Amath 482 Hw 5 - Background Separation: DMD

Ofir Kedar

Mar 17, 2021

Abstract

Tasked with the challenge of separating the foreground from the back ground of a video, we use Dynamic Mode Decomposition (DMD) to determine the steady state of the system (video), and predict what it should look like in future frames. By deconstructing the original matrix of frames over time we are able to reconstruct it through dynamically iterating over it's independent modes.

1 Introduction and Overview

We are presented with two different videos; one of cars racing across a stretch of a track, and the other of a skier traversing down a mountain. Our objective is to mathematically separate the moving components (foreground) in each video from the stationary setting they occur in (background). We will use Dynamic Mode Decomposition to to separate the two things in each video.

2 Theoretical Background

2.1 Dynamic Mode Decomposition (DMD)

DMD simply put, predicts future data. It is the data driven equivalent to a mathematical solution representing a known model. By this I mean that we take DMD takes large sets of data, and constructs a linear solution to what future data should look like, if it is to accurately represent the patterns formed in previously collected data in the set. It does this by approximating the modes of the **Koopman Operator**. The overarching idea here, while not necessarily intuitive to implement, is actually quit simple in nature; multiply a previous data sample at time **j** by some global matrix of this system **A**, such that the product results in my next data sample in time. [1]

$$\text{Koopman Operatot} : x_{j+1} = A * x_j \tag{1}$$

The challenging part comes in finding this matrix A, such that our proposed linear system is true for the following;

$$\text{Data Samples in Time} : X_j^k = [U(x,t_j) \quad U(x,t_{j+1}) \quad . \ . \ . \quad U(x,t_k)]. \tag{2}$$

$$\text{Koopman Operator Condition} : X_1^{M-1} = [x_1 \quad A*x_1 \quad A^2*x_1 \quad . \ . \ . \quad A^{M-2}*x_1)]. \tag{3}$$

Using these rules we deduce that;

$$\text{Enlarge Data Set by One Sample} : X_2^M = A*X_1^{M-1} + \boldsymbol{r}e_{M-1}^T = A*U*\Sigma*V^* + \boldsymbol{r}e_{M-1}^T \tag{4}$$

Where **U**, **Σ**, and **V** are the result of computing the SVD of X_1^{M-1}, $\boldsymbol{e_{M-1}}$ is a vector with all zeros except a 1 at the (M - 1)st component to account for the size difference in matrices, and **r** is the residual vector. Apply inverse matrices to format in Similar Matrix form and we get

$$\text{Similar Matrix Form} : U*A*U^* = U^**X_2^M*V*\Sigma^{-1} \tag{5}$$

In a controlled data set we dissect the original matrix X, from the first, to second-to-last element to find X_1^{M-1} and from the second to the last element to find X_2^M. This means that the right hand side of the equation is easily commutable. We right hand side S_{tilda}

$$\text{Similar Matrix Form} : U * A * U^* = S_{tilda} \tag{6}$$

This means that we now know the eigen values of A, since using properties of similar matrices we know that they are the same as that of S_{tilda}. We call the eigen vectors of S_{tilda} and A, and call **y** the eigen vectors of S_{tilda}. Using S_{tilda} and μ we can compute y.

$$\text{Eigen Vectors of } S_{tilda} : S_{tilda\_k} * y_k = \mu_k * y_k \tag{7}$$

Now that y is solved for, we can finally find the eigen vectors of A which is out last piece before constructing our currently jumbled puzzle.

$$\text{Eigen Vectors of A} : \psi = U * y_k \tag{8}$$

$$\text{Expansion Coefficient} : b = \psi^+ x_1 \tag{9}$$

$$\text{Continuity of } \mu_k : \omega = ln(\mu_k)/\Delta t \tag{10}$$

Above, we define b at t = 0, given the initial conditions. Ψ is the matrix who's columns are composed of the individual ψ_k vectors, and Ψ^+ is the pseudo-inverse of Ψ. Now we can take all of our pieces, and construct our desired function. This is the actual function used to determine a prediction of data the data samples at time t based off of everything we just did to find our missing components in the Koopman Operator.

$$\text{Continual Multiplication Function of A} : X_{DMD} = \sum_{k=1}^{K} (b_k * \psi_k * e^{\omega_k * t}) = \Psi * diag(e^{\omega_k * t}) * \boldsymbol{b} \tag{11}$$

3 Algorithm Implementation and Development

3.1 Dynamic Mode Decomposition (DMD)

Luckily for the purpose of implementation, the mathematical definition of the DMD method, is in itself an algorithm, its definition relies on a step by break down of each component, which it then uses to build the next next component of the puzzle. This means that if you understand the theoretical implications of the mathematics, it's quite easy to implement the method algorithmically, by computing each step explained in the definition. Here we also reduce out matrix such that we only use as many modes as needed. This speeds up the computation process greatly.

