AMATH 482 Homework 5

Jeremy Lu

March 17, 2021

Abstract

Using the techniques of Dynamic Mode Decomposition (DMD), we can perform image and video processing in the form of separating the foreground and the background of videos. Given two videos, one of cars driving by during a race, and one of a skier going down a steep slope, we apply the DMD and successfully separate the video into a background and foreground video.

1 Introduction and Overview

Dynamic Mode Decomposition is a form of dimensionality reduction that does not rely on a data model. This means that we can take advantage of low-dimensionality in experimental data without having to rely on a given set of governing equations.
Given two 960×540 videos, we can convert them into $518400 \times f$ matrices, where f is the number of frames present. Performing DMD on the video data, we can separate the resulting matrice into a low-rank DMD along with the summation of sparse-DMD's, which represents the background and foreground respectively. The resulting images and videos reflect the effectiveness of one of DMD's many applications.

2 Theoretical Background

2.1 Dynamic Mode Decomposition

Dynamic Video Decomposition operates on snapshots of spatio-temporal data, which is perfect for videos, as they are just frames put together in time series. Given N spatial points saved per unit and M snapshots, The DMD method works by approximating the modes of the Koopman Operator. The Koopman Operator is a linear, time-independent operator $\mathbf{A}$ such that:

$$x_{j+1} = \mathbf{A}x_j,$$

where j indicates a specific point in the time data, and $\mathbf{A}$ maps our data from time j to $j+1$. Essentially, multiplying by our Koopman operator $\mathbf{A}$ will advance our data to the next time point. This operator is not only restricted to the local time, and can be used to extrapolate as well.
To construct a Koopman Operator for the data, consider the matrix:

$$\mathbf{X}_1^{M-1} = [x_1 \;\; x_2 \;\; \dots \;\; x_{M-1}],$$

where x_j denotes the data snapshot at time j. Writing this in terms of the Koopman Operator gives:

$$\mathbf{X}_1^{M-1} = [x_1 \;\; \mathbf{A}x_1 \;\; \dots \;\; \mathbf{A}^{M-2}x_1].$$

This can be further rewritten as:

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T,$$

where e_{M-1}^T is a vector of all zeros except for a 1 at the $M-1$ component. This equation means that it is possible to represent the next data snapshot as a sum of the product of the Koopman Operator and the previous data snapshot and a residual.

Rather than compute $\mathbf{A}$ directly, it is computationally simpler to find a similar matrix, which will share the same eigenvalues as $\mathbf{A}$. First, applying the Singular Value Decomposition (SVD) to $\mathbf{X}_1^{M-1}$ gives $\mathbf{U}\Sigma\mathbf{V}^*$, resulting in:

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{U}\Sigma\mathbf{V}^* + \mathbf{r}e_{M-1}^T.$$

$\mathbf{A}$ is chosen in a way such that the columns in $\mathbf{X}_2^M$ can be written as linear combinations of the columns of $\mathbf{U}$ (the POD modes). Since $\mathbf{r}$, the residual vector, is orthogonal to the POD basis, this gives $\mathbf{U}^*\mathbf{r}$=0. As a result, this gives:

$$\mathbf{U}^*\mathbf{X}_2^M = \mathbf{U}^*\mathbf{A}\mathbf{U}\Sigma\mathbf{V}^*.$$

To isolate $\mathbf{U}^*\mathbf{A}\mathbf{U}$, this equation is multiplied by $\mathbf{V}$ and Σ^{-1} resulting in:

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\Sigma^{-1}.$$

This right side, which is know entirely from input data, can be rewritten as $\tilde{\mathbf{S}}$ and is similar to $\mathbf{A}$. Thus if $\mathbf{S}$ has eigenvector y, then $\mathbf{A}$ will have eigenvector $\mathbf{U}y$. The eigenvector/eigenvalue pairs of $\tilde{\mathbf{S}}$ satisfy

$$\tilde{\mathbf{S}}y_k = \mu_k y_k,$$

and therefore the eigenvectors of $\mathbf{A}$, called the DMD modes, are defined as:

$$\psi_k = \mathbf{U}y_k.$$

With this information, the data $\mathbf{X}$ can have each of its time snapshots written as:

