AMATH 482 Homework 5

Maxime Didascalou

February 17, 2021

Abstract

This project walks through the steps for analysing videos and separating the foreground (objects that are moving) and background (what stays still throughout) using Dynamic Mode Decomposition. The process gives a foreground video where only the moving parts are present and are represented in white, and the background video is essentially a picture where the moving objects have disappeared.

1 Introduction and Overview

Sometimes it is very difficult for the naked eye to notice small changes in the pixels of a video. This paper will describe the process of trying to isolate those changes so that they are easier to catch. We are going to analyse two videos, one of car race from Monte Carlo, and another where a skier is jumping down a mountain. The goals are to be able to identify components of the video that stay still throughout in order to construct two separate videos: one where we see only what wasn't moving in the original video, and one where we only see the moving parts.
We will first go through some theoretical background that is necessary to understand the steps that were taken, then we will go through my implementation of the several algorithms that were used, and finally we will discuss the results.

2 Theoretical Background

2.1 SVD decomposition

The SVD makes it possible to write any matrix $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{1}$$

This comes from the fact that:

$$\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{\Sigma} \tag{2}$$

Where $\mathbf{U}$ is a matrix containing all the principle semiaxes (scaled to have a length of 1) that would appear if we applied $\mathbf{A}$ to the unit circle, $\mathbf{\Sigma}$ contains all the lengths of these semiaxes (from biggest to smallest), and $\mathbf{V}$ contains the vectors (from the unit circle) that give these principle semiaxes when multiplied by $\mathbf{A}$. This makes it so that $\mathbf{V}$ and $\mathbf{U}$ are both orthonormal (so their inverse equals their transpose) and $\mathbf{\Sigma}$ is a diagonal matrix where each of the scaling factors are arranged in decreasing order (they are called the singular values).
The SVD is great in practice because it tells us what components of $\mathbf{A}$ are most important: big singular values mean that there is a lot of variance in their respective principal component (columns of $\mathbf{U}$) so if only the first few singular values are "significant" you can truncate the matrices to keep only the most important aspects. [1]

2.2 Dynamic Mode Decomposition

Say you have an n by m matrix $\mathbf{X}$, where each column represents the state of something at time j (so j can go all the way up to m), where each time interval dt is the same. Now construct the matrices $\mathbf{X}_1^{m-1}$ (everything except the last column) and $\mathbf{X}_2^m$ (everything except the first column). We now want to construct a matrix $\mathbf{A}$ such that

$$\mathbf{X}_2^m = \mathbf{A}\mathbf{X}_1^{m-1}. \tag{3}$$

so that we can predict what will happen in the future. We can take the SVD of $\mathbf{X}_1^{m-1}$, and after some rearranging we get

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}_2^m\mathbf{V}\mathbf{\Sigma}^{-1} = \tilde{\mathbf{S}} \tag{4}$$

So $\tilde{\mathbf{S}}$ and $\mathbf{A}$ are similar. This is important because our goal is not to compute $\mathbf{A}$ because some data sets are enormous and so it would be a waste of time and memory, but to use components of $\mathbf{A}$ to understand the behavior of $\mathbf{X}$. We know that similar matrices share the same eigenvalues, and if $\mathbf{y}$ is an eigenvector of $\tilde{\mathbf{S}}$, $\mathbf{U}\mathbf{y}$ is an eigenvector of $\mathbf{A}$. So the eigenvectors of $\mathbf{A}$ (DMD modes) are

$$\varphi_k = \mathbf{U}\mathbf{y}_k, \tag{5}$$

where $\mathbf{y}_k$ and μ_k are the eigenvectors and eigenvalues of $\tilde{\mathbf{S}}$ respectively. Now we can switch to continuous time and we get

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k\varphi_k e^{\omega_k t} = \Phi\text{diag}(e^{\omega_k t})\mathbf{b}, \tag{6}$$

where K is the rank of $\mathbf{X}_1^{m-1}$. Notice that you can reduce the rank after doing the SVD by only keeping the most important components to reduce computation time. b_k is the initial amplitude of mode k (can be found from initial condition $\mathbf{x}_1 = \Phi\mathbf{b}$), and $\omega_k = \ln(\mu_k)/dt$. [1]

