Khrisna Kamarga 3/15/2019

# AMATH 482 Homework 5

## Section 1. Introduction and Overview

In this assignment, we separate the background and the foreground of a video clip using the concept of Dynamic Mode Decomposition (DMD). With DMD, we are able to rank reduce each frames in the video clip, which allows us to reconstruct the video using the low rank matrices. Furthermore, we can use the low DMD ranks to reconstruct only the static background of a video, where we can then obtain the foreground by subtracting the original data with the low rank DMD modes. Ultimately, this technique is able to separate the foreground and the background of types of videos recorded, such as traffic video and a dinner table activity video.

The videos used in this project is recorded by a Samsung Galaxy S8 phone, and the traffic video is downloaded from Youtube.

## Section 2. Theoretical Background

To compute the Dynamic Mode Decomposition of a video, each frame which is a picture has to be flattened into a vector. Each snapshot (m x n) is then appended to the collumns (m*n, 1) of the measurement data matrix **X** (m*n, length(t)). In order for the DMD to work, each snapshot should have a constant time step. We continue by preparing **X1** which is the first to the second to last timeframe of **X** and **X2** which is the second to last timeframe of **X**. Then, we can correlate **X1** to **X2** by using the realtionship described in Equation 1.

$$\frac{d\mathbf{x}}{dt} = \mathcal{A}\mathbf{x}$$

Equation 1. Continuous Dynamics of a Differential Equation

A_tilde is a linear operator that can be computed by multiplying **X2** with the pseudo-inverse of **X1**, which is computationally expensive. Therefore, SVD can be used to formulate the dynamics between the the **X1** and **X2**, and to guarantee that the system is described in a basis which is equipped with unique eigenvalues. The equation described before is equivalent to the differential equation dx/dt = A*x, which means that this is simply an eigenvalue problem, where the eigenvalues are simply the diagonals of the S matrix from the SVD. We can predict the equation governing the change of **X** through time described by the equation below.

$$\mathbf{x}(t) = \sum_{k=1}^{n} \phi_k \exp(\omega_k t) b_k = \mathbf{\Phi} \exp(\mathbf{\Omega} t)\mathbf{b}$$

Equation 2. Locally Linear Dynamic System Equation

Since measurement data are discrete, the discrete equation equivalent to the previous equation is described below.

$$\mathbf{x}_k = \sum_{j=1}^{r} \boldsymbol{\phi}_j \lambda_j^k b_j = \boldsymbol{\Phi}\boldsymbol{\Lambda}^k \mathbf{b}.$$

Equation 3. Discrete Linear Dynamic System Equation

In this equation, b corresponds to the coefficients of the intial condition in the eigenvector basis. Phi corresponds to the collections of the eigenvectors in its collumns, and Lambda are the eigenvectors. We can then predict the state of interest k-steps in the future using this equation.

The DMD algorithm works by providing a low-rank eigen-decomposition (Kutz's DMD Notes) of the matrix A_tilde that optimally fits the measured trajectory described in $\mathbf{x_k}$ based on the L2 norm. To put the system in the time domain, we can set omega = ln(eigenvalue_i)/dt, in which eigenvalue_i is the eigenvalue for the i-th diagonal entry.

For the purpose of this assignment, we assume that the background is relatively static, which means that the magnitude of the omega term should be close to 0. This means that after obtaining the low rank reconstruction of the data matrix **X**, we are able to obtain the static components of the video in $\mathbf{X_{LowRank}}$, which contains the background information.

To recover the foreground, we can then simply subtract the background from the foreground by performing $\mathbf{X_{sparse}} = \mathbf{X} - \mathbf{X_{LowRank}}$. This algorithm is implemented in this project to obtain both the background and the foreground.

## Section 3. Algorithm Implementation and Development

The background foreground separation algorithm is implemented in MATLAB, where we first have to load the video by placing the video in the same directory as the .m file. We can use VideoReader('name of the video'), which will assign a variable to a video object. Then, the user can specify the amount of frames to be analyzed, as long as it is within the range of the video. The time step (dt) is defined by accessing taking the reciprocal of the frame rate contained in the video object. Next, the pixel resolution of the expected frames are loaded in m and n. If the user is interested to analyze the frame in the middle of the video, then the user can specify the amount of frames to be skipped in the for loop that only proceeds to the next frame in the video object. We can then prepare the big **X** matrix whose collumns contains the flattened pixels in a frame for each time step. Each of the frames can be resized to the desired size to adjust the computation time, then it is reshaped into a vector which is appended to the collumn of **X**.