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k\psi_k e^{k}t = \Psi\mathrm{diag}(e^{\omega_k t})\mathbf{b},$$

where K is the rank of $\mathbf{X}_1^{M-1}$, b_k is the initial amplitude of each mode, Ψ contains the eigenvectors ψ_k as its columns, and $\omega_k = \ln(\mu_k)/\delta t$, allowing us to have a continuous equation. This continuous equation for x allows for data snapshots that are interpolations between collected time steps, as well as extrapolations for predicting outside the bounded times. To calculate b_k, simply note that at $t = 0$, we have:

$$x_1 = \Psi\mathbf{b} \implies b = \Psi^{\dagger}x_1,$$

where $\Psi^{\dagger}$ is the pseudoinverse of the matrix Ψ.

2.2 Video Decomposition

The DMD spectrum of frequencies can be used to subtract background modes. For ω_p, where $p \in 1, 2, \ldots, l$, and $||\omega_p|| \approx 0$, $\mathbf{X}_{DMD}$ can be written as:

$$\mathbf{X}_{DMD} = b_p\varphi_p e^{\omega_p t} + \sum_{j\neq p} b_j\phi_j e^{\omega_j t} \tag{1}$$

The first term in this equation represents the background, while the summation on the right is the foreground video. They are also referred to as the Low-Rank DMD of $\mathbf{X}$ and the Sparse DMD of $\mathbf{X}$, respectively.
Since the terms of the low-rank of DMD are complex, we can acquire real-valued elements of the sparse DMD by using:

$$\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparce}} = \mathbf{X} - \left|\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Low\text{-}Rank}}\right|.$$

This might result in negative values for $\mathbf{X}_{\mathrm{DMD}}^{\mathrm{Sparce}}$, although this can be remedied through MATLAB's `mat2gray` function, which is described in Appendix A.

3 Algorithm Implementation and Development

We begin the process of extracting foreground and background information by first reading in each frame of the videos and storing them in the matrix `X`. Both videos are 960×540, with 369 and 453 frames each, so `X` has shape $518400 \times f$, where f is the appropriate number of frames, depending on video. Additionally, we have t, which represents the time stamps of each frame of the video, as well as δt, representing the time elapsed between each frame.

3.1 DMD Algorithm

After we have our data, we can find the DMD expression for `X`.

Algorithm 1: Dynamic Mode Decomposition of `X`

X1 = `X`(:, 1:$f-1$)
X2 = `X`(:, 2:f)
Perform SVD to get X1 = UΣV*
Compute $\tilde{S}$ with U*X2VΣ^{-1}
Extract the eigenvectors y and eigenvalues μ from $\tilde{S}$
Find ω using $\dfrac{\ln(\mu)}{\delta t}$
Eigenvectors of Koopman Factor is Ψ = U y_k
Find initial conditions with $b = \Psi^{\dagger} x_1$
Calculate modes of U with: $x_1 e^{\omega t}$ where t are the recorded time stamps
DMD of `X` = $\Psi\cdot$ (modes of U)

3.2 Video Decomposition

Now with the DMD of `X` calculated, we find the value of p such that $||\omega_p||$ is closest to 0. The column of the DMD of `X` which corresponds to ω_p become sthe background, and we can find the Sparse DMD of `X` using `X` - abs(background vector). Finally, we can store each column of the Sparse DMD as a frame of the foreground video. Note that for each column, we wrap it in the `mat2gray` command, which will remove negative pixel values, resulting in visible images for MATLAB.

