Assignment 5

Background Subtraction in Video Streams

Author: Sofia Chen

Abstract: This assignment is to use the Dynamic Mode Decomposition method on the video clips *ski drop.mov* and *monte carlo.mov* containing a foreground and background object and separate the video stream to both the foreground video and a background.

Section I. Introduction and Overview

This assignment is to perform a Dynamic Mode Decomposition method on two video clips. We want to separate the foreground and background in the videos. We convert them into data matrix and reshape them to analyze. We first apply the SVD to the matrix. I also make the singular value spectrum to look for its rank.

After I perform the above, I do the Dynamic Mode Decomposition. We will then have the low-rank reconstruction of our data, which is also the background of our video. Next, we will get the foreground part by subtracting the low-rank reconstruction from the original matrix.

Section II. Theoretical Background

SVD:

The *singular value decomposition* of the matrix A is: $\boldsymbol{A = U\Sigma V^*}$, where $U \in R^{m\times m}$ and $V \in R^{n\times n}$ are unitary matrices, and $\Sigma \in R^{m\times m}$ is diagonal.

The values σ_n on the diagonal of Σ are called the *singular values* of the matrix A. The vectors u_n which make up the columns of U are called the *left singular vectors* of A. The vectors v_n which make up the columns of V are called the *right singular vectors* of A. U and V are unitary matrices that simply lead to rotation and Σ scales an image as prescribed by the singular values.

DMD:

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.

To construct the appropriate Koopman operator that best represents the data collected, we will consider the matrix $X_1^{M-1} = [x_1\ x_2\ x_3 \dots x_{M-1}]$, where we use the shorthand x_j to denote a snapshot of the data at time t_j.

The Koopman operator allows us to rewrite this as $X_1^{M-1} = [x_1\ Ax_1\ A^2x_1 \dots A^{M-2}x_2]$. 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.

Hence, we have (in theory at least) a way of relating the first M − 1 snapshots to x1 using just the Koopman operator/matrix. We can write the above in matrix form to get $X_2^M = AX_1^{M-1} + re_{M-1}^T$, where e_{M-1} is the vector with all zeros except a 1 at the $(M-1)$st component. That is, A applied to each column of X_1^{M-1}, given by x_j, maps to the corresponding column of X_2^M, given by x_{j+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. We are going to circumvent finding A directly by finding other matrices with the same eigenvalues.

First, let's use the SVD to write $X_1^{M-1} = U\Sigma V^*$. Then, from above we get $X_2^M = AU\Sigma V *. + re_{M-1}^T$. 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 of 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 byU^* on the left gives: $U^* X_2^M = U^* AU\Sigma V^*$. Then, we can isolate for U^*AU by multiplying by V and then $\Sigma - 1$ on the right to get $U^*AU = U^* X_2^M V\Sigma^{-1} = \tilde{S}$. Remember, everything on the right side is known from the input data.

From what we know about singular values, K is the rank of the data matrix X_1^{M-1}, which essentially tells us the number of dimensions that the data varies in. Ideally, we would like K to be small relative to N and M.

Notice that $\tilde{S}$ and A are related by applying a matrix on one side and its inverse on the other. This means they are similar. If y is an eigenvector of $\tilde{S}$, then Uy is an eigenvector of A. So, let's write the eigenvector/eigenvalue pairs of $\tilde{S}$ 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 have got all we need to describe continual multiplications by A. We can just expand in our eigenbasis to get $x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega k t} = \Psi diag(e^{\omega k t}) b$.

As discussed in the important note 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. We just take the linear dynamics governed by A, compute the eigenvalues and eigenvectors, and then write the dynamics in terms of a linear combination of exponential growth/decay/oscillation in the direction of each eigenvector. We know that at time $t = 0$ in the above formula we have to get x_1 because this was our initial condition to generate the other x_m. Therefore, taking $t = 0$ in the above gives $x_1 = \psi_b \Rightarrow$

$b = \Psi^{\dagger} x_1$, where Ψ† is the pseudoinverse of the matrix Ψ.

