AMATH482 homework5

Rui Zhu

March 2021

Abstract

In this assignment we will use the Dynamic Mode Decomposition method on two video clips containing a foreground and background object and separate the video stream to both the foreground video and a background.

1 Introduction and Overview

The aim of the **Dynamic Mode Decomposition** (DMD) method is to take advantage of low dimensionality in experimental data without having to rely on a given set of governing equations. DMD will allow us to forcast the data by finding a basis of spatial modes for which the time dynamics are just exponential functions (with potentially complex exponents). The DMD method is relatively efficient (with the cost of SVD algorithm) and does not make assumptions about the data itself.

We are given two video clips. The DMD will return a low-rank approximate reconstruction of our data (which represents the background of our video clips). By subtracting this low rank approximation from the original data, we will be able to get the sparse reconstruction (which represents the foreground object). However, this process might result in our sparse matrix having negative values, which would not make sense in terms of having negative pixel intensities. In this case, we will save these residual negative values in a matrix R and add back to the low-rank matrix.

2 Theoretical Background

2.1 The Setup

The DMD will need snapshots of data at M points in time and N points in space from a dynamical system. Each frame in our video clips will be represented as t_m where $m = 1, ..., M-1$ such that data snapshots are evenly spaced in time by a fixed Δt. The full snapshot matrix X is then denoted as

$$X = [U(x,t_1), U(x,t_2), ..., U(x,t_M)] \text{ where}$$

$$U(x,t_m) = \begin{bmatrix} U(x_1,t_m) \\ U(x_2,t_m) \\ \vdots \\ U(x_n,t_m) \end{bmatrix} \tag{1}$$

2.2 The Koopman Operator

The DMD method approximates the modes of the Koopman operator. The Koopman operator A is a linear, time-independent operator such that $x_{j+1} = Ax_j$ where the j indicates the specific data collection time and A is the linear operator that maps the data from time t_j to t_{j+1}. The vector x_j is an N-dimensional vector of the data points collect at time j. That is, applying A to a snapshot of data will advance it forward in time by Δt. In short, we are asking for a linear mapping from one timestep to the next, even though the dynamics of the system are likely nonlinear.

2.3 Dynamic Mode Decomposition (DMD)

To construct the appropriate Koopman operator that best represents the data collected, we will consider the matrix

$$X_1^{M-1} = [x_1, x_2, x_3, ..., x_{M-1}]$$

where the notation X_1^{M-1} is columns 1 through $M-1$ of the full snapshot matrix X. Then the Koopman operator allows us to rewrite this as

$$X_1^{M-1} = [x_1, Ax_1, A^2x_1, ..., A^{M-2}x_1]$$

The columns are formed by applying powers of A to the vector x_1, and are said to form the basis for the Krylov subspace. This gives us a way of relating the first $M-1$ snapshots to x_1. Then we can get

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

where e_{M-1} is a vector with all zeros expect a 1 at the $(M-1)$th component. That is, A applied to each column of X_1^{M-1}, given by x_m, maps to the corresponding column of X_2^M , given by x_{m+1}, but the final point x_M wasn't included in our Krylov basis, so we add in the residual (or error) vector r to account for this. Our goal is to find the unknown matrix A which can be reached by finding another matrix with the same eigenvalues and eigenvectors.

We will begin with SVD to write $X_1^{M-1} = U\Sigma V^*$, where $U \in C^{N*K}, \Sigma \in R^{K*K}, V \in C^{M-1*K}$. Here the K is the rank of X_1^{M-1}, which essentially tells us the number of dimensions that the data varies in. We did not use the full matrix Σ due to the goal of dimensionality reduction. Then, according to above, we have

$$X_2^M = AU\Sigma V^* + re_{M-1}^T$$

We are going to choose A in such a way that the columns in X_2^M can be written as linear combinations of the columns of U. This is the same as requiring that they can be written as linear combinations o the POD modes. Hence, the residual vector r must be orthogonal to the POD basis, giving that $U*r = 0$. Multiplying the above equation through by U^* on the left gives:

$$U^*X_2^M = U^*AU\Sigma V^*$$

We than isolate

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

where the elements contained by $\tilde{S}$ are all known. Sincen that $\tilde{S}$ and A are related by applying a matrix on one side and its inverse on the other, $\tilde{S}$ and A have the same eigenvalues: if y is an eigenvector of $\tilde{S}$ then Uy is an eigenvector of A. The eigenvector/eigenvalue pairs of $\tilde{S}$ is written as:

