MAST 680 Assignment 1: Background Subtraction in Video Streams

Will Sze

February 12, 2023

Abstract

Dynamic Mode Decomposition can be used to linearize a system that is completely nonlinear. It can also be used to separate background and foreground in video streams. In this paper, a method is developed to reconstruct the background and foreground of two video streams plus a test video for validation using DMD. The method was able to retrieve the background information with slight ghosting present, as well as the foreground by subtracting the background from the original video. Furthermore, some artifact effects were discussed.

1 Introduction and Overview

Modeling phenomena in the real world can be a huge challenge. Often, we find ourselves having to deal with non-linear systems which are inherent in a world where chaos is predominant. A common method of solving nonlinear systems is to approximate the system linearly which is many times easier to solve. A method capable of linearizing a system of low-dimensionality is called Dynamic Mode Decomposition, or DMD for short[1]. DMD allows us to forecast from one-time point to the next by taking in existing data[1]. Particularly, we can use a characteristic from this method to separate multi-scale signals into slow and fast frequencies[1]. An application of this method is object tracking in a video stream where the background can be removed[2].

1.1 Problem Statement

The current paper demonstrates the capabilities of DMD to separate the background (slow) and foreground (fast) of a short video stream. Particularly, two real video streams with file names `monte_carlo_low.mp4` and `ski_drop_low.mp4` were given where there is a clear stationary background and an object that is moving in the foreground.

2 Theoretical Background

Following the notes of [1], the following theory is collected. DMD linearizes a system by transforming it into a matrix multiplication of A onto a current time frame to obtain the next time frame. Any Data matrix with shape M by N can be separated into matrices X and Y where every column of Y represents the next time step of columns of X. The size M is associated to spatial variables and N is associated to time variables.

$$Y_n = X_{n+1} \tag{1}$$

Here, the subscript $n \in \{1, 2, 3, .., N-1\}$ are the columns. The DMD method tells us that Y can be obtained through the multiplication of X by a matrix A called the DMD matrix hence linearizing the system.

$$Y = AX \tag{2}$$

Since X and Y are not necessarily square matrices, to find this DMD matrix, we can apply the pseudo-inverse of X onto Y.

$$A = YX^{\dagger} \tag{3}$$

The pseudoinverse $X^{\dagger}$ is computed by first finding the singular value decomposition (SVD) of X and using equation 2) with inverse matrix multiplication.

$$X = U\Sigma V^{*} \tag{4}$$

Here, U and V are unitary matrices resulting from the separation of space and time domain and Σ is a diagonal matrix containing the singular values σ representing the level of importance. The star notation denotes the conjugate transpose of the matrix. The DMD matrix indicates the necessary transformation in order to reach the next time step. Using the same technique as those used in linear models, i.e finding the eigenvalues (ψ) and eigenvectors (μ) of A, we can reproduce the original data matrix by linear combination of the eigensolution with exponents.

$$Data_{DMD} = \sum_{m=1}^{M} c_m \psi_m e^{\omega_m t} \tag{5}$$

X_{DMD} is the reconstructed data matrix, c_m are constants which can be found using initial conditions, and $\omega_m = \ln(\mu_m)/\triangle t$ are the frequencies with $\triangle t$ being the time increment. To separate the fast and slow timescales, a threshold λ can be set to separate $\|\omega_m\|$. Smaller frequencies represent the slow timescales and larger frequencies represent he fast timescales. Thus, we can write the data matrix as a combination of two separate eigensolutions: one which groups frequencies for the slow evolution and another for the fast evolution.

$$Data_{DMD} = \sum_{p=1} c_p \psi_p e^{\omega_p t} + \sum_{j \neq p} c_j \psi_j e^{\omega_j t} \tag{6}$$

$p \in \{1, 2, 3, .., l\}$ represents the indices for slow timescale and j represents the indices for the fast timescale. When dealing with large sets of data, it can be computationally intensive to find the DMD matrix and compute the eigenvectors and eigenvalues. Thus we can begin by finding a low rank data matrix using SVD which will approximate the information.