4 Computational Results

4.1 Monte Carlo Race

In Figure 1 we can see the background image on the left, compared to the first frame of the video. Here it is evident that the race cars and checkered flag, both of which are moving throughout the video, are removed.

Figure 1: Background extracted (left) vs. First frame of the video (right)

In Figure 2, we have frames from the foreground video. The cars are very clear, as well as the flag on the left, which is waving around. There is also some traces of the background, which is due to slight camera movements. Overall, the primary moving objects in the video are captured with our technique.

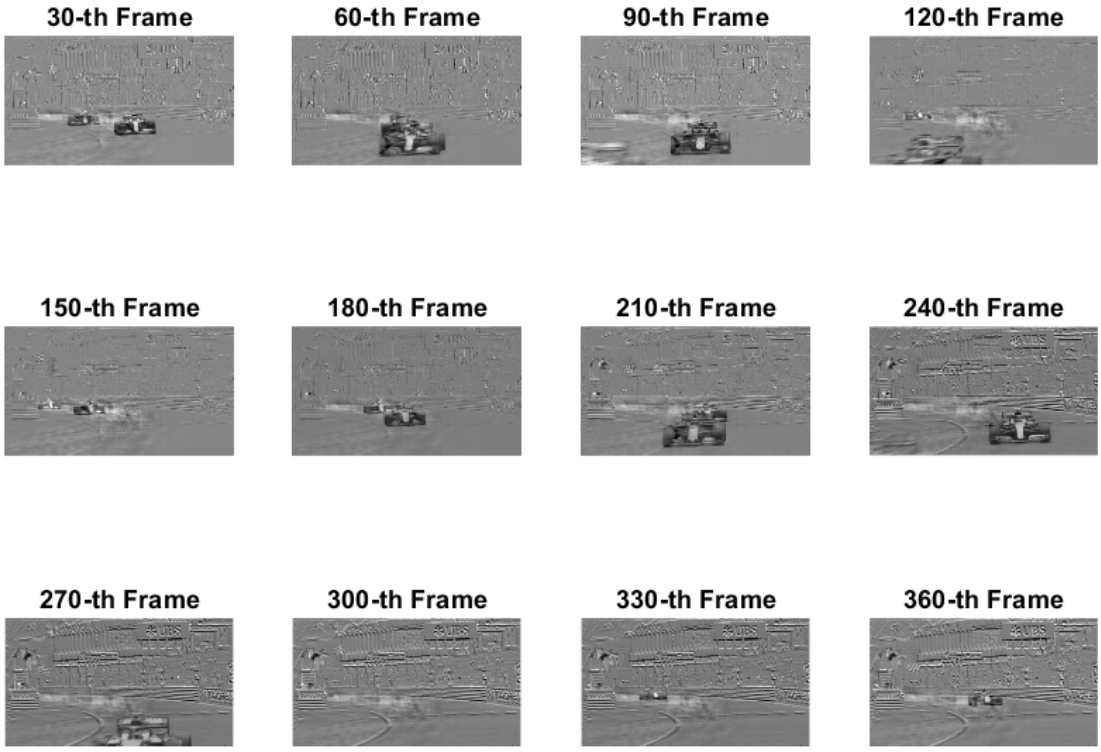

Figure 2: Frames from the foreground video

4.2 Ski Drop

In figure 3, we can see that the background is devoid of the skiier (might have to zoomin in!). Meanwhile, the original in the video has the skiier slightly above the tree in the middle. Here, DMD again performs well, in removing the skiier from the video.

Figure 3: Background extracted (left) vs. Frame of the video (right)

Figure 4 shows frames of the foreground video. Although somewhat hard to see, the skiier is moving down the hill, with other streaks of white being the snow left from his path.

