AMATH 482 HW 5

Liyuan Tang

March 2019

Abstract

In this homework, we are going to use the Dynamic Mode Decomposition method to take a video clip containing a foreground and background object and separate the video stream to both the foreground video and a background. I downloaded three videos online and used some pieces to test the algorithm.

1 Introduction

I used three pieces of video downloaded from Youtube. The first one is from Beijing 2008 Olympics Badminton Men's Singles final[3], I used the clip from 27:01 to 27:12. The second one is a clip from the collection of Xu Xin's table tennis competitions, I used the clip from 0:20 to 0:31[1]. The last one is Professor Kutz's talk on 2017 Workshop on Brain Dynamics and Neurocontrol Engineering at Washington University in St. Louis[2]. I used the clip from 3:00-3:11. For each video, I will apply the DMD method to seperate the foreground video and the background. Then I will plot the image of the original video, foreground and background at some specific frames to see how DMD works and compare the results for three videos.

2 Theoretical Background

2.1 Dynamic Mode Decomposition

DMD is a powerful technique to discover dynamical systems from high dimensional data [4]. It constructs the best linear fit model dynamical system to the nonlinear dynamical system which contains the data. Considering we are using the data collected from

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t; \mu), \tag{1}$$

where $\mathbf{x}$(t) is a vector that represents the dynamical system at time t, with parameter μ in the system. The DMD procedure constructs the linear dynamical system as

$$\frac{d\mathbf{x}}{dt} = \mathcal{A}x. \tag{2}$$

And we can construct an analogous discrete-time system:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k \tag{3}$$

where $\mathbf{x}_k = \mathbf{x}(k\Delta t)$ as the column of $\mathbf{x}$ at a specific time frame and $\mathbf{A} = \exp(\mathcal{A}\Delta t)$. $\mathcal{A}$ is the matrix in the continuous-time dynamics in Equation (2). And we can solve this system by using the eigenvalues

and eigenvectors of the discrete time map $\mathbf{A}$:

$$x_k = \sum_{j=1}^{r} \phi_j \lambda_j^k b_j = \mathbf{\Phi \Lambda}^k \mathbf{b} \tag{4}$$

where λ_j and ϕ_j are the eigenvalues and eigenvectors, $\mathbf{b}$ are the coefficients of the initial condition $\mathbf{x}_1$ in the eigenvector basis. The DMD algorithm generates a low-rank eigen-decomposition of the matrix $\mathbf{A}$ which extracts key low-rank features of high-dimensional systems. It can be interpreted as a regression of data onto locally linear dynamics $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$ that $\mathbf{A}$ minimizes $||\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k||_2$ and k is 1 to $m-1$.

As a result, it is possible to arrange m snapshots into 2 large matrices:

$$\begin{aligned} \mathbf{X} &= [\mathbf{x} \quad \mathbf{x}_2 \quad \mathbf{x}_3 \cdots \mathbf{x}_{m-1}] \\ \mathbf{X}' &= [\mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \cdots \mathbf{x}_m]. \end{aligned} \tag{5}$$

So we can write the Equation (3) as X' $\approx$ AX, and the best fit matrix A is $\mathbf{A} = X'X^\dagger$ where $X^\dagger$ is the Moore-Penrose pseudo-inverse. However, the matrix $\mathbf{A}$ might be high-dimensional so we may hard to represent or decompose. As a result, we should project the data to a low-rank subspace defined by POD modes and solve for low-dimensional $\tilde{A}$ instead of directly computing $\mathbf{A}$ which is what I am going to do in next section.

3 Algorithm Implementation and Development

First, we need to load the data from the video. I used "X = VideoReader('badminton.mp4'); " which creates object X to read video data. I created an empty matrix "video" to store my data. Then, I used a while loop to extract all the data which is "while hasFrame(X)". If there is a video frame available to read from the file, then it will read available video frame by using "data = readFrame(X);". Then, I converted it under grayscale by using "data = rgb2gray(data)" and reshaped it to a column vector by using "data = data(:). Finally, I store the data in the matrix "video", each column of "video" indicates the data at different frame. After extracting all data, I convert the "video" from uint8 to double precision by using "video = double(video)".

