Background Subtraction in Video Streams
Automatic Motion Extraction using Dynamic Mode Decomposition Algorithms

Dino de Raad

April 11, 2019

Abstract
The DMD is used to subtract background images from streams of video data. An overview of the DMD method is given in both theoretical and algorithmic detail. Several examples are given and limitations are discussed.

I Introduction and Overview

The Dynamic Mode Decomposition (DMD) is used to extract foreground and background objects in a video. Foreground objects are any objects in the video that move, and background objects are those that generally stay still. This method can then be used to separate moving and non-moving components of videos for analysis.

II Theoretical Background

Overview

The Dynamic Mode Decomposition (DMD) has been used as an equation-free structure to extract the dynamics of highly complex, potentially non-linear spatio-temporal systems for future state prediction. It combines the space dimension reduction techniques of the Principle Component Analysis with the time-series analysis of the Fourier transform to form a regression.

Details

We collect m data snapshots of length n $\mathbf{x}_k$ at times t_k and construct the matrices

$$\mathbf{X} = \begin{bmatrix} | & | & & | & | \\ \mathbf{x_1} & \mathbf{x_2} & \cdots & \mathbf{x_{m-2}} & \mathbf{x_{m-1}} \\ | & | & & | & | \end{bmatrix}, \quad \mathbf{X}' = \begin{bmatrix} | & | & & | & | \\ \mathbf{x_2} & \mathbf{x_3} & \cdots & \mathbf{x_{m-1}} & \mathbf{x_m} \\ | & | & & | & | \end{bmatrix}$$

so that a local linear solution may look like $\mathbf{X}' \approx \mathbf{AX}$. Additionally, the time between snapshots is a fixed Δt. $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{X}'\mathbf{X}^\dagger$ where $\mathbf{X}^\dagger$ is the pseudo-inverse of $\mathbf{X}$. This solution for $\mathbf{A}$ minimizes the Frobenius norm in $||\mathbf{X}' - \mathbf{AX}||_F$. The key thing to note is that the rank of $\mathbf{A}$ is at most $m-1$ since it is constructed from a linear combination of the $m-1$ snapshots in $\mathbf{X}'$, which means it is possible to construct the much smaller matrix $\tilde{\mathbf{A}}$, the $r \times r$ projection onto the Principle Component modes.

Algorithm

Broadly, the DMD can be summarized in the following algorithm. More explicit details on implementation are given in the section titled 'Algorithm Implementation and Development'.

1. The Singular Value Decomposition (SVD) is taken of $\mathbf{X}$:

$$\mathbf{X} = \mathbf{USV}^*$$

.

2. $\mathbf{A}$ is given by
$$\mathbf{A} = \mathbf{X'V\Sigma^{-1}U^*}$$
but may be large and intractable, thus we use:
3. $\tilde{\mathbf{A}}$ is given by $\mathbf{UAU}^*$ or
$$\tilde{\mathbf{A}} = \mathbf{U^*X'V\Sigma^{-1}}$$
so that the low dimensional system
$$\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k$$
on the PC modes can be reconstructed via
$$\mathbf{x}_k = \mathbf{U}\tilde{\mathbf{x}}_k$$
4. Compute the Eigendecomposition of $\tilde{\mathbf{A}}$:
$$\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W\Lambda}$$
where the columns of $\mathbf{W}$ are the eigenvectors and the diagonal of $\mathbf{\Lambda}$ are the eigenvalues λ_k.
5. The eigendecomposition of $\mathbf{A}$ can be reconstructed from (4). First, the eigenvalues of A are also the diagonal of $\mathbf{\Lambda}$. The eigenvectors are given by the columns of $\mathbf{\Psi}$:
$$\mathbf{\Psi} = \mathbf{X'V\Sigma^{-1}W}$$

Finally, we can now project the DMD solution into the future using
$$x(t) = \mathbf{\Psi}\exp{(\mathbf{\Omega}t)}\mathbf{b}$$
where the initial condition $\mathbf{b}$ is given by
$$\mathbf{b} = \mathbf{\Phi}^{\dagger}x_1$$
and $\omega_k = \ln\lambda_k/\Delta t$ in $\mathbf{\Omega}$.