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

Thus giving the eigenvectors of A, called the **DMD modes**, by

$$\psi_k = Uy_k$$

Now we've got all we need to describe continual multiplications by A! We can just expand in our eigenbasis to get

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

As discussed above, K is the rank of X_1^{M-1} . The b_k are the initial amplitude of each mode, and the matrix Ψ contains the eigenvectors ψ_k as its columns.

2.4 Background Subtraction using DMD

Assume that ω_p where $p \in 1, 2, ..., l$ satisfies $||\omega_p|| = 0$, and that $\omega_j, j \neq p$ is bounded away from zero, gives us:

$$X_{DMD} = b_p \psi_p e^{\omega_p t} + \textstyle\sum_{j \neq p} b_j \psi_j e^{\omega_j t}$$

where the first element represents the Background video, and the second element represents the foreground video.

This will produce a X_{DMD} matrix with the same dimensions as our original X matrix, but each term of the DMD reconstruction is complex: $b_p \psi_p e^{\omega_p t}$, though they sum to a real-valued matrix.

To solve the above problem, we calculate the DMD's approximate low-rank reconstruction according to:

$$X_{DMD}^{Low-Rank} = b_p \psi_p e^{\omega_p t}$$

Since

$$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$$

Then the DMD's approximate sparse reconstruction can be calculated with real-valued elements only as follows:

$$X_{DMD}^{Sparse} = \textstyle\sum_{j \neq p} b_j \psi_j e^{\omega_j t} = X - |X_{DMD}^{Low-Rank}|$$

where $| \dots |$ yields the modulus of each element within the matrix. The negative values in X_{DMD}^{Sparse} can be then put into a $n * m$ matrix R (same size as our X matrix) and then be added back to $X_{DMD}^{Low-Rank}$.

This way the magnitudes of the complex values from the DMD reconstruction are accounted for, while maintaining the important constraints that

$$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$$

so that none of the pixel intensities are below zero, and ensuring that the approximate low-rank and sparse DMD reconstructions are real-valued.

3 Algorithm Implementation and Development

- Load the mp4 videos using `VideoReader`, obtain the length of the video, number of frames with parameters of `VideoReader`
- Process each frame of the current video, convert this image matrix to grayscale using `rgb2gray` and convert its values from *uint8* to *double*
- Scale down the size by 25% of each frame for better computational performance using `imresize`, and save the data into a column using `reshape`
- Define $\Delta t = 1/Framerate$
- Split the processed data matrix into two matrices **X1** and **X2**, where **X1** includes the snapshots from frame 1 to the second last frame and **X2** includes the snapshots from frame 2 to the last frame. This step is for determining how to get from **X1** to **X2** using a linear transformation **A**
- Take Singular Value Decomposition (SVD) of **X1** using `svd()` and plot the diagonal values of the Σ matrix (divided by the sum of the diagonal values) to see how much energy each singular value captured. We then determine the number of significant modes based on the graph, and saved this value as `K`

- Truncate our U,Σ, and V matrices from SVD to $U \in C^{N*K}, \Sigma \in R^{K*K}, V \in C^{M-1*K}$
- Calaulate $\tilde{S} = U^* X_2^M V \Sigma^-1$
- Take eigen value decomposition of $\tilde{S}$ to get its eigenvalues and eigenvectors
- Calculate DMD modes Φ by multiplying the eigenvectors of S against U; ω
- Use pseudoinverse (Φ) to get initial conditions `y0`
- Calculate DMD solutions at each time step using `y0.*exp(omega*t(iter))`
- Obtain matrix R for the residual negatives values of our sparse matrix, add R to the low rank DMD approximation and subtract R from sparse matrix

