AMATH 482 HW 5

Dynamic Mode Decomposition : Video Processing

Zachary McNulty
zmcnulty, ID: 1636402

Abstract: The Dynamic Mode Decomposition (DMD), like the SVD, is useful for decomposing a system into a more fundamental set of components. The components of DMD highlight the spatio-temporal properties of the given system, allowing us to study its dynamics over time and make predictions about the future state of the system. In this paper, we will use the spatio-temporal properties of DMD to separate a video into two components: a high-frequency moving subject and a low-frequency, staionary background.

University of Washington
March 2019

1 Introduction and Overview

A dynamical system is some system whose dynamics over time are dictated by some fundamental set of equations, often a set of ODEs which may include nonlinearities, that rely on some subset of the variables in the system. These systems are found throughout all areas of scientific study, but there are many situations under which the governing equations for a given system are not known. Rather, a data-driven approximation of these dynamics is required to further explore the system and possibly infer its fundamental structure. While the Singular Value Decomposition (SVD) provides a set of principal components that can describe the patterns in the data as well as possible, it does so without considering possible spatio-temporal patterns. Thus, the components found in the basis returned may not be the exact individual signals that make up the input but rather some combination of them. Furthermore, the SVD is restricted in some sense by the requirement that these components be completely orthogonal. Much like the SVD, the Dynamic Mode Decomposition (DMD) seeks to capture the low-rank structure of a given system. By providing a new, carefully chosen coordinate system to map data into, the DMD breaks down a system into a set of fundamental components. In the case of DMD, these components are the underlying spatio-termporal properties of the system. By not requiring that the modes of the system be orthogonal, DMD may be able to extract more information about the fundamental signals of the system. As such, the DMD is a great tool for performing a linear approximation to the dyanmics of the system. In this paper, we use DMD to extract two of the primary spatio-temporal components of a video: its moving subject and its stationary background.

2 Theoretical Background

The Singular Value Decomposition is a diagonalization of a given matrix A that focuses on the rotations/reflections and stretching a vector undergoes when transformed by A. Formally, for any matrix $A \in \mathbb{C}^{m\times n}$ there exists a diagonalization of the form:

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

where $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are unitary matrices, $\Sigma \in \mathbb{R}^{m\times n}_{\geq 0}$ is a diagonal matrix, and V^* represents the complex conjugate transpose of V. As U and V are unitary matrices their columns each form an orthonormal basis for their respective space and the matrices have the convenient property that $UU^* = U^*U = I_m$ and $VV^* = V^*V = I_n$. The diagonal entries of Σ are called the **singular values** of A. Intuitively, this breaks down the transformation A into three fundamental transformations: a rotation/reflection V^* within the the domain of A, a scaling of the components of the space X by Σ, and another rotation/reflection of the space within the codomain of A. To understand the importance of the SVD, first consider the covariance matrix for a data matrix X:

$$C_X = \frac{1}{n-1}XX^T \tag{2}$$

In this matrix, entry (i, j) corresponds to the covariance between row i and row j of X (data measurement i and j). If $i \neq j$, then a high value in entry (i, j) suggests the two data measurements vary in similar ways and are thus likely redundant measurements of the same feature in the data. Consequentially, a low value at (i, j) suggests the two data measurements are fairly independent. If $i = j$, the diagonal of C_X, then entry (i, j) describes the variance in data measurement i. Generally, data measurements with a high variance are assumed to capture important features of the data while those with low variance do not. To see the importance of the SVD, consider the covariance matrix for $Y = U^*X = \Sigma V^*$ where U, Σ, V are from the SVD of X:

$$C_Y = \frac{1}{n-1}YY^T = \frac{1}{n-1}\Sigma^2 \tag{3}$$

As C_Y is diagonal, all its off diagonal entries are zero: projecting the data X onto U^* produces a transformed data set ΣV^* with completely independent measurements, eliminating redundancy in the data. Similarly, if we chose $Y = XV = U\Sigma$ we find the same covariance matrix, showing us $U\Sigma$ also forms an independent set. As the columns of U and V form bases for the codomain and domain of X respectively, each give an ideal, independent coordinate system for each space. If the data samples in X are stored as rows, V gives the ideal coordinate system and U the coordinates of each sample in that system. Else, if the samples are stored as columns then U gives the ideal coordinate system and V the coordinates of each sample in that system. Furthermore, the corresponding singular value σ in the diagonal of Σ ranks the importance of each direction

