Homework 4: Background Subtraction in Video Streams
AMATH 482: Computational Methods for Data Analysis, Winter 2017

Sean Lam
seanlam@uw.edu

March 3, 2017

Abstract

This report investigates the use of dynamic mode decomposition (DMD) for separating video background and foreground. The background is the approximate low rank DMD reconstruction, and the foreground is the approximate sparse DMD reconstruction. The DMD method is very robust and fast, and can be used in applications that require video to be separated in real time.

I. Introduction and Overview

Computer vision is a rapidly growing field, with one of its applications being real-time video surveillance. One big aspect of video surveillance is being able to subtract the background from a video stream, which is relatively the same for each frame, leaving the foreground to detect moving objects. The hardest part about video surveillance is that it has to be able to process in real-time with decent quality. This calls for very robust algorithms that can run without huge processing power.

DMD is a powerful tool used for analyzing the dynamics of nonlinear systems without knowing their underlying equations. All it requires is time series data. In this application, the time series data are the individual video frames, which are essentially snapshots of a nonlinear dynamical system. The background can be thought of the modes that are not changing (low frequency), and the foreground are the rest of the modes that are changing much faster than the background modes.

II. Theoretical Background

The Dynamic Mode Decomposition is essentially a combination of spatial dimensionality-reduction techniques, in this case the singular value decomposition with Fourier transforms in time.

Suppose we have a set of time series data with n data points collected at a given time, with m total samplings in time, evenly spaced by Δt. Let $\mathbf{x_j}$ be a column vector of the n data points collected at time t_j, j = 1, 2, ... m. We can represent this data as a dynamical system:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x_1} & \mathbf{x_2} & \dots & \mathbf{x_{m-1}} \\ | & | & & | \end{bmatrix} \tag{1}$$

$$\mathbf{X'} = \begin{bmatrix} | & | & & | \\ \mathbf{x_2} & \mathbf{x_3} & ... & \mathbf{x_m} \\ | & | & & | \end{bmatrix} \tag{2}$$

Our goal is to solve the system

$$\mathbf{X'} \approx \mathbf{AX} \tag{3}$$

where $\mathbf{A}$ is the Koopman operator, the best fit linear operator that will solve the underdetermined system.

$$\mathbf{A} = \mathbf{X'X}^{\dagger} \tag{4}$$

The singular value decomposition for $\mathbf{X}$ can be used for dimensionality reduction. The singular value decomposition decomposes a matrix into three constitutive components,

$$\mathbf{X} = \mathbf{U\Sigma V}^{*} \tag{5}$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary orthogonal bases, and $\mathbf{\Sigma}$ is a diagonal matrix. From the SVD, we can get

$$\mathbf{A} = \mathbf{X'V\Sigma^{-1}U}^{\dagger} \tag{6}$$

A rank reduction of A can be computed by

$$\mathbf{\tilde{A}} = \mathbf{U}^{\dagger}\mathbf{X'V\Sigma^{-1}} \tag{7}$$

The DMD modes $\mathbf{\Phi}$, also known as the dynamic modes, are the eigenvectors of $\mathbf{\tilde{A}}$, and each DMD mode corresponds to a particular eigenvalue $\mathbf{\Lambda}$.

$$\mathbf{\tilde{A}W} = \mathbf{W\Lambda} \tag{8}$$

The DMD modes are given by columns of $\mathbf{\Phi}$

$$\mathbf{\Phi} = \mathbf{X'V\Sigma^{-1}W} \tag{9}$$

With the low rank approximations of both the eigenvalues and eigenvectors, the projected future solution can be constructed at all future times is given by

$$\mathbf{X}(t) = \sum_{k=1}^{r} \phi_k e^{\omega_k t} b_k \tag{10}$$

where

$$\omega_k = ln(\lambda_k)/\Delta t \tag{11}$$

are the Fourier modes and b_k where

$$\mathbf{b} = \mathbf{\Phi}^{\dagger}\mathbf{X} \tag{12}$$

is the initial amplitude of each mode.

