# AMATH 482 Homework 5

lewisxy

March 17, 2021

**Abstract**

In this paper, we demonstrated the possibility of separating backgrounds and foregrounds of short video clips using Dynamic Mode Decomposition (DMD).

## 1 Introduction and Overview

Videos often contains moving objects (foreground) and a stationary background (when camera position fixed). It is useful in many situations to separate the background and foreground of a video. In this paper, we explore the possibly of using Dynamic Mode Decomposition (DMD) to achieve this, by treating each frame of the video as the data collected from a dynamic system.

## 2 Theoretical Background

### 2.1 Koopman Operator

Given a discrete dynamic system with $x_k$ as $k$th state, we use a linear transform $T(x) = Ax$ to model the transformation from the current state to the next. This linear operator is known as Koopman Operator.

$$x_{k+1} = T(x_k) = Ax_k \tag{1}$$

The above dynamic system modeled by Koopman operator is a system of linear *difference* equations, which has similar properties to system of linear *differential* equations as we will explore later. At a high level, this linear model can capture the solution of exponential growth/decay and oscillations in the form of linear combinations of exponential of complex numbers, which is analogous to the solutions to systems of differential equations $x(t) = e^{tW}x_0$.

### 2.2 Similarity

Similarity is a concept in linear algebra. Square matrices of the same size $A, B$ are similar if and only if there is an invertible matrix $C$ of the same size such that

$$A = C^{-1}BC \tag{2}$$

Similarity is commutative, since equation (2) implies $B = CAC^{-1}$. Let $v, \lambda$ be an eigenvector and eigenvalue of $A$, we have

$$Av = \lambda v \tag{3}$$

Because of similarity, if we rewrite $A = C^{-1}BC$ and multiply $C$ on the left of both side of equation (3), we have

$$B(Cv) = \lambda(Cv) \tag{4}$$

This means $Cv$ is an eigenvector of $B$ with eigenvalue $\lambda$. This is only one of the many useful properties of similarity.

## 2.3 Dynamic Mode Decomposition (DMD)

Given data $(x_1, ..., x_n)$ collected with fixed temporal interval $\Delta t$ such that $x_k = x(t_0 + k\Delta t)$, Dynamic Mode Decomposition (DMD) aims to approximate the Koopman operator $A$ such that $x_{k+1} = Ax_k$. While this assumes the underlying dynamics represented by the data is linear, we can also use this approach to model nonlinear systems since they are approximately linear in small time scale (this even allows us to reasonably predict the near future of the system). This is particularly useful for data collected from an unknown underlying dynamics.

To approximate $A$, we first need to formulate $m$ slice of $n$ dimensional data $(x_1, ..., x_m)$ into 2 $n \times m - 1$ matrices: $X_1^{M-1}$, those columns contain $x_1$ to $x_{m-1}$, and $X_2^M$, those columns contain $x_2$ to $x_m$. With these data, we need to find $A$ such that

$$X_2^M = AX_1^M + re_{M-1}^T \tag{5}$$

$e_{M-1}$ is a $m - 1 \times 1$ vector that is all zero except for the last row that has value 1. So $re_{M-1}^T$ is a matrix that only last column has value $r$ and zero on all other columns. This is to accommodate the fact that we don't actually have the data $x_m$ in $X_1^{M-1}$, and we denote this residual as $r$.

If we take the SVD of $X_1^M$ to be $USV^*$, we are essentially rewriting the columns of $X_1^{M-1}$, or $x_1, ..., x_m$ in the basis of $U$. Since $r$ cannot possibly be represented by these vectors (by definition of residual), $r$ must be orthogonal to all columns of $U$, or $U^*r = 0$. Therefore, if we multiply $U^*$ to the left of equation (5) and do some rearrangements, we have

$$U^*X_2^M = U^*AU\Sigma V^* \tag{6}$$

$$U^*X_2^M V\Sigma^{-1} = U^*AU \tag{7}$$

Of course, since we are taking SVD, we can also perform dimension reduction by truncating columns of $U$, diagonals of $\Sigma$ and rows of $V$. Because DMD works best in short term approximations, we assume $m < n$. This means $U \in \mathbb{C}^{n\times k}, \Sigma \in \mathbb{R}^{k\times k}, V \in \mathbb{C}^{m-1\times k}, k \in [1, m-1]$.