2.3 Background/Foreground Separation

Say you have a matrix $\mathbf{X}$ where each column is a frame of a video rearranged into a vector, and $\mathbf{t}$ contains the time for each frame. Assume that $||\omega_p||$ is close to 0 and that all the other $||\omega||$s aren't. Then

$$\mathbf{X}_{DMD} = b_p\varphi_p e^{\omega_p \mathbf{t}} + \sum_{k\neq p}^{K} b_k\varphi_k e^{\omega_k \mathbf{t}}, \tag{7}$$

where the first part is the background video (lets call it $\mathbf{X}_{\text{background}}$), and the second part is the foreground video (lets call it $\mathbf{X}_{\text{foreground}}$). However, these individual parts don't add up to real valued matrices. To bypass that we first compute $\mathbf{X}_{\text{background}}$, and then we find

$$\mathbf{X}_{\text{foreground}} = \mathbf{X} - |\mathbf{X}_{\text{background}}|. \tag{8}$$

This could result in negative values which is not good in our case since pixels can't have negative values, so we construct a matrix $\mathbf{R}$ that consists of these negative values and we do:

$$\mathbf{X}_{\text{background}} \leftarrow \mathbf{R} + |\mathbf{X}_{\text{background}}| \tag{9}$$

$$\mathbf{X}_{\text{foreground}} \leftarrow \mathbf{X}_{\text{foreground}} - \mathbf{R} \tag{10}$$

3 Algorithm Implementation and Development

These are the steps I took to separate the foreground and background from each video:

1. Load data/construct $\mathbf{X}$.

2. Implement DMD algorithm.

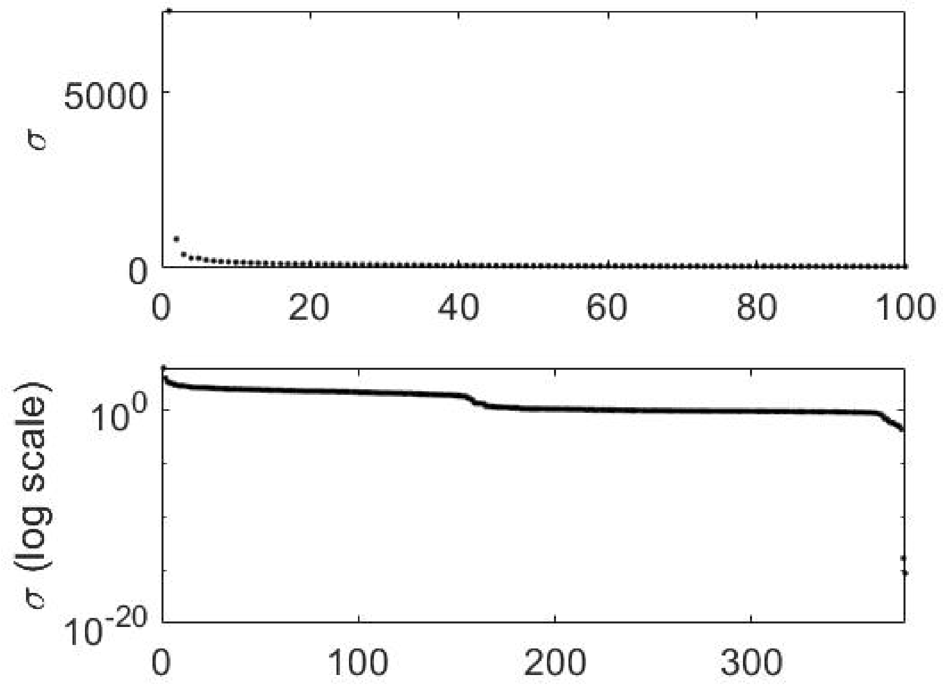

Figure 1: Singular values of **X1**

3. Separate foreground and background.

Here I will go through the steps I took with the Monte Carlo video.

Algorithm 1: Loading data/constructing **X**

```
Import movie
$m$ = number of frames
$dt = 1/$ framerate
for $k = 1 : m$ do
    reshape frame $k$ into column vector
end for
put all the frames in one matrix $\mathbf{X}$
$\mathbf{X1}$ = first $m-1$ columns of $\mathbf{X}$
$\mathbf{X2}$ = last $m-1$ columns of $\mathbf{X}$
Compute SVD of $\mathbf{X1}$
```

We can now look at the singular values to see if we can reduce the rank. As we can see in Figure 1, most of the information is contained in the first few components since the first singular value is much bigger than the rest. So we can get away with a big rank reduction. I used rank = 25 for the remainder of the process.