The next step is to subtract the background modes.

$$\mathbf{X}_{DMD} = b_p \varphi_p e^{\omega_p \mathbf{t}} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j \mathbf{t}} \tag{13}$$

The first term of $\mathbf{X}_{DMD}$ is the DMD's approximate low-rank reconstruction, which is the background video. The value of p is the mode that has its $||\omega_p|| \approx 0$. The second term is the DMD's approximate sparse reconstruction, which is the foreground video, where $||\omega_j|| \; \forall \; j \neq p$ is bounded away from zero.

Each term of the DMD reconstruction is complex, which poses a problem because videos have to be constructed with real numbers. To fix this, the algorithm takes the modulus of each element within the matrix.

The real-valued elements of the approximate sparse reconstruction is

$$\mathbf{X}_{DMD}^{Sparse} = \mathbf{X} - |\mathbf{X}_{DMD}^{Low-Rank}| \tag{14}$$

This may result in the sparse reconstruction to have some negative values in some of its elements, but negative pixel intensities makes no sense. To account for this, the residual negative values can be put into a $n \times m$ matrix $\mathbf{R}$ and then added back into $\mathbf{X}_{DMD}^{Low-Rank}$

$$\mathbf{X}_{DMD}^{Low-Rank} \Leftarrow \mathbf{R} + |\mathbf{X}_{DMD}^{Low-Rank}| \tag{15}$$

$$\mathbf{X}_{DMD}^{Sparse} \Leftarrow \mathbf{X}_{DMD}^{Sparse} - \mathbf{R} \tag{16}$$

$$\mathbf{X}_{DMD} = \mathbf{X}_{DMD}^{Sparse} + \mathbf{X}_{DMD}^{Low-Rank} \tag{17}$$

III. Algorithm Implementation and Development

First, the video is loaded into a 4 dimensional matrix: rows $\times$ columns $\times$ RGB $\times$ frames. The video is then grayscaled for faster computational time. Since the video is pretty long, I only took a 30 second clip, which was sufficient for DMD reconstruction. I then proceeded to only keep every other frame, making the video 15 fps instead of the original 30 fps. This was also to reduce computational time without affecting the DMD approximation dramatically.

I then created the $\mathbf{X}$ and $\mathbf{X}'$ matrices in equation 1 and 2, and took the singular value decomposition of $\mathbf{X}$, keeping 95% of the accuracy from the singular values. $\tilde{A}$ is then calculated with equation 7, and then the eigendecomposition is computed. From this, $\mathbf{\Phi}$ (equation 9), the DMD modes, $\mathbf{\Lambda}$, the discrete-time eigenvalues, and $\mathbf{\Omega}$ (equation 11), the Fourier modes. $\mathbf{b}$ is then computed with equation 12. Finally, $\mathbf{X_{DMD}}$ was computed using equation 10.

The next step is to compute the background video (DMD low-rank approximation), which is the first term on the right hand side of equation 13. The value of p was found by finding the ω_p value closest to 0. Then, the DMD sparse approximation was found by subtracting the original video by the background video. The negative residuals were taken care of as shown in equations 15 and 16.

IV. Computational Results

The first 30 seconds of the traffic video was inputted into the DMD algorithm, to output the DMD reconstruction shown in Figure 2 (second column). Keeping 95% of the information from the SVD, the DMD reconstruction is almost identical to the original video (first column). The Fourier modes ω_p are plotted on Figure 1. The mode corresponding to the background is the one that is closest to zero, because the background never "moves". In this case, the background mode was $p = 266$, with an ω_p value of 0.005188. The DMD background approximation is also shown on Figure 2 (third column), although it is a little brighter than the original video. The background video is an almost perfect reconstruction of the road and scenery without any cars. Subtracting the original video by the background video while accounting for negative residuals gives the DMD sparse approximation, also shown on Figure 2 (fourth column). One flaw is that the darker vehicles are hard to spot on the sparse reconstruction because the background is also

