Hanze Zhang
Dr. Bramburger
AMATH 482
Assignment 5
03/17/2021

Abstract:

If we are given a video, which is consisted of an object moving in the foreground and a steady background, is it possible that we can extract its foreground and background individually? In this assignment, we will be trying to accomplish the above task using Dynamic Mode Decomposition (DMD) with two given videos: a ski-drop video and a monte-carlo video. In the result, we successfully separated the background and foreground for both videos.

Introduction and Overview:

The aim of DMD is to take advantage of low dimensionality in experimental data without having to rely on a given set of governing equations. DMD can help us predict what our data will be in the future. Meanwhile, it allows us to extract dynamical movements from a high dimensional, complex dataset, such as two videos given in this assignment.

In the first video, we have a skier skiing down from a snow ramp. In this video, the skier is the dynamical system that we want to extract, and the rest is the background. In the second video, we have an automobile racing in the Monte-Carlo event. In this video, the car that is turning is the dynamical system that we want to extract, and the rest is the background.

Theoretical Background:

The most important technique we will be using in this assignment is DMD. DMD is a

data-driven method that approximates the modes of the Koopman operator. The Koopman operator A is a linear, time-independent operator such that:

$$x_{j+1} = Ax_j \qquad (1)$$

Here, A is the linear operator that maps the data from time t_j to t_j. The vector x_j is a N-dimensional vector of the data points collected at time j. If we apply A to a snapshot of our data, we will advance it forward in time by Δt. With the above equation, we will be able to map from one timestep to the next, even though the dynamics of the system are likely nonlinear.

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

$$X_1^{M-1} = [x_1\ x_2\ \dots\ x_{M-1}] \qquad (2)$$

With Koopman operator, the above becomes:

$$X_1^{M-1} = [x_1\ Ax_1\ A^2x_1\ \dots\ A^{M-2}x_1] \qquad (3)$$

Here, x_j represents the snapshot of data at time j. The columns in equation (3) form the basis for the Krylov subspace. In matrix form:

$$X_2^M = AX_1^{M-1} + re_{M-1}^T \qquad (4)$$

Here, e_{M-1} is the vector with all zeros except a 1 at the $(M-1)$st component. Notice that in equation (4), the final point x_M was not included in our Krylov basis, so we need to add in the residual (or error) vector to account for this. Now, our task is to find our A. Since the matrix A can be completely understood by its eigenvalues and eigenvectors, we can find A just by finding another matrices with the same eigenvalues.

If we apply the singular value decomposition (SVD) on X_1^{M-1}:

$$X_1^{M-1} = U\Sigma V^* \qquad (5)$$

$$X_2^M = AU\Sigma V^* + re_{M-1}^T \qquad (6)$$

If we choose A such that its columns in X_2^M can be written as linear combinations of the columns of U, the residual vector must be orthogonal to the POD basis, $U^*r = 0$. Thus,

$$U^* X_2^M = U^* A U \Sigma V^* \quad (7)$$

$$U^* A U = \underbrace{U^* X_2^M V \Sigma^{-1}}_{=:\tilde{S}} \quad (8)$$

Now, note that $\tilde{S}$ and A are similar in this case, they should have the same eigenvalues. If y is an eigenvalue of $\tilde{S}$, then Uy is the eigenvalue of A.

$$\tilde{S} y_k = \mu_k y_k \quad (9)$$

$$\psi_k = U y_k \quad (10)$$

ψ_k is the eigenvectors of A, and we call it DMD modes. If we expand our eigenbasis:

$$x_{DMD}(t) = \sum_{k-1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t}) b \quad (11)$$

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. Additionally, $\omega_k = \frac{\ln(\mu_k)}{\Delta t}$. Now, we can compute b if we set $t = 0$:

$$x_1 = \Psi b \Longrightarrow b = \Psi^{\dagger} x_1 \quad (12)$$

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

The DMD spectrum of frequencies can be used to subtract background modes. Assume ω_p, where $p \in \{1,2,\ldots,l\}$, satisfies $\|\omega_p\| \approx 0$, and that $\|\omega_p\|\ \forall\, j \neq p$ is bounded away from 0, then we have:

$$X_{DMD} = \underbrace{b_p \varphi_p e^{\omega_p t}}_{Background\ Video} + \underbrace{\sum_{j \neq p} b_j \varphi_j e^{\omega_j t}}_{Foreground\ Video} \quad (13)$$

Here, X_{DMD} should have the same dimension as our original data matrix X. However, each term of the DMD reconstruction is complex. Consider calculating the DMD's approximate low-rank reconstruction with the method we described above:

$$X_{DMD}^{Low-Rank} = b_p \varphi_p e^{\omega_p t} \quad (14)$$

Since we know that $X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$, we can find X_{DMD}^{Sparse}:

$$X_{DMD}^{Sparse} = \sum_{j \neq p} b_j \varphi_j e^{\omega_j t} = X - \left| X_{DMD}^{Low-Rank} \right| \quad (15)$$

Finally, this may result in X_{DMD}^{Sparse} having negative values in some of its elements, which would not make sense in terms of having negative pixel intensities. To resolve this, we can put negative values into a residual $n * m$ matrix R and then create our low-rank and sparse reconstructions as follows:

$$X_{DMD}^{Low-Rank} \leftarrow R + \left|X_{DMD}^{Low-Rank}\right| \qquad (16)$$

$$X_{DMD}^{Sparse} \leftarrow X_{DMD}^{Sparse} - R \qquad (17)$$

This way the magnitudes of the complex values from the DMD reconstruction are accounted for, while maintaining the important constraints that $X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$ and ensuring that none of the pixel intensities are below zero, as well as the approximate low-rank and sparse DMD reconstructions are real-valued.