Video Analysis

The DMD can be applied to video analysis by supposing that background areas of a video satisfy $||\omega_p|| \approx 0$ for one $p \in \{1, 2, \cdots l\}$. This assumption is made on the basis that $\text{Re}(\omega_p)$ controls the decay or expansion of pth frequency and $\text{Im}(\omega_p)$ controls its oscillatory frequency. Then we can define
$$\mathbf{X}_{\text{Foreground}} = \sum_{j\neq p} b_j\boldsymbol{\phi}_j e^{\omega_j t}$$
and
$$\mathbf{X}_{\text{Background}} = b_p\boldsymbol{\phi}_p e^{\omega_p t}$$
and importantly make the assumption that $\mathbf{X}_{\text{Foreground}}$ is a sparse matrix. From the above, we construct the matrices in the following way:
$$\mathbf{X}_{\text{Foreground}} = \mathbf{X} - |\mathbf{X}_{\text{Background}}|$$
The only problem is that although $\mathbf{X}$ is a real matrix and is equal to the sum of the foreground and background matrices, it is possible that the foreground and background matrices contain negative values which don't make sense in the context of image reconstruction. We can solve this through the creation of the negator matrix $\mathbf{R}$ which contains all of the negative values in $\mathbf{X}_{\text{Foreground}}$. We then correct the matrices like so:
$$|\mathbf{X}_{\text{Background}}| + \mathbf{R} \to \mathbf{X}_{\text{Background}}$$
and
$$\mathbf{X}_{\text{Foreground}} - \mathbf{R} \to \mathbf{X}_{\text{Foreground}}$$
so that the results make sense as images. This perturbation is only so that the results are real and positive; they in no way improve image quality.

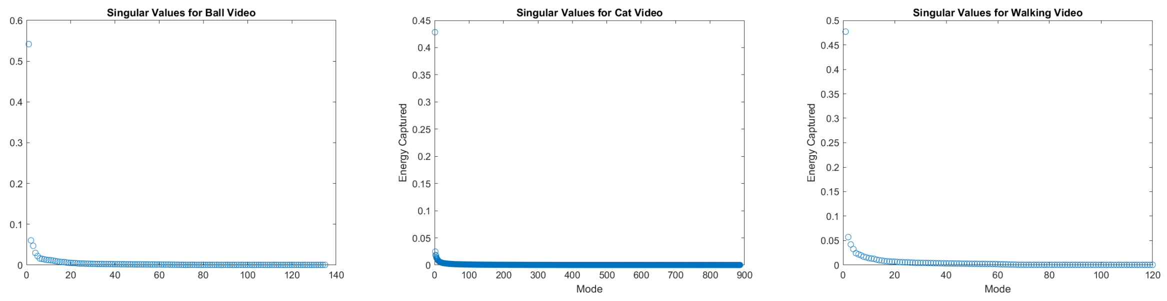


Figure 1: Singular Values for Each of the Videos. 20 modes capture 99% of the energy in all of the videos, but most were well approximated even with 5.

III Algorithm Implementation and Development

III.I Algorithm 1: DMD

The Dynamic Mode Decomposition is the main algorithm of this paper. The goal is to take data snapshots `X1` and `X2`, a goal truncation of `r`, and the time-step (between `X1` and `X2`) `dt` and perform the DMD, thereby extracting the matrix `Phi`, the vector of complex values `omega`, the vector of eigenvalues `lambda`, the vector of initial conditions (or magnitudes of `Phi`) `b`, the reconstructed matrix `Xdmd`, and the singular value vector `s`. Implementation is as follows:

1. Perform the SVD on `X1`. More details on the sVD can be found in papers 3 and 4, suffice it to say that in Matlab, we might implement it like so:

   ```
   [U, S, V] = svd(X1, 'econ');
   ```

2. Take `r` to be the minimum of the input `r` and the column number of `U` (in the untruncated case).
3. Truncate the columns and rows of `S`, and the columns `U` and `V` so that they are rank `r`.
4. We can compute `Atilde` as $\mathtt{U}^T$`*X2*V/S` where `U`,`S`, and `V` are `r`-truncated.
5. Compute the eigenvalues and eigenvectors of `Atilde` (in Matlab, `eig` can be used) as the diagonal matrix `D` and the columns of `W`. `lambda` is just the diagonal of `D` and `omega` is the natural log of `lambda` divided by `dt`.
6. `Phi` can now be reconstructed as `X2*V/S*W`.
7. The first column of `X1` is used as the initial condition, and we obtain `b = Phi\x1`. Since `Phi` is unlikely to be square, this is equivalent to the multiplication of `x1` with the pseudo-inverse of `Phi`.
8. We reconstruct the time dynamics of the DMD by performing `(b.*exp(omega*t))` for all `t` of interest and storing as the columns of `time_dynamics`.
9. The reconstructed matrix is simply `Xdmd = Phi*time_dynamics`.
10. `s` is simply the diagonal of `S` (prior to truncation).

IV Computational Results

Video Descriptions

Supplemental video descriptions have been included since first, the videos themselves cannot be included in this paper and second, the video resolutions are low in order to speed the computations up.

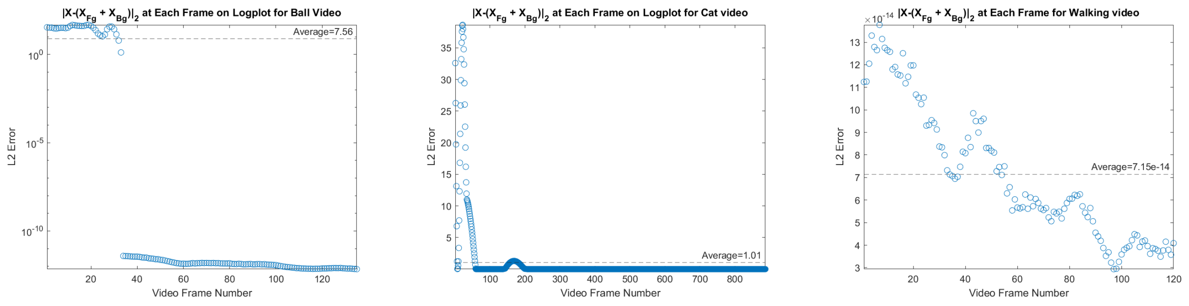


Figure 2: L2 Error at each Frame for Each Video. The error is very low for most of the videos, especially as the video progresses.

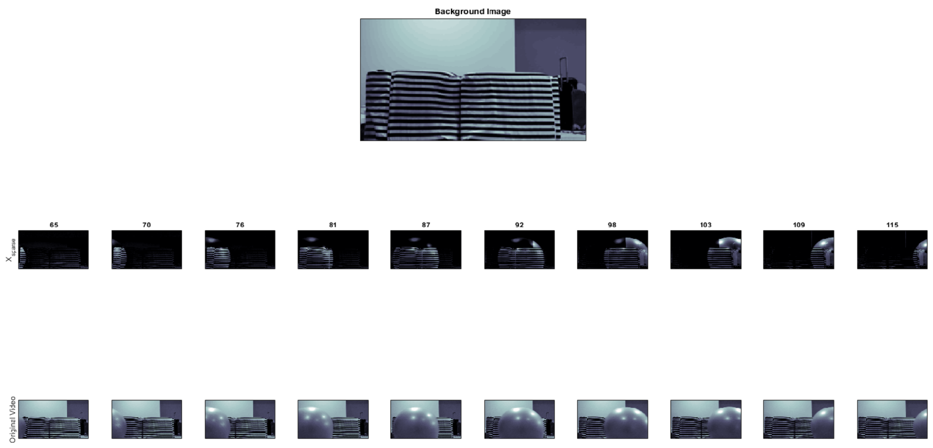
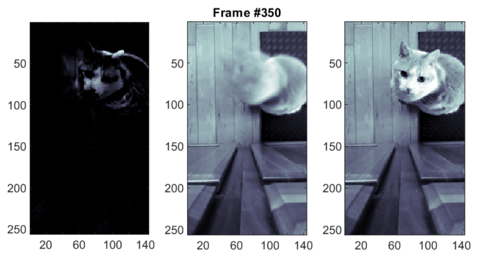


Figure 3: Still Frames from the Ball Video and Extractions. On top is the low rank approximation for **X**, the middle row are frames selected from the video depicting the sparse **X**, and finally the botton row is the original video at the times depicted.