We can continue to compute $A$ from here, but since we only care about its eigenvalues and eigenvectors, we don't have to isolate $A$. Let $S = U^*X_2^M V\Sigma^{-1}$. Notice $S$ is a *similar* matrix of $A$, and we can find the eigenvalues and eigenvectors using the properties we talked about earlier (equation (4)). Let $\mu_k, v_k$ be the $k$th eigenvalue and eigenvector of $S$, we have $\mu_k$ and $\phi_k = Uv_k$ to be the $k$th eigenvalue and eigenvector of $A$.

With all the above, we can write the closed form approximation of the underlying dynamic system as

$$x(t) = \Phi e^{t\Omega} b \tag{8}$$

where $\Phi$ is the matrix those columns are eigenvector $\phi_1, ...m\phi_k$, and $\Omega$ is the diagonal matrix those values are continuous eigenvalues of $\omega_1, ..., \omega_k$ for each $\omega_i = \ln(\mu_i)/\Delta t$, and $b$ is the initial condition, which can be solved from the data (for example, $\Phi b = x_1$ if we assume $x_1$ is taken at $t = 0$).

We can evaluate this function at the same $t$ as original data test the approximation, or evaluate at a different $t$ to predict from the data. We can also analyze the eigenvalues $\omega$ and eigenvectors $\Phi$ to gain insight of the underlying system represented by the data.

# 3 Algorithm Implementation and Development

In this section, we describes the different steps to separate the background and foreground from a video. Implementations are provided in the form of MATLAB code in Appendix B, explanations of selected MATLAB functions are provided in Appendix A.

## 3.1 Preparation

We first need to prepare the data. The raw data is in the form of video frames, and each video frame contains 3 matrices of width by height of the video, each represents the RGB value of pixels of that frame. For the purpose of this paper, we convert each frame to a grayscale image using `rgb2gray` from MATLAB and flattens the image to a vector. Therefore, we have $m \times n$ data matrix those columns are the frames of the video ($m =$ number of frames, $n =$ width $\times$ height, $\Delta t = 1/$frame rate). We then formulates $X_1^{M-1}$ to be the first $m - 1$ columns and $X_2^M$ to be the last $m - 1$ columns of the data matrix.

### 3.2 Separation

We then run the Dynamic Mode Decomposition (DMD) algorithms as mentioned above without reducing dimensions ($k = m - 1$). Since the eigenvalue represents the extent of the change of a certain mode over time and background does not change in the video, we can find a subset of the eigenvalues $\omega_p$ such that $|\omega_p| \approx 0$ and reconstruct the video as the following (evaluate at the same $t$ as in original video based on frame rate).

$$x_{\text{bg}}(t) = \sum_{p \in P} \phi_p e^{\omega_p t} b \tag{9}$$

Of course, since the result can contain complex number, we can take the absolute value of it. To get the foreground video, we can just subtract it from the reconstruction using all $\omega$ (or the original data matrix).

$$x_{\text{fg}}(t) = x(t) - x_{\text{bg}}(t) \tag{10}$$

Since this subtraction can resulted in negative values, we will need some ways to project it to positive domain to be properly displayed as an image (we will explore this later).

## 4 Computational Results

We separated the background from 2 videos: monte_carlo_low.mp4 and ski_drop_low.mp4. Both videos are about 6 seconds long in real time and have 300-500 frames. Due to limited memory, we first down sampled the video by a factor of 2. Then, we perform the above mentioned algorithm for preparation and separation. In both video, we used 3 modes to reconstruct the background by finding the smallest 3 $\omega$ value.To reconstruct the foreground, we tested various different approaches.