in this coordinate system based on the data's variance along that direction. Note for any given matrix X we can reconstruct it given these coordinates:

$$X = \sum_{i=1}^{min(m,n)} u_j \sigma_j v_j^* \tag{4}$$

where u_j, σ_j, v_j are the jth column of U, Σ, V respectively. Since the singular values σ_j are a non-increasing sequence, we see each successive term is increasingly less important to the structure of X. Thus, if the singular values decrease rapidly, we can get a good approximation of X with only the first few terms of this sum. This is a low-rank approximation of the matrix X, a simplified form of the dynamics of the system that well-captures its behavior.

The goal of the Dynamic Mode Decomposition is to describe a dynamical system:

$$\frac{dx}{dt} = f(x, t; \mu) \tag{5}$$

in a data driven way where the governing equations, f, are often not known and likely include nonlinearities. Instead, we have a series of measurements of the system in the form:

$$y_k = g(x_k) \tag{6}$$

where x_k is our state variable of interest and our measurements y_k indirectly measure these states. In the most simple case, $y_k = x_k$ and our measurements directly measure the system's state.

$$X = \begin{bmatrix} x_1 & x_2 & ... & x_{n-1} \end{bmatrix} \qquad X' = \begin{bmatrix} x_2 & x_3 & ... & x_n \end{bmatrix} \tag{7}$$

$$\frac{dx}{dt} = Ax \rightarrow AX \approx X' \rightarrow A \approx X'X^\dagger \tag{8}$$

DMD begins by developing the best linear approximation to the system as possible, as we see in **Equations 7,8** above. By arranging the state of our system x_t at each point in time as the columns of X and X', we can stagger these two matrices to encode the transition in time between two states x_i and x_{k+1}. Then, the matrix A that can best take column n in X, x_k, to column n in X', x_{k+1}, will be the best linear approximation of the dynamics of our system: for every possible state, it explains as much of the transition to the next state in a linear way as possible. Given the linear system in **Equation 8**, the solution is well-known to be:

$$x(t) = \sum_{k=1}^{n} b_k \phi_k e^{\omega_k t} \tag{9}$$

where ϕ_k and ω_k are the corresponding eigenvectors/eigenvalues of A respectively and b_k are the coefficients of the intial condition in the eigenvector basis: $\Phi b = x(0) = x_1$. Rather than work with an arbitrary coordinate system however, we can project our data onto the principal components obtained via SVD. Note that if $X = U\Sigma V^*$ then:

$$A = X'X^\dagger = X'V\Sigma^{-1}U^* \tag{10}$$

When X is tall and skinny, A is a massive matrix so it is only computationally feasible to work with the matrix A constructed from a low-rank approximation of X, projecting A onto the first r principal components to make $\tilde{A}$.

$$\tilde{A} = U_r^* A U_r = U_r^* X' V_r \Sigma_r^{-1} \tag{11}$$

From here, we can construct the eigendecomposition of our dynamics matrix A from our low-rank approximation.

$$\tilde{A}W = W\Lambda \tag{12}$$

$$\tilde{\Phi} = X'V_r\Sigma_r^{-1}W \tag{13}$$

As we can see from **Equation 9**, eigenvalues with positive real parts cause system to drastically change in time. In these situations, the model poorly predicts future events outside the time-frame of the data and some kind of data transformation may be required.