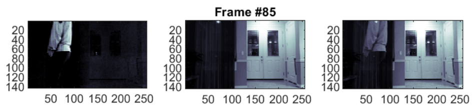
1. **Video 1, "Ball":** A large yoga ball is rolled in front of a background of black and white horizontal stripes. The ball begins off-screen, rolls from left to right dominating the frame, and ends its trajectory off-screen. Pictured in Figure 3.
2. **Video 2, "Cat":** A sitting cat dominates the frame Through the course of a long video, the cat's head moves but not his body. The video is shot from above. Pictured in Figure 4a.
3. **Video 3, "Walk":** A person with a light shirst and dark pants walks from right to left across the frame. The background is complicated, containing a bright door on the right half and dark set of curtains on the left. Pictured in Figure 4b.

Results

Figures shown are of video frame stills where motion is extracted, including the $\mathbf{X}_{\text{Foreground}}$, $\mathbf{X}_{\text{Background}}$, and original images. As we can see from the error results (Figure 2), not only are these approximations good in an L2 sense, but also from the other figures we see that motions of interest are extracted. In general there is some error at the beginnings of the videos, but this smooths out over the course of the videos. Additionally, in general the DMDs provide significant dimensionality reduction with only a dozen modes needed rather than hundreds (Figure 1). Finally, there are clear drawbacks to using this method with complicated foreground objects on complicated background designs. The texture of the background can be seen through the silhouette of the moving foreground, especially the more compicated these shapes become.



(a) Still Frames from the Cat Video and Extractions.



(b) Still Frames from the Walk Video and Extractions.

Figure 4: Stills from two of the videos. In each of (a) and (b), the leftmost image is the sparse representation of **X**, the middle is the low rank **X**, nad the right is the original video. The Cat video, (a), is the only example of a video in this set where the foreground object stayed in frame for the duration of the video. It is also the only video where a blurred image of the foreground appears in the background.

While this creates a very nice visual effect, this is not practical for many applications, so reservations need to be made about the utility of the raw DMD approach. One way of improving on the approach is to look to where motion is not occurring and using that as a mask on the original video. In others, changing contrast or inverting pixel colors can work.

Finally, the omega values (Figure 5) are of some note due to their symmetry property, although this may only save marginal computational cost. It is also worth noting that, at least for the videos in the dataset, there is an omega for which our assumption that omega is close to 0 holds. However, it is important to note that the rest of the omegas have real parts that are slightly negative and so would decay in future state prediction, leading to only the dominant background image in the long term.

V Summary and Conclusions

The algorithm is especially excellent at capturing the edges of moving objects, but when backgrounds and foregrounds are complicated, the results vary from the expected clean separation of background and foreground. Instead, the complicated backgrounds may shine through the motion of the foreground. Additionally, the omegas are exhibit paired behaviour which is of further interest. This method is optimal in high contrast situations, situations where the object of interest is quite bright with respect to the background, or situations where the edges of the object are well defined. Thus, this method may be used in conjunction with others such as the wavelet transform that can yield high contrast.