Because the reconstructed matrix for background is complex, we take the absolute value of all values (absolute value for complex number is the "length" of the "vector" on complex plane). This is shown in figure 1c and 2c (the absolute value approach behaves well and almost all values are within the range of $[0, 1]$, which is the same as input). We also tried to watch the resulted video. As expected, because the first 3 eigenvalues are small, the video is essentially a still image (all frames are pretty much the same vector).

$$x_{\text{bg,final}}(t) = |x_{\text{bg}}(t)| \tag{11}$$

To extract the foreground, we rely on the principle of equation (10). Using either the absolute value of the fully reconstructed matrix from DMD or the original data matrix gives visually similar results. However, if we perform the subtraction naively, the matrix will contain complex and negative values. One way to fix this is to take the *absolute value of the difference.*

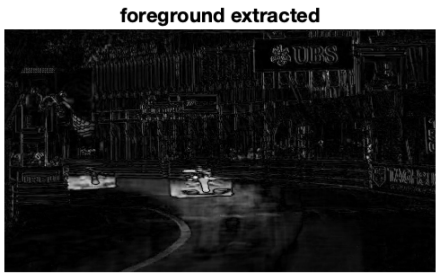
$$x_{\text{fg,final}}(t) = |x(t) - x_{\text{bg}}(t)| \tag{12}$$

This approach is shown in 1a and 2a. It successfully separates the foreground objects, but because the background is black (values close to 0 in grayscale), the contrast, especially for parts of object that is originally dark (has a small grayscale value), is not great. To fix this issue, we tried another approach.

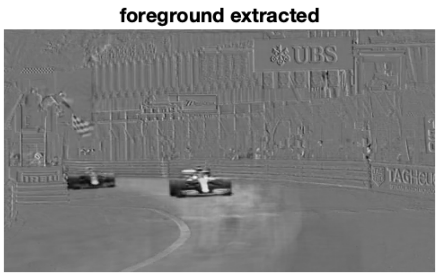
$$\text{normalized}(x) = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{13}$$

$$x_{\text{fg,final}}(t) = \text{normalize}(|x(t)| - |x_{\text{bg}}(t)|) \tag{14}$$

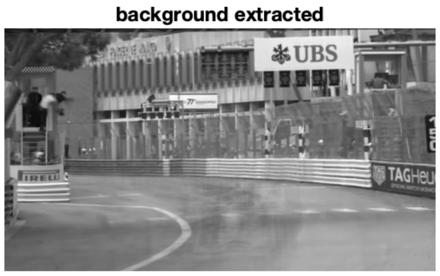
Instead of taking the absolute value of the difference, we take the *difference of absolute value.* There will be negative values in the data, but we can correct this by normalize it to the range of $[0, 1]$ using one of the techniques from machine learning (13). The results is shown in 1b and 2b. Because of the presence of negative values, zeros are normalized to about 0.5 in the resulted image, which is why the background looks gray (gray color has a grayscale value of about 0.5). This normalization preserves the contrast for all values (and is also how `imshow` in MATLAB display data that is not in range of $[0, 1]$ when explicitly asked to (see appendix)). However, this normalization is potentially sensitive to outliers. If there exist an extreme value in the foreground, we will lost all contrast because the range of data inflated by the outlier is projected to



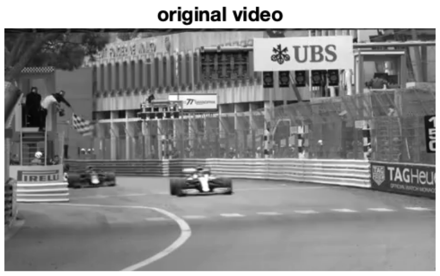
(a) Foreground of the first frame using *absolute-value-of-difference* approach