I separated the matrix in two matrices as "X1 = video(:, 1:end-1); X2 = video(:, 2:end)" like the Equation (5) for next calculation. I generated some variables for convenience such as the width and height of the video, and number of frames by checking the number of columns the matrix. I calculated *dt* by using 1 / X.FrameRate which determined the time between two frames.

Then, I took the singular value decomposition of X1 which generated $\mathbf{U}$, Σ and $\mathbf{V}$. I plotted the energy percentage by using the diagonal of Σ like the last two homework which helps me to determine the rank r to truncate those three matrix. So the SVD performs a low-rank truncation of the data. Then, I computed $\tilde{A}$ which is the r $\times$ r projection of the full matrix A on the modes instead of calculating the full matrix A in order to increase efficiency.

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

In Matlab, I used "Atilde = Ur'*X2*Vr / Sr". Then, I calculated the matrix of eigenvalue and eigenvectors of by using "[W, D] = eig(Atilde);". We can reconstruct the eigendecomposistion of A using W and D with the equation:

$$\Phi = X' V \Sigma^{-1} W. \tag{7}$$

Φ is the matrix whose columns are the DMD modes. In Matlab, I used "Phi = X2*Vr/ Sr*W". I created

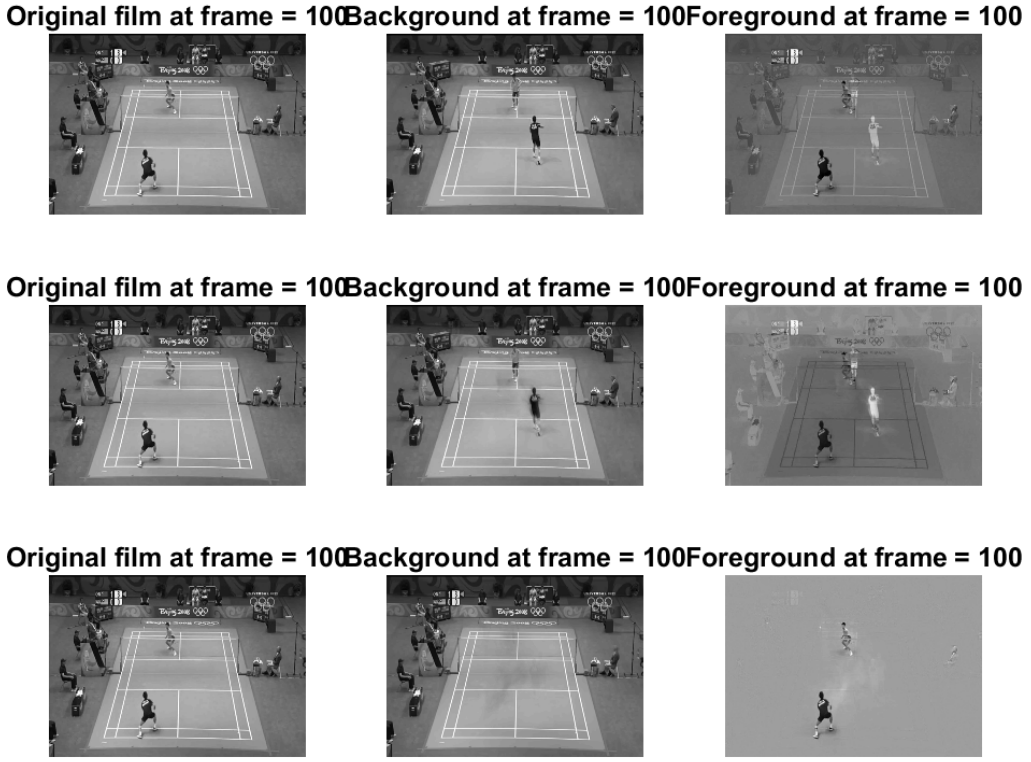

Figure 1: Comparison between different rank. The first row used full rank. The second row used r = 5. The last row only used r=1.

some variables for convenience: "lambda = diag(D); omega = log(lambda)/dt". So lambda is discrete-time DMD eigenvalues and omega is the continuous-time DMD eigenvalues. If the absolute value of omega is close to 0, it indicates the background. So I sorted the absolute value of omega, in order to find some small value and their index by using "[val, ind] = sort(abs(omega));". Then, we can approximate the result at time t by the equation

$$x(t) \approx \sum_{k=1}^{r} \phi_k e^{\omega_k t} b_k = \mathbf{\Phi e^{\Omega t} b} \tag{8}$$

where b is the vector of initial coefficient b_k.