Appendix A MATLAB functions

Function 1

`DMD.m`
Computes the DMD as described in Algorithm Implementation and Development.

```
function [Phi,omega,lambda,b,Xdmd,s] = DMD(X1,X2,r,dt)
% function [Phi,omega,lambda,b,Xdmd,s] = DMD(X1,X2,r,dt)
% Computes the Dynamic Mode Decomposition of X1, X2
%
% CODE FROM DR. KUTZ (DMD NOTES)
```

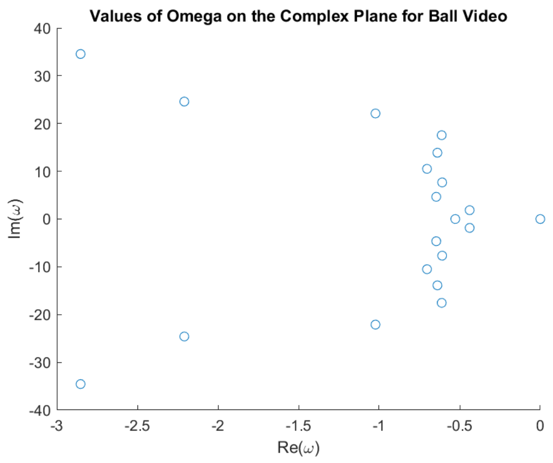


Figure 5: The ωs seem to exhibit a nice symmetry along the imaginary axis. Only two omegas will not be paired in this way, and one of those will be the one closest to $\mathbf{0}$. Similar patterns are exhibited in other ω spectra plots.

```
% with minor modifications
%
% INPUTS:
% X1 = X, data matrix
% X2 = X', shifted data matrix
% Columns of X1 and X2 are state snapshots
% r = target rank of SVD
% dt = time step advancing X1 to X2 (X to X')
%
% OUTPUTS:
% Phi, the DMD modes
% omega, the continuous-time DMD eigenvalues
% lambda, the discrete-time DMD eigenvalues
% b, a vector of magnitudes of modes Phi
% Xdmd, the data matrix reconstrcted by Phi, omega, b
%% DMD
[U, S, V] = svd(X1, 'econ');
r = min(r, size(U,2));
U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
Atilde = U_r' * X2 * V_r / S_r; % low-rank dynamics
[W_r, D] = eig(Atilde);
Phi = X2 * V_r / S_r * W_r; % DMD modes
lambda = diag(D); % discrete-time eigenvalues
omega = log(lambda)/dt; % continuous-time eigenvalues
%% Compute DMD mode amplitudes b
x1 = X1(:, 1);
b = Phi\x1;
%% DMD reconstruction
mm1 = size(X1, 2); % mm1 = m - 1
time_dynamics = zeros(r, mm1);
t = (0:mm1-1)*dt; % time vector
for iter = 1:mm1
```

```
    time_dynamics(:,iter)=(b.*exp(omega*t(iter)));
end
Xdmd = Phi * time_dynamics;

s = diag(S);
end
```

Appendix B MATLAB codes

hw5rgbvid2X.m:

```
% This script processes the videos to make them grayscale and have
% reasonable aspect ratio

% Phone Pixel Aspect Ratio: 1920x1080p
% This is 2 million color pixels per video frame
% Can record in 1280x720p
% 921600 color pixels

% These aspect ratios are both 16:9
% Choose 256x144, reasonable 16:9 aspect ratio

clear variables; close all; clc;

videos = {'20190202_143602', '20190312_125426', '20190312_181033', '20190312_195723', '20190313_165031'
vidfmt = '.mp4';
savfmt = '.mat';

w = 256;
h = 144;

for video = videos

    vid = [cell2mat(video) vidfmt];
    v = VideoReader(vid);

    numframes = double(int16(fix(v.FrameRate*v.Duration)));

    X = zeros([h*w numframes]);

    for j = 1:numframes
        frm = readFrame(v);
        gryfrm = rgb2gray(frm);
        ar = [h w];
        switch cell2mat(video)
            case '20190312_125426'
                ar = [w h];
        end
        Xfrm = imresize(gryfrm, ar);
        imshow(Xfrm); drawnow
        X(:,j) = double(reshape(Xfrm, [h*w 1]));
        if ~hasFrame(v)
            break
        end
    end

    save(['X' cell2mat(video) savfmt],'X')
end
```