The DMD spectrum of frequencies can be used to subtract background modes. Specifically, assume that ω_p, where $p \in \{1, 2, \ldots, \ell\}$, satisfies $\|\omega_p\| \approx 0$, and that $\|\omega_j\| \; \forall \; j \neq p$ is bounded away from zero. Thus,

$$\mathbf{X}_{\text{DMD}} = \underbrace{b_p \boldsymbol{\varphi}_p e^{\omega_p \mathbf{t}}}_{\text{Background Video}} + \underbrace{\sum_{j \neq p} b_j \boldsymbol{\varphi}_j e^{\omega_j \mathbf{t}}}_{\text{Foreground Video}} \tag{1}$$

Assuming that $\mathbf{X} \in \mathbb{R}^{n \times m}$, then a proper DMD reconstruction should also produce $\mathbf{X}_{\text{DMD}} \in \mathbb{R}^{n \times m}$. However, each term of the DMD reconstruction is complex: $b_j \boldsymbol{\varphi}_j \exp(\omega_j \mathbf{t}) \in \mathbb{C}^{n \times m} \; \forall j$, though they sum to a real-valued matrix. This poses a problem when separating the DMD terms into approximate low-rank and sparse reconstructions

because real-valued outputs are desired and knowing how to handle the complex elements can make a significant difference in the accuracy of the results. Consider calculating the DMD's approximate low-rank reconstruction according to

$$\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} = b_p \boldsymbol{\varphi}_p e^{\omega_p \mathbf{t}}.$$

Since it should be true that

$$\mathbf{X} = \mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} + \mathbf{X}_{\text{DMD}}^{\text{Sparse}},$$

then the DMD's approximate sparse reconstruction,

$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \sum_{j \neq p} b_j \boldsymbol{\varphi}_j e^{\omega_j \mathbf{t}},$$

can be calculated with real-valued elements only as follows...

$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \mathbf{X} - \left| \mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} \right|,$$

where $|\cdot|$ yields the modulus of each element within the matrix. However, this may result in $\mathbf{X}_{\text{DMD}}^{\text{Sparse}}$ having negative values in some of its elements, which would not make sense in terms of having negative pixel intensities. These residual negative values can be put into a $n \times m$ matrix $\mathbf{R}$ and then be added back into $\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}}$ as follows:

$$\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} \leftarrow \mathbf{R} + \left| \mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} \right|$$
$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} \leftarrow \mathbf{X}_{\text{DMD}}^{\text{Sparse}} - \mathbf{R}$$

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

$$\mathbf{X} = \mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} + \mathbf{X}_{\text{DMD}}^{\text{Sparse}},$$

so that none of the pixel intensities are below zero, and ensuring that the approximate low-rank and sparse DMD reconstructions are real-valued. This method seems to work well empirically.

Section III. Algorithm Implementation and Development

To start, we use the MATLAB function videoReader to read in the two videos. We then turn the video into gray scale, cast it into double, and reshape it to store the data in the matrix X, by processing each frames of the videos. Next, we split the X into X_1^M and X_2^M, which are snapshots of the data from the first frame to the second-last frame, and the snapshots of the data from the second frame to the last frame.

Then, we do the SVD to the X_1^M. After plotting the singular value spectrum, we can find that approximately the first 2 of the singular values is dominating the *ski_drop* video, and the first 3 of the singular values is dominating the *monte_carlo* video. So, we will use these two features (referring the modes) for the following demonstration. Next, we will truncate our U, Σ, V to this rank and then calculate the $\tilde{S}$.

We then get the eigenvalues and eigenvectors of $\tilde{S}$ to calculate the DMD modes ψ and ω. We use the pseudoinverse to get initial value $y0$. Then we can calculate the DMD's approximate low-rank reconstruction, which is our background video, corresponding to ω is close to 0. And we get the foreground video by subtracting the low-rank reconstruction from the original matrix. We also need to rescale it into the range 0 to 1 to get rid of the negative pixels.