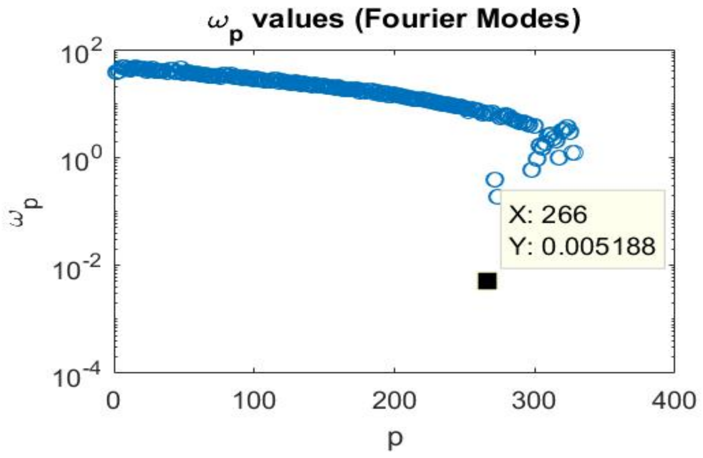

Figure 1: These are the Fourier modes for the traffic video. The corresponding p value for the background is p = 266, with an ω_p value of 0.005188.

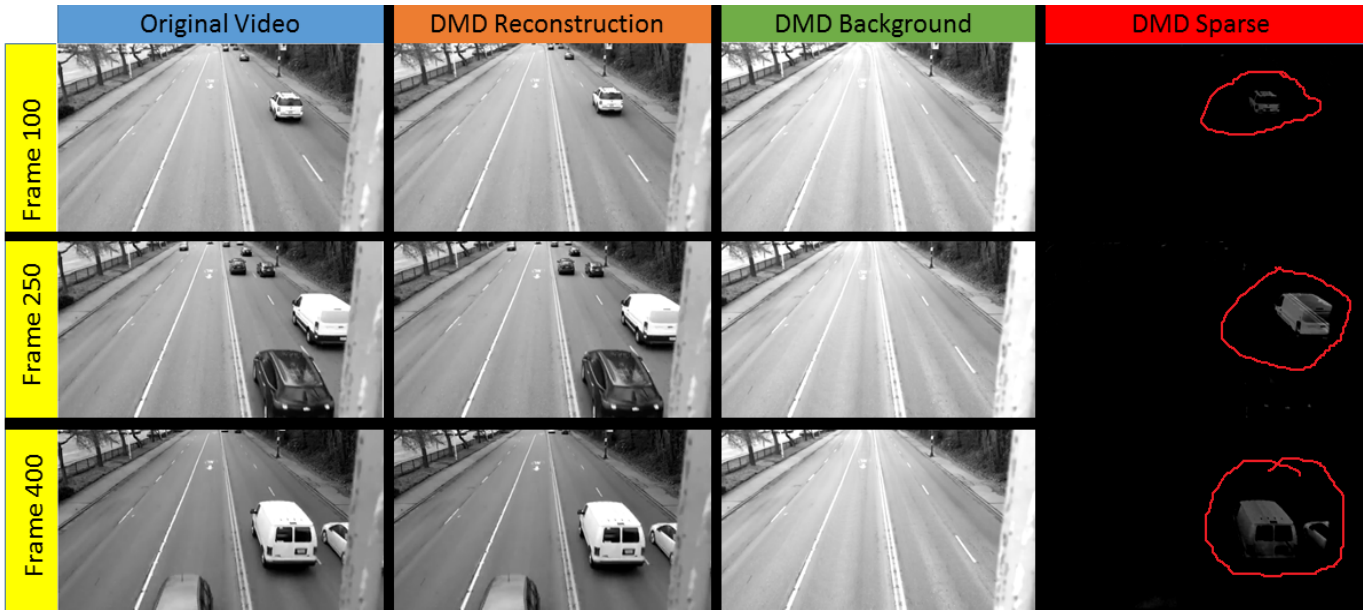

Figure 2: These are selected frames from the traffic video DMD. The columns, from left to right, are the 100th, 250th, and 400th frame for the original video, DMD reconstruction, DMD low-rank approximation (background), and DMD sparse approximation (foreground). Circled are the visible moving objects.

blacked out, so the dark vehicles are camouflaged in there. A solution to this problem would be a colored DMD reconstruction, which will take much longer to compute.