Algorithm 2: Loading data/constructing **X**

```
Keep first 25 columns of $\mathbf{U}$ and $\mathbf{V}$
Keep first 25 rows and columns of $\mathbf{\Sigma}$
$\mathbf{S} = \mathbf{U}^T\mathbf{X2V\Sigma}^{-1}$
$\mathbf{eV}$ = eigenvectors of $\mathbf{S}$
$\mu$ = eigenvalues of $\mathbf{S}$
$\omega = \ln(\mu)$*framerate
$\Phi = \mathbf{UeV}$
```

As we can see in Figure 2, there are two eigenvalues that are truly close to 0, so we can start the

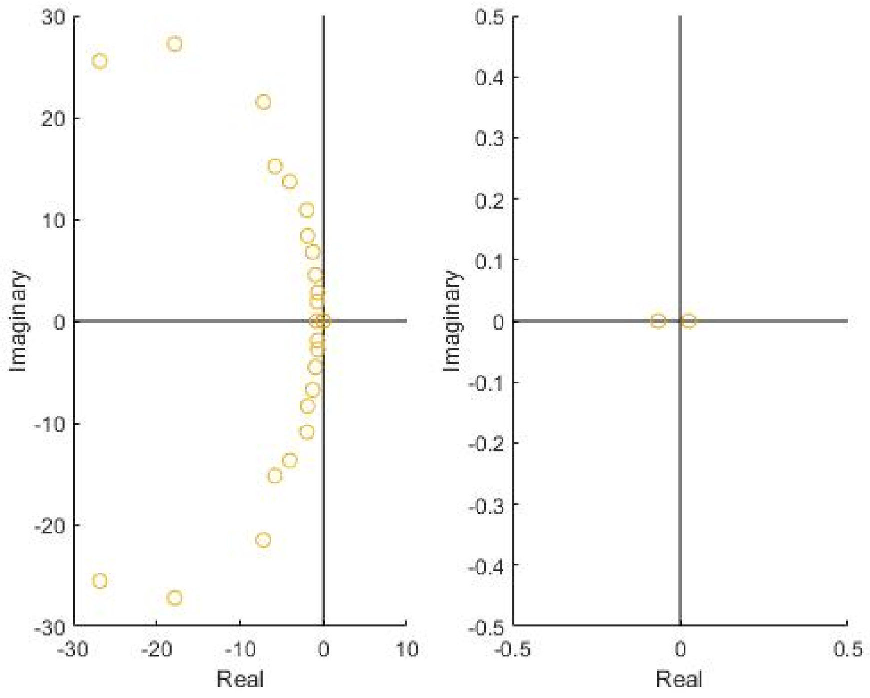

Figure 2: Eigenvalues of **S** (and **A**). Right: zoomed in on the origin.

foreground/background separation as described in theoretical background:

Algorithm 3: Loading data/constructing **X**

```
$thresh$ = threshold based on plot of $|\omega|$s
$\omega_{bg}$ = vector of all $\omega_j$ for $|\omega_j|$ less than $thresh$
$\Phi_{bg}$ = columns of $\Phi$ corresponding to $\omega_{bg}$
solve $\Phi_{bg}\mathbf{b} = \mathbf{x1}$ for $\mathbf{b}$
construct $modes$ using $\mathbf{b}$ and $\omega_{bg}$ (more details in Appendix B Listing 3)
$\mathbf{X}_{bg} = \Phi_{bg}$modes
$\mathbf{X}_{fg} = \mathbf{X1} - |\mathbf{X}_{bg}|$
R = $\mathbf{X}_{fg} - |\mathbf{X}_{fg}|$
R = R/2
$\mathbf{X}_{bg}$ = R $+|\mathbf{X}_{bg}|$
$\mathbf{X}_{fg} = \mathbf{X}_{fg}-$ R
```

We can now put each frame back into matrix form for both $\mathbf{X}_{fg}$ and $\mathbf{X}_{bg}$ (similar to Algorithm 1) and view the foreground and background videos.

4 Computational Results

We can see in Figure 3 that the background stays the same in all the frames, and the white parts (with some error) in the foreground videos represent the parts of the video where there is some movement (e.g flag, cars, crowd). We see the same thing in Figure 4 where the snow that is flying in the air is depicted as white and everything else is black.