4 Computational Results

4.1 Video 1: Monte Carlo

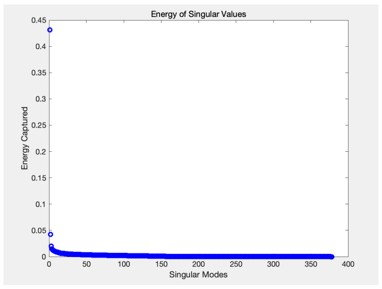

Figure 1: Energy Diagram of Singular Values

Based on the plot above, we set our $K = 2$ since only the first two modes are significant.

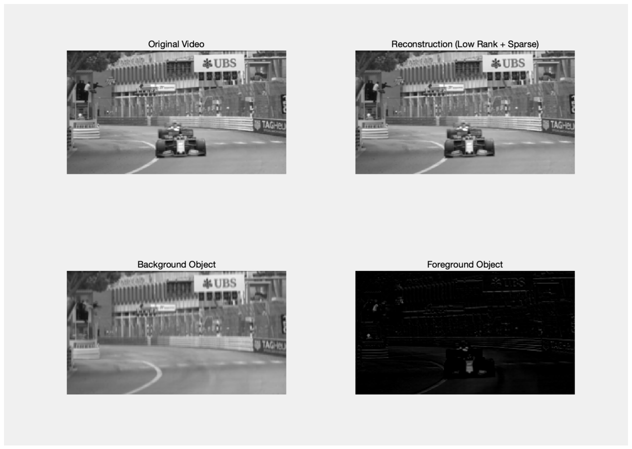

Figure 2: Video Separation and Reconstruction

From the plot above, we can see the foreground (race car) is relative visible and clearly separated from the background. And the reconstruction using the low-rank and sparse matrix look very similar to the original video, which implies a satisfying performance of DMD.

4.2 Video 2: Ski Drop

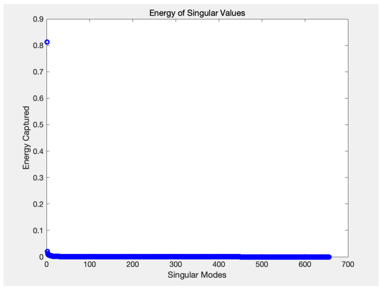

Figure 3: Energy Diagram of Singular Values

Based on the plot above, we set our $K = 2$ since only the first two modes are significant.

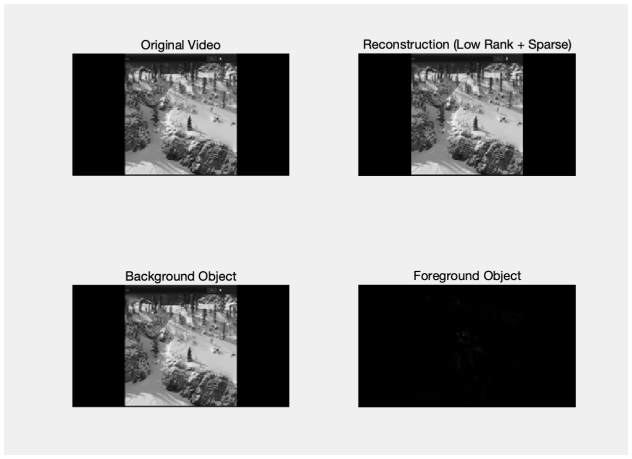

Figure 4: Video Separation and Reconstruction

From the plot above, we can see the foreground (tourist) is relative visible and somehow separated from the background. It is not as good as the racing car case above. This might be caused by the small size of the foreground object, which results in a less satisfying performance of our DMD. Also, from the plot above the flying snow around the tourist is also identified as foreground object, which makes the tourist even harder to separate. The reconstruction using the low-rank and sparse matrix look similar to the original video, which implies an OK performance of DMD.

