AMATH 482: Homework 5

Alex Omusoru

Due: March 17, 2021

Abstract

Using Dynamic Mode Decomposition (DMD) on two video clips allowed the seperation of the video clips into foreground and background objects. This was completed through the seperation of the DMD matrix into the low-rank DMD matrix, and the remaining sparse DMD matrix.

1 Introduction and Overview

The intent of this report was to seperate two video clips, one from a car race in monte carlo, and the other of a skier, into the respective background and foregrounds. This was completed using first a Dynamic Mode Decomposition, and then seperating the DMD modes based on the eigenvalues that were approximately 0 and those that weren't. These two groups became the foreground and the background videos.

2 Theoretical Background

2.1 Dynamic Mode Decomposition

Dynamic Mode Decomposition involves the calculation of the Koopman operator for the video, in an attempt to represent a linear mapping from one point in time, to a second forward point in time after a shift of δt. The Koopman operator is defined as some matrix A where:

$$x_{j+1} = Ax_j$$

Where t_j is some time, and t_{j+1} indicates the next discrete time slot.
To determine A for DMD, consider two matrices,

$$X_1^{M-1} = [x_1 x_2 ... x_{M-1}]$$

and

$$X_2^M = [x_2 x_3 ... x_M]$$

We define the Koopman operator A to be

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

By conducting SVD on $X1$ to get U, Σ, V, we can rewrite above as

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

We then define

$$S_{yk} = \mu_k y_k$$

and

$$X_{]textDMD}(t) = \Phi diag(e^{\omega_k t})b$$

2.2 DMD Analysis for Video Seperation

We can extract the low-rank DMD matrix, in such a way that

$$X_{\text{low rank}} = \beta_p \phi_p e^{\omega_p t}$$

and then approximate

$$X_{\text{sparse}} = X - |X_{\text{low rank}}|$$

Which can then be used for the decomposition of the video.

3 Algorithm Implementation and Development

After setting up and importing the videos, each video was subjected to the same procedures. Each frame was converted to grayscale, then into a column vector, and then combined into a m x n matrix X, where each column of X is a frame of the video.
Based on the description above, the DMD components were constructed (see *Algorithm: Defining DMD Components* for general algorithmic approach). Then the low-rank and sparse DMD matrices were constructed, as described above (see *Algorithm: Constructing Low-Rank and Sparse Matrices* for general algorithmic approach)
Finally, the low-rank and sparse matrices were each converted back into videos and compared to the original.

Algorithm: Defining DMD Components

```
given n = number of frames, X = m x n matrix of frames

t = vector of frame timestamps
X1 = first n-1 columns of X
X2 = last n-1 columns of X

Complete SVD of X1
Construct S matrix

mu = eigenvalues of S
Construct omgega, Phi and beta
```

Algorithm: Constructing Low-Rank and Sparse Matrices

```
given X = video matrix
Identify c = modes with omega approximately 0

for i in c
    Xlr += beta(i) * Phi(:, i) * exp(omega(i) * t(i))
end

Xs = X - abs(Xlr)

where Xlr is low-rank matrix and Xs is sparse matrix
```

4 Computational Results

Two videos for both the race and the skier videos were extracted. A frame was compared among the three videos, for the race and the skier, respectively.
The race had some inconsistencies in the videos (see **Figure 1**), however it can be seen that the low-rank approximation emphasises the details of the cars, while the sparse matrix emphasises the background.

Figure 1: Side-by-side of Race Original Frame, Low-Rank and Sparse

Figure 2: Side-by-side of Skier Original Frame, Low-Rank and Sparse

The skier was much clearer (see **Figure 2**), and it is seen that the low-rank matrix acurately represents the background, and the sparse matrix is relatively effective in identifying the skier, by reducing the influence of the background substantially.

5 Summary and Conclusions

It is concludeded that DMD is effective at foreground and background video removal, however this report failed to ascertain that completely. While the skier was well extracted, the racecars were not extracted well, resulting in a blurring of the foreground and background videos, rather than two seperate pieces.

Appendix A MATLAB Functions

- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `B = reshape(A,sz)` reshapes A using the size vector, sz, to define size(B). For example, reshape(A,[2,3]) reshapes A into a 2-by-3 matrix. sz must contain at least 2 elements, and prod(sz) must be the same as numel(A).
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `v = VideoWriter(filename)` creates a VideoWriter object to write video data to an AVI file with Motion JPEG compression.
- `I = mat2gray(A,[amin amax])` converts the matrix A to a grayscale image I that contains values in the range 0 (black) to 1 (white). amin and amax are the values in A that correspond to 0 and 1 in I. Values less than amin are clipped to 0, and values greater than amax are clipped to 1.
- `v = VideoWriter(filename)` creates a VideoWriter object to write video data to an AVI file with Motion JPEG compression.

Appendix B MATLAB Code

Main

```
%% Reset Workspace
clc; clear; close all;

isSkiDrop = 0; % 1 for true, 0 for false

%% import and read generally

ski = VideoReader('ski_drop_low.mp4');
monte = VideoReader('monte_carlo_low.mp4');

video = ski;
skiX = zeros(video.Height*video.Width, video.NumFrames);
for i = 1:video.NumFrames
    skiX(:, i) = reshape(rgb2gray(read(video, i)), [video.Height*video.Width, 1]);
end

video = monte;
monteX = zeros(video.Height*video.Width, video.NumFrames);
for i = 1:video.NumFrames
    monteX(:, i) = reshape(rgb2gray(read(video, i)), [video.Height*video.Width, 1]);
end


%% Generate DMD Parts
video = monte;
X = monteX;

dt = 1/video.FrameRate;
t = linspace(0, video.Duration, video.NumFrames);

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;
b = Phi\X1(:, 1);

%% Low-Rank and Sparse Video Decomposition
low_ranks = (find(abs(omega) < 0.05))';

X_low = zeros(video.Height*video.Width, video.NumFrames);
for i = low_ranks
   X_low = X_low + b(i)*Phi(:, i)*exp(omega(i)*t(i));
end
```

```
X_sparse = X - abs(X_low);
%% Low Rank to Video
vidX = abs(X_low);

lowVid = VideoWriter('monteLow.avi');
lowVid.FrameRate = video.FrameRate;

open(lowVid)
for i = 1:video.numFrames
    img = mat2gray(reshape(vidX(:, i), [video.Height, video.Width]));
    writeVideo(lowVid, img);
end
close(lowVid)

%% Sparse to Video
vidX = X_sparse;

lowVid = VideoWriter('monteSparse.avi');
lowVid.FrameRate = video.FrameRate;

open(lowVid)
for i = 1:video.numFrames
    img = mat2gray(reshape(vidX(:, i), [video.Height, video.Width]));
    writeVideo(lowVid, img);
end
close(lowVid)
```