hw5.m:

```
clear variables; close all; clc;

%% Video 1: Ball
w = 256;
h = 144;
framerate = 29.9887;
dt = 1/framerate;

Xname = 'X20190312_181033.mat';
% starts at about 70
% ends at about 150
load(Xname)

X = X(:, 20:155);

frames = size(X,2);

r = 20;

tic
X1 = X(:,1:end-1);
X2 = X(:,2:end);
[Phi,omega,lambda,b,Xdmd,s] = DMD(X1,X2,r,dt);
toc

X(:,1) = [];

Xsparse = X - abs(Xdmd);

R = zeros(size(Xsparse));
R(Xsparse<0) = Xsparse(Xsparse<0);

Xsparsedmd = Xsparse - R;
Xlowrankdmd = Xdmd + R;

figure(1)
colormap(bone)
for i = 1:frames-1
    subplot(1,3,1)
    im = Xsparsedmd(:,i);
    imagesc(reshape(im, [h w])); axis image
    subplot(1,3,2)
    imagesc(reshape(real(Xdmd(:,i)), [h w])); axis image
    subplot(1,3,3)
    imagesc(reshape(X(:,i), [h w])); axis image
    title(i)
    drawnow
end
```

```
figure(1)
subplot(3,10,1:10)
imagesc(reshape(real(Xdmd(:,1)), [h w])); axis image
title('Background Image')
xticks([])
yticks([])

frm = fix(linspace(65,frames-21,10));

for j = 10:-1:1
    subplot(3,10,10+j)
    i = frm(j);
    im = Xsparsedmd(:,i);
    imagesc(reshape(im, [h w])); axis image
    title(i)
    xticks([])
    yticks([])
end
ylabel('X_{sparse}')

for j = 10:-1:1
    subplot(3,10,20+j)
    i = frm(j);
    imagesc(reshape(X(:,i), [h w])); axis image
    xticks([])
    yticks([])
end
ylabel('Original Video')

fig11 = figure(11);
plot(s/sum(s), 'o')
title('Singular Values for Ball Video')
saveas(fig11, 'svball.png')

fig111 = figure(111);
scatter(real(omega), imag(omega))
xlabel('Re(\omega)')
ylabel('Im(\omega)')
title('Values of Omega on the Complex Plane for Ball Video')
saveas(fig111, 'omegaball.png')
%%
fig1111 = figure(1111);
E = zeros([1 frames-1]);
for i = 1:frames-1
    E(i) = norm(X(:,i) - (Xsparsedmd(:,i) + Xlowrankdmd(:,i)),2);
end
semilogy(E, 'o');
yline(mean(E), '--', {sprintf('Average=%.2f',mean(E))})
title('|X-(X_{Fg} + X_{Bg})|_2 at Each Frame on Logplot for Ball Video')
ylabel('L2 Error')
xlabel('Video Frame Number')
axis tight
```

```
saveas(fig1111, 'errorball.png')

%% Video 2: Cat
w = 144;
h = 256;
framerate = 29.9887;
dt = 1/framerate;

Xname = 'X20190312_125426.mat';

load(Xname)

frames = size(X,2);

r = 20;

tic
X1 = X(:,1:end-1);
X2 = X(:,2:end);
[Phi,omega,lambda,b,Xdmd,s] = DMD(X1,X2,r,dt);
toc

X(:,end) = [];

Xsparse = X - abs(Xdmd);
R = Xsparse.*(Xsparse<0);
Xsparsedmd = Xsparse - R;
Xlowrankdmd = Xdmd + R;

fig2 = figure(2);
colormap(bone)
for i = 1:frames-1
   subplot(1,3,1)
   im = Xsparsedmd(:,i);
   %im(abs(im)<40) = 0;
   imagesc(reshape(im, [h w])); axis image
   subplot(1,3,2)
   imagesc(reshape(abs(Xdmd(:,i)), [h w])); axis image
   title(sprintf('Frame #%d', i))
   subplot(1,3,3)
   imagesc(reshape(X(:,i), [h w])); axis image