$$\mathbf{b} = \Phi^{\dagger} x_1. \tag{9}$$

$\Phi^{\dagger}$ is the pseudo-inverse of Φ and x_1 is the first column in the data. At this point, we have all the data to implement Equation (8), but instead of calculating the result at time t, I tried to approximate the result at all future times by using matrices. I created the vector t to represent time. Then I used a for loop from 1 to length(t). During each iteration, I used "temp = (b.*(exp(omega*t(iter)))).'); time_dynamics(:,iter) = sum(temp, 2)" which gave me $e^{\Omega t} b$ at a specific time and stored it in a column in the matrix "time_dynamics". At the end, I used "X_lowRank = Phi*time_dynamics;" which is the resulting matrix, and each column of X_lowRank represents the solution at a specific time t.

Then, consider the equation

$$\mathbf{X_{DMD}} = b_p \varphi_p e^{\omega_p t} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t} \tag{10}$$

where the first part is background and second part is the moving foreground. We can approximate the low rank reconstruction of the DMD by

$$\mathbf{X}_{\text{DMD}}^{\text{Low Rank}} = b_p \varphi_p e^{\omega_p t} \tag{11}$$

which we already calculated above. Due to

$$\mathbf{X} = \mathbf{X}_{\text{DMD}}^{\text{Low Rank}} + \mathbf{X}_{\text{DMD}}^{\text{Sparse}}, \tag{12}$$

the DMD's approximate sparse reconstruction is

$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \sum_{j \neq p} b_j \varphi_j e^{\omega_j t} = \mathbf{X} - |\mathbf{X}_{\text{DMD}}^{\text{Low Rank}}| \tag{13}$$

in order to get real-valued elements. I calculated X_sparse which is "X_sparse = video - abs(X_lowRank)".

I started with full rank which r = length(S), but it didn't give me a good result (First row in Figure 1). So I chose several r and used different thresholds for the omega (Figure 1). In the end, I found $r = 1$ gives me the best result, with only one omega.

In the spec, it also says that if there are some negative values in "X_sparse", we should add them to a new matrix $\mathbf{R}$. Then, we should modify the matrices by

$$\begin{aligned} \mathbf{X}_{\text{DMD}}^{\text{Sparse}} &= \mathbf{X}_{\text{DMD}}^{\text{Sparse}} - R, \\ \mathbf{X}_{\text{DMD}}^{\text{Low Rank}} &= R + |\mathbf{X}_{\text{DMD}}^{\text{Low Rank}}|. \end{aligned} \tag{14}$$

However, when I used "pcolor" to visualize my results, I think the image is better without modifying data (Figure 2). So I chose not to modify my data. Finally I used "videoPlayer = vision.VideoPlayer" to play the video based on my data of "X_sparse" and "X_lowRank". And I plotted the videos at some specific frames to compare the results.

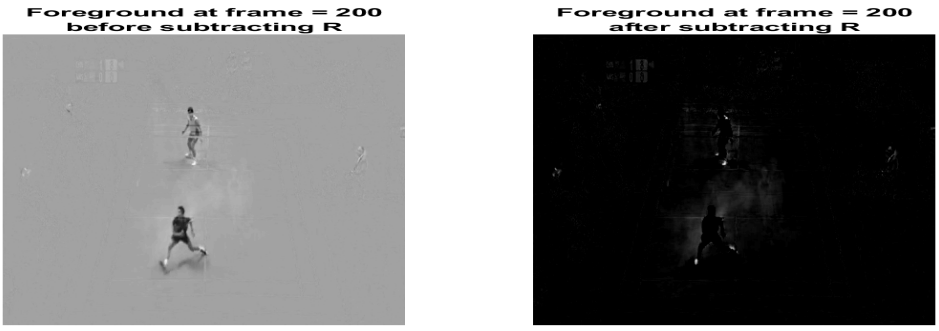

Figure 2: Foreground before and after subtracting R, the residual negative values. Left image is made before subtracting R, right image is made after subtracting R