If the user wishes to preview the video clip, a for loop that draws each frame with pcolor() is implemented.

Proceeding to the DMD part, we start by preparing **X1** and **X2** matrices, which corresponds to the first until the second last data and the second until the last data matrix respectively. After preparing the time vector that has the same number of elements as the total frames and with dt as the timestep, we can use the function DMD provided by Professor Kutz to obtain the $\mathbf{X}_{\text{Low Rank}}$ which represents the static background of the video. The function takes is **X1**, **X2**, the amount of rank reduction, and dt. The rank reduction is set specific to the video. If the video is very stable with a clear background, then we can just set r = 1. However, if there are more than one background, r has to be higher than 1 to accommodate all the backgrounds.

To compute the foreground, we can simply subtract the data matrix **X** with the modulus (absolute value) of the elements of the background matrix, $\mathbf{X}_{\text{Low Rank}}$. It was suggested that the resulting foreground matrix $\mathbf{X}_{\text{sparse}}$ should be removed of it's negative value so called the residual **R**. This is done by creating a new matrix **R** and setting it to $\mathbf{X}_{\text{sparse}}$. Then, we can set the elements that are positive to zero to only obtain the residual matrix. The residual matrix then can be added to the backgronud matrix, and we can subtract **R** from the foreground matrix $\mathbf{X}_{\text{sparse}}$ to handle the negative value case, which plays a role in setting the minimum brightness in the imshow() command for each frame.

Based on experience, I commented out the part of using **R** to compute the foreground and the background since it makes the foreground frames weaker and it adds some of the foreground to the background matrix.

In order to reduce computational power, we can perform SVD to reduce the rank of each frames in the video. We can achieve that by only including the first chosen amount of terms that corresponds to the principal components, then reconstruct the frame by multiplying U*S*V'.

Lastly, in order to visualize, 3 matrices: background, foreground, and their summation is created to be visualized with imshow(). I also provided the option to draw each frames using pcolor(), which will result in more obvious hints of the background inside the moving foreground object.

## Section 4. Computational Results

Implementing DMD results in the successful attempt to separate static background from the foreground. Three videos are constructed using this algorithm. Below are the types of videos processed and a snapshot of the videos:

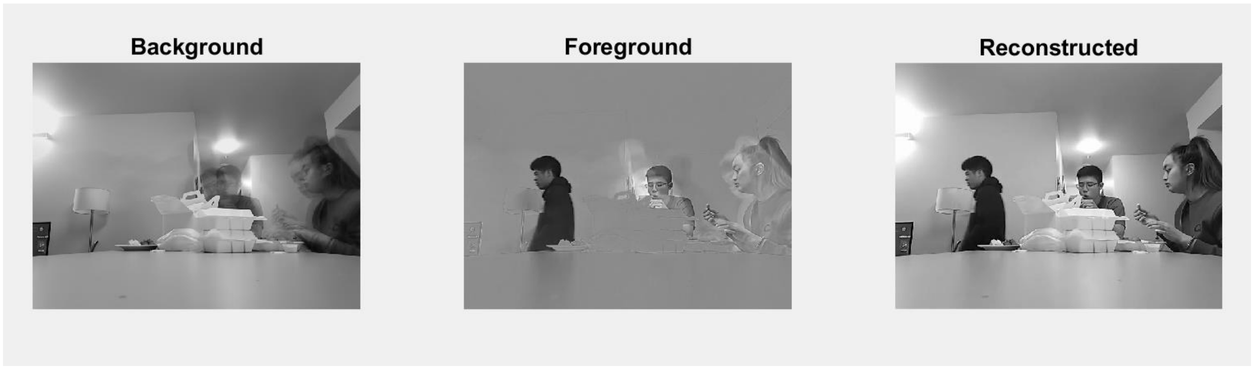

Figure 1. Hardworking college students eating dinner

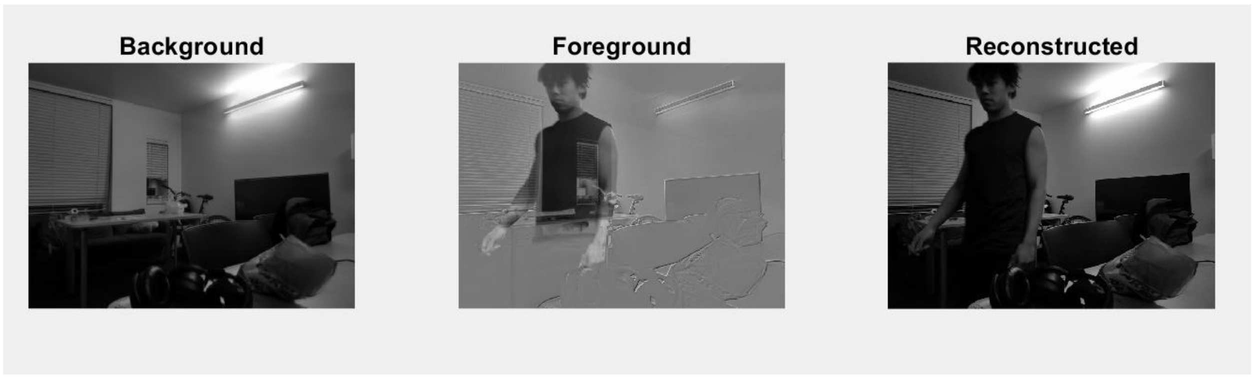

Figure 2. Sleepy college student walking across a messy living room

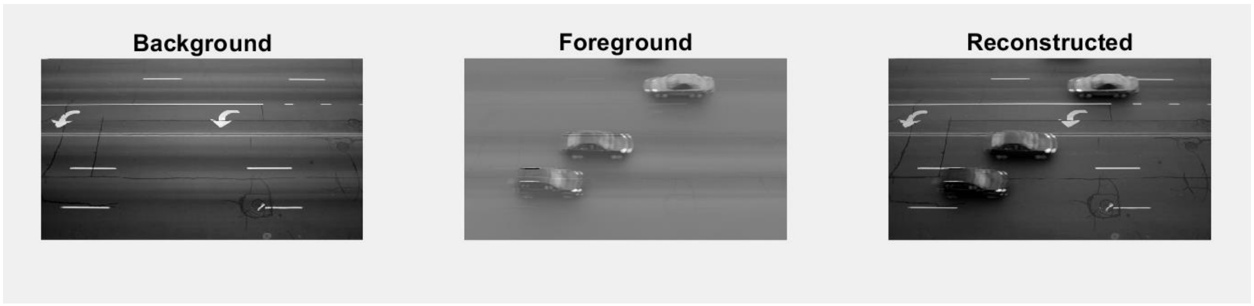

Figure 3. Bird's eye view of a traffic activity (imported from Youtube)

Figure 2 and Figure 3 shows perfect background frames which barely consists of noise, while in Figure 1, the background is a little noisy. An explanation for this occurrence is because Figure 2 and Figure 3 both have static backgrounds, while Figure 1's background consists of two people sitting still with tiny movements that are picked up by the background matrix.

For Figure 1 and Figure 2, 10 DMD modes are necessary to isolate the foreground while only 1 DMD mode is necessary for Figure 3. Figure 1 and 2 have fairly dynamic backgrounds with tiny continuous changes in the small details, which corresponds to the need for more backgrounds to be included which is described in the DMD rank. On the other hand, Figure 3 has a perfectly still camera recording a very static background, which means that there is only 1 absolute background. If we use ranks much higher than 10 for Figure 1 and Figure 2, the background will include shadows of the foreground person's movements, which is due to overfitting the backgrounds in the video inputs, which will worsen the quality of the background frames.

The background and foreground matrices which are visualized in the figures above are not post-processed by removing the residual matrix **R** from the foreground matrix **Xsparse** and adding it back to the background matrix **Xdmd**. Based on experience plotting the videos, using the residual matrix will include some features from the foreground to the background, and a loss of some parts of the foreground object. An example of the foreground image obtained by including **R** in the calculation is showin in Figure 4. Below. We can see that the background is closer to the shade of black since there are no negative values that sets the minimum color shading value lower. However, we lose some profiles of the foreground such as the end of the car seen on the left side of Figure 4, where the shape is no longer clear.

