AMATH 482 Homework 5:
Background Subtraction in Video Streams

Amelia Nathan

March 17, 2021

Abstract

This investigation highlights Dynamic Mode Decomposition as a computational technique to process video streams. Tasked with isolating foreground and background videos, DMD with the addition of some filtering manipulation was successful.

1 Introduction and Overview

1.1 Introduction to Dynamic Mode Decomposition

Dynamic Mode Decomposition (DMD), like Singular Value Decomposition, Principal Component Analyis, and Proper Orthogonal Decomposition, focuses on low-rank approximations to analyze data. DMD is especially useful in data-based algorithms when we do not have equations to describe a relationship or situation and also enables prediction. DMD can be used in a variety of applications, for example, video processing and isolation, which will be the focus of this investigation.

1.2 Problem Overview

In this investigation we are tasked with separating the foreground and background from videos and reconstructing two videos with the isolated foreground and backgrounds. To do so we are to use Dynamic Mode Decomposition, building on previously explored methods like Singular Value Decomposition.

2 Theoretical Background

2.1 Foundational Ideas in Linear Algebra

A key theory directly related to matrices that gives the SVD significance in practice and motivates DMD is the following (from lecture notes):

If A is a matrix of rank r, then A is the sum of r rank 1 matrices:

$$A = \Sigma_{j=1}^{r} \sigma_j u_j v_j^* \tag{1}$$

where $u_j v_j^*$ is the outer product and the resulting matrix is rank 1. Ultimately, this foundational idea leads to the possibility of dimensionality reduction which, for high rank matrices, greatly reduces computational time. This notion of approximation greatly informs analytical methods. As will be discussed in the next theoretical background section, Singular Value Decomposition is based on this idea of dimensionality reduction and approximation.

Dynamic Mode Decomposition rests on this notion of dimension reduction as well. As an algorithm, DMD tries to utilize the low-dimensionality in experimental data. This will be expanded on in the theoretical section dedicated to DMD.

2.2 Singular Value Decomposition

Singular value decomposition is a factorization technique that stretches and transforms vectors by using two unitary matrices U and V and a diagonal matrix Σ. The simple definition of the SVD is given by the following equation where A is the matrix being deconstructed:

$$A = U\Sigma V^*, \tag{2}$$
$$U \in \mathbb{R}^{mxm}, \quad V \in \mathbb{R}^{nxn}, \quad \Sigma \in \mathbb{R}^{mxn}$$

The matrix Σ is a diagonal matrix with singular values along the diagonal. U and V are unitary and geometrically, do the following as described in lecture notes:

- Multiplying by V^* facilitates rotation
- Multiplying by Σ facilitates stretching
- Multiplying by U facilitates proper orientation

As previously mentioned, Σ contains singular values. U and V are unitary matrices; the columns of U contain left singular vectors and the columns of V are right singular vectors, all with respect to the factored matrix A. In finding the SVD, it is important to note that eigenvalues and eigenvectors form a key component. Calculating the eigenvalues of A^TA and AA^T is a key step as the singular values are the squares of these eigenvalues (both of these matrix products have the same eigenvalues).

2.3 Dynamic Mode Decomposition

Dynamic Mode Decomposition ultimately functions by creating a basis of spatial modes; unlike other methods we have studied, these spatial modes do not have to be orthogonal. Another key characteristic of the basis is that time is expressed as exponential functions where the exponents can be real or complex. This basis formation simplifies data into essentially a linear system of differential equations.

Setting up data to use in DMD, we have:

N = number of spatial points saved per unit time snapshot
M = number of snapshots taken
Time must be linearly spaced and consistent, where we have:

$$t_{m+1} = t_m + \delta t, \qquad m = 1, ..., M-1, \qquad \delta t > 0 \tag{3}$$

$$\text{Snapshots are: } U(x, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \vdots \\ U(x_n, t_m) \end{bmatrix} \tag{4}$$

$$\text{Snapshots into column matrices: } X = [U(x, t_1), U(x, t_2)...U(x, t_M)] \tag{5}$$

$$X_j^k = [U(x, t_j), U(x, t_{j+1})...U(x, t_k)]. \tag{6}$$

Involved in finding the desired basis is an approximation of modes of the Koopman operator which is independent of time and linear, shown as A in the following equation. j refers to the time of data collection and x_j is an N-dimensional collection of data points from time j.