   drawnow
   if i == 350
       saveas(fig2, 'cat.png')
   end
end

fig22 = figure(22);
plot(s/sum(s), 'o')
title('Singular Values for Cat Video')
xlabel('Mode')
ylabel('Energy Captured')
saveas(fig22, 'svcat.png')
```

```
fig2222 = figure(2222);
E = zeros([1 frames-1]);
for i = 1:frames-1
    E(i) = norm(X(:,i) - (Xsparsedmd(:,i) + Xlowrankdmd(:,i)),2);
end
plot(E, 'o');
yline(mean(E), '--', {sprintf('Average=%.2f',mean(E))})
title('|X-(X_{Fg} + X_{Bg})|_2 at Each Frame for Cat video')
ylabel('L2 Error')
xlabel('Video Frame Number')
axis tight

saveas(fig2222, 'errorcat.png')


%% Video 3: Walking
w = 256;
h = 144;
framerate = 29.9887;
dt = 1/framerate;

Xname = 'X20190312_195723.mat';

load(Xname)
X = X(:,100:end-60);

frames = size(X,2);

r = 5;

tic
X1 = X(:,1:end-1);
X2 = X(:,2:end);
[Phi,omega,lambda,b,Xdmd,s] = DMD(X1,X2,r,dt);
toc

X(:,end) = [];

Xsparse = X - real(Xdmd);
R = zeros(size(Xsparse));
R(Xsparse<0) = Xsparse(Xsparse<0);
Xsparsedmd = Xsparse - R;
Xlowrankdmd = Xdmd + R;

fig3 = figure(3);
colormap(bone)
for i = 1:frames-1
   subplot(1,3,1)
   im = Xsparsedmd(:,i);
   imagesc(reshape(im, [h w])); axis image
   subplot(1,3,2)
   imagesc(reshape(real(Xdmd(:,i)), [h w])); axis image
   title(sprintf('Frame #%d', i))
   subplot(1,3,3)
   imagesc(reshape(im+real(Xdmd(:,i)), [h w])); axis image
   drawnow
   if i == 85
       saveas(fig3, 'walk.png')
   end
end

fig33 = figure(33);
plot(s/sum(s), 'o')
title('Singular Values for Walking Video')
xlabel('Mode')
ylabel('Energy Captured')
saveas(fig33, 'svwalk1.png')
%%
fig3333 = figure(3333);
E = zeros([1 frames-1]);
for i = 1:frames-1
    E(i) = norm(X(:,i) - (Xsparsedmd(:,i) + Xlowrankdmd(:,i)),2);
end
plot(E, 'o');
yline(mean(E), '--', {sprintf('Average=%.2e',mean(E))})
title('|X-(X_{Fg} + X_{Bg})|_2 at Each Frame for Walking video')
ylabel('L2 Error')
xlabel('Video Frame Number')
axis tight

saveas(fig3333, 'errorwalk.png')

%% Video 4: Walking
w = 256;
h = 144;
framerate = 29.9887;
dt = 1/framerate;

Xname = 'X20190313_165031.mat';

load(Xname)

frames = size(X,2);

r = 40;

tic
X1 = 255-X(:,1:end-1);
X2 = 255-X(:,2:end);
[Phi,omega,lambda,b,Xdmd,s] = DMD(X1,X2,r,dt);
toc

X(:,end) = [];

Xdmd = Xdmd;

Xsparse = X - real(Xdmd);
R = zeros(size(Xsparse));
```

```
R(Xsparse<0) = Xsparse(Xsparse<0);
Xsparsedmd = Xsparse - R;
Xlowrankdmd = Xdmd + R;

figure(4)
colormap(bone)
for i = 1:frames-1
   subplot(1,3,1)
   im = Xsparsedmd(:,i);
   imagesc(reshape(im, [h w])); axis image
   subplot(1,3,2)
   imagesc(reshape(real(Xlowrankdmd(:,i)), [h w])); axis image
   subplot(1,3,3)
   imagesc(reshape(X(:,i), [h w])); axis image
   title(i)
   drawnow
end

figure(44)
plot(s)
```