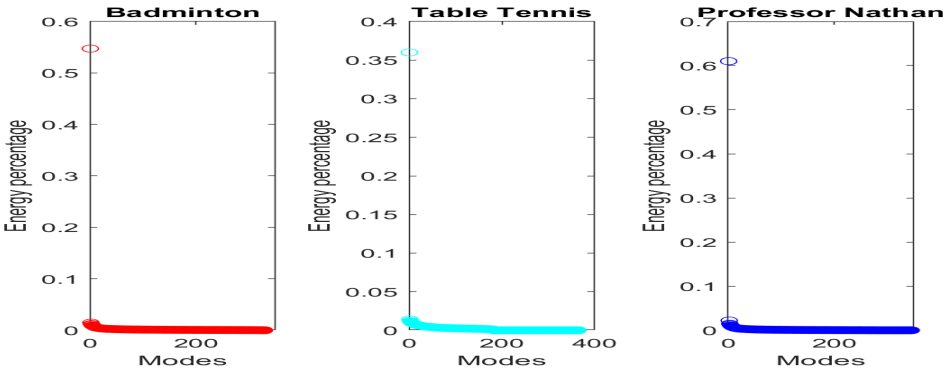

Figure 3: Singular values for three videos from SVD

4 Computational Results

Comparing the results in Figure 4, 5 and 6. I found the video of badminton and table tennis extracted a clear static background. They also generated a clear foreground of the athletes. But in the video of Professor Kutz, there are still some shadows remaining in the background while the extraction of the foreground is relatively clear. This might because he stood at that place for a while which confused the algorithm. But in the first two cases, athletes are always moving and have large body movements so the results are relatively better.

Overall, the algorithm did pretty well on those three videos, especially for the first two cases. It extracted the moving people from the static background. And it generated a relatively clear background and tracked the movements of people as the foreground.

5 Conclusion

In this homework, I applied Dynamic Mode Decomposition to three videos to separate the moving objects as the foreground and the static background. The first video is from the finals of badminton

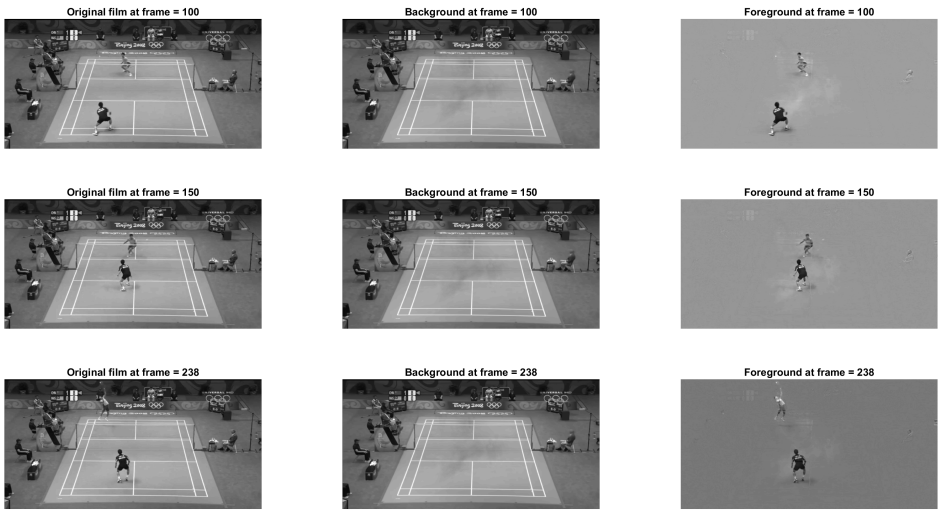

Figure 4: The original video, background and foreground of the badminton video at 100th, 150th and 238th frame. Each row represents the figure in the same frame