(b) Foreground of the first frame using *difference-of-absolute-value* approach



(c) Background image



(d) First frame of original video

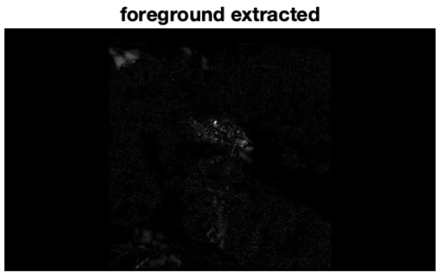
Figure 1: Analysis of Monte Carlo Video

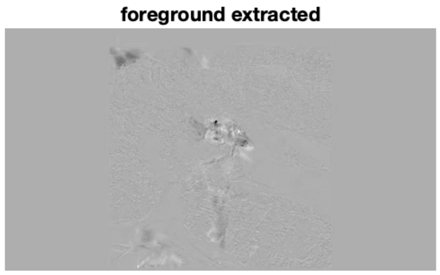
$[0, 1]$. We did not run into this issue when using the 2 videos mentioned above, but it's a potential problem and needs further research to resolve.

We also tried the approach given by the assignment of subtracting all foreground values that are negative (referred to $R$) and add them to background. The problem of this approach is that $R$ is not stationary with respect to time, which contaminates the background. Since $R$ is suppose to be part of the foreground, we lost foreground information when subtracting it. Overall, this approach performs worse than the above mentioned approaches in all major aspects (and thus not shown here).

## 5 Summary and Conclusions

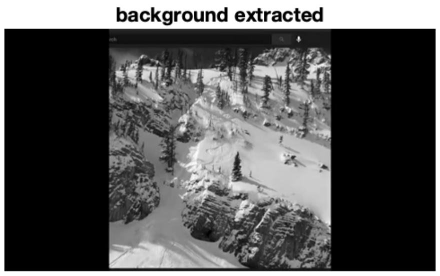
In this paper, we demonstrated the possibility of separating background and foreground from a video using Dynamic Mode Decomposition (DMD). From the sample video, DMD works great at distinguishing stationary background and moving object (foreground). Since DMD gives us an explicit function that approximates the data, we can evaluate it any reasonable $t$ to get extra information. This allows DMD to be potentially useful for interpolation problem as well (for example, interpolate frames in a video). Of course, DMD is not without drawbacks. Linear approximation that DMD relies on only works in short duration for nonlinear systems. This means the quality of reconstruction will drop if we are processing a longer and more complex video. The computation of SVD and eigenvalue decomposition is expensive in term of both time and memory. There are also several engineering problems related to dealing with negative and complex values and normalization needed to be solved to make this approach more robust. Overall, DMD is an excellent data-driven method for analyzing and dissecting complex dynamic systems, and analyzing videos is one of its many potential applications.

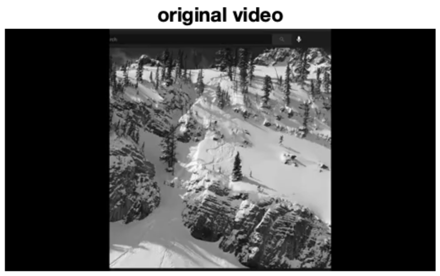




(a) Foreground of the first frame using *absolute-value-of-difference* approach

(b) Foreground of the first frame using *difference-of-absolute-value* approach





(c) Background image

(d) First frame of original video

Figure 2: Analysis of Ski Drop Video

## Appendix A  MATLAB Functions

- `[U,S,V] = svd(A, 'econ')`: perform reduced SVD on the given matrix $A$, return `U`, `S`, `V` of the decomposed matrix such that $A = USV^T$.
- `[V,D] = eig(A, B)`: perform generalized eigenvalue decomposition on the given matrices $A, B$, return `V`, `D`, such that $AV = BVD$.
- `imshow(A, [])`: display matrix $A$ (2 dimensional array) in grayscale. The value in the matrix is converted to grayscale value of $[0, 1]$ normalized based on the normalization (13).
- `rgb2gray(A)`: convert an image $A$ with dimension width $\times$ height $\times\, 3$ to a grayscale image of dimension width $\times$ height, computing each pixel using a predefined linear transform that maps RGB values to grayscale value.