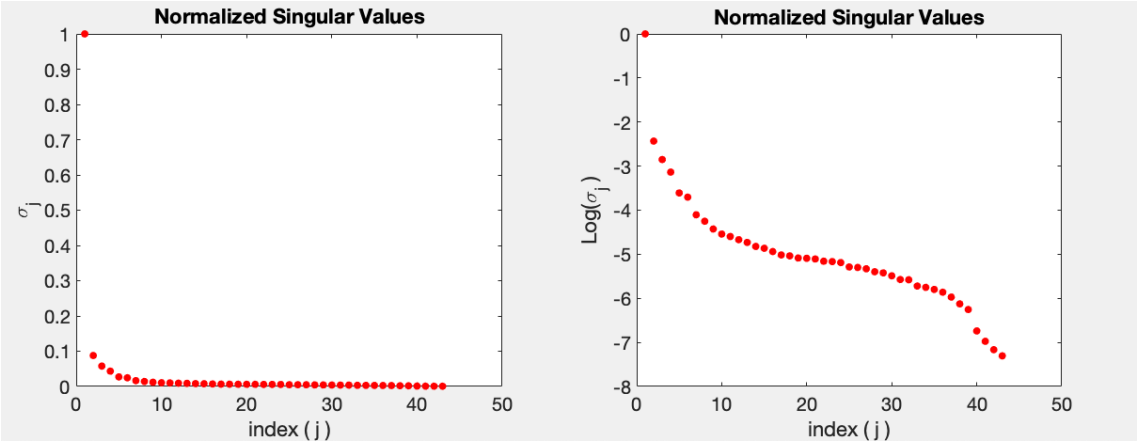

Figure 1: Singular values for the data matrix X storing each frame of the car video. As we can see, there is a single dominant mode and all modes past index 10 are below the level of machine precision at 10^{-5}

3 Algorithm Implementation and Development

1. Load the video and pre-allocate memorfor storing frames (Appendix B: code 12).
2. Iterating through each frame of the video, convert the frame to black & white, reshape it into a column vector, and store it as a column of our data matrix X (Appendix B: code 14-48).
3. Take the DMD of the data matrix X
   (a) Generate the staggered matrices $X1, X2$ where columns $x_j^1 = x_{j+1}^2$ from $X1, X2$ respectively and take the SVD of $X1$ (Appendix B: code 58).
   (b) Plot the singular values and choose a low-rank approximation that captures at least 90% of the energy of the system (Appendix B: code 62-79).
   (c) Using the low-rank approximation of $X1$, calculate $\tilde{A}$. From $\tilde{A}$ calculate the relevant Φ and its associated frequencies $\vec{\omega}$ as well as the initial conditions $\vec{y_0}$ (Appendix B: code 82-90).
4. Find the $\omega_i \in \vec{\omega}$ such that $||\omega_i|| \approx 0$, the minimum frequency. This low frequency will be the stationary background (Appendix B: code 93-94).
5. Calculate the low-rank DMD $X\_lowrank$ and the sparse DMD X_sparse (Appendix B: code 100-101).
6. To compensate for the fact that X_sparse may have negative entries, and negative pixel values do not make sense, generate a matrix R that contains these negative entries. Set these entries to zero and add the corresponding entries to $X\_lowrank$ to ensure that $|X\_lowrank| + X\_sparse = X$ still holds (Appendix B: code 103-108).
7. Plot the corresponding low-rank and sparse DMD deconstructions of the data matrix X (Appendix B: code 114-153).
8. Repeat this process again after inverting the pixel values before performing DMD. Revert the pixels prior to plotting. We have found this sometimes improves the performance of DMD in this case.