$$x_{j+1} = Ax_j \tag{7}$$

Using the Koopman operator we can transform how we write the data collected from:

$$X_1^{M-1} = [x_1, x_2, ..., x_{M-1}] \tag{8}$$

to

$$X_1^{M-1} = [x_1, Ax_1, A^2x_1, ..., A^{M-2}x_1] \tag{9}$$

which is the basis for the Krylov Subspace, the space of a vector multiplied by increasing powers of a matrix. Rewriting this again and then incorporating an SVD of our data matrix, we have (with e_{M-1} as a residual vector):

$$X_2^M = AX_1^{M-1} + re_{M-1}^T \tag{10}$$

$$X_2^M = AU\Sigma V^* + re_{M-1}^T \tag{11}$$

$$\text{Using } U^*r = 0, \qquad U^*X_2^M = U^*AU\Sigma V^* \tag{12}$$

$$U^*AU = U^*X_2^M V\Sigma^{-1} = \tilde{S}(\text{input data}) \tag{13}$$

Note that $\tilde{S}$ and A are similar, meaning they are related by multiplying one by a matrix and the other by the inverse of that matrix, and therefore have the same eigenvalues and share eigenvectors. Finding the eigenvalues and eigenvectors we uncover the DMD modes, can describe our matrix with A as a repeated multiplier, and solve for b:

$$\tilde{S}y_k = u_k y_k \tag{14}$$

$$\psi_k = Uy_k \tag{15}$$

$$X_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t})b \tag{16}$$

$$t = 0, \qquad x_1 = \Psi b \rightarrow b = \Psi^{\dagger} x_1, \tag{17}$$

$\Psi^{\dagger}$ = pseudoinverse of Ψ which contains eigenvectors as columns.

In the final equation K refers to the rank of X_1^{M-1}. As will be seen in the algorithm implementation section next, using as low a rank as is necessary to capture sufficient information is a key goal in DMD as has been in previous investigations this quarter like PCA. We know that, for example, using rank 5 refers to first 5 singular values along the diagonal of our Sigma matrix. K provides information about how the data varies and using a low rank improves computation time and storage requirements as we capture most of the information in as few modes as possible.

3 Algorithm Implementation and Development

Described below in algorithm 1, to implement DMD we first have to initially process the data by reading in each movie frame, converting to grayscale, and creating the necessary matrices for DMD as previously described in the theoretical section. Algorithm 1 also involves performing singular value decomposition to help us better understand the data and how many ranks are necessary to use in our subsequent analysis.

Algorithm 1: Data Organization and Set Up

Read in movie using `VideoReader`
for each frame in video **do**
  Convert colored frame to grayscale using `rgb2gray`
  Reshape frame data to a column and add to cumulative matrix, X
end for
Create X_1^{M-1} and M_2^M from X
Use `svd()` to perform singular value decomposition and find U, Sigma, and V matrices
Define and plot a vector of the singular values

Following the implementation of algorithm 1, the key step of reducing the rank of our U, Sigma, and V

matrices must occur. To do so, I viewed the energy distribution of singular values by plotting, shown in the computational results section. After determining the rank necessary and truncating U, Sigma, and V matrices accordingly, algorithm 2 is implemented to finish DMD and construct our sparese and low-rank matrices.

Algorithm 2: Dynamic Mode Decomposition

Define dt as 1 and t as a vector of each video frame to ensure linear time steps as required by DMD
Define $\tilde{S} = U^* X_2^M V \Sigma^{-1}$ and compute eigenvalues and eigenvectors.
Calculate the pseudoinverse and DMD modes
Calculate residual and form low-rank and sparse matrices
Apply filters to refine background and foreground
Plot using `imshow()` and by reshaping again

After implementing Dynamic Mode Decomposition and attaining my low rank and sparse matrices as shown in the two algorithms, I made several attempts at improving the foreground videos using different filters. I started with the algorithm described in lecture, by simply adding and subtracting the residual to lowrank and sparse matrices. This worked well for the background in both cases. However, I found my implementation for the foreground did not isolate the car and skier sufficiently, so I continued to try filters utilizing maximums, minimums, and thresholds for the sparse matrix, details of which can be seen in the appendix and the results in the next section.