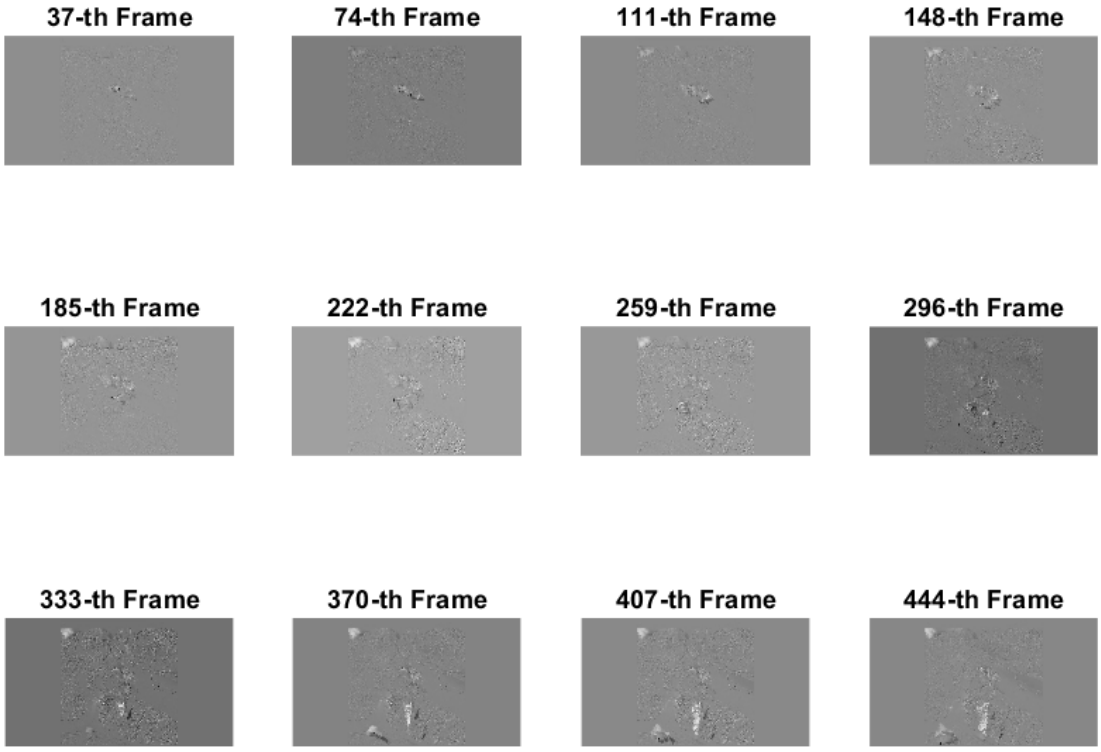

Figure 4: Frames from the foreground video

5 Summary and Conclusions

Overall, the use of DMD proves effective in separating the background and foreground from inputted videos. For the Monte Carlo video, we successfully extracted the moving flag and race cars, while in the ski drop video, the skiier and moving snow were successfully removed from the background.

Appendix A MATLAB Functions

- `abs(X)`: Returns the absolute value of a given input `X`.
- `close(X)`: Closes an instance of VideoWriter `X` when finished editting.
- `diag(X)`: Returns the diagonal entries of a given a matrix `X`.
- `eig(X)`: Returns a column vector of eigenvalues corresponding to the matrix `X`.
- `exp(X)`: Return e^X given an input `X`.
- `im2double(X)`: Returns an image converted to double precision, given an image `X`.
- `imshow(X)`: Returns the maximum value in a given vector `X`.
- `length(X)`: Returns the length of the inputted array `X`.
- `linspace(X,d,Y)`: Returns an array that ranges from `X` to `Y` with steps of length `d`.
- `log(X)`: Returns the natural logarithm given an input `X`.
- `mat2gray(X)`: Returns a grayscale image with values bounded by $(0, 1)$ that is converted via inputted matrix `X`.
- `read(X)`: Returns pixel and RGB data extracted from an inputted video file `X`.
- `reshape(X, [m n])`: Returns an `m` $\times$ `n` matrix given an inputted array `X`.
- `rgb2gray(X)`: Returns a grayscale image converted from the input image ED`X`.
- `size(X)`: Returns the dimensions of a given matrix `X`.
- `svd(X)`: Performs SVD on a given matrix `X` and returns U, Σ, and V.
- `VideoReader(X)`: Returns a video object after reading the input file `X`.
- `VideoWriter`: Begans an instance of a video writer to file location `X`.
- `writeVideo(X, I)`: Adds frame/image `I` to the inputted video file `X`.
- `zeros(m,n)`: Returns an `m` $\times$ `n` matrix of zeroes.