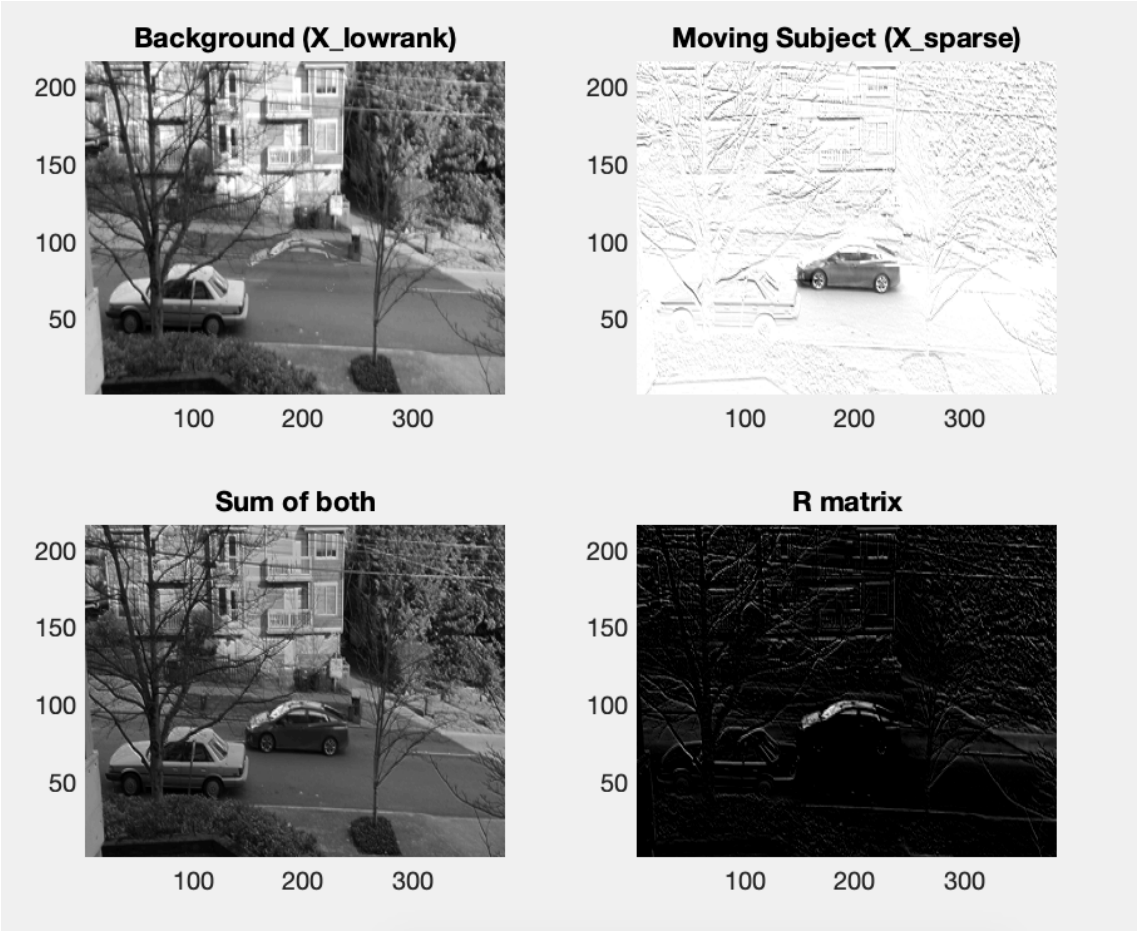

Figure 2: DMD decomposition of car video into two parts: high frequency areas X_sparse (moving objects) and low frequency areas X_lowrank (background). Interestingly, the former matrix captures many of the edges in the video. The R matrix is the correction matrix mentioned in step 6 of the implementation.

.

4 Computational Results

The first video we tried this technique on was a short clip of a single car traveling down a street. For this video, we found that working with the inverted pixel values produced a better separation of foreground and background. Once loading in this video and placing each frame into a column of our data matrix, we took the SVD of that matrix. This gave us the singular values seen in **Figure 1**. Clearly, there is a single dominant singular value, but a few of the others may be relevant. Either way, we can see from the log plot graph that the singular values beyond index 15 are below machine precision of 10^{-5}. A quick check shows these singular values capture over 90% of the energy of the system. Thus, we decided to use a rank 15 approximation of our system after taking the SVD. In **Figure 2**, we see the results of the DMD decomposition. Interestingly enough, not only does the sparse reconstruction capture the moving car, but it also captures many of the edges of the scene. This is likely because the video was taken from a hand-held phone and the camera shake created the illusion that these areas are moving and thus not part of the low frequency background. Furthermore, the R matrix we used to store negative entries in X_sparse also appears to capture the car quite well. In fact, we can still somewhat of a mirage of the car still in the background. It is possible that this feature was added to the background when adding R to $X\_lowrank$ as part of part 5 of our implementation. By setting $R = 0$, we saw that this mirage vanishes, supporting the idea that this artifact is being added to the background by our procedure.

The next video follows several birds all trying to get a meal from a conveniently placed pile of seeds. We can see in **Figure 3** that we have similar behavior in the singular values as in the previous case: a fairly dominant first singular value followed by a quick decline in magnitude. Once

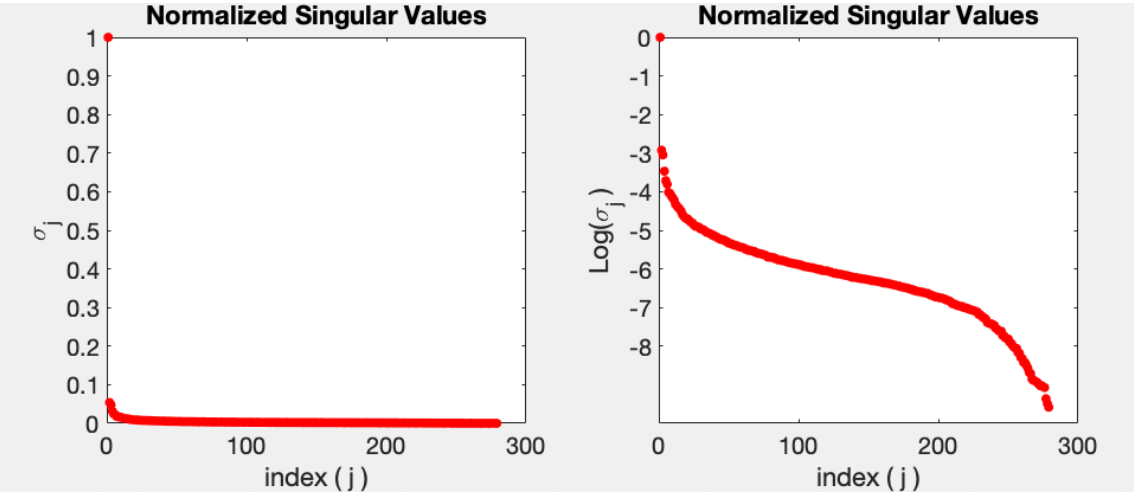

Figure 3: Singular values for the data matrix X storing each frame of the car video. As we can see, there is a single dominant mode and all modes past index 10 are below the level of machine precision at 10^{-5}

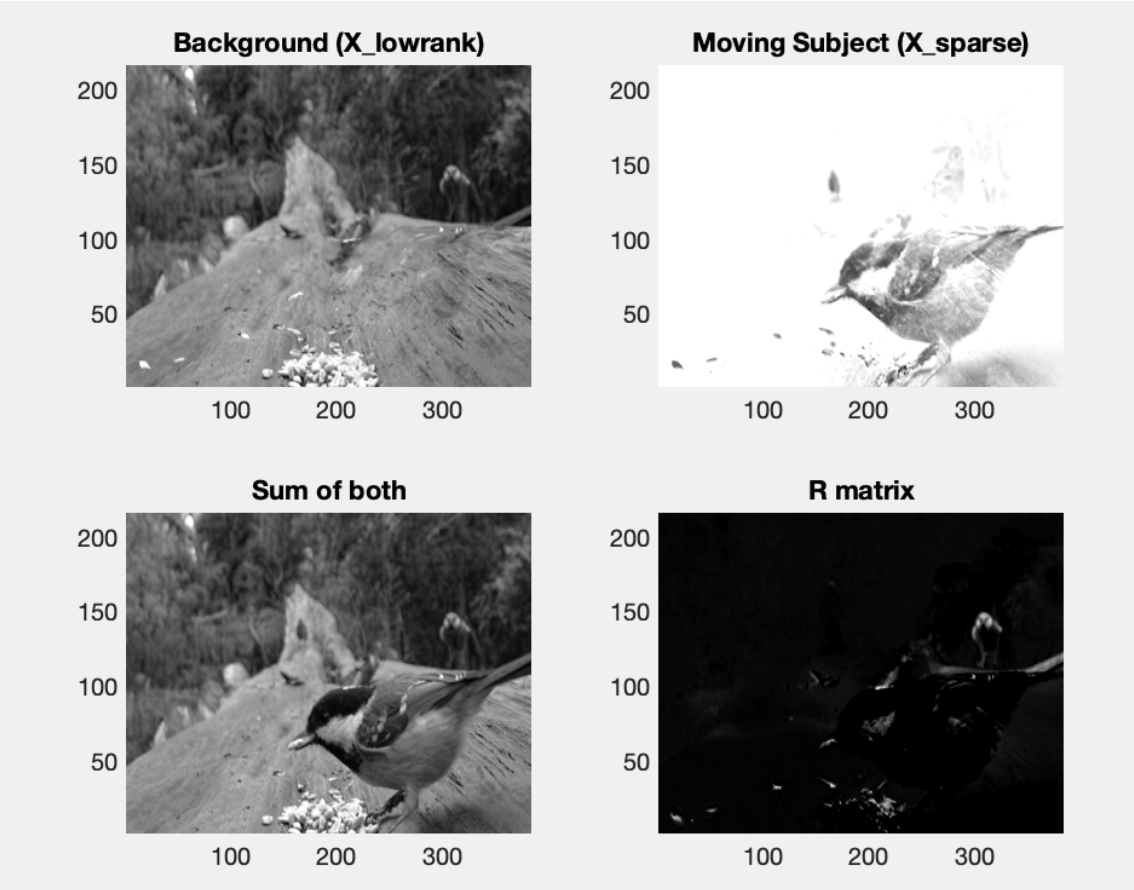

Figure 4: DMD decomposition of bird video into two parts: high frequency areas X_sparse (moving objects) and low frequency areas X_lowrank (background). The R matrix is the correction matrix mentioned in step 6 of the implementation. We again see that this matrix seems to capture a lot of the moving subjects in the video.

more, we chose a low-rank approximation with 115 modes (out of 279) as it captured at least 90% of the energy of the system and greatly reduced the system's size. We can see in the right side of **Figure 3** that all modes past this point are well below the level of machine precision, so this should be a fairly accurate approximation of the system. As we see in **Figure 4**, this approximation does a great job at separating the stationary background and the moving foreground subject (the bird). Unlike what we saw in **Figure 2**, there are no defined edges. This is likely because in this video the

camera is fixed in place, greatly reducing the amount of camera shake. This supports our earlier hypothesis that this may have been the cause of the edges appearing in **Figure 2**. Interestingly, we can again see that the R matrix captures the moving subject.

5 Summary and Conclusions

The SVD is a powerful tool for extracting fundamental structure from a system, but like all tools it has its uses as well as its limitations. Some of these shortcomings include the fact that principal components in SVD must be orthogonal, placing a restriction on the types of modes that can occur together. Furthermore, SVD ignores some spatio-temporal information, causing it to misinterpret some signals that occur in this domain. In this context, the DMD performs much better. By encoding each mode with a specific frequency, it can capture spatio-temporal information that SVD would typically miss. This kind of context often comes into play when studying dynamical systems in a data-driven way with unknown governing equations. As we saw in this paper, the spatio-temporal properties can be used quite well to separate the moving foreground from the stationary background. However, like many image processing techniques, DMD struggles to differentiate true movement from camera-shake. This suggests other pre-processing steps may be necessary to improve the quality of the input before DMD can be used or that we may need to be more selective in which modes we choose to reconstruct our foreground.

6 Appendix A

Below is a brief summary of the MATLAB functions I used during this project and their functions.

svd(X): Calculates the Singular Value Decomposition of the given matrix X, returning U, Σ, V.

imcomplement(I): Given a black & white image matrix I, this returns a new image matrix with every pixel value inverted. So for each pixel value p: $p_{new} = 255 - p_{old}$

pcolor(X): plots a colored representation of the given matrix.

flipud(X): Flips a matrix top to bottom to be compatible with pcolor.

7 Appendix B

```
% AMATH 482 HW 5: Dynamic Mode Decomposition
% Zachary McNulty

% DMD for dynamic data?

%% Loading the video clips
clear all; close all; clc;
%video = VideoReader('input_files/car.MOV');
%video = VideoReader('input_files/newtcradle.mp4');
%video = VideoReader('input_files/mythbusters.mp4');
%video = VideoReader('input_files/watermelon.mp4');
video = VideoReader('input_files/parallel_bars.mp4');

flip_contrast = 1; % flip contrast of pixels for analysis

%get(video) gives info on video file

frame_skip = 2; % take every "frame_rate"th frame
resolution_reduction = 0.2; % reduce resolution by this factor.
imheight = video.Height*resolution_reduction;
imwidth = video.Width*resolution_reduction;
num_frames = ceil(video.Duration * video.FrameRate / frame_skip);
dt = frame_skip / video.FrameRate; % used later
% t = 0: dt :video.Duration;
dt = 1;
t = 1:num_frames;

v = zeros(num_frames, imheight, imwidth); % stores frames as rows of 2D
    matrices
X = zeros(imheight*imwidth, num_frames); % stores frames as 1D columns


frame = 1;
index = 1;
while hasFrame(video)
    next_frame = readFrame(video);
    if mod(index, frame_skip) == 0
        x = imresize(next_frame, resolution_reduction);
        %imshow(x);
        if flip_contrast
            v(frame, :, :) = imcomplement(rgb2gray(x)); % imcomplement
        else
            v(frame, :, :) = rgb2gray(x); % dont imcomplement
        end
        X(:, frame) = reshape(v(frame,:,:), [imheight*imwidth, 1]);
        frame = frame + 1;
    end
    index = index + 1;
end

%% demean data!
% mean_data = mean(mean(X));
% X = X - mean_data;

%% Make X and X' matrices
clc; close all;

%%%%% body of DMD %%%%%%%%%
```