$$Data_s = U_s \Sigma_s V_s^{*} \tag{7}$$

The subscript s denotes the s rank of the matrices. The singular values are ordered from highest to lowest, thus we can choose the first s values which can reconstruct the Data matrix approximately. To know how much of the original matrix is reconstructed with the low rank SVD, we can calculate the ratio of the Frobenius norm between the low rank singular values and the entire singular values.

$$\frac{\|\Sigma_s\|_F}{\|\Sigma\|_F} = \frac{\sum_{i=1}^{s} \sigma_i^2}{\sum_{j=1}^{r} \sigma_j^2} \tag{8}$$

Equation 8 can be thought of as the energy level of the original data matrix. Thus we can use this notion to find a value of s such that it meets a minimum energy requirement. For dimension $M >> N$, the new low rank Data can be calculated by taking the inverse of U_s which reduces the M dimension to the rank number s. This will reduce tremendously the computation time needed for the DMD matrix, and its eigenvalues and eigenvectors.

$$Data_{S \times N} = U_s^{*} Data = \Sigma_s V_s^{*} \tag{9}$$

3 Algorithm Implementation and Development

A code was written in Octave to perform DMD on the given video streams and solve the problem. The functions referenced in later sections can be found in Appendix A and the code in Appendix B. Considering Octave was made using the same language as Matlab, the script can be run with Matlab with some exceptions stated in Appendix A. During the development of the algorithm, three videos were used. Two of them were the real video streams provided and one was a simpler test video created to validate the algorithm. The entire algorithm is separated into three parts: the initial video conversion, the background reconstruction and the foreground reconstruction.

3.1 Video Conversion

To compute DMD on a video, the video must be imported and converted into a 2D matrix (refer to algorithm 1). The imported video stream is read as individual frames changing through time. Each frame is stored as matrix of size m by n which are the number of pixels. In addition, each pixel has 3 colors (red, green and blue). To reduce the size of the data, the video is converted into grayscale which has only 1 value per pixel rather than 3. Then each frame is reshaped into a single column forming M pixels and all frames are grouped together forming N frames. This produces a Data matrix $D \in \mathbb{R}^{M \times N}$.

Since the M component is very large, the low-rank data matrix is obtained following equations 7) to 9). Here the SVD function from Octave was used to retrieve matrices U, Σ and V. To find a low rank s, a loop was implemented to continuously check the energy level from equation 8 and update s until a satisfactory energy level is achieved. In our case, the level was set to 99.99% giving very good approximation. The s value obtained for `monte_carlo_low`, `ski_drop_low` and test data was 143, 60 and 80 respectfully. The data matrix was then multiplied by the inverse of U to get a new data matrix where M is now reduced to s. This reduced matrix now becomes $D_{s \times N} \in \mathbb{R}^{s \times N}$.

Algorithm 1: Video Conversion and Low Rank Approximation

```
Import data from filename.mp4
for i = 1 : Numberofframes do
    Convert to grayscale
    Reshape frame into M x 1 matrix
    Put frame into i^th column of D_MxN
end for
Calculate SVD on D_MxN
initialize s = 1
while Energy < 0.9999 do
    s ← s + 1
    Compute Energy
end while
D_sxN = U_s' D_MxN
```