Algorithm 1: Dynamic Mode Decomposition (DMD)

```
v = VideoReader('video file name');
video = read(v);
width height rgbval frames= size(video);

for i = 1:frames do
   x = rgb2gray(video(:,:,:,i));vidframes(:,i) = double(reshape(x(:,:),width*height,1));
end for

X1 = vidframes(:,1:end-1);
X2 = vidframes(:,2:end);
U,S,V= svd(X1,'econ');

diagS = diag(S)/sum(diag(S))*100;
percent = 0;
i = 1;
while percent ¡ 80 do
   percent = percent + diagS(i);i = i+1;
end while
feature = i-1;

U = U(:,1:feature);
S = S(1:feature,1:feature);
V = V(:,1:feature);

%% Stild's eig vals are equal to the eig vals of arbitraty matrix A
% A = (the koopman opperator)
Stild = U'*X2*V/S;
eigVec, eigVal= eig(Stild);

Mu = diag(eigVal); % contains eig val of dmd
dt = 1; omega = log(Mu)/dt; %continous eig val
Phi = U*eigVec; % eig vec of matrix A (the koopman opperator)
% Now we have found eig vec and eig val of A

y0 = Phi1(:,1); % pseudoinverse to get initial conditions
t = 1:frames - 1;
u_modes = zeros(length(y0),length(t));

for iter = 1:length(t) do
   u_modes(:,iter) = y0.* exp(omega * t(iter));
end for
u_dmd = Phi * u_modes;
```

4 Computational Results

By applying the Continual Multiplication Function of A (equation 11) to the frames in the video, we can separate the background and foreground, and view them separately. Below are labeled snap shots of the edited and unedited videos at the same point in time (frame 200 of each video), showing the original frames, background only, and foreground only, with some of the foreground pictures adjusted to extenuate the contrast allowing them to be isolated and located more easily by eye.

4.1 Numerical Breakdown

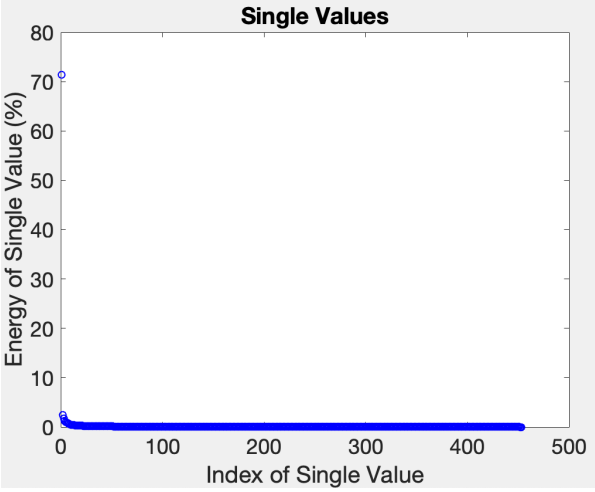

Figure 1: Singular Values of Ski Video

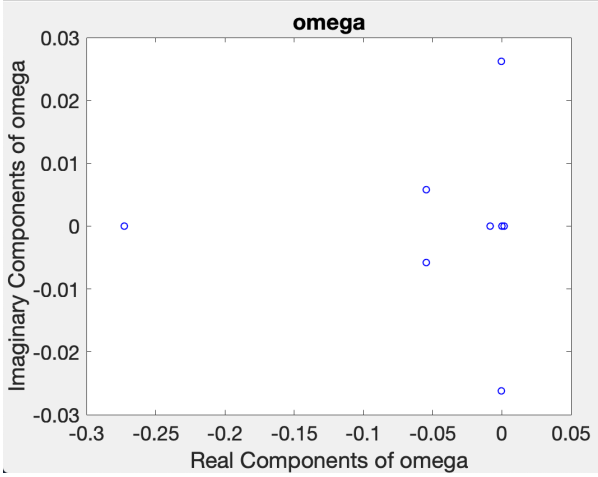

Figure 2: Omega Values of Ski Video

Its clear in the above chart that much of the information about this particular system (Ski video) in contained in the first mode, holding over 70%. upon further inspection it can quickly be computed that using the first 8 modes of the system we are able to gather 80% of the information. This reduces our system down from over 450 modes to allow much faster computation. We can then find the the continuous single values in time and plot their real vs imaginary values, these are the omega values. Notice that these are selected along the diagonal of the Single value matrix, and that they are filtered to give a real value smaller than or equal to 0.1. If there were real numbers much greater than 0, it would cause the DMD to increase the magnitude of its predicted values exponentially and would eventually cause it to "explode".