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

[U2,Sigma2,V2] = svd(X1, 'econ');

threshold = 0.90;
r = find(cumsum(diag(Sigma2) ./ sum(diag(Sigma2))) > threshold   ,1);
%r = size(Sigma2, 1); % full rank
figure(1)
subplot(121)
plot(diag(Sigma2)./max(diag(Sigma2)), 'r.', 'markersize', 20);
title('Normalized Singular Values')
xlabel('index ( j )')
ylabel('\sigma_j')
yticks(0:0.1:1)
set(gca, 'fontsize', 20);
subplot(122);
plot(log(diag(Sigma2)./max(diag(Sigma2))), 'r.', 'markersize', 20);
title('Normalized Singular Values')
xlabel('index ( j )')
ylabel('Log(\sigma_j )')
yticks(-8:1:0)
set(gca, 'fontsize', 20);


U=U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);
Atilde = U'*X2*V/Sigma;
[W,D] = eig(Atilde);
Phi=X2*V/Sigma*W;

mu=diag(D);
omega=log(mu)/dt;

y0 = Phi\X(:, 1);  % pseudo-inverse initial conditions


[min_omega, min_index] = min(abs(omega));
foreground_modes_indices = find(abs(omega) > min_omega);
% omega(foreground_modes_indices)

%%
clc; close all;

X_lowrank = y0(min_index).*Phi(:, min_index).*exp(omega(min_index).*t);
X_sparse = X - abs(X_lowrank);

R = X_sparse .* (X_sparse < 0); % places all negative entries in R

% R = zeros(size(X_sparse)); % But why?

X_lowrank2 = abs(X_lowrank) + R; % R stores moving object? Cancel out
    from background and add to foreground
X_sparse2 = X_sparse - R;

% Recreate the Foreground and Background

clc; close all;

figure(2)
set(gcf, 'Position',  [100, 100, 1000, 1000])
for frame = 1:size(X_lowrank2, 2)
    if flip_contrast
```

```
118         lowrank = imcomplement(reshape(X_lowrank2(:, frame), [imheight,
                imwidth]));
119         sparse = imcomplement(reshape(X_sparse2(:, frame), [imheight,
                imwidth] ));
120         rj = imcomplement(reshape(R(:, frame), [imheight, imwidth]));
121      else
122
123         lowrank = reshape(X_lowrank2(:, frame), [imheight, imwidth]);
124         sparse = reshape(X_sparse2(:, frame), [imheight, imwidth] );
125         rj = reshape(R(:, frame), [imheight, imwidth]);
126      end
127
128     subplot(221)
129     %imshow(uint8(lowrank));
130     pcolor(flipud(lowrank)), shading interp, colormap(gray);
131     title('Background (X\_lowrank)')
132     set(gca, 'fontsize', 20);
133
134     subplot(222)
135     %imshow(uint8(sparse));
136     pcolor(flipud(sparse)), shading interp, colormap(gray);
137     title('Moving Subject (X\_sparse)')
138     set(gca, 'fontsize', 20);
139
140     subplot(223)
141     %imshow(uint8(sparse + lowrank + rj));
142     pcolor(flipud(sparse + lowrank)), shading interp, colormap(gray);
143     title('Sum of both')
144     set(gca, 'fontsize', 20);
145
146     subplot(224)
147     %imshow(uint8(abs(rj)));
148     pcolor(flipud(rj)), shading interp, colormap(gray);
149     title('R matrix')
150     set(gca, 'fontsize', 20);
151
152     drawnow;
153 end
```