Jonathan Lu (1438364)
HW4 – Background Subtraction in video streams

This project involved taking a video stream and separating it into foreground and background videos using Dynamic Mode Decomposition (DMD). The goal of such data-driven models is to take advantage of low dimensionality without having to rely on governing equations or formulas. DMD relies on the SVD but with a Fourier Transform in time. This tool is so powerful, as it can do a lot of things given so little information. Only experimental data drives the understanding of the system.

Sec I: Introduction and Overview

Suppose that we only have data, and we do not know how the system functions. Then, from the data, how do we extract the motion of a system and its formula? Dynamic mode decomposition is a new data based algorithm that requires no governing equation but rather snapshots of data in time. It is robust in a sense that it can predict the future (in terms of data), while PCA and POD cannot. The limitation is that data sampling must be done in equally spaced time intervals.

Video streams are perfect for the DMD method, because the frames of videos are equally spaced in time and every frame can be vectorized as pixel data.

Sec II: Theoretical Background

Given *n* data points collected at a given time, with a total of *m* samplings in time over evenly spaced time sets Δt. The data can be organized into two matrices:

$$X_1^{m-1} = [x_1\ x_2\ \dots\ x_{m-1}\ ]$$
$$X_2^{m} = [x_2\ x_3\ \dots\ x_{m}\ ]$$

The Koopman Operator A maps the data at time *j* to the time *j+1* such that $x_{j+1} = Ax_j$. The DMD algorithm estimates the Koopman Operator A that best represents the data in X_1^{m-1} such that:

$$X_1^{m-1} = [x_1\ Ax_1\ A^2x_1\ \dots\ A^{m-2}x_1\ ]$$

The DMD method I use in this project is based off of the Jonathan Tu Algorithm.
J-Tu (Jonathan Tu) Algorithm:

1. Do SVD on data matrix to get U, Σ, V matrices.
2. $A = X'X^t = X'V\ \Sigma^{-1}U^*$
    a. where X' represents X_2^m, X^t represents the pseudo-inverse.
    b. However, due to low rank projection, better to work with Å
3. $\text{Å} = U^*AU = U^*X'V\ \Sigma^{-1}$
    a. Where Å = A embedded in low-rank U space.
4. $\text{Å}W = W\Lambda$
    a. Do eigendecomposition to find eigenvalues in the snapshots of data.

   b. Can tell us a lot about the data: oscillations (imaginary eigenvalues), growing (positive), decaying (negative).
5. $\Phi = \mathrm{X}' V\, \Sigma^{-1} W$ where $x = \Phi e^{W \Lambda \mathrm{t}} b$
   a. Where Φ are the DMD modes, x = data approximation

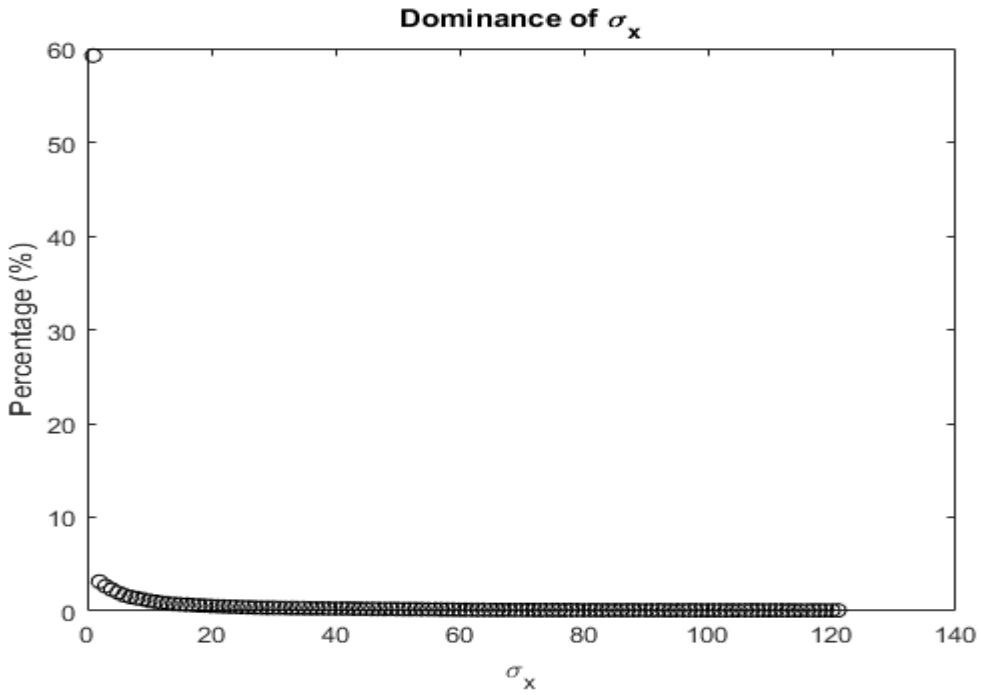
Essential to the J-Tu Algorithm is that because $AX_1^{m-1} \approx X_2^m \approx \mathring{\mathrm{A}} X_1^{m-1}$, then some of the eigenvalues of matrix Å approximate the eigenvalues of Koopman operator *A*. In addition, the DMD modes are not orthogonal. This means that DMD doesn't approximate as well with low-rank approximation compared to PCA. However, the modes can be more meaningful because each mode is associated with a sinusoidal behavior in time. With just this, it can predict how the motion of something will be into the future. It is also worth noting that even though we can extrapolate data with DMD, there is a limit on how far we can do it before the accuracy falls off. This relies on the eigenvalues, where an eigenvalue > 1 will have the data eventually blow up.