## Appendix B MATLAB Code

```
% Warning: this program consumes a large amount (> 10GB) of memory
%% setup
clear; close all; clc;

%% read video to matrix
%v = VideoReader('data.nosync/ski_drop_low_low.mp4');
v = VideoReader('data.nosync/monte_carlo_low_low.mp4');
n_frame = round(v.Duration * v.FrameRate);
data = zeros(v.Width * v.Height, n_frame);
for i=1:n_frame
    tmp_ = readFrame(v);
    tmp_ = double(rgb2gray(tmp_)) / 255;
    data(:, i) = tmp_(:);
end

%% preparing data
dt = 1 / v.FrameRate;
t = linspace(0, v.Duration, n_frame);
X1 = data(:,1:end-1);
X2 = data(:,2:end);

%% performing SVD
[U, Sigma, V] = svd(X1,'econ');
S = U'*X2*V*diag(1./diag(Sigma));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV;

%% create DMD solution (for all)
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions

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

%% reconstruct background
% fixed cutoff value (depends on both length and resolution of video)
%idx_bg = abs(omega) < 10;
% first 3 smallest values (depends only on length of video)
tmp_ = sort(abs(omega));
idx_bg = abs(omega) <= tmp_(3);

y0_bg = Phi(:, idx_bg)\X1(:,1); % pseudoinverse to get initial conditions

u_modes_bg = zeros(length(y0_bg),length(t));
for iter = 1:length(t)
   u_modes_bg(:,iter) = y0_bg.*exp(omega(idx_bg)*t(iter));
end
u_dmd_bg = Phi(:, idx_bg)*u_modes_bg;
```

```
%% get real values for foreground and background
% below are different approaches, uncomment them to sum
% make both foreground (X_sparse) and background (X_low_rank) nonnegative
% approach 1: absolute value of difference
u_dmd_fg = abs(u_dmd - u_dmd_bg);
%u_dmd_fg = abs(data - u_dmd_bg); % using original video also works
u_dmd_bg = abs(u_dmd_bg);

% approach 2: difference of absolute value (with normalization)
% u_dmd_bg = abs(u_dmd_bg);
% u_dmd_fg = abs(u_dmd) - u_dmd_bg;
%u_dmd_fg = data - u_dmd_bg; % using original video also works

% linear standardization (this preserve the scale of differences)
% (x - min(x)) / (max(x) - min(x)) for each feature
% imshow automatically does this if using imshow(img, [])
u_dmd_fg = (u_dmd_fg - min(u_dmd_fg, [], 1)) ./ ...
    (max(u_dmd_fg, [], 1) - min(u_dmd_fg, [], 1));

% approach 3: residual (I don't think this works)
% u_dmd_fg = abs(u_dmd) - abs(u_dmd_bg);
% r = (u_dmd_fg < 0).*u_dmd_fg;
% u_dmd_fg(u_dmd_fg < 0) = 0;
% u_dmd_bg = abs(u_dmd_bg) + r;

%% visualize background
figure
imshow(reshape(u_dmd_bg(:, 1), v.Height, v.Width), [])
title('background extracted');

% play video
%implay(reshape(u_dmd_bg, v.Height, v.Width, n_frame))

%% visualize foreground
figure
imshow(reshape(u_dmd_fg(:, 1), v.Height, v.Width), [])
% disable adaptative
%imshow(reshape(u_dmd_fg(:, 1), v.Height, v.Width))
title('foreground extracted');

% play video
%implay(reshape(u_dmd_fg, v.Height, v.Width, n_frame))

%% visualize original frame
figure
imshow(reshape(data(:, 1), v.Height, v.Width), [])
title('original video');

% play video
%implay(reshape(data, v.Height, v.Width, n_frame))
```

Listing 1: Algorithm for separating background and foreground from video clips