Algorithm Implementation and Development:

First, we use the function $VideoReader$ to load our two videos into our workspace in MATLAB. We then calculate $dt = \frac{1}{Frame\ rates}$ and find our vector t. Next, we convert each frame in the video to grayscale using $rgb2gray$ and its values to double using $im2double$. Finally, we call the function $reshape$ to put each frame into a column, and this will form our data matrix X.

Now, our first step is to create the matrices X_1^{M-1} and X_2^M. Here, we make X_1^{M-1} represents the snapshots from the first frame to the second last frame, while X_2^M represents the data from the second frame to the last frame. Our goal is to find a linear transformation A which can get us from X_1^{M-1} to X_2^M. Then, we need to apply SVD on X_1^{M-1}, which can help us find the eigenvalues of $\tilde{S}$. After we obtain three matrices $U, Sigma, V$, we can plot the energy spectrum to determine the number of significant modes, and this number is referred as r. Based on r we found, we can truncate our three matrices so they only contain information of significant modes. Then, we can compute the matrix $\tilde{S}$ based on equation (8) we mentioned above. Once we find $\tilde{S}$, we can use the function eig along with the function $diag$ to obtain its eigenvalues

and eigenvectors. Then, we can find the matrices of omega and phi. Here, the vector omega still contains our eigenvalues, but in another form, while the vector phi contains the eigenvectors of the matrix A. After we find omega and phi, we can start create our DMD solution. We can get our initial condition by doing the pseudoinverse, and then get the solution of each time point by $b .* \exp(omega * t(iter))$.

Now, we can obtain our $X_{DMD}^{Low-Rank}$ matrix by multiplying the vector phi with the time dynamics matrix we just created. Then, we want to find all negative values and make it a matrix R. Based on the equation $(15), (16), (17)$, we can find real-valued $X_{DMD}^{Low-Rank}$ and X_{DMD}^{Sparse} by adding R to $X_{DMD}^{Low-Rank}$ and subtracting R from X_{DMD}^{Sparse}. Finally, we can add $X_{DMD}^{Low-Rank}$ and X_{DMD}^{Sparse} together to reconstruct our video.

We repeat the above steps for both videos.

Computational Results:

- **Monte-Carlo:**

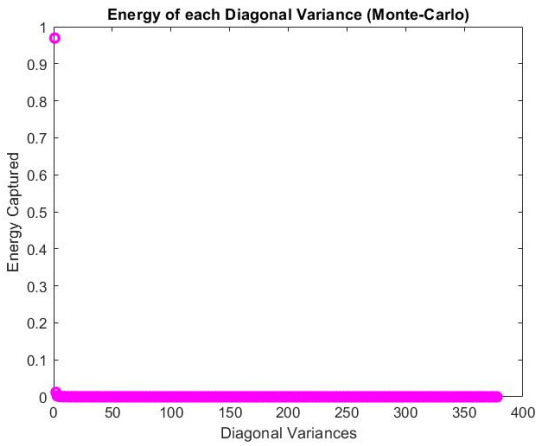

Figure 1 - Energy captured for each mode for monte-carlo video

From figure 1, we can observe that there are only two non-zero values, indicating that only two modes are significant. Thus, we can truncate our matrices to only rank 2.

After applying DMD, we obtain the following pictures (each captured at the 150th

frame):

Figure 2 - Background without R adding on it (Monte-Carlo) **Figure 3 - Background with R adding on it (Monte-Carlo)**

Figure 4 - Foreground without R adding on it (Monte-Carlo) **Figure 5 - Foreground with R adding on it (Monte-Carlo)**

Figure 6 - Reconstructed frame (Monte-Carlo)

From Figure 3 and Figure 4, we can observe that in our sparse reconstruction for the foreground, they are pretty similar no matter we incorporated the R matrix or not. In both figures, we can barely see the car, but we can still clearly see some shape of things in the background. This indicates that we do separated something from the background. As for the reconstruction for the background, adding the matrix R makes the body of the car reappear. If we do not incorporate the matrix R into our reconstruction for the background, DMD does pretty well in separating the background. In Figure 5, our reconstructed frame is pretty much the same as the

original frame.