Sec III: Algorithm Implementation and Development

I had one function (processVid) that processed the .mov file to a 4-d data matrix. Then I had another function (vid2Data) that converted the 4-d matrix into a data matrix *A* containing all the frames, which each frame reshaped as a row vector. After performing SVD on this data matrix, I took note of the modes. I wanted to see what modes were relevant to the system. After looking at the singular value plot (see sec IV), I can conclude that a rank 20 system would be the best. As the first 20 dominant modes took up the majority of the graph. Thus I chose to project the data matrix onto the low rank embedded U space (of only 20 columns), and got *Atilde* (Step 3 of J-Tu Algorithm). Then I did eigendecomposition and found the eigenvalues of *Atilde* (Step 4). With this, I could find the modes of the data (Step 5), and also the *b* value. With the *b* value, I proceeded to find the eigenvalues of *D* that corresponded to the lowest absolute value. Then I proceeded to multiply *b* by the column vectors of *Phi* that corresponds to the respective eigenvalue, which generates an *mx1* vector *A1.* Then I multiply *A1* by vector *d* which contains the log-scaled version of the eigenvalues. This gives me the low-rank matrix, *xLow.* I construct the sparse matrix *xSparse* which is essentially the data matrix minus the low rank matrix. However, some elements of *xSparse* are negative, which corresponds to a negative pixel intensity, which doesn't make sense. So I removed the negative elements (called the residuals *R*), and added it to the low rank matrix. With the two matrices, *xLow* (background) and *xSparse* (foreground), we can reconstruct the gray scaled video separately by reshaping it back to a 3-d matrix.

Sec IV: Computational Results

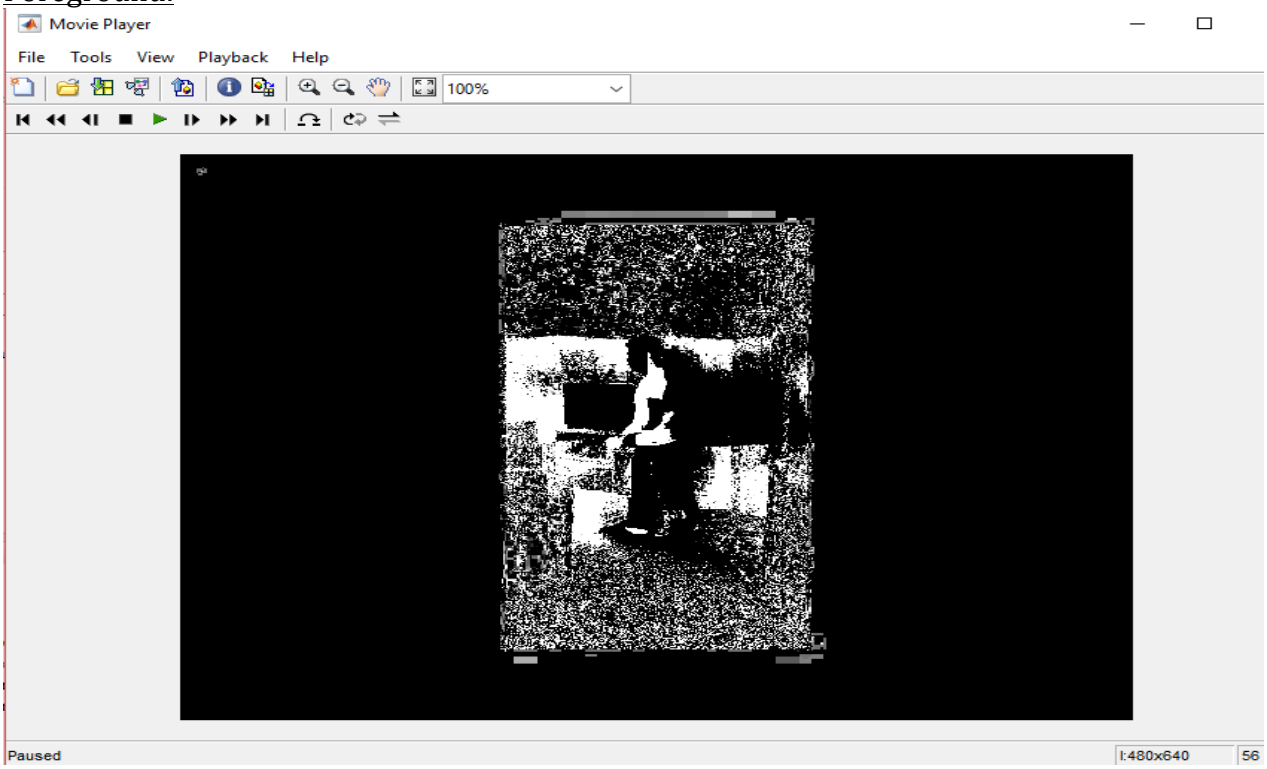
Based on the graph of the modes, I saw that the first few nodes provided most of the data, but there was still a lot of other modes that were comprised between 0.5-5% of the data:



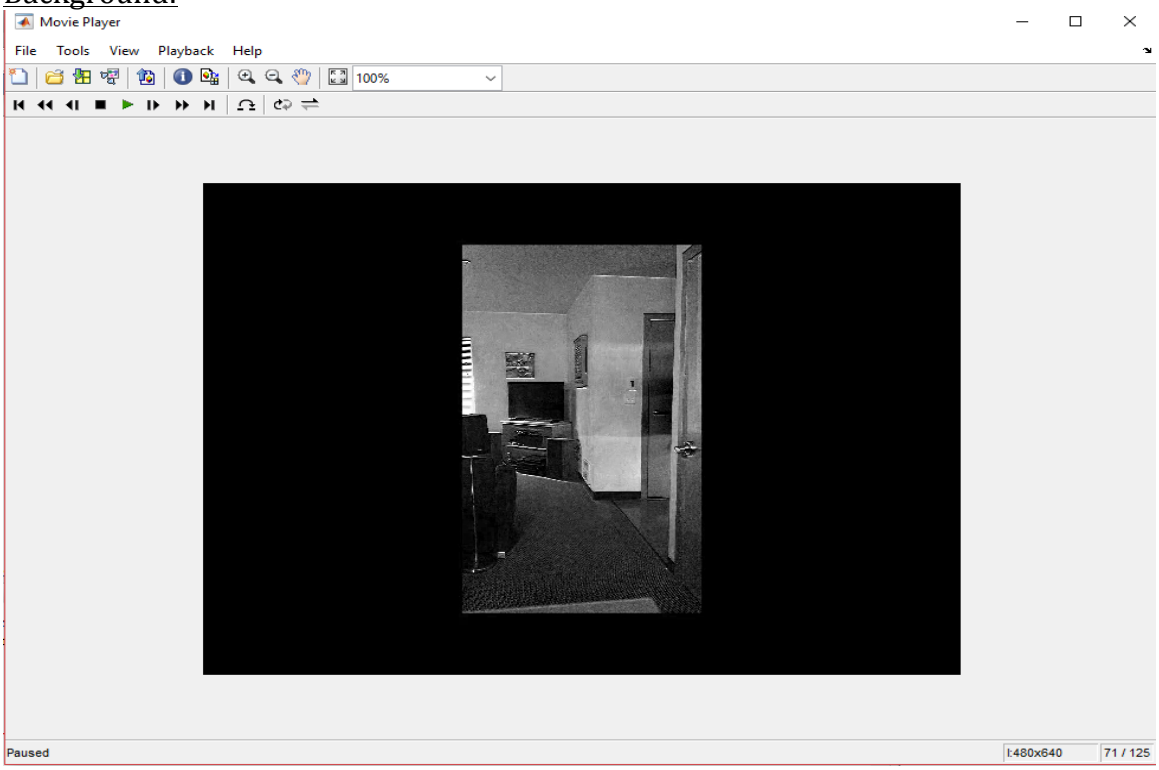
Therefore, based on the graph above, I decided to go with a rank 20-system to approximate my data.

After running the program, I was able to retrieve the videos for the separate foreground and background.

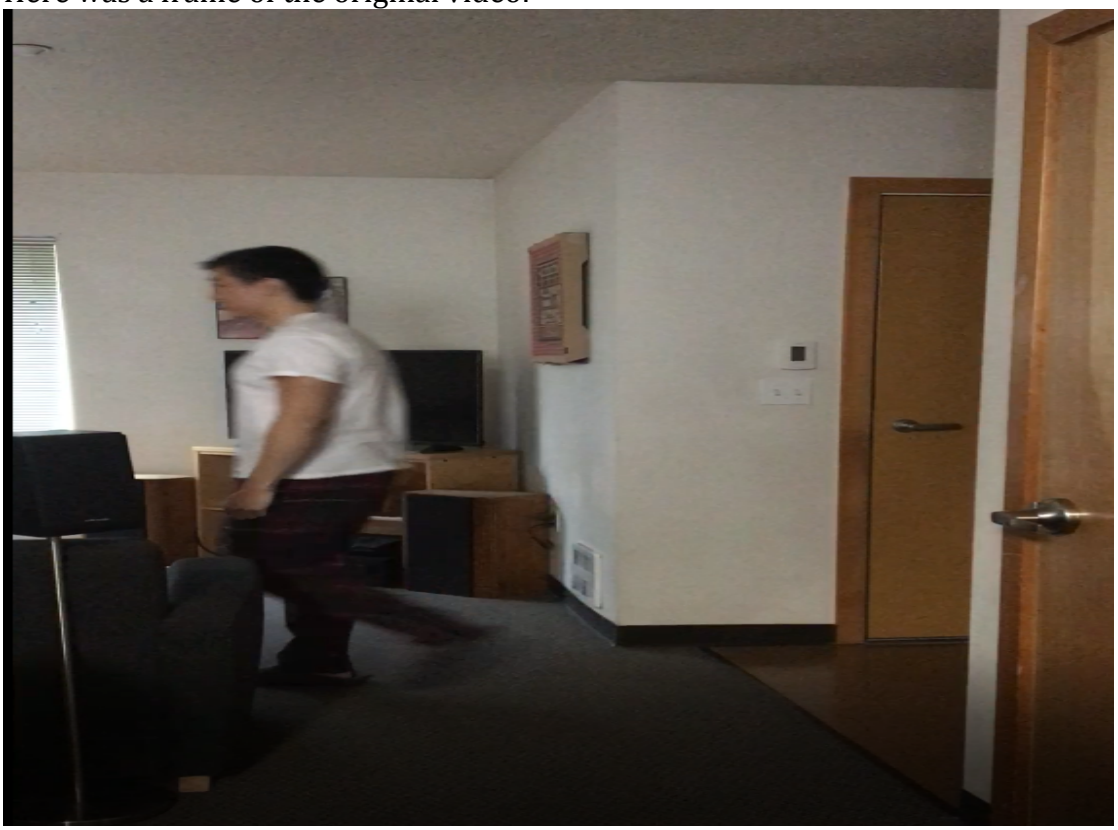
Foreground:



Background:



Here was a frame of the original video:



Note: my initial video was shot in 1920x1080, but due to size limitations, I used imresize() to reduce the resolution by 4x.

Sec V: Summary and Conclusions

Choosing a rank is very important in separating the two video streams. With a small rank that doesn't capture the data fully, both streams may be blurry and not be fully separated. With a very high rank, the streams will be well defined but the program will take a much longer time to process the information and run. Knowing the data is very important.

Separating video streams may be important in information technology and security systems. It may be beneficial to capture the motion of an object, and even predict the future location of the object up to a certain amount.

Appendix A: Important Matlab Functions

double() – used to convert from unit8 precision to double precision
rgb2gray() – used to convert rgb values to gray scale
reshape() – reshape data to row vector to add to data matrix
imresize() – resize image frame
svd() – singular value decomposition.
eig() – retrieve eigenvalues and eigenvectors of a matrix.

Appendix B: Code

processVid():

```
% processes .mov files into a matlab matrix that contains pixel width,
% height, rgb scale, and frame number.
function vidFrames = processVid(vid)

obj = VideoReader(vid);
vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');
for k = 1:numFrames
    mov(k).cdata = vidFrames(:,:,:,k);
    mov(k).colormap = [];
end
% for j = 1:numFrames
%     Z=frame2im(mov(j));
%     imshow(Z); drawnow
% end
```

vid2Data():

```
% turns the video matrix into readable data - returns A which holds pixels
% values in double precision, grayscaled, and reshaped, per frame
function A = vid2Data(vid)

numFrames = size(vid, 4);

%%  turn resulting 4-d video matrix to 3-d grayscale matrix (w,h,numFrames)
for k = numFrames:-1:1 % note: start backwards to initialize size of g
    g(:, :, k) = imresize(rgb2gray(vid(:, :, :, k)), 0.25);
end

A = [];

%% turn 3-d grayscale matrix into double precision, reshape to row vector,
% add to data matrix.
for k = numFrames:-1:1
    R = reshape(double(g(:,:,k)), [], 1);
    A = [A R];
end
```

main():

```
% Jonathan Lu
% AMATH 482
% HW3 - Identifying Music
% 2/16/17
close all; clc

vidFrames = processVid('test1.mov');
X = vid2Data(vidFrames);

t=1:121; %298 frames
dt = 1; %1 frame

X1 = X(:, 1:end-1);
X2 = X(:, 2:end);
[U2,S2,V2] = svd(X1, 'econ');

figure(1)
plot(100*diag(S2)/sum(diag(S2)), 'ko')
xlabel('\sigma_x')
ylabel('Percentage (%)')
```

```
21 -    title('Dominance of \sigma_x')
22
23 -    r = 50; %rank
24 -    U=U2(:,1:r);
25 -    S=S2(1:r,1:r);
26 -    V=V2(:,1:r);
27
28      %% DMD J-Tu Algorithm %%
29 -    Atilde = U'*X2*V/S; % embed into low rank space
30 -    [W,D] = eig(Atilde);
31 -    Phi = X2*V/S*W; %U*W, DMD modes
32
33 -    mu = diag(D); % eigenvalues
34 -    omega = log(mu)/dt; % linear scale of eigenvalues.
35
36 -    u0 = X(:,1); %project onto b, take first snapshot
37 -    b = Phi\u0; %pseudo-inv initial conditions
38
39 -    [M,I] = min(abs(omega));
40 -    A1 = Phi(:,I)*b(I);
41
42 -    d = zeros(1,121);
43 -    for j = 1:121
44 -       d(j) = exp(M*j);
45 -    end
46
47 -    xLow = A1*d; %calculate low rank matrix
48 -    xSparse = X1-abs(xLow);
49 -    R = xSparse; %construct residuals
50 -    R(xSparse>0)=0;  %set all pos values of resid to 0
51 -    xSparse(xSparse<0) = 0; %remove residuals from xsparse
52 -    xLow = abs(xLow) + R; %low rank foreground info
53 -    x_f = reshape(xLow, 480, 270, 121);
54 -    x_b = reshape(xSparse, 480, 270, 121);
55 -    implay(x_f)
56 -    implay(x_b)
```