4 Computational Results

Seen below are the singular values and eigenvalues for each test case. The Eigenvalue plots provided information about the foreground and background, with the omega values near 0 representing background information. As can be seen, the first singular value in both cases contain most of the information about the data. However, to ensure I had enough information I decided to use the first 5 modes, before the difference between successive modes became visually indistinguishable. In hindsight I would have preferred to implement a summation of singular values up to a threshold to determine the ideal number of modes to use rather than a visual test with the plot. I do believe the rank 5 implementation I did use was effective, although coding in a method to determine the rank as I did in previous assignments would be more versatile and reliable.

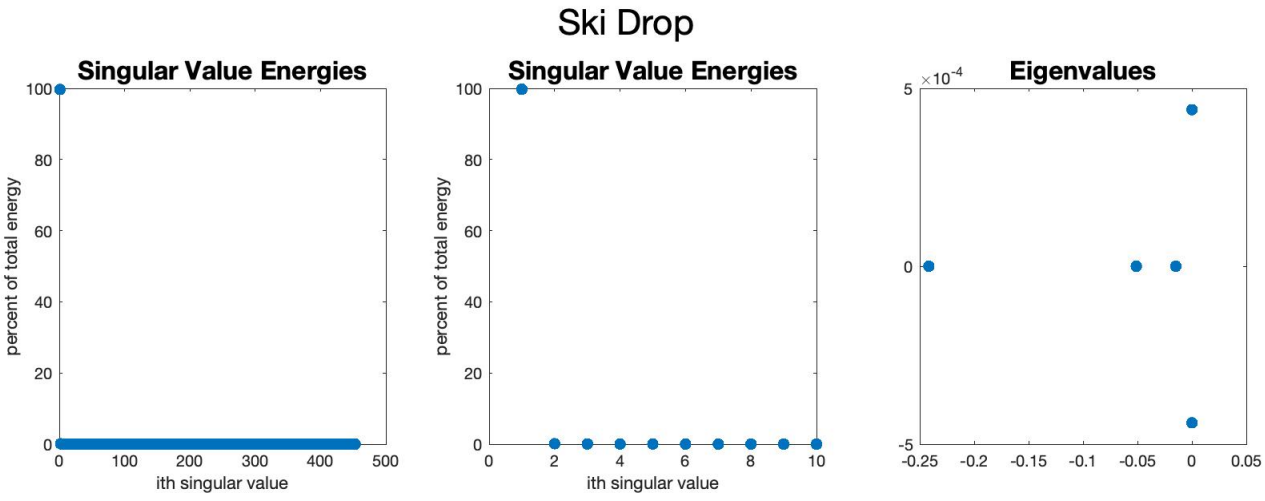


Figure 1:

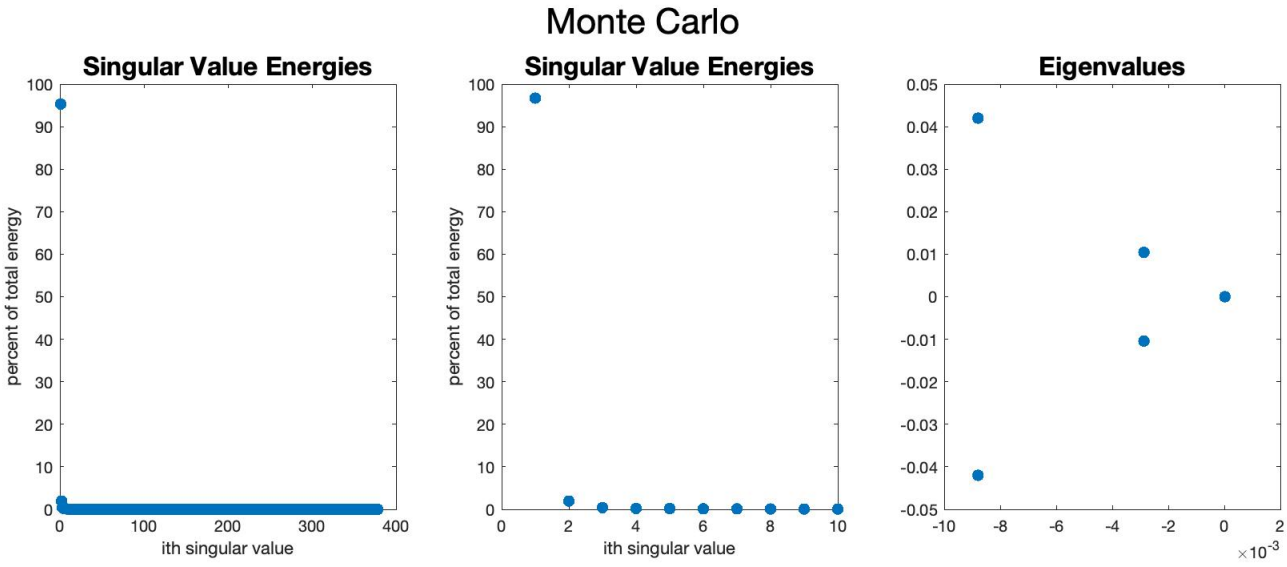


Figure 2:

Pictured below are the filtering results. In addition to the method from lecture, I tried two additional filters for the Skier and one for the race car. I also tried `imadjust()` but did not notice significant change in results, although the contrast might have been slightly more apparent. Ultimately, the third method was most effective in isolating the skier as it did not black out the entire frame, providing a little more context for the skier as they move down the mountain. While the second method did isolate the skier well in white, it was more difficult to detect the context. For the race car, the lecture method was sufficient in emphasizing the car and erasing most of the background. The second method I tried did an even better job isolating the car, but it emphasized contrast a lot that led to some details in the background being picked up on as well. When watching the reconstructed videos, I think the third method for the skier and second for the car make the foreground easiest to follow. To better show the skier results, I zoomed in on the skier to compare the different filtering methods. I then include three frames with normal zoom to demonstrate how the skier appeared in context of the video, since it was a very small foreground piece to isolate. I've also shown the reconstruction for both cases, for which the method described in class, by combining the low rank and sparse matrices with the residual, was successful.

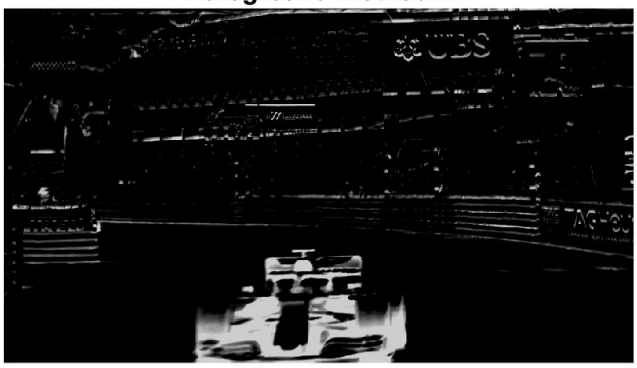
5 Summary and Conclusions

Ultimately, DMD was effective in separating the foreground and background videos. While manipulating filtering to get the best foreground was necessary, this investigation has shown the power of DMD and importance of dimensionality reduction.