3.2 Background Reconstruction

Following equation 1, $D_{s \times N}$ were separated into the matrices X_s and Y_s with size $s \times N - 1$. A pseudoinverse function is applied to X_s where it can be multiplied on the right side of Y_s to obtain the DMD matrix A. Applying the *eig* function from octave onto the DMD matrix A allows us to find the eigenvalues and eigenvectors needed to reconstruct the video. First, the frequencies ω were calculated using log of the eigenvalues over a time interval of $dt = \frac{1}{framerate}$ where the frame rate was around 60 fps for the video streams provided and 30 fps for the test data. The frequencies were then sorted from small to large using the sort function and a threshold was determined by iteratively choosing a value which gives the best result. Here, the threshold was chosen to be 0.01 allowing only 1 eigenvalue to be used to reconstruct the background since it allowed the best background reconstruction. The algorithm then loops through the unsorted $\|\omega\|$ verifying if it is under the threshold and performs the left summation of equation 6. The constants c_p were calculated by taking the inverse of the eigenvectors and multiplying by the first column of X_s which are the initial conditions at t = 0. Since the order of $\|\omega\|$ are preserved, the eigenvector will correspond to its ω and c_p. The resulting background matrix has complex values due to the eigenvalue computation. The real portion can be taken for the video.

3.3 Foreground Reconstruction

The foreground is reconstructed knowing that the total video stream is a combination of the background and the foreground. Thus we can find the foreground by taking the absolute value of the background

Algorithm 2: DMD Matrix and Background Reconstruction

Separate $D_{s\times N}$ into X_s and Y_s
Calculate A = Y_s×pseudoinverse(X_s)
Calculate eigen vectors and eigen values
Calculate ω
Sort $\|\omega\|$
Determine λ
Calculate constants of eigensolution
for $i = 1 : s$ **do**
  if $\|\omega_i\| < threshold$ **then**
    $D_{s,background} \leftarrow D_{s,background} + c_i\psi_i e^{\omega_i t}$
  end if
end for
Reverse to $M \times N$ matrix $D_{background}$ using $U_s D_{s,background}$ and take real part

reconstruction and subtracting it from the original low rank data.

$$Data_{Foreground} = Data_{Total} - \|Data_{Background}\| \tag{10}$$

Doing so will result in negative values in some of the pixels. These pixels are those that do not belong to the background. Thus a residual matrix which gathers all negative values below a threshold is created which could then be subtracted back to the foreground reconstruction. If elements in the residual are exactly the negative numbers from the foreground, this has an effect of turning all foreground elements into zero. However we want to accentuate the foreground elements, thus an integer is multiplied to the residual matrix to bring those elements into the positive. More specifically, those with larger negatives are multiplied whereas the smaller values are left as is. This is intended to reduce the noise and highlight the moving objects with higher contrast. An integer value of 3 and a threshold value of -40 work well. The new modified residual will just be named R.

$$Data_{Foreground} = Data_{Foreground} - R \tag{11}$$

Algorithm 3: Foreground Reconstruction

Calculate foreground using $D_{total} - abs(D_{background})$
Gather all negative values in a separate matrix R
Set all negative values inside R below a threshold to an integer multiplied by itself
$D_{foreground} \leftarrow D_{foreground} - R$

4 Computational Results

The algorithm successfully extracted the background from both the `monte_carlo_low.mp4` and the `ski_drop_low.mp4` video. The algorithm was also performed on a test data video which is just a moving box. We can see from figure 1 the objects that were moving are now not visible, however, slight ghosting is present. We can see a slight blur of where the moving object is support to be. The ghosting is particularly present on the test data where we can see a faint version of the moving box. This might be due to a relatively small number of frames. It would be interesting to see what effect the number of frames would bring to the quality of the reconstruction.

For the foreground, the algorithm was able to extract the moving objects. However, some parts of the background were also extracted, particularly the contours of the background objects (see figure 1). This may be due the background being only a single frame. When it was subtracted from the original video which contained noise (camera vibrating), it caused the outlines appear. The camera in `monte_carlo_low.mp4` was seen shifting quite frequently which resulted in more white areas popping out. However, if we look at the

Figure 1: Comparison between original (left), background (middle) and foreground (right) reconstruction of monte_carlo_low (top), ski_drop_low (middle) and Test data (bottom) at frame 1/4 of total frame count.

ski drop or the test data video, the camera is close to, or in the case of the test video perfectly, still which allows the background to be completely cancelled. The only artifact from the ski video is the falling snow and the cloud which were captured in the background allowing it to appear in the foreground.