Next, the first 30 seconds of the scooter video was inputted into the DMD algorithm, to output the DMD reconstruction shown in Figure 4 (second column). This video has less moving objects, and the objects are much smaller compared to the traffic video because the people in the video scooter quite far from the camera. However, the DMD still produces similar results

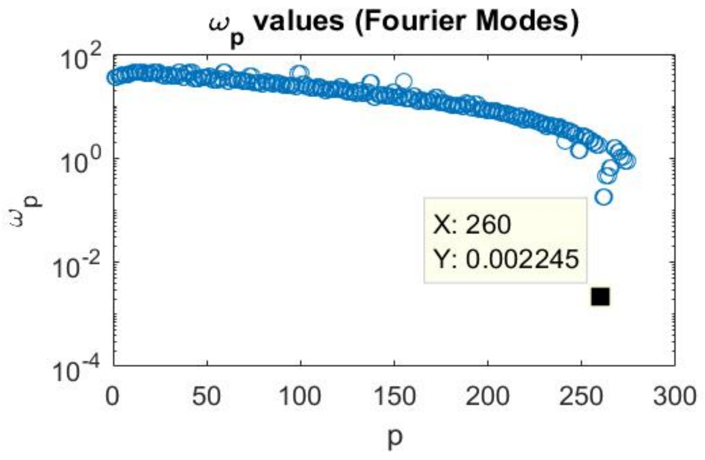

Figure 3: These are the Fourier modes for the scooter video. The corresponding p value for the background is p = 260, with an ω_p value of 0.002245.

as the traffic video. Keeping 95% of the information from the SVD, the DMD reconstruction is almost identical to the original video (first column). The Fourier modes ω_p are plotted on Figure 3. In this case, the background mode was $p = 260$, with an ω_p value of 0.002245. The DMD background approximation is also shown on Figure 4 (third column), although it is a little brighter than the original video. The background video is an almost perfect reconstruction of the parking lot and scenery without the riders. Subtracting the original video by the background video while accounting for negative residuals gives the DMD sparse approximation, also shown on Figure 4 (fourth column). Looking closely at the sparse reconstruction, the scooter rider can be seen, albeit barely. This is caused by a bad choice in clothing; both riders were wearing dark clothes, so it blended in with the black background. However, we can tell that the DMD algorithm worked successfully because on the sparse reconstructions the parts of the body not covered in dark clothing (forearms, lower leg) can be seen. The assumption can be made that the algorithm captured the dark clothing as well, but is just not shown clearly in the Figure. Again, a colored reconstruction would be the solution to this problem, but would be much more computationally heavy.

V. Summary and Conclusions

This project investigated the dynamic mode decomposition, an algorithm based on spatial dimensionality-reduction techniques, such as the singular value decomposition with Fourier transforms in time. The DMD was used to reconstruct the low-rank approximation (background) and sparse approximation (foreground) of videos with a fixed camera position. The results show the robustness of the algorithm, being able to separate foregrounds that are very small, such as the riders in the scooter video, with fast computation time.

This algorithm is a very viable way to process real-time surveillance, because it can reconstruct backgrounds almost perfectly with less than a minute of video stream, and then all it has to do to separate the foreground by a simple matrix subtraction of the original video by the background video, assuming the background stays the same during the duration of the surveillance video.