4.2 Car Video Results

Figure 3: Original Snap Shot of Cars

Figure 4: Background Snap Shot of Cars

Figure 5: Foreground Snap Shot of Ski

4.3 Ski Video Results

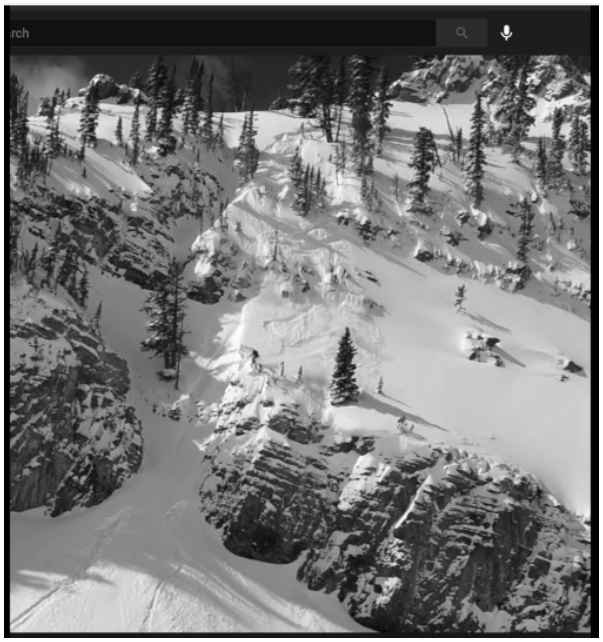

Figure 6: Original Snap Shot of Ski

Figure 7: Background Snap Shot of Ski

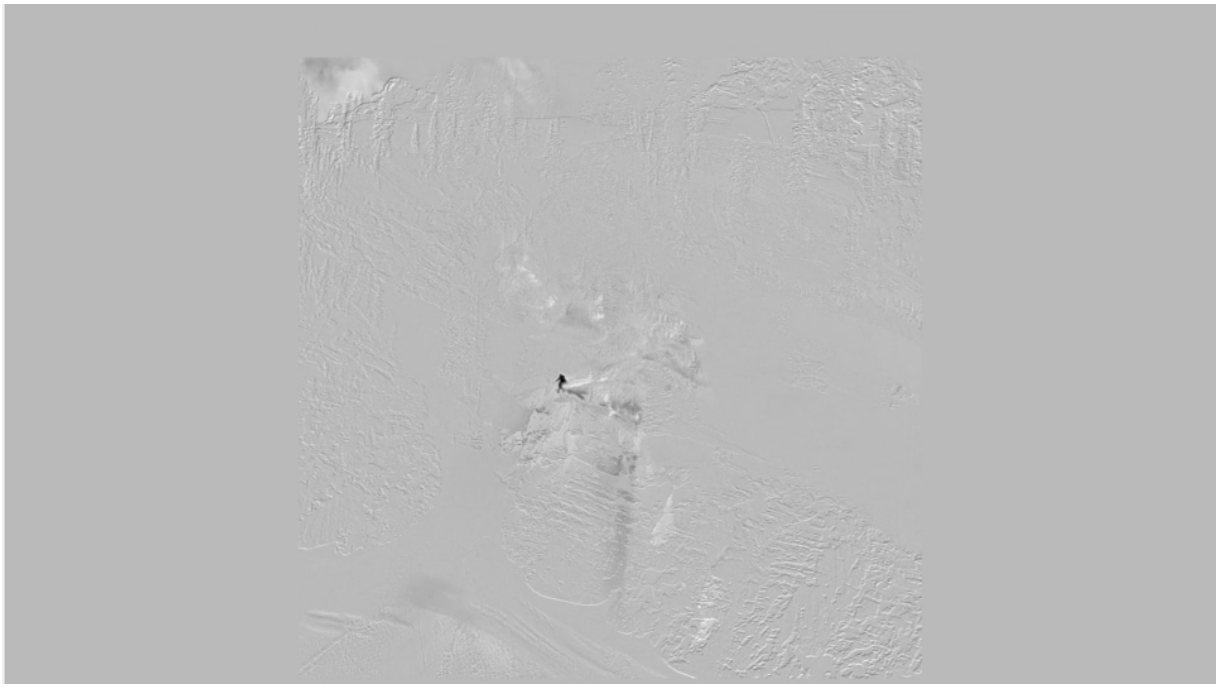

Figure 8: Foreground Snap Shot of Ski

5 Summary and Conclusions

With static space through time thought of as a steady state, we can separate moving objects from a static background in a video Using Dynamic Mode Decomposition. Here we relying on SVD to analyse the Principal Components of a video. Using the matrix that forms each frame in the video we can dynamically extrapolate what future steady frames should look like, and through this, reconstruct each frame such that it excludes information of space that changes over time, aka; takes out moving objects. By contrasting these newly constructed matrices of each frame with the ones that compose the original video, we can then extract the information of only the moving space, use this to view moving objects without their background. Since this entire technique is dependant on only one SVD and one over all linear system computation, it is able to achieve very fast separation. An interesting thought experiment here is if this method could then be used in green screen technology. Recording a video on a blatantly static, simple, and high contrast background, could allow for easy high quality extraction of the foreground where. The background could then be set to all 0's, in each frame. Taking a newly desired background you would be able to set the indices of the new background matrix to 0 at the location of the none zero indices of the foreground, then simply add the two matrices to effectively place your foreground into a new back ground in each frame.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `[images, labels] = mnist_parse('a', 'b')` takes ubyte files a and b and returns then in uint8 and double files respectively
- `images = im2double(images);` takes an image as input, and returns an image of class double
- `reshape(X,M,N) or reshape(X,[M,N])` returns the M-by-N matrix whose elements are taken columnwise from X. An error results if X does not have M*N elements.
- `OBJ = VideoReader(FILENAME)` constructs a multimedia reader object, OBJ, that can read in video data from a multimedia file.
- `DATA = read(OBJ)` reads the requested item from the input stream.
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale intensity image I.
- `imshow(I) displays the grayscale image I`
- [U,S,V] = svd(X) produces a diagonal matrix S, of the same dimension as X and with non-negative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V'.