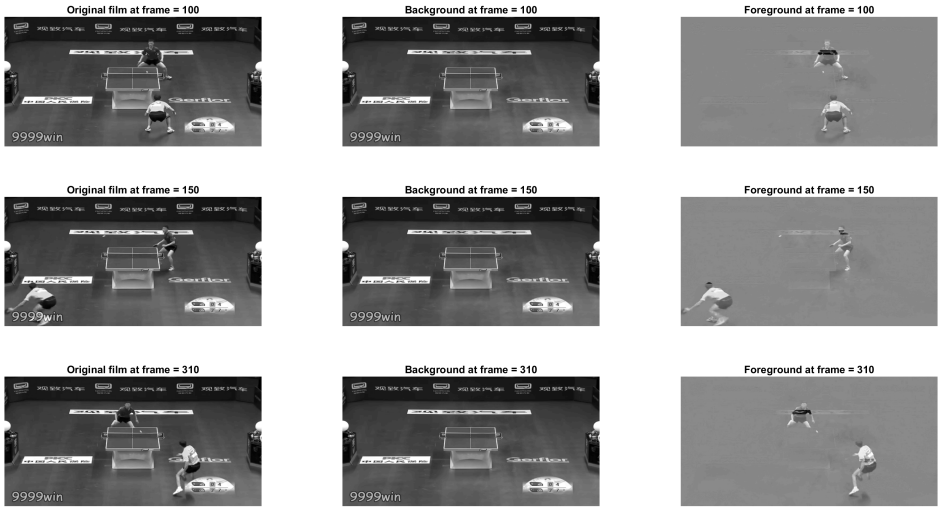

Figure 5: The original video, background and foreground of the table tennis video at 100th, 150th and 310th frame. Each row represents the figure in the same frame

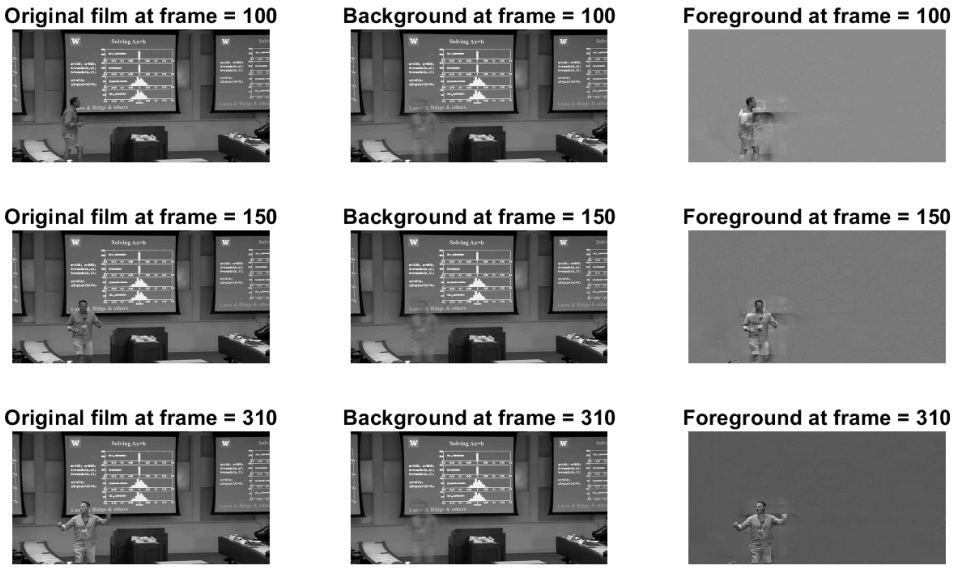

Figure 6: The original video, background and foreground of the Professor Kutz's video at 100th, 150th and 310th frame. Each row represents the figure in the same frame

competition, the second video is from a table tennis competition and the last video is from Professor Kutz's talk. This algorithm separates the matrix for the background, $\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low\ Rank}}$ and the matrix for the foreground, $\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparse}}$.I plotted the results at three different frames. The it works really well for first two videos since the athletics have large movements all the time so it clearly extract them from the background. However, for the Kutz's video, it remains some shadow in the background. This might because he stood there for a while so it takes that as a part of the background. Overall, this algorithm works pretty well on all three videos.

References

[1] 9999win. xuxin . https://www.youtube.com/watch?v=KvUCtTy8NeE. Jun 28, 2016.

[2] BrainDynamics. J nathan kutz. https://www.youtube.com/watch?v=5LOpkRsqRRw. Jul 26, 2017.

[3] Ivan Jiang. Beijing 2008 olympics badminton msf lin dan vs lee chong wei. https://www.youtube.com/watch?v=5POGztly9B4t=1632s. Nov 27, 2011.

[4] J. Nathan Kutz. *AMATH 582 Computation Methods for Data Analysis.* 2010.

6 Appendix A

1. X = VideoReader(files); It creates object v to read video data from the file named as files. And X contains several information such as X.width, X.height which are the height and width of the video frame, X.FrameRate which is number of frames per second and X.Duration which is the length of the file.