Besides video surveillance, the DMD has tons of applications and uses, including diagnostics of complex data, state estimation and future state prediction, and control. It can categorize modern

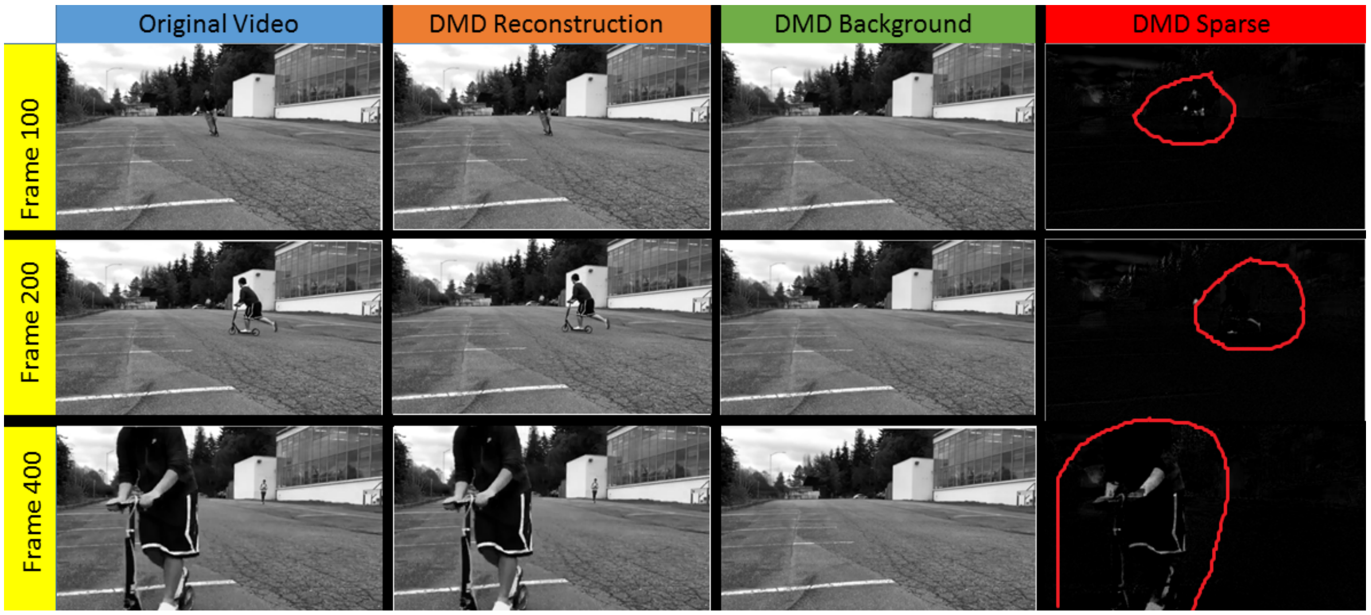

Figure 4: These are selected frames from the scooter video DMD. The columns, from left to right, are the 100th, 250th, and 400th frame for the original video, DMD reconstruction, DMD low-rank approximation (background), and DMD sparse approximation (foreground). Circled are the moving objects.

systems of interest such as turbulent fluid flow, an epidemiology system, stock market growth, or the climate, as a high-dimensional nonlinear dynamical system in space and time. From there, it can then transform these systems into simple representations based on spatio-temporal coherent structures.

Appendix A: MATLAB functions used and brief implementation explanation

v = VideoReader(file) - creates object v to read video data from file
vInfo = get(v) - gets all the properties of video v, including frames rate, video size, etc
frame = readFrame(v) - read the next available video frame from the file associated with v
grayFrame = rgb2gray(colorFrame) - turn a colored image from RGB to grayscale
resizedFrame = imresize(originalFrame, dimensions) - resizes the original frame into the corresponding numbers of rows and columns specified by the dimensions parameter
reshapedFrame = reshape(originalFrame, rows, columns) - reshape the original frame into a matrix with the specified size in rows and columns
vid = mat2vid(matrix) - Takes in a matrix of column vectors for each frame of a video, and reshapes it to a video matrix that can be played
matrix = vid2mat(video) - Takes in a video matrix that can be played, and reshapes it into a matrix of column vectors for each frame of a video, for computing
[U, S, V] = svd(X) - takes the singular decomposition of X and outputs the corresponding **U**, **Σ**, and **V** matrices
implay(video) - plays a video from a video matrix on the MATLAB video player