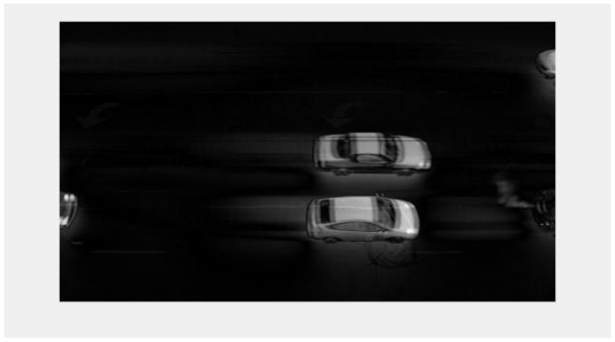

Figure 4. **Xsparse** postprocessed using **R**

If our goal is just to make the background darker, taking an absolute value to **Xsparse** works as well. When making the reconstructed video shown in Figure 1, 2, and 3, I reduced the rank of each frame to only the first 100 out of 400 terms to obtain reconstructions that are similar to the original video. In order to achieve perfect resemblance of the reconstruction, up to 350 out of 400 modes are necessary. The video which is the hardest to be rank reduced is the traffic video and one possible explanation is because it contains a lot of car movements (foreground).

A final remark is that visualizing these frames using pcolor() works worse than using imshow(), which results in frames where the objects has the texture of the background matrix, and this detail is very apparent and clear, unlike when imshow() is used.

## Section 5. Summary

In conclusion, Dynamic Mode Decomposition is a perfect tool to separate backgrounds from foregrounds. For the best result, the video input should be shot in a static position, and should record videos with static backgrounds. If the video is shaky and there exists multiple backgrounds, the small changes especially near the edges of objects will be picked up by the foreground matrix. If there exists multiple backgrounds, we can just adjust the rank reduction r accordingly, with more background included with increasing r.

This algorithm would synergize with filtering algorithm from the topic of Time Frequency Analysis. DMD provides a computationally quick algorithm that eliminates unwanted background data, which is very useful in object tracking in computer vision.

## Appendix A: MATLAB Functions Implementation Explanation

**`v = VideoReader('FileName')`**

Returns a video object containing all the information about the video such as each frames, the frame rate, the video resolution, etc.

**`X = zeros(m, n)`**

Creates a matrix of size m by n containing zeros. Useful to allocate memory to a matrix.

**`vidFrame = readFrame(v)`**

Reads the current frame in the video object.

**`Imresize(v, size)`**

Reduces the resolution of each frame by the factor of size.

**`[U S V] = svd(X, 'econ')`**

Returns the SVD matrices of X using computationally cheap algorithm.

**`lambda = diag(X)`**

Returns the diagonal elements of matrix X.

**`Reshape(X, m, n)`**

Reshapes X into an m by n matrix.

**`[Phi w lambda b Xdmd] = DMD(X1, X2, r, dt)`**

Author: Professor Kutz. Computes the DMD modes based on the matrices X1 and X2 (X1 delayed by 1). Reduces the rank to r with the timestep of dt.

**`R(R>0) = 0;`**

Sets the elements of R to zero if the element is positive.