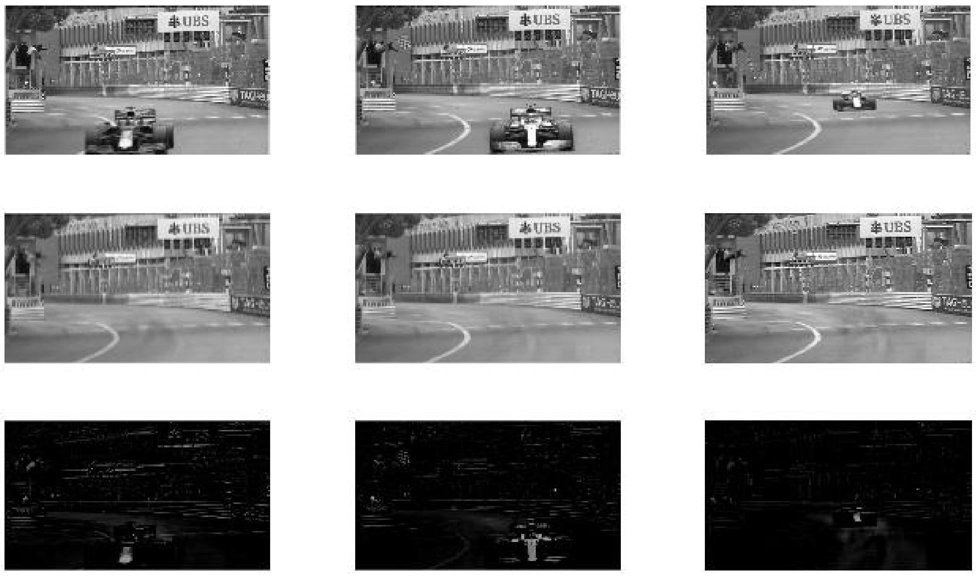

Figure 3: Here is a comparison of the original Monte Carlo video (row 1) with the DMD background (row 2) and foreground (row 3) videos for the frames 100 (col 1), 250 (col 2), and 370 (col 3). Note: Brightness for foreground pictures was artificially increased.

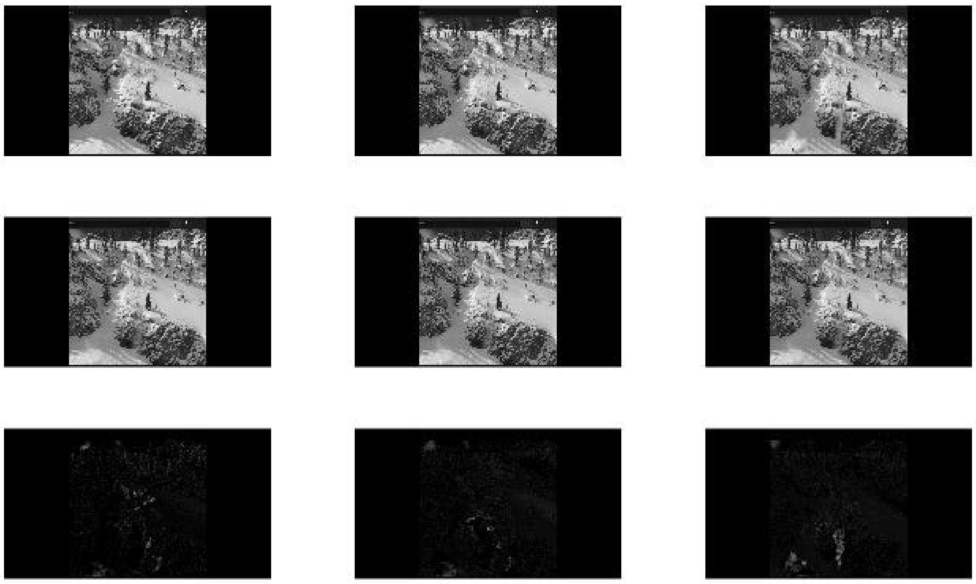

Figure 4: Here is a comparison of the original ski jump video (row 1) with the DMD background (row 2) and foreground (row 3) videos for the frames 120 (col 1), 290 (col 2), and 420 (col 3). Note: Brightness for foreground pictures was artificially increased.

5 Summary and Conclusions

In conclusion we can see that dynamic mode decomposition is effective in separating the foreground and background from videos as we can see from figures 3 and 4. It is also an effective way of doing it since it is possible to work in lower rank approximations which drastically reduces computation time and memory usage.