5 Summary and Conclusions

We performed DMD on two video clips with a moving foreground object and a static background object to create a low-rank (background) and sparse (foreground) approximation of them. In both cases two significant modes from SVD have successfully capture most of the energy, and the results of separation are also satisfying. However, the portion of foreground object of video 2: Ski Drop is way smaller than the portion of foreground object of video 1: Racing Car, and the flying snow around the ski tourist also makes it harder for DMD to separate.

Appendix A MATLAB Functions

- `VideoReader(.mp4)` creates a VideoReader object which contains informations of the input mp4 file such as length (duration) or frame rate.

- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I 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 $A = U * \Sigma * V'$.
- `uint8(X)` converts X (`double`) into type `uint8` so that we can view the result matrix using `imshow`

Appendix B MATLAB Code

```
clear all; close all; clc
%% Data Import & Setup
mc = VideoReader('monte_carlo_low.mp4');
ski = VideoReader('ski_drop_low.mp4')

% dt = 1/mc.Framerate;
% t = 0:dt:mc.Duration;
% vidFrames = read(mc);
% numFrames = get(mc, 'numberOfFrames');

dt = 1/ski.Framerate;
t = 0:dt:ski.Duration;
vidFrames = read(ski);
numFrames = get(ski, 'numberOfFrames');

for j = 1:numFrames
    mov(j).cdata = vidFrames(:,:,:,j);
    mov(j).colormap = [];
end

X = [];

% scale the frames down by 1/4 to improve speed
for k = 1:numFrames
    x = frame2im(mov(k));
    X = [X, reshape(double(rgb2gray(imresize(x,0.25))),[135*240,1])];
end

%% DMD
X1 = X(:,1:end-1);
X2 = X(:,2:end);

[U,Sigma,V] = svd(X1,'econ');

plot(diag(Sigma)/sum(diag(Sigma)), 'bo', 'Linewidth', 2);
title("Energy of Singular Values");
ylabel("Energy Captured");
xlabel("Singular Modes");

%truncate to rank r
r = 2;
Ur = U(:,1:r);
Sigmar = Sigma(1:r,1:r);
Vr = V(:,1:r);
```

```matlab
S = Ur'*X2*Vr*diag(1./diag(Sigmar));


[eV,D] = eig(S); %compute eigenvalues & eigenvectors
mu = diag(D); %extract eigenvalues
omega = log(mu)/dt;
Phi = Ur*eV;

bar(abs(omega))
title("Absolute value of Omega");


%Create the DMD solution
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions

X_modes = zeros(length(y0),length(t)-1);
for iter = 1:(length(t)-1)
    X_modes(:,iter) = y0.*exp(omega*t(iter));
end
X_dmd = Phi*X_modes;

%Create Sparse and Low-Rank Matrix
Xsparse = X1-abs(X_dmd);

R = Xsparse.*(Xsparse<0);

X_lowrank = R + abs(X_dmd);
X_sparse = Xsparse-R;

X_r = X_lowrank + X_sparse;

%% Display
% recon = reshape(X_r, [135,240,378]);
% background = reshape(X_dmd, [135,240,378]);
% foreground = reshape(X_sparse, [135,240,378]);
% original = reshape(X1, [135,240,378]);

recon = reshape(X_r, [135,240,453]);
background = reshape(X_dmd, [135,240,453]);
foreground = reshape(X_sparse, [135,240,453]);
original = reshape(X1, [135,240,453]);

subplot(2,2,1)
imshow(uint8(original(:,:,50)));
title("Original Video");

subplot(2,2,2)
imshow(uint8(recon(:,:,50)));
title("Reconstruction (Low Rank + Sparse)");

subplot(2,2,3)
imshow(uint8(background(:,:,50)));
title("Background Object");
```

```
subplot(2,2,4)
imshow(uint8(foreground(:,:,50)));
title("Foreground Object");
```