Appendix B Additional Graphs

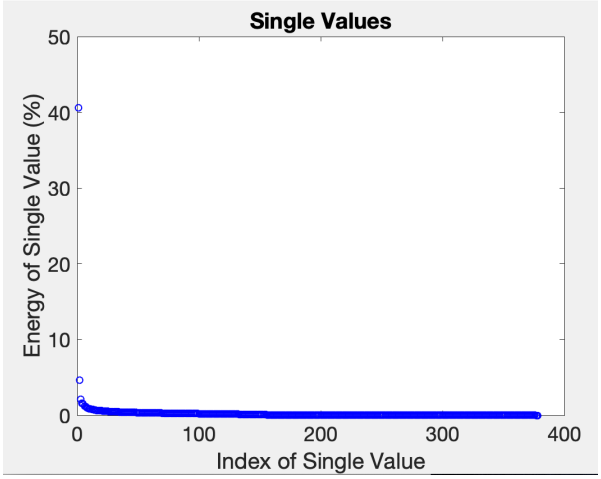

Figure 9: Singular Values of Car Video

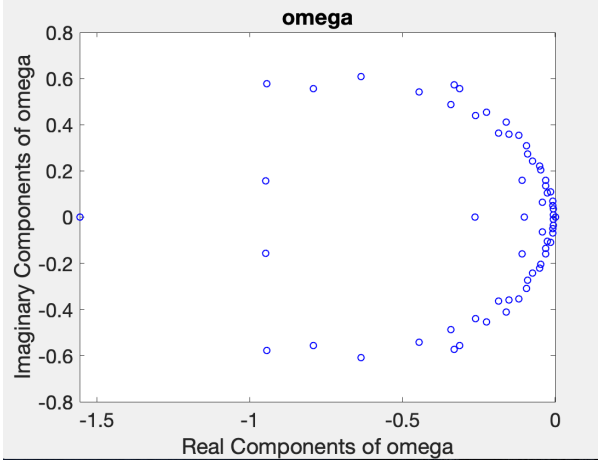

Figure 10: Omega Values of Car Video

Appendix C MATLAB Code

```
%% Assignement 5

% must clear all to chage between videos
clear all;
close all;
clc;

%% audio Indicators
WarnWave = [sin(1:0.05:150), sin(1:0.1:150)];
Audio1 = audioplayer(WarnWave, 22050); % done section
WarnWave = [sin(1:0.1:200), sin(1:0.15:200), sin(1:0.2:200)]; % done code
Audio2 = audioplayer(WarnWave, 22050);
% play(Audio1);
% play(Audio2);

%%  Load Car Vid                                  - not in use
v = VideoReader('monte_carlo_low.mp4');
video = read(v);
[width height rgbval frames] = size(video);
done = 1
play(Audio1);


%% Load Ski Vid                                   - in use
v = VideoReader('ski_drop_low.mp4');
video = read(v);
[width height rgbval frames] = size(video);
done = 1
play(Audio1);


%% Reshape picture to matrix
for i = 1:frames
    x = rgb2gray(video(:,:,:,i));
    vidframes(:,i) = double(reshape(x(:,:),width*height,1));
end
done = 2
play(Audio1);


%% Compute SVD
X1 = vidframes(:,1:end-1);
X2 = vidframes(:,2:end);
[U,S,V] = svd(X1,'econ');
s = S;
done = 3
play(Audio1);
```

Listing 1:

```
%% Extract Necessary Modes
diagS = diag(s)/sum(diag(s))*100;
percent = 0;
i = 1;
while percent < 80
    percent = percent + diagS(i);
    i = i+1;
end
feature = i-1;


U = U(:,1:feature);
S = S(1:feature,1:feature);
V = V(:,1:feature);
done = 4
play(Audio1);


%%  Stild's eig vals are equal to the eig vals of arbitraty matrix A
%   A is the (the koopman opperator)
Stild = U'*X2*V*diag(1./diag(S));
[eigVec, eigVal] = eig(Stild);

Mu = diag(eigVal); % contains eig val of dmd
dt = 1;
omega = log(Mu)/dt; %continous eig val

Phi = U*eigVec; % eig vec of matrix A (the koopman opperator)
% Now we have found eig vec and eig val of A

y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
t = 1:frames - 1;

omegalater = omega;
omega = omegalater (abs(omega) <= 0.1);