Background

Foreground Method 1

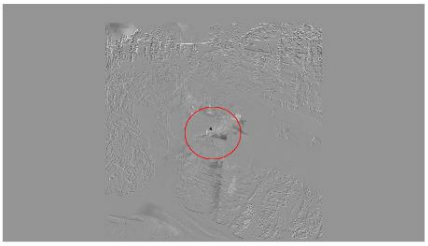
Foreground Method 2



Reconstructed

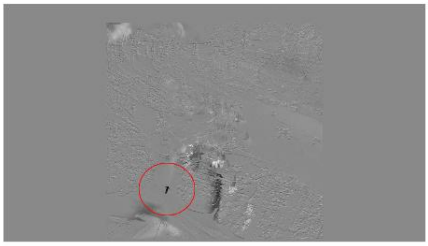
Ski Drop Method 3

Frame 150



Frame 225

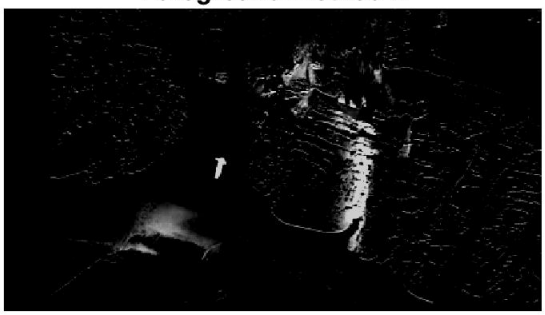
Frame 300



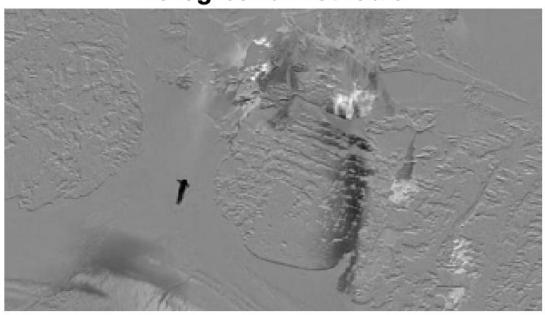
Background

Foreground Method 1

Foreground Method 2



Foreground Method 3



Reconstructed Image

Appendix A MATLAB Functions

- `min(X)` Finds the minimum value within X
- `sum(X)` Finds the sum of values in X
- `mean(X)` Finds the mean of values in X
- `[x, y] = size(X)` Returns the dimensions of a matrix X in terms of number of rows and columns
- `[U, S, V] = svd(A)` Computes the singular value decomposition of matrix A, returning a diagonal matrix S and two unitary matrices U and V such that X = U*S*V'
- `diag(V)` Creates a diagonal matrix with vector V along the diagonal
- `double` Casts values as type double
- `ones, zeros(m,n)` Creates a matrix of ones or zeros with dimensions m x n.