Appendix B MATLAB Code

```
%% Read Image
%540x960x369

v = VideoReader('monte_carlo_low.mp4');
colorFrames = read(v);
    for f = 1:369
        J = rgb2gray(colorFrames(:,:,:,f));    % CONVERT COLOR TO GRAY
        X(:,f) = J(:);                         % GRAY FRAMES
    end

%% Perform DMD

X = im2double(X);
t = linspace(0,379/60,379);
dt = t(2)-t(1);
```

```
vid_length = size(X, 2);
X1 = X(:, 1:end-1);
X2 = X(:, 2:end);

[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

y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
umodes = zeros(length(y0),vid_length);
for iter = 1:vid_length
    umodes(:,iter) = y0.*exp(omega*t(iter));
end
udmd = Phi*umodes;

%% Background image

background = reshape(udmd(:,1), [540 960]);
imshow(background);

%% sparse dmd

sparse = X-abs(udmd(:,1));

%% Create foreground video

video = VideoWriter('monte_carlo_foreground.avi'); %create the video
   object
open(video); %open the file for writing
for ii=1:369 %where N is the number of images
  foreground = reshape(sparse(:,ii), [540 960]);
  I = mat2gray(foreground); %read the next image
  writeVideo(video,I); %write the image to file
end
close(video); %close the file

%% Snapshots of frame from foreground video

figure();
t = tiledlayout(3,4,'TileSpacing','Compact','Padding','Compact');
for i=1:12
    nexttile
    imshow(mat2gray(reshape(sparse(:,30*i), [540 960])));
    title([num2str(i*30),'-th Frame'])
end
saveas(gcf, 'monte_carlo_frames.png')

%% Read Image
%540x960x454
```

```
v = VideoReader('ski_drop_low.mp4');
colorFrames = read(v);
for f = 1:454
        J = rgb2gray(colorFrames(:,:,:,f));   % CONVERT COLOR TO GRAY
        X(:,f) = J(:);                        % GRAY FRAMES
    end
%% Perform DMD

X = im2double(X);
t = linspace(0,454/60,454);
dt = t(2)-t(1);
vid_length = size(X, 2);
X1 = X(:, 1:end-1);
X2 = X(:, 2:end);

[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

y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
umodes = zeros(length(y0),vid_length);
for iter = 1:vid_length
    umodes(:,iter) = y0.*exp(omega*t(iter));
end
udmd = Phi*umodes;

%% Show background

background = mat2gray(abs(reshape(udmd(:,1), [540 960])));
imshow(background);

%% lowrank dmd
sparse = X-abs(udmd(:,1));

%% Creating foreground video

video = VideoWriter('ski_drop_foreground.avi'); %create the video object
open(video); %open the file for writing
for ii=1:453 %where N is the number of images
  foreground = reshape(sparse(:,ii), [540 960]);
  I = mat2gray(foreground); %read the next image
  writeVideo(video,I); %write the image to file
end
close(video); %close the file

%% Snapshots of frames from foreground video
figure();
t = tiledlayout(3,4,'TileSpacing','Compact','Padding','Compact');
```

```
for i=1:12
    nexttile
    imshow(mat2gray(reshape(sparse(:,37*i), [540 960])));
    title([num2str(i*37),'-th Frame'])
end
saveas(gcf, 'ski_drop_frames.png')
```