It is to note, however, if the foreground is extracted the same way as for the background but taking ω_j where $j \neq p$ then the result becomes significantly different than the ones discussed in the previous paragraph. It seems the moving object in each frame is spread faintly across multiple frames and the objects seemingly move backwards (see figure 2). The reason for this remains unknown. Further on this topic, if we look at the difference in the DMD matrix computation from Grosek and Kutz[2], and apply it here where it becomes equation 12, a slight improvement can be seen.

$$\tilde{A} = U^*_{x,s} Y \Sigma^{-1}_{x,s} V_{x,s} = U^*_{x,s} A U_{x,s} \tag{12}$$

Here, $U_{x,s}$,$\Sigma_{x,s}$ and $V_{x,s}$ comes from the low rank SVD of X. Working with equation 12 is slightly different than working with equation 9 and then preceding to find equation 3 from section 2 of this current paper. This resulted in the test data foreground reconstruction appearing to have a better result albeit still in reverse. The other two video streams remain qualitatively the same as previously described.

5 Summary and Conclusions

In summary, the DMD computations, which can linearize a non-linear system, can also serve to separate out the background and foreground. However, care must be taken when separating the foreground since the same technique separating the background cannot be applied to the foreground. One should use the background and the original video to reconstruct the foreground. Good foreground reconstruction can be achieved, however the quality is still dependent on the amount of noise in the data. Simple filtering tricks, such as defining a pixel intensity threshold or others, could be explored to improve on bringing out the objects of interest.

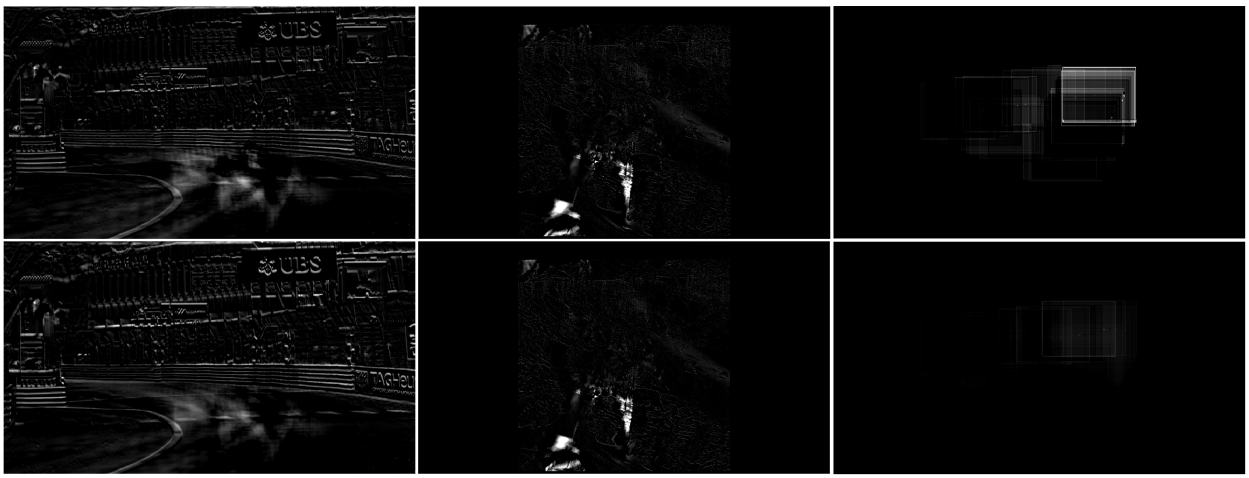

Figure 2: Comparison of foregrounds between current DMD matrix computation (top row) and DMD matrix computation of [2] (bottom row) at frame 1/4 of total frame count.

References

[1] J. Jason Bramburger. *Data-Driven Methods for Dynamic Systems*. Concordia University, 2023.

[2] Jacob Grosek and J. Nathan Kutz. *Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video*. 2014. DOI: 10.48550/ARXIV.1404.7592. URL: https://arxiv.org/abs/1404.7592.

Appendix A Premade Functions Used

The following premade functions are found in Octave which are the same as in Matlab. The only exception is the "number of frames" property where in Octave is written `NumberOfFrames` whereas in Matlab is written `NumFrames`. In addition, to be able to use everything associated to the VideoReader function, the video package needs to be installed once and loaded every time Octave is opened. To do this, use `pkg install -forge video` and `pkg load video`. VideoReader is already a built-in function in Matlab.

- `vid = VideoReader("filename.mp4")` gathers all appropriate information from the video including each frame, the number of frames, the frame rate, video height and width (pixels), etc. These propreties can be accessed using `vid.`.
- `grayImage = rgb2gray(readFrame(vid))` converts each pixel containing of RGB values to a single value by a weighted sum. These values represent the grayscale of each pixel.
- `movie(h, mov, 1, frameRate)` displays the frames inside `mov` structure in the figure box `h`. The number 1 indicates the number of loops and `frameRate` indicates the video fps.
- `[U,S,V] = svd(X,'econ')` returns the unitary matrices `U` and `V`, and the singular values of matrix X. The argument `'econ'` removes all zeros rows or columns to make the matrices compact.
- `round(x)` rounds the value `x` down to the nearest integer.
- `imshow(mov(k).cdata,mov(k).colormap)` displays the k^{th} frame in `mov` structure with its color map onto a figure window.
- `pinv(X)` returns the pseudoinverse ($V\Sigma^{-1}U^*$) of `X` matrix.
- `[V, D] = eig(A)` returns the eigenvectors of matrix `A` in each columns of `V` and a diagonal matrix `D` containing the eigenvalues of matrix `A`.

- `diag(X)` returns a vector containing all the diagonal values inside matrix `X`.
- `abs(x)` returns the norm of `x`.
- `sort(X)` returns a vector sorted from smallest to largest of vector `X`.
- `rem(x, n)` returns the remainder of a number `x` divided by an integer `n`.
- `real(x)` returns the real numbers of a complex number `x`.