Appendix B: MATLAB codes

Traffic Video
trafficDMD.m

```
clear all; close all; clc;
v = VideoReader('traffic.mp4');
vInfo = get(v);
fps = round(vInfo.FrameRate);
dimensions = [vInfo.Height vInfo.Width];
counter = 0;

%make dimensions smaller
scale = 1;
dimensions = dimensions / scale;
while hasFrame(v)
    counter = counter + 1;
    frame = readFrame(v);
    frame = rgb2gray(frame);
    video(:,:,counter) = imresize(frame, dimensions);
end

[rows, cols, n] = size(video);
vid = zeros(rows * cols, n);
%start and stop time
start = 0; %seconds
stop = 30; %seconds
n1 = start * fps + 1;
n2 = stop * fps;
n = n2 - n1;
counter = 0;
newvid = zeros(rows*cols, n);
for j = n1:n2
   currentFrame = video(:,:,j);
   counter = counter + 1;
   newvid(:,counter) = reshape(currentFrame, rows * cols, 1);
end

% play the new video
playvid = mat2vid(newvid,dimensions);
%implay(uint8(playvid));
vid = vid2mat(playvid);

%if want to take every f frames
f = 2;
fps = fps / f;
counter = 0;
newvid = [];
for j = 1:n
   if (mod(j-1,f) == 0)
```

```
        counter = counter + 1;
        newvid(:,counter) = vid(:,j);
    end
end
vid = newvid;
implay(uint8(mat2vid(vid, [rows cols])))

% Collect X',X
X1 = vid(:,1:end-1);
X2 = vid(:,2:end);

[U,S,V] = svd(X1,'econ');
lambda = 100*(diag(S))/sum(diag(S));

r = 0;
accuracy = 0;
while accuracy < 95
    r = r + 1;
    accuracy = accuracy + lambda(r);
end

%truncate rank
Ur = U(:,1:r);
Sr = S(1:r,1:r);
Vr = V(:,1:r);

% Find New Basis
A = Ur.'*X2*Vr / Sr; % low-rank dynamics
[W,D] = eig(A);
phi = X2*Vr/Sr*W; % DMD modes
lambda = diag(D); % discrete-time eigenvalues
omega = log(lambda)*fps; % continuous-time eigenvalues
figure(1)
semilogy(abs(omega),'o')
xlabel('p')
ylabel('\omega_p')
title('\omega_p values (Fourier Modes)')

% Compute DMD mode amplitudes b
x1 = X1(:,1);
b = phi\x1;
mm1 = size(X1, 2); % mm1 = m - 1
r = length(lambda);
time_dynamics = zeros(r,mm1);
t = (0:mm1-1)/fps; %time vector
for j = 1:mm1
    time_dynamics(:,j) = b.*exp(omega*t(j));
end
viddmd = phi * time_dynamics;
```

```
n = size(viddmd,2);
videodmd = zeros(rows, cols, n);
for j = 1:n
   videodmd(:,:,j) = reshape(viddmd(:,j),rows,cols);
end
videodmd = real(videodmd);

% background
[~,p] = min(abs(omega));
back = b(p) * phi(:,p) * exp(omega(p) * t);
realback = abs(back);
n = size(back,2);
for j = 1:n
   backvid(:,:,j) = reshape(realback(:,j),rows,cols);
end
implay(uint8(backvid))

%% DMD Sparse
R = back - realback;
sparse = X1 - round(realback) - abs(R);
dmd = back + sparse;
dmdvid = mat2vid(dmd, [rows cols]);
sparsevid = mat2vid(sparse, [rows cols]);
implay(uint8(sparsevid))
implay(uint8(dmdvid))
```

