(1:3)*delta;
width = length(framenums)+1;

subplot(2, width, 1)
imshow(back(:,:,framenums(1)))
title(strcat('Background Frame ', num2str(framenums(1))))

subplot(2, width, width+1)
imshow(frames(:,:,framenums(1)))
title(strcat('Frame ', num2str(framenums(1))))

for i=1:length(framenums)
    subplot(2, width, i+1)
    imshow(fore(:,:,framenums(i)))
    title(strcat('Foreground Frame ', num2str(framenums(i))))

    subplot(2, width, width+i+1)
    imshow(frames(:,:,framenums(i)))
    title(strcat('Frame ', num2str(framenums(i))))
end
```