Homework 5: Background Subtraction in Video Streams

- Aditya Karan

Abstract

Dynamic mode decomposition (DMD) is a dimensionality reduction algorithm developed by Peter Schmid in 2008. DMD is a powerful new technique for discovery of dynamical systems from high-dimensional data. For a time series data, DMD computes a set of modes each of which is associated with a fixed oscillation frequency and decay/growth rate. Due to the intrinsic temporal behaviors associated with each mode, DMD differs dimensionally reduction methods such as principal component analysis, which computes orthogonal modes that lack predetermined temporal behaviors.

1. Introduction and Overview

By using a smartphone, we were able to capture different videos our purposes. The first video is a recording of a background in front of Applied Mathematics department and a car passing in front of it. The second video is a recording of my friend passing by.

The objective of this homework is to calculate the low-rank DMD of the dynamical system (captured video) so that we can use that to separate foreground and background parts of the video.

2. Theoretical Background

2.1 Dynamic Mode Decomposition (DMD)

Suppose we have a dynamical system (1.1) and two sets of data (1.2, 1.3):

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \mu) \quad (1.1)$$

$$X = \begin{bmatrix} \vdots & \vdots & \vdots \\ x_1 & x_2 & x_3 \\ \vdots & \vdots & \vdots \end{bmatrix} \quad (1.2)$$

$$X' = \begin{bmatrix} \vdots & \vdots & \vdots \\ x_1' & x_2' & x_3' \\ \vdots & \vdots & \vdots \end{bmatrix} \quad (1.3)$$

so that $\mathbf{x_k}' = \mathbf{F}(\mathbf{x_k})$ where **F** is the map in (1.2) corresponding to the evolution of (1.1) for time Δt. DMD computes the leading eigendecomposition of the best-fit linear operator **A** relating the data $\boldsymbol{X}' \approx \boldsymbol{A}\boldsymbol{X}$:

$$\boldsymbol{A} = \boldsymbol{X}'\boldsymbol{X}^{\dagger}$$

The DMD modes, also called dynamic modes are the eigenvectors of **A**, and each DMD mode corresponds to a particular eigenvalue of **A.**

3. Algorithm Implementation and Development

The DMD algorithm takes advantage of low dimensionality of data in order to make a low rank approximation of the linear mapping that best approximates the nonlinear dynamics of the data collected for the system. Once this is done, a prediction of the future state of the system is achieved for all time.

By using a smartphone, we can capture a video sequence consisting of a moving object in a static background. We read the entire video file in a matrix and then take a transpose such that columns of the matrix represent each video frame. We can then perform an SVD analysis of the matrix. By plotting the energy diagram, we can calculate the rank truncation number.

Through the rank truncation number, we can create appropriate truncated matrices and using them calculate the eigenvalues and eigenvectors of the system. After defining a time scale we can then calculate the dmd modes of the system. After reshaping each of the column vector if the dmd matrix we ca then plot them to get the desired result.

4. Computational Results

For video 1 which consists of a car passing in front of the AMATH department:

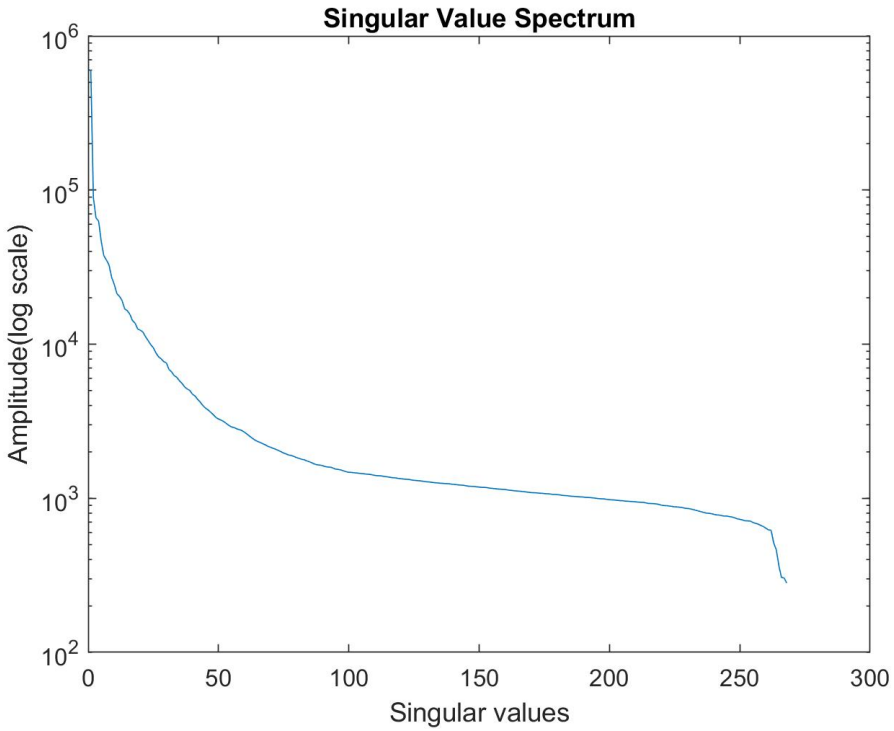

Fig1a – Singular Value Spectrum

Fig 1b – Original Frame

Fig 1c – Background Created with DMD

Fig 1d –Created with Sparse DMD

For video 2 which involves my friend passing

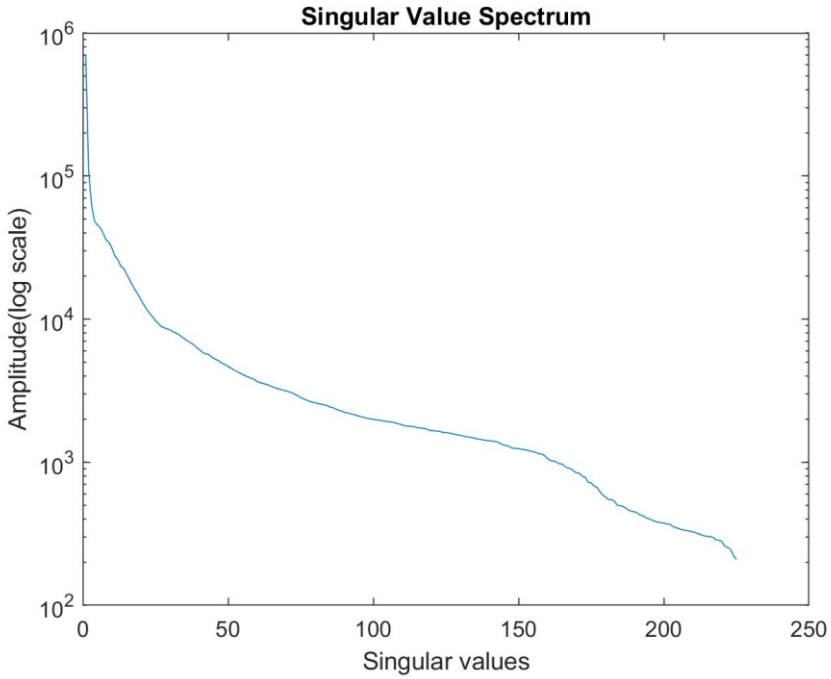

Fig2a – Singular Value Spectrum

Fig 2b – Original Frame

Fig 2c – Background Created with DMD

Fig 1d –Created with Sparse DMD

5. Summary and Conclusions

Using DMD we were able to separate the foreground and background parts of the video. Also, by testing the algorithm on various videos we can see that in cases where the foreground's intensity is less than background intensity (in terms of brightness) the background info will overshadow the foreground.

When we are subtracting our absolute value of our low rank matrix from our matrix data we might get negative values which do not make sense since pixel intensities cannot be negative. These terms can be coined as residual negative values. By adding these residual values back makes real and positive background but in no way makes the background better looking.

Appendix

1)**videoReader**

- v = videReader(file) uses a videoReader object to read files containing video data. The object contains information about the video file and enables the user to read data from the video.

2)**rgb2gray**

- I = rgb2gray(RGB) converts the tricolor image RGB to the grayscale intensity image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance

3)**svd**

- [U, S, V]= svd() performs a singular value decomposition of matrix A

4)**imshow**

- imshow(I) displays the grayscale image I in a figure. Imshow optimizes figure axes, and image object properties for image display.

Appendix B (Source Code)

The only difference would be file name which would be changed while using VideoReader.

```
clear all; clc

data_matrix = [];

v = VideoReader('IMG_0831.MOV');
vidFrames = read(v);
numFrames = get(v, 'numberOfFrames');

for i = 1:numFrames
    frame = vidFrames(:, :, :, i);
    frame = double(rgb2gray(frame));
    data_matrix(i,:) = reshape(imresize(frame, [264,
400]),[1,264*400]);
end

%%
data_matrix = data_matrix.';
X1 = data_matrix(:, 1:end - 1);
X2 = data_matrix(:, 2:end);
[U, S, V] = svd(X1, 'econ');
%%
semilogy(diag(S))
xlabel('Singular values')
ylabel('Amplitude(log scale)')
title('Singular Value Spectrum')
%%
r = 4;
U = U(:, 1:r);
S = S(1:r, 1:r);
V = V(:, 1:r);

Atlide = U'*X2*V/S;
[W, D] = eig(Atlide);
Phi = X2*V/S*W;
%%
dt = 1/3.9;
mu = diag(D);
omega = log(mu)/dt;
%%
x1 = data_matrix(:, 1);
```

```
b = Phi\x1;

mm1 = size(X1, 2);
u_modes = zeros(r, mm1);
t = (0:mm1 - 1)*dt;
for iter = 1:mm1
    u_modes(:, iter) = (b.*exp(omega*t(iter)));
end
X_dmd = Phi*u_modes;
%%
for k = 1:225
    y = reshape(X_dmd(:, k), [264 400]);
    imshow(uint8(y));
    pause(0.001);
end
%%
X_sparse = X1 - abs(X_dmd);
for m = 1:225
    y = reshape(X_sparse(:, m), [264, 400]);
    imshow(uint8(y));
    pause(0.001);
end
%%
R_Matrix = X_sparse.*(X_sparse < 0);
X_sparse_dmd = X_sparse - R_Matrix;
X_bg = R_Matrix + abs(X_dmd);
X_fg = X_sparse - R_Matrix;
for m = 1:225
    y = reshape(X_sparse_dmd(:, m), [264, 400]);
    imshow(uint8(y));
    pause(0.001);
end
```