## Appendix B: MATLAB Code

```
% Khrisna Kamarga
% AMATH 482 - Homework 5
clear all; close all; clc;

v = VideoReader('FroggerHighway.mp4'); % reads the input video

frames = 400; % specify the amount of frames desired to be played
dt = 1/v.frameRate; % the delta time between frames

m = 108*4; % vertical size of the frame
n = 192*4; % horizontal size of the frame

%for car: 0.1 m = 108, n = 192, 0.4 X4
%my phone: 1 m = 480, n = 640

X = zeros(m*n, frames); % data matrix

% enter how many frames to be skipped
for j = 1:500
    readFrame(v);
end

% the frames used for the video clip
for i = 1:frames
    vidFrame = imresize(rgb2gray(readFrame(v)), 0.4); % resize frame
    X(:,i) = reshape(double(vidFrame), m*n, 1); % vectorize
end

%% Preview of the video clip
for i = 1:frames
    pcolor(flipud(reshape(X(:,i), m, n))); colormap gray, shading interp;
    drawnow;
end

%% Dynamic Mode Decomposition
clc;
X1 = X;
X2 = X;
X1(:,end)=[]; %first until last-1
X2(:,1) = []; %second until last
t = dt*(1:frames); %the time

[Phi, w, lambda, b, XlowRank] = DMD(X1, X2, 1, dt);

Xsparse = X - abs(XlowRank); % foreground
%% Comment out to recompute Xsparse using the residual matrix
% R = Xsparse;
% R(R>0) = 0;
% XlowRank = R + abs(XlowRank);
% Xsparse = Xsparse - R;

%% Reconstruction
[U, S, V] = svd(X, 'econ');
```

```
r = 350;
% Energy Plot
lambda = diag(S); % discrete-time eigenvalues
plot(lambda);
U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
Xr = U_r*S_r*V_r.';

%% Visualizing the result using pcolor
clc; close all;
for i = 1:frames
    i % show the current frame number
    subplot(1,3,1)
    pcolor(flipud(reshape(abs(XlowRank(:,i)), m, n))); colormap gray, shading
interp;
    title("Background");
    subplot(1,3,2)
    pcolor(flipud(reshape(abs(Xsparse(:,i)), m, n))); colormap gray, shading
interp;
    title("Foreground");
    subplot(1,3,3)
    pcolor(flipud(reshape(abs(Xr(:,i) + Xsparse(:,i)), m, n))); colormap
gray, shading interp;
    title("Reconstructed")
    drawnow;
end

%% Preparing the Video Matrices
clc; close all
for j = 1:frames
    background(:,:,j) = reshape(XlowRank(:,j),m,n);
end
for j = 1:frames
    foreground(:,:,j) = reshape(Xsparse(:,j),m,n);
end
for j = 1:frames
    reconstruction(:,:,j) = reshape(Xr(:,j),m,n);
end

%% Visualizing using imshow()

for i = 1:frames
    i %show the current frame number
    subplot(1,3,1)
    imshow(background(:,:,i),[]);
    title("Background")
    subplot(1,3,2)
    imshow(foreground(:,:,i),[]);
    title("Foreground")
    subplot(1,3,3)
    imshow(reconstruction(:,:,i),[]);
    title("Reconstructed")
    drawnow
end
```

## DMD Function

```
% Author: Professor Kutz

function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% Computes the Dynamic Mode Decomposition of X1, X2
%
% INPUTS:
% X1 = X, data matrix
% X2 = X’, shifted data matrix
% Columns of X1 and X2 are state snapshots
% r = target rank of SVD
% dt = time step advancing X1 to X2 (X to X’)
%
% OUTPUTS:
% Phi, the DMD modes
% omega, the continuous-time DMD eigenvalues
% lambda, the discrete-time DMD eigenvalues
% b, a vector of magnitudes of modes Phi
% Xdmd, the data matrix reconstrcted by Phi, omega, b
%% DMD
[U, S, V] = svd(X1, 'econ');
r = min(r, size(U,2));
U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
Atilde = U_r' * X2 * V_r / S_r; % low-rank dynamics
[W_r, D] = eig(Atilde);
Phi = X2 * V_r / S_r * W_r; % DMD modes
lambda = diag(D); % discrete-time eigenvalues
omega = log(lambda)/dt; % continuous-time eigenvalues
%% Compute DMD mode amplitudes b
x1 = X1(:, 1);
b = Phi\x1;
%% DMD reconstruction
mm1 = size(X1, 2)+1; % mm1 = m - 1
time_dynamics = zeros(r, mm1);
t = (0:mm1-1)*dt; % time vector
for iter = 1:mm1
time_dynamics(:,iter)=(b.*exp(omega*t(iter)));
end
Xdmd = Phi * time_dynamics;
```

## Appendix C: References

Footage, B. B. (2012, January 24). Beachfront B-roll: Frogger Highway (Royalty Free Footage). Retrieved from https://www.youtube.com/watch?v=YCPBXbG4-vY

Kutz. (n.d.). Dynamic Mode Decomposition. Retrieved March 15, 2019.