Appendix B Octave Code

```
%OCTAVE Video Conversion VERS 7 (FINAL)
%Clear workspace
clear; close all; clc

%Load video package for Octave (remove if using Matlab)
pkg load video;

pause(0.1);

fprintf('\nImporting Video...\n');
tic
  %Imports video file into matlab
  vid = VideoReader("TestData2.mp4");

  %Collects video dimensions
  vidHeight = vid.Height;
  vidWidth = vid.Width;
  vidNFrames = vid.NumberOfFrames;
toc

%Movie Struct to collect all original frames to run movie using movie
    function
mov = struct('cdata', zeros(vidHeight,vidWidth,1,'uint8'), 'colormap',gray
    (255));

%Movie struct for reshaping and conversions
mov2 = struct('cdata', zeros(vidHeight*vidWidth,1,'uint8'), 'colormap',
    gray(255));

%Movie struct for postprocessing results
mov_LowRank = struct('cdata', zeros(vidHeight*vidWidth,1,'uint8'), '
    colormap',gray(255));
mov_Back = struct('cdata', zeros(vidHeight,vidWidth,1,'uint8'), 'colormap'
    ,gray(255));
mov_Fore = struct('cdata', zeros(vidHeight,vidWidth,1,'uint8'), 'colormap'
    ,gray(255));
mov_all = struct('cdata', zeros(vidHeight,vidWidth,1,'uint8'), 'colormap',
    gray(255));

%Initializing (All In One) matrix to hold all frames into a 2D space-time
    matrix
DATAMovie = zeros(vidWidth*vidHeight,vid.NumberOfFrames);
```

```
35
36 %figure number tracking
37 nfigure = 1;
38
39 [M,N] = size(DATAMovie);
40
41 %Collects each frame into cdata
42 fprintf('\nConverting video to Grayscale...\n');
43 tic
44   k = 1;
45   while hasFrame(vid)
46       grayImage = rgb2gray(readFrame(vid));
47
48       mov(k).cdata = grayImage; %Stores the gray frames into the structure
             ;
49       mov(k).colormap = gray(255); %Applies Gray Colormap to every frame
50       k = k+1;
51
52   end
53 toc
54
55 %View original video
56 fprintf('\nDisplaying Gray Video...\n');
57 tic
58    hf = figure(nfigure);
59    set(hf, 'position', [150 150 vidWidth vidHeight]);
60    movie(hf, mov, 1, vid.FrameRate);
61    nfigure = nfigure + 1;
62 toc
63
64
65 fprintf('\nReshaping Gray Video into 2D Matrix...\n');
66 tic
67   for k = 1:vidNFrames
68       mov2(k).cdata = reshape(mov(k).cdata,[],1);
69       mov2(k).colormap = gray(255);
70       DATAMovie(:,k) = [mov2(k).cdata];
71   end
72 toc
73
74 %Computing SVD for DATAMovie
75 fprintf('\nCalculating SVD...\n');
76 tic
77   [U_DATA,S_DATA,V_DATA] = svd(DATAMovie,'econ');
78 toc
79
80 %initializes variables for low rank calculations with energy
81 s_ = 1;
82 EnergyTarget = 0.9999;
83
84 sum_Sigma_s = 0;
85 sum_Sigma_all = 0;
86 Energy = 0;
87
```

```
%Searches for a rank value that will make the Energy bigger than or equal to 99.99% to have good approximation.
fprintf('\nSearching for low rank s number achieving 99.99 percent Energy ...\n');
tic
  while(Energy < 0.9999)
    s_ = s_+1;
    sum_Sigma_s = 0;
    sum_Sigma_all = 0;
    for i = 1:s_
      sum_Sigma_s = sum_Sigma_s + S_DATA(i,i)^2;
    end
    for i = 1:vid.NumberOfFrames
      sum_Sigma_all = sum_Sigma_all + S_DATA(i,i)^2;
    end

    Energy = sum_Sigma_s/sum_Sigma_all;
  end

  fprintf('\n\tLow Rank s number achieved: %d', s_);
  fprintf('\n\twith Energy level = %d\n', Energy);
toc
clear('sum_Sigma_s');
clear('sum_Sigma_all');


fprintf('\nComputing and displaying Low Rank movie...\n');
tic
  %Computes the low rank approximation s = s_
  DATAMovie_s = U_DATA(:,1:s_)*S_DATA(1:s_,1:s_)*V_DATA(:,1:s_)';

  % View the low rank approximated video
  for k = 1:vidNFrames
      mov2(k).cdata = [cast(DATAMovie_s(:,k),'uint8')];
      mov_LowRank(k).cdata = reshape(mov2(k).cdata,vidHeight,vidWidth);
      mov_LowRank(k).colormap = gray(255);
  end

  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth vidHeight]);
  movie(hf, mov_LowRank, 1, vid.FrameRate);
  nfigure = nfigure + 1;
toc

fprintf('\nDisplaying Low Rank image at 1/4 in the movie...\n');
tic
  FrameInterest = round(vidNFrames*0.25);
  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth vidHeight]);
  imshow(mov_LowRank(FrameInterest).cdata,mov_LowRank(FrameInterest).
     colormap);
  nfigure = nfigure + 1;
toc