Section IV. Computational Results

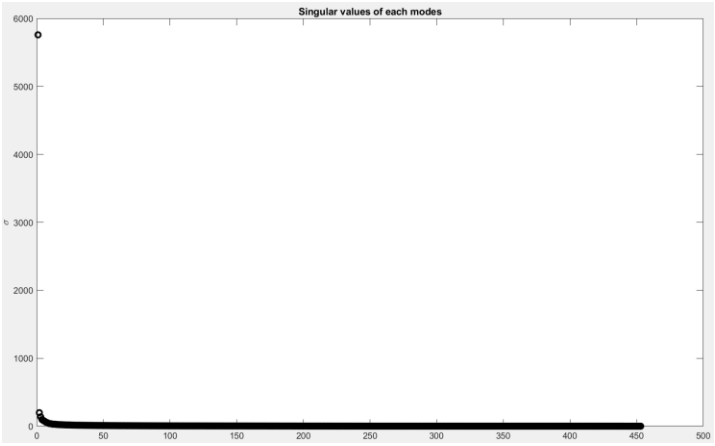

Figure 1: Singular value spectrum of the SVD of the ski_drop data.

Figure 2: Original video of ski drop

Figure 3: Background video of ski drop

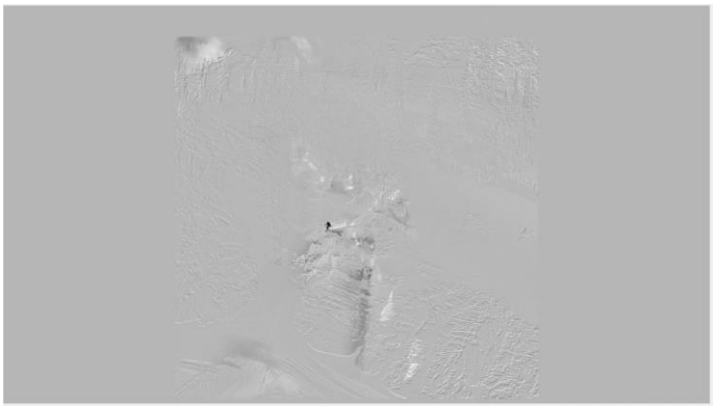

Figure 4: Foreground video of ski drop

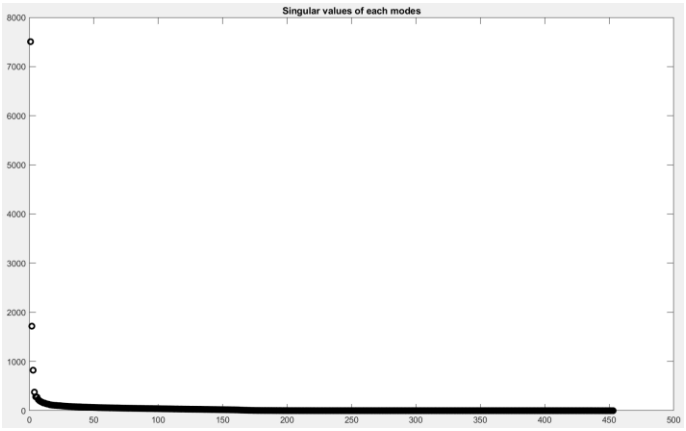

Figure 5: Singular value spectrum of the SVD of the monte carlo data.

Figure 6: Original video of monte carlo

Figure 7: Background video of monte carlo

Figure 8: Foreground video of monte carlo

Section V. Summary and Conclusions

We first use the SVD to analysis the data. We get then rank by looking at the singular value spectrum. We truncate our U, Σ, V to this rank and then calculate the $\tilde{S}$. We then get the eigenvalues and eigenvectors of $\tilde{S}$ to calculate the DMD modes ψ and ω. We use the pseudoinverse to get initial value $y0$. Then we can calculate the DMD's approximate low-rank reconstruction, which is our background video. Finally, we get the foreground video by subtracting the low-rank reconstruction from the original matrix.