2. hasFrame(X); It determine if frame is available to read. It will return logical 1 if there is a video frame available to read from the file. Otherwise, it returns 0 which is false.

3. readFrame(X); It reads the next available video frame from the file X.

4. [W, D] = eig(Atilde). It returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding eigenvectors.

5. [val, ind] = sort(A); It returns the sorted value from smallest to largest in val and their index in the vector A.

6. videoPlayer = vision.VideoPlayer; It returns a video player object to play a video.

7 Appendix B

```
clear all; close all; clc;
%% load data
X = VideoReader('badminton.mp4');
video = [];
while hasFrame(X)
    data = readFrame(X);
    data = rgb2gray(data);
    data = data(:);
    video = [video data];
end
video = double(video);

width = X.width;
height = X.height;
dt = 1 / X.FrameRate;

X1 = video(:, 1:end-1); X2 = video(:, 2:end);
frame = size(video, 2) - 1;
[U, S, V] = svd(X1, 'econ');
plot(diag(S)/sum(diag(S)), 'ro');

%
r = 1;
Ur = U(:, 1:r);
Sr = S(1:r, 1:r);
Vr = V(:, 1:r);
Atilde = Ur'*X2*Vr / Sr;
[W, D] = eig(Atilde);
Phi = X2*Vr/ Sr*W;
lambda = diag(D);
omega = log(lambda) / dt;
[val, ind] = sort(abs(omega));
omega = omega(ind(1));

%%
t = linspace(0, X.Duration, size(video, 2));
x1 = video(:, 1);
b = Phi\x1;
time_dynamics = zeros(r,length(t));
for iter = 1:length(t)
    temp = (b.*(exp(omega*t(iter))).');
    time_dynamics(:,iter) = sum(temp, 2);
end

X_lowRank = Phi*time_dynamics;
X_sparse = video - abs(X_lowRank);

% % Residuals
% R = zeros(length(X_sparse), size(X_sparse, 2));
%
% for i = 1:length(X_sparse)
%     for j = 1:frame
%         if X_sparse(i, j) < 0
%             R(i,j) = X_sparse(i,j);
%         end
```

```
%     end
% end
% X_sparse = X_sparse - R;
% X_lowRank = abs(X_lowRank) + R;
video_sparse = reshape(X_sparse, [height, width, frame+1]);
video_lowRank = reshape(X_lowRank, [height, width, frame+1]);

%% make video
videoPlayer = vision.VideoPlayer;
for j = 1:frame
    videoFrame = video_sparse(:,:,j);
    %videoFrame = video_lowRank(:,:,j);
    videoPlayer(uint8(videoFrame));
    pause(dt);
end

%% plot results
i = 100;
figure(1)
subplot(3,3,1);
pcolor(flip(reshape(video(:,i), [height, width]))); shading interp; ...
    colormap(gray);axis off;
title('Original film at frame = 100');
subplot(3,3,2);
pcolor(flip(video_lowRank(:,:,i))); shading interp; colormap(gray);axis off;
title('Background at frame = 100');
subplot(3,3,3);
pcolor(flip(video_sparse(:,:,i))); shading interp; colormap(gray);axis off;
title('Foreground at frame = 100');
%% frame = 150;
i = 150;
subplot(3,3,4);
pcolor(flip(reshape(video(:,i), [height, width]))); shading interp; ...
    colormap(gray);axis off;
title('Original film at frame = 100');
subplot(3,3,5);
pcolor(flip(video_lowRank(:,:,i))); shading interp; colormap(gray);axis off;
title('Background at frame = 100');
subplot(3,3,6);
pcolor(flip(video_sparse(:,:,i))); shading interp; colormap(gray);axis off;
title('Foreground at frame = 100');
%% frame = 238
i = 238;
subplot(3,3,7);
pcolor(flip(reshape(video(:,i), [height, width]))); shading interp; ...
    colormap(gray);axis off;
title('Original film at frame = 100');
subplot(3,3,8);
pcolor(flip(video_lowRank(:,:,i))); shading interp; colormap(gray);axis off;
title('Background at frame = 100');
subplot(3,3,9);
pcolor(flip(video_sparse(:,:,i))); shading interp; colormap(gray);axis off;
title('Foreground at frame = 100');
```