```

```
%------------ Starting Separation DMD into background and foreground ------------%

DATA_SxN = U_DATA(:,1:s_)'*DATAMovie_s;

fprintf('\nCreating Original and Low Rank (s) X and Y matrices...\n');
tic
  for k = 1:vidNFrames-1
      Xs(:,k) = [DATA_SxN(:,k)];
      Ys(:,k) = [DATA_SxN(:,k+1)];
  end
toc

fprintf('\nObtaining DMD Matrix A...\n');
tic
  %Calculating SVD(X) to find pseudoinv(X) (same as pinv(X))
  %[U,S,V] = svd(Xs, 'econ');
  %S_inv = inv(S);

  %Low rank DMD Matrix using Ys * pseudoinv(X)
  A = Ys*pinv(Xs);%V*S_inv*U';
toc

%------------Uncomment to use alternate DMD Matrix and comment above ----------------%
%%fprintf('\nCreating Alternate DMD Matrix...\n');
%%tic
%%  %Creating X and Y matrices out of the original Data Matrix
%%  for k = 1:vidNFrames-1
%%      X(:,k) = [DATAMovie(:,k)];
%%      Y(:,k) = [DATAMovie(:,k+1)];
%%  end
%%
%%  [U,S,V] = svd(X,'econ');
%%  A = U(:,1:s_)'*Y*V(:,1:s_)*inv(S(1:s_,1:s_));
%%  Xs = U(:,1:s_)'*X;
%%toc



dt = 1.0/vid.FrameRate;
t = (0:dt:dt*(vidNFrames-1))';

%%dt = 1.0;
%%t = (0:1:N-1)'; %Same result as using time except scaled smaller.

fprintf('\nObtaining Eigen Vectors and values and Omega of DMD Matrix A ...\n');
tic
  [e_Vect, e_Val] = eig(A); %finds eigenvectors and eigenvalues of A
  e_Val_D = diag(e_Val); %Take all the diagonals of eigenvalues
  omega = log(e_Val_D)/dt;
  absOmega = abs(omega);
  absOmegaSorted = sort(absOmega);
```

```
toc

fprintf('\nPlotting Omega sorted...\n');
tic
  figure(nfigure);
  plot(absOmegaSorted, '.', 'MarkerSize', 20);
  nfigure = nfigure + 1;
toc

%Lambda
Threshold = 0.01;

%Initializing Matrices for Results
e_Val_Back = zeros([size(e_Val)]);
e_Vect_Back = zeros([size(e_Vect)]);
e_Val_Fore = zeros([size(e_Val)]);
e_Vect_Fore = zeros([size(e_Vect)]);

D_Back = zeros(M,N);
D_Fore = zeros(M,N);

Ds_Back = zeros(s_,N);
Ds_Fore = zeros(s_,N);
Ds_all = zeros(s_,N);

fprintf('\nSeparating Background and Foreground Matrices with Threshold: %
    d...\n',Threshold);
  %Constants of eigensolutions using t = 0 and the first column of Xs
  b = pinv(e_Vect)*Xs(:,1);
  Percentage = 1/length(A)*100;
tic
  for i = 1:length(A);

    if(absOmega(i) < Threshold)
      %Background Separation
      e_Val_Back(i,i) = e_Val(i,i);
      e_Vect_Back(:,i) = e_Vect(:,i);
      Ds_Back = Ds_Back + b(i)*e_Vect_Back(:,i).*exp(omega(i)*t)';  %exp(1
          i*imag(omega(i))*t);
    else
      %Foreground Separation
      e_Val_Fore(i,i) = e_Val(i,i);
      e_Vect_Fore(:,i) = e_Vect(:,i);
      Ds_Fore = Ds_Fore + b(i)*e_Vect_Fore(:,i).*exp(omega(i)*t)';  %exp(1
          i*imag(omega(i))*t);
    end

    %Full Reconstrucion
    Ds_all = Ds_all + b(i)*e_Vect(:,i).*exp(omega(i)*t)';

    Completed = rem(i,10);
    if(Completed == 0)
      fprintf('\n Completed: %d', round(i*Percentage));
      disp('%');
```

```
    end

    %Calculate and display estimated time to complete Loop
    if(i == 1)
      FirstLoop_t = toc;
      fprintf('\nEstimated Duration = %d minutes\n',FirstLoop_t*length(A)
          /60);
    end

  end
toc

fprintf('\nConverting Back to MxN matrices...\n');
tic
  D_Back_og = U_DATA(:,1:s_)*Ds_Back;
  D_Fore_og = U_DATA(:,1:s_)*Ds_Fore;