Appendix A. MATLAB functions used and brief implementation explanation

- ***reshape*** - *Reshape array*: This MATLAB function reshapes A using the size vector, sz, to define size(B).
- ***rgb2gray*** - *Convert RGB image or colormap to grayscale*: This MATLAB function converts the true color image RGB to the grayscale image I.
- ***svd*** - *Singular value decomposition*: This MATLAB function returns the singular values of matrix A in descending order.
- ***im2double*** - *Convert image to double precision*: This MATLAB function converts the image I to double precision.
- ***VideoReader*** - *Read video files*: Use a VideoReader object to read files containing video data.

Appendix B. MATLAB codes

Assign5.m:

```
ski = VideoReader('ski_drop_low.mp4');
mon = VideoReader('monte_carlo_low.mp4');
ski_frames = read(ski);
mon_frames = read(mon);

%% ski drop
dt = 1/ski.Framerate;
t = 0:dt:ski.Duration;
numFra = get(ski, 'numberOfFrames');

for i = 1:numFra
    I = rgb2gray(ski_frames(:,:,:,i));
    I = im2double(I);
    X(:,i) = reshape(I,[],1);
end

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

[U,Sig,V] = svd(X1,'econ');
sig = diag(Sig);

figure(1)
plot(sig, 'ko', 'Linewidth', 2)
ylabel('\sigma')
```

```
title('Singular values of each modes')

% trancate to rank r
r = 2;
ur = U(:, 1:r);
sr = Sig(1:r, 1:r);
vr = V(:, 1:r);
S = ur'*X2*vr/sr;
[eV, D] = eig(S);
mu = diag(D);
omega = log(mu)/dt;
phi = ur*eV;
y0 = phi \ X1(:,1);

u_modes = zeros(length(y0), length(t));
for iter = 1:length(t)
    u_modes(:, iter) = y0.*exp(omega*t(iter));
end
u_dmd = phi*u_modes;

bg = u_dmd;
fg = X - abs(u_dmd);
fg = (fg-min(fg))./(max(fg)-min(fg));

figure(2)
imshow(reshape(X(:,200), 540, 960));
figure(3)
imshow(reshape(bg(:,250), 540, 960));
figure(4)
imshow(reshape(fg(:,250), 540, 960));

%% monte carlo
dt = 1/mon.Framerate;
t = 0:dt:mon.Duration;
numFra = get(mon, 'numberOfFrames');

for i = 1:numFra
    I = rgb2gray(mon_frames(:,:,:,i));
    I = im2double(I);
    X(:,i) = reshape(I,[],1);
end

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

[U,Sig,V] = svd(X1,'econ');
sig = diag(Sig);

figure(5)
plot(sig, 'ko', 'Linewidth', 2)
ylabel('\sigma')
title('Singular values of each modes')

% trancate to rank r
r = 3;
ur = U(:, 1:r);
sr = Sig(1:r, 1:r);
vr = V(:, 1:r);
S = ur'*X2*vr/sr;
[eV, D] = eig(S);
mu = diag(D);
omega = log(mu)/dt;
phi = ur*eV;
y0 = phi \ X1(:,1);

u_modes = zeros(length(y0), length(t));
for iter = 1:length(t)
    u_modes(:, iter) = y0.*exp(omega*t(iter));
end
u_dmd = phi*u_modes;

bg = u_dmd;
fg = X - abs(u_dmd(:,end-1));

figure(6)
imshow(reshape(X(:,200), 540, 960));
figure(7)
imshow(reshape(bg(:,50), 540, 960));
figure(8)
imshow(reshape(fg(:,50), 540, 960));
```