Appendix B MATLAB Code

```
% AMATH 482
% Assignment 5 - Video Foreground and Background Isolation


movie_name = 'monte_carlo_low.mp4';

vid = VideoReader(movie_name);
numFrames = vid.NumFrames;
for i = 1:numFrames
    colored_frame = read(vid,i);
    % imshow(colored_frame);
    x = rgb2gray(colored_frame);
    x = imadjust(x);
    x_reshaped = reshape(x,[],1);
    data(:,i) = x_reshaped;
end
data = double(data);
data1 = data(:,1:end-1);
data2 = data(:,2:end);

[U, Sigma, V] = svd(data1, 'econ');


lambda = diag(Sigma).^2; % vector of singular values

subplot(1,3,1)
plot(1:length(lambda), lambda/sum(lambda)*100,
'.','MarkerFaceColor','#0072BD', 'MarkerSize', 20);
title('Singular Value Energies','Fontsize', 16);
xlabel('ith singular value')
ylabel('percent of total energy')
subplot(1,3,2)
plot(1:10, lambda(1:10,:)/sum(lambda(1:10))*100, '.','MarkerFaceColor',
'#0072BD', 'MarkerSize', 20);
title('Singular Value Energies','Fontsize', 16);
xlabel('ith singular value')
ylabel('percent of total energy')


rank = 5;
U = U(:,1:rank);
Sigma = Sigma(1:rank,1:rank);
V = V(:, 1:rank);



dt = 1;
t = 1:numFrames-1;
S = U'*data2*V*diag(1./diag(Sigma));
[ev, D] = eig(S);
mu = diag(D);
omega = log(mu)/(dt); % the omegas close to 0 represent dmd modes that best
describe background

subplot(1,3,3)
plot(omega, '.', 'MarkerSize', 20)
```

```
title('Eigenvalues','Fontsize', 16)
omega = omega.*(abs(omega)<0.05);
Phi = U*ev;
Phi = Phi(:,abs(omega)<0.05);
y0 = Phi\data1(:,1); % pseudoinverse
y0 = y0.*(abs(omega)<0.05);
sgtitle('Monte Carlo', 'Fontsize', 22)
u_modes = zeros(length(y0),length(t));
for j = 1:length(t)
    u_modes(:,j) = y0.*exp(omega*t(j));
end
u_dmd = Phi*u_modes;
%% For cars
x_lowrank = (abs(u_dmd));
x_sparse = data1 - x_lowrank;

r = x_sparse.*(x_sparse < 0);
x_sparse = uint8(x_sparse - r);
% x_sparse = uint8(x_sparse.*(x_sparse>700) - r).*3 - uint8(50);
x_lowrank_fr = uint8(x_lowrank + r);
reconstruction_car = uint8(x_sparse + x_lowrank_fr);
x_lowrank = uint8(x_lowrank); % assignment says add r, but in practice this
adds the car back in
figure(7)
subplot(2,2,1)
imshow(reshape(x_lowrank(:,100),[],vid.Width)) % background
title('Background', 'Fontsize', 16)
subplot(2,2,2)
imshow(reshape(x_sparse(:,100),[],vid.Width))
title('Foreground Method 1', 'Fontsize', 16)

x_lowrank = (abs(u_dmd));
x_sparse = data1 - x_lowrank;
r = x_sparse.*(x_sparse < 0);
x_sparse = uint8(x_sparse.*(x_sparse>700) - r).*3 - uint8(50); % second
method
subplot(2,2,3)
imshow(reshape(x_sparse(:,100),[],vid.Width))
title('Foreground Method 2', 'Fontsize', 16)
subplot(2,2,4)
imshow(reshape(reconstruction_car(:,100),[],vid.Width))
title('Reconstructed', 'Fontsize', 16)
%%
figure(2)
for frame = 1:numFrames - 1 % foreground
    imshow(reshape(x_sparse(:,frame),[],vid.Width))
end

% plot reconstruction

%% For Skier

x_lowrank = (abs(u_dmd));
x_sparse = data1 - x_lowrank;
figure(5)
r = x_sparse.*(x_sparse < 0);
%subplot(2,2,2)
```

```
x_sparse = uint8(x_sparse - r); % original, but cannot see skier as they
blend into the black
imshow(reshape(x_sparse(:,300),[],vid.Width))
title('Foreground Method 1', 'Fontsize', 16)
% x_lowrank = uint8(x_lowrank); % assignment says add r
x_lowrank = uint8(x_lowrank + r);
reconstruction = x_sparse + x_lowrank;
imshow(reshape(reconstruction(:,300),[],vid.Width))
title('Reconstructed Image', 'Fontsize', 20)
%%

subplot(2,2,1)
imshow(reshape(x_lowrank(:,300),[],vid.Width)) % background
title('Background', 'Fontsize', 16)

x_lowrank = (abs(u_dmd));
x_sparse = data1 - x_lowrank;

r = x_sparse.*(x_sparse < 0);

x_sparse = uint8(x_sparse.*(x_sparse>500) - r).*5 - uint8(50); % method 2 -
magnifies the skier
subplot(2,2,3)
imshow(reshape(x_sparse(:,300),[], vid.Width))
title('Foreground Method 2', 'Fontsize', 16)

% foreground method 3
x_lowrank = (abs(u_dmd));
x_sparse = data1 - x_lowrank;

r = x_sparse.*(x_sparse < 0);
x_sparse = (x_sparse - min(x_sparse(:,:))./(max(x_sparse(:,:)) -
min(x_sparse(:,:)));
subplot(2,2,4)
imshow(reshape(x_sparse(:,300),[], vid.Width))
title('Foreground Method 3', 'Fontsize', 16)

%%
figure(2)
for frame = 1:numFrames-1
    imshow(reshape(x_sparse(:,frame),[], vid.Width)) %foreground
end
%%
figure(6)
subplot(1,3,1)
imshow(reshape(x_sparse(:,150),[], vid.Width))
title('Frame 150', 'Fontsize', 16)
subplot(1,3,2)
imshow(reshape(x_sparse(:,225),[], vid.Width))
title('Frame 225', 'Fontsize', 16)
subplot(1,3,3)
imshow(reshape(x_sparse(:,300),[], vid.Width))
title('Frame 300', 'Fontsize', 16)
sgtitle('Ski Drop Method 3', 'Fontsize', 20)
```