- **Ski-drop video:**

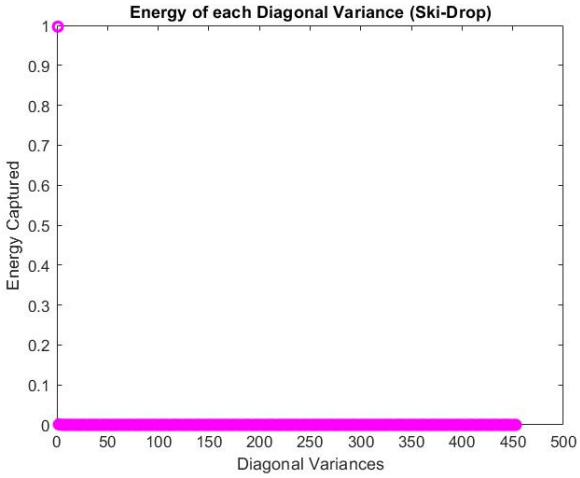

Figure 7 - Energy Captured for each mode (Ski-drop Video)

From figure 7, we can observe that there are only one non-zero values, indicating that only one modes are significant. Thus, we can truncate our matrices to only rank 1.

After applying DMD, we obtain the following pictures (each captured at the 150th

frame):

Figure 8 - Background without R adding on it (Ski)

Figure 9 - Background with R adding on it(Ski)

Figure 10 - Foreground without R adding on it (Ski)

Figure 11 - Foreground with R adding on it (Ski)

Figure 12 - Reconstructed frame (Ski)

From Figure 10 and Figure 11, we can observe that in our sparse reconstruction for the foreground, they are pretty similar no matter we incorporated the R matrix or not. In both figures, we can barely see the skier, but we can still clearly see some shape of things in the background. This indicates that we do separated something from the background. As for the reconstruction for the background, adding the matrix R makes the skier reappear. If we do not incorporate the matrix R into our reconstruction for the background, DMD does pretty well in separating the background. In Figure 12,

our reconstructed frame is pretty much the same as the original frame.

Summary and Conclusions:

If we compare two results, we can observe that DMD does a better job in separating the background and foreground with closer objects (Monte-Carlo automobiles). If we use it to separate a dynamic movements in a far place, the result might not be as good as expected. However, in general, DMD still does very well in separating a background and foreground, and capturing the dynamic movements.

Appendix I (MATLAB Functions):

- $diag(X)$: Create diagonal matrix or get diagonal elements of matrix
- $[W, D] = eig(A)$: returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that $A * V = V * D$.
- $VideoReader$: Create object to read video files
- $imshow$: Display image
- $reshape(X, sz)$: reshapes A using the size vector, sz, to define $size(B)$.
- $[u, s, v] = svd(X)$: $svd(A)$ performs a singular value decomposition of matrix A, such that $A = U * S * V'$.
- $Im2uint8(X)$: Convert image to 8-bit unsigned integers
- $im2double(I)$: converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. im2double rescales the

output from integer data types to the range [0,1].

- $im2gray$: Convert RGB image to grayscale

Appendix II (MATLAB Code):

```
% Clear workspace
clear all; close all; clc;

%% import videos
vid_ski_drop = VideoReader('ski_drop_low.mp4');
% vid_ski_drop = VideoReader('monte_carlo_low.mp4');

dt = 1/vid_ski_drop.Framerate;
t = 0:dt:vid_ski_drop.Duration;
vidFrames_ski = read(vid_ski_drop);
numFrames_ski = get(vid_ski_drop,'numberOfFrames');

frame = im2double(vidFrames_ski(:,:,:,1));
X = zeros(size(frame,1) * size(frame,2), numFrames_ski);

for j = 1:numFrames_ski
    frame = vidFrames_ski(:,:,:,j);
    frame = im2double(rgb2gray(frame));

    X(:,j) = reshape(frame, 540 * 960, []);
    % imshow(frame); drawnow
end

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

[U, Sigma, V] = svd(X1,'econ');
lambda = diag(Sigma).^2;

% Plot SVD Results
% Singular values1
figure(1)
plot(diag(Sigma),'ko','Linewidth',2)
ylabel('\sigmaj')

figure(2)
plot(1:453, lambda/sum(lambda), 'mo', 'Linewidth', 2);
title('Energy of each Diagonal Variance (Ski-Drop)');
xlabel('Diagonal Variances'); ylabel('Energy Captured');

%%
r = 1;
U = U(:, 1:r);
```

```
Sigma = Sigma(1:r, 1:r);
V = V(:, 1:r);

%%
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;

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

%% Sparse
X_sparse = X1 - abs(udmd(:,size(udmd, 2)));
neg_vals = X_sparse < 0;
R = X_sparse .* neg_vals;

udmd_new = R + abs(udmd(:,size(udmd, 2)));
X_sparse_new = X_sparse - R;

X_reconstructed = X_sparse_new + udmd_new;

%% Show
temp =    reshape(X_reconstructed, [size(frame, 1), size(frame, 2), length(t) - 1]);
imshow(im2uint8(temp(:,:,150)))
for i = 1:453
    imshow(im2uint8(temp(:,:,i)))
end
title("Total Reconstruction (Sparse + Low Rank)");
```