Scooter Video
dmdScooter.m

```
clear all; close all; clc;
v = VideoReader('scooter.mp4');
vInfo = get(v);
fps = round(vInfo.FrameRate);
dimensions = [vInfo.Height vInfo.Width];
counter = 0;

%make dimensions smaller
scale = 1;
dimensions = dimensions / scale;
while hasFrame(v)
    counter = counter + 1;
    frame = readFrame(v);
    frame = rgb2gray(frame);
    video(:,:,counter) = imresize(frame, dimensions);
end

[rows, cols, n] = size(video);
vid = zeros(rows * cols, n);
%start and stop time
start = 0; %seconds
stop = 30; %seconds
```

```
n1 = start * fps + 1;
n2 = stop * fps;
n = n2 - n1;
counter = 0;
newvid = zeros(rows*cols, n);
for j = n1:n2
   currentFrame = video(:,:,j);
   counter = counter + 1;
   newvid(:,counter) = reshape(currentFrame, rows * cols, 1);
end

% play the new video
playvid = mat2vid(newvid,dimensions);
%implay(uint8(playvid));
vid = vid2mat(playvid);

%if want to take every f frames
f = 2;
fps = fps / f;
counter = 0;
newvid = [];
for j = 1:n
   if (mod(j-1,f) == 0)
       counter = counter + 1;
       newvid(:,counter) = vid(:,j);
   end
end
vid = newvid;
implay(uint8(mat2vid(vid, [rows cols])))

% Collect X',X
X1 = vid(:,1:end-1);
X2 = vid(:,2:end);

[U,S,V] = svd(X1,'econ');
lambda = 100*(diag(S))/sum(diag(S));

r = 0;
accuracy = 0;
while accuracy < 95
    r = r + 1;
    accuracy = accuracy + lambda(r);
end

%truncate rank
Ur = U(:,1:r);
Sr = S(1:r,1:r);
Vr = V(:,1:r);

% Find New Basis
```

```
A = Ur.'*X2*Vr / Sr; % low-rank dynamics
[W,D] = eig(A);
phi = X2*Vr/Sr*W; % DMD modes
lambda = diag(D); % discrete-time eigenvalues
omega = log(lambda)*fps; % continuous-time eigenvalues
figure(1)
semilogy(abs(omega),'o')
xlabel('p')
ylabel('\omega_p')
title('\omega_p values (Fourier Modes)')

% Compute DMD mode amplitudes b
x1 = X1(:,1);
b = phi\x1;
mm1 = size(X1, 2); % mm1 = m - 1
r = length(lambda);
time_dynamics = zeros(r,mm1);
t = (0:mm1-1)/fps; %time vector
for j = 1:mm1
    time_dynamics(:,j) = b.*exp(omega*t(j));
end
viddmd = phi * time_dynamics;

n = size(viddmd,2);
videodmd = zeros(rows, cols, n);
for j = 1:n
   videodmd(:,:,j) = reshape(viddmd(:,j),rows,cols);
end
videodmd = real(videodmd);

% background
[~,p] = min(abs(omega));
back = b(p) * phi(:,p) * exp(omega(p) * t);
realback = abs(back);
n = size(back,2);
for j = 1:n
   backvid(:,:,j) = reshape(realback(:,j),rows,cols);
end
implay(uint8(backvid))

%% DMD Sparse
R = back - realback;
sparse = X1 - round(realback) - abs(R);
dmd = back + sparse;
dmdvid = mat2vid(dmd, [rows cols]);
sparsevid = mat2vid(sparse, [rows cols]);
implay(uint8(sparsevid))
implay(uint8(dmdvid))
```

Column vector video frames to sized video mat2vid.m

```
%column vectors of video frames to sized video
function vid = mat2vid(mat, dimensions)
    [~, frames] = size(mat);
    vid = zeros(dimensions(1), dimensions(2), frames);
    for j = 1:frames
        vid(:,:,j) = reshape(mat(:,j),dimensions(1),dimensions(2));
    end
end
```

Sized video to column vector video frames
vid2mat.m

```
%sized video to column vector video frames
function mat = vid2mat(vid)
    [rows, cols, frames] = size(vid);
    mat = zeros(rows*cols,frames);
    for j = 1:frames
        mat(:,j) = reshape(vid(:,:,j),rows*cols,1);
    end
end
```