References

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

Appendix A MATLAB Functions

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix `A`, such that `A` = `U*S*V'`.
- `B = reshape(A,sz1,...,szN)` reshapes `A` into a `sz1`-by-...-by-`szN` array where `sz1`,...,`szN` indicates the size of each dimension.
- `[V,D] = eig(A,B)` returns diagonal matrix `D` of generalized eigenvalues and full matrix `V` whose columns are the corresponding right eigenvectors, so that `A*V` = `B*V*D`.
- `v = VideoReader(filename)` creates object `v` to read video data from the file named `filename`.
- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array `X`.
- `Y = uint8(X)` converts the values in `X` to type `uint8`. Values outside the range `[0,28-1]` map to the nearest endpoint
- `I2 = im2double(I)` converts the image `I` to double precision. `I` can be a grayscale intensity image, a truecolor image, or a binary image. `im2double` rescales the output from integer data types to the range `[0, 1]`.
- `imshow(I)` displays the grayscale image `I` in a figure. `imshow` uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.

Appendix B MATLAB Code

```
clear all; close all; clc;

%%
ski = VideoReader('monte_carlo_low.mp4');
%ski = VideoReader('ski_drop_low.mp4');
video = read(ski);
[n m uint frames] = size(video);
dt = 1/ski.Framerate;
t = 0:dt:ski.Duration;

for j = 1:frames
    Reshaped = reshape(im2double(rgb2gray(video(:,:,:,j))), n*m, 1);
    ski_matrix(:,j) = ski_reshape;
end
X1 = ski_matrix(:,1:end-1);
X2 = ski_matrix(:,2:end);

%% SVD of X1 and Computation of ~S
[U, Sigma, V] = svd(X1,'econ');
```

Listing 1: Algorithm 1

```
%%
subplot(2,1,1)
plot(diag(Sigma),'k.','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 100])
ylabel("\sigma")
subplot(2,1,2)
semilogy(diag(Sigma),'k.','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 frames-1])
ylabel("\sigma (log scale)")
%%
r = 25;
U = U(:,1:r);
Sigma = Sigma(1:r,1:r);
V = V(:,1:r);

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;
```

Listing 2: Algorithm 2

```
%%
subplot(1,2,1)
hold on
plot([-30 10], [0 0], 'k')
plot([0 0],[-30 30],'k')
plot(real(omega), imag(omega),'o')
ylabel("Imaginary")
xlabel("Real")
subplot(1,2,2)
hold on
plot([-30 0.5], [0 0], 'k')
plot([0 0],[-30 30],'k')
plot(real(omega), imag(omega),'o')
ylabel("Imaginary")
xlabel("Real")
xlim([-0.5 0.5])
ylim([-0.5 0.5])

%%

thresh = 0.1;
bg = find(abs(omega) < thresh);
omega_bg = omega(bg);
phi_bg = Phi(:,bg);

b = phi_bg\X1(:,1);
t = t(1:end-1);

u_modes = zeros([length(omega_bg) length(t)]);

for j = 1:length(t)
    u_modes(:,j) = b.*exp(omega_bg*t(j));
end

X_bg = phi_bg*u_modes;
```

Listing 3: Algorithm 3: part 1

```
%%
for j = 1:length(t)
    X_bg_vid(:,:,:,j) = uint8(reshape(X_bg(:,j),[],960)*255);
end

for j=1:length(t)
  frame=X_bg_vid(:,:,:,j);
  imshow(frame); drawnow
end

%% Subtract background to get foreground

X_fg = X1 - abs(X_bg);
R = X_fg - abs(X_fg);
R = R./2;
X_bg = R + abs(X_bg);
X_fg = X_fg - R;

%%

for j = 1:length(t)
    X_fg_vid(:,:,:,j) = uint8(reshape(X_fg(:,j),[],960)*255);
end

%%
for j=1:length(t)
  frame=X_fg_vid(:,:,:,j);
  imshow(frame*1.5); drawnow
end
```

Listing 4: Algorithm 3: part 2, watching foreground and background videos

```
%%
subplot(3,3,1)
imshow(rgb2gray(video(:,:,:,100)))
subplot(3,3,2)
imshow(rgb2gray(video(:,:,:,250)))
subplot(3,3,3)
imshow(rgb2gray(video(:,:,:,370)))
subplot(3,3,4)
imshow(X_bg_vid(:,:,:,100))
subplot(3,3,5)
imshow(X_bg_vid(:,:,:,250))
subplot(3,3,6)
imshow(X_bg_vid(:,:,:,370))
subplot(3,3,7)
imshow(X_fg_vid(:,:,:,100)*1.3)
subplot(3,3,8)
imshow(X_fg_vid(:,:,:,250)*1.3)
subplot(3,3,9)
imshow(X_fg_vid(:,:,:,370)*1.3)
```

Listing 5: Comparing Original, foreground, and background