  D_all_og = U_DATA(:,1:s_)*Ds_all;

  %D_all = D_Back_og + D_Fore_og;
  D_Back = real(D_Back_og);
  D_Fore = real(D_Fore_og);
  D_all = real(D_all_og);
toc


%Comment this section to view foreground separation without using
%Background substraction with residual
D_Fore = DATAMovie_s - abs(D_Back_og);
D_Fore_Copy = D_Fore;

fprintf('\nCreating Residual Matrix...\n');
tic
  R = zeros([size(D_Fore)]);

  D_Fore_Copy(D_Fore_Copy<0) = 0;
  R = D_Fore - D_Fore_Copy;
  R(R<-40) = 3*R(R<-40);%-40 for test data

  D_Fore = D_Fore - R;

  D_Fore(D_Fore>180) = 255;
toc
%-----------------------------------------------------------------%



%--------- View Reconstruction Videos ---------%

fprintf('\nConverting and Displaying Background Video...\n');
tic
  for k = 1:vidNFrames
      mov2(k).cdata = [cast(D_Back(:,k),'uint8')];
      mov_Back(k).cdata = reshape(mov2(k).cdata,vidHeight,vidWidth);
      mov_Back(k).colormap = gray(255);
```

```
  end

  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth vidHeight]);
  movie(hf, mov_Back, 1, vid.FrameRate);
  nfigure = nfigure + 1;
toc

fprintf('\nConverting and Displaying Foreground Video...\n');
tic
  for k = 1:vidNFrames
      mov2(k).cdata = [cast(D_Fore(:,k),'uint8')];
      mov_Fore(k).cdata = reshape(mov2(k).cdata,vidHeight,vidWidth);
      mov_Fore(k).colormap = gray(255);
  end

  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth vidHeight]);
  movie(hf, mov_Fore, 1, vid.FrameRate);
  nfigure = nfigure + 1;
toc

fprintf('\nConverting and Displaying All Video...\n');
tic
  for k = 1:vidNFrames
      mov2(k).cdata = [cast(D_all(:,k),'uint8')];
      mov_all(k).cdata = reshape(mov2(k).cdata,vidHeight,vidWidth);
      mov_all(k).colormap = gray(255);
  end

  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth vidHeight]);
  movie(hf, mov_all, 1, vid.FrameRate);
  nfigure = nfigure + 1;
toc


%--------- View Reconstruction Images ---------%

fprintf('\nDisplaying Background image at 1/4 in the movie...\n');
tic
  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth vidHeight]);
  imshow(mov_Back(FrameInterest).cdata,mov_Back(FrameInterest).colormap);
  nfigure = nfigure + 1;
toc

fprintf('\nDisplaying Foreground image at 1/4 in the movie...\n');
tic
  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth vidHeight]);
  imshow(mov_Fore(FrameInterest).cdata,mov_Fore(FrameInterest).colormap);
  nfigure = nfigure + 1;
toc
```

```
348
349 fprintf('\nDisplaying all image at 1/4 in the movie...\n');
350 tic
351   hf = figure(nfigure);
352   set(hf, 'position', [150 150 vidWidth vidHeight]);
353   imshow(mov_all(FrameInterest).cdata,mov_all(FrameInterest).colormap);
354   nfigure = nfigure + 1;
355 toc
356
357 %Clear Variables to relieve memory
358 clear('D_Back_og');
359 clear('D_Fore_og');
360 clear('D_all_og');
361 clear('Ds_Back');
362 clear('Ds_Fore');
363 clear('Ds_all');
364 clear('Xs');
365 clear('Ys');
366 clear('DATAMovie');
367 clear('DATAMovie_s');
368 clear('DATA_SxN');
369 clear('U_DATA');
370 clear('FirstLoop_t');
371 clear('Completed');
372 clear('Percentage');
373 clear('S_DATA');
374 clear('V_DATA');
375 clear('grayImage');
```