y0 = y0(abs(omegalater) <= 0.1);
Phi = Phi(:,abs(omegalater) <= 0.1);
% Now we have found eig vec and eig val of A

        % in lecture version
u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
    u_modes(:,iter) = y0.* exp(omega * t(iter));
end
u_dmd = Phi * u_modes;
%   end of DMD alg
done = 5
play(Audio1);
```

Listing 2:

```
%%
LR = abs(u_dmd);
LRski = LR;
Xsp = X1 - LR;
err = Xsp .* (Xsp < 0);
done = 6
play(Audio1);


%%

    % LR_err = background
LR_err = uint8(LR + err);
    % Xsp_err = foreground
Xsp_err = uint8(Xsp - err);
done = 7
play(Audio1);


%%

    % LR_err = backgrounf
LR = uint8(LR);
    % Xsp = foreground
Xsp = Xsp_err;
done = 8
play(Audio1);


%% Play Background
close all;

for (i = 1: frames-1)
    image(:,:,i) = reshape(LR(:,i),width,height);
end

% for (i = 1: frames-1)
%     imshow(image(:,:,i));
%     i
% end
done = 9
play(Audio1);


%% Before skier forground to make white
Xsp = X1 - LRski;
Xsp = (Xsp - min(Xsp))./(max(Xsp) - min(Xsp));
done = 10
play(Audio1);
```

Listing 3:

```
%% Play Foreground
close all;

for (i = 1: frames-1)
    image(:,:,i) = reshape(Xsp(:,i),width,height);
end
% for (i = 1: frames-1)
%     imshow(image(:,:,i));
%     i
% end
done = 11
play(Audio1);


%% Comparing Snapshots
close all;

snap = 200;
image_og = rgb2gray(video(:,:,:,snap));

% original snap shot
figure(1);
imshow(image_og);

figure(2);
% edited snap shot back
image = reshape(LR(:,snap),width,height);
imshow(image);

figure(3);
% edited snap shot back
image = reshape(Xsp(:,snap),width,height);
imshow(image);
done = 12
play(Audio1);


%% Plotting Singular Values
figure(4)
plot((diag(s)/sum(diag(s))*100), 'ob')
set(gca,'Fontsize',18)
title('Single Values')
xlabel('Index of Single Value')
ylabel('Energy of Single Value (%)')
done = 13
play(Audio1);
```

Listing 4:

```
%% Plotting Omega
figure(5)
plot(real(omegalater), imag(omegalater), 'ob')
set(gca,'Fontsize',18)
title('omega')
xlabel('Real Components of omega')
ylabel('Imaginary Components of omega')
done = 14
play(Audio2);
```

Listing 5: