AMATH 482 Homework 5

Seoyoung Cho

March 17, 2021

Abstract

Dynamic Mode Decomposition (DMD) is a data-based algorithm to re-express large spatial-temporal datasets with more efficient subspaces. One may find DMD is nothing different from Singular Value Decomposition (SVD). Indeed, DMD utilizes the low-rank SVD representation of data, but it also incorporates eigendecomposition to take account of both space and time information in data. The goal of DMD is to reconstruct the data with a lower dimension, but without knowing the governing equation of the model. In this paper, we explore the use of DMD to separate the video frames into background and foreground images. This will give an idea of reconstructing lower-rank data with only the experimental data.

1 Introduction and Overview

This assignment centers on Dynamic Mode Decomposition (DMD), another powerful method in data analysis to reduce the dimensionality of data. One thing special about this method is that we don't need governing equation of the target model. One example use of DMD that this paper focuses on is background and foreground separation. Given two different videos "monte_carlo_low.mp4" and "ski_drop_low.mp4", the task is to separate the foreground and background videos using DMD. Note that due to run-time issues during the implementation process, we use the low-resolution version of the original videos. The first video shows the cars racing on a track and the second video shows a person skiing down a hill. The background video basically represents the image features that don't vary throughout the whole video and the foreground video represents the moving objects. Hence in our videos, the cars racing and the person skiing would be the foreground features. With the given videos, the data matrix will contain the snapshot of each video frame in a 1-D column vector, so the matrix size will be pixels by frames. Since the DMD takes account of spatial-temporal data, the pixel in a frame represents the space data and each frame represents the time data. In this paper, we discuss the implementation process of applying DMD to the data matrix and the result after separating the background/foreground videos.

2 Theoretical Background

2.1 Dynamic Mode Decomposition (DMD)

Among the wide variety of low-rank matrix decomposition methods in data analysis, Dynamic Mode Decomposition stands a unique ground of data-based algorithm. What this implies is that applying DMD doesn't require the availability of governing equation of the model, but instead focuses on the snapshots of experimental data. Given spatial-temporal data of N spatial points per time snapshot and M time snapshots, DMD turns the problem into linear systems of differential equations by finding the basis of spatial modes and yielding the exponential time dynamics. Given the snapshots of the data, the setup for DMD is as follows:

The time data collected at regularly spaced intervals:

$$t_{m+1} = t_m + \Delta t, \qquad m \in [1, M-1], \qquad \Delta t > 0 \tag{1}$$

The snapshots:

$$U(x, t_m) = \begin{bmatrix} U(x1, t_m) \\ U(x2, t_m) \\ \vdots \\ U(x_n, t_m) \end{bmatrix}, \qquad n \in [1, N] \tag{2}$$

For our assignment, each row of $U(x, t_m)$ is the pixel of the frame at specific video play time. Then for each $m \in [1, M]$, our data matrix takes form as below:

$$X = \begin{bmatrix} U(x,t_1) & U(x,t_2) & \cdots & U(x,t_M) \end{bmatrix} \tag{3}$$

where the columns of X are each frame of the video. We also consider sub-matrix X_j^k which is the time-shifted matrix of columns j through k of the original X. With the above setup, we introduce the concept of the "Koopman Operator" to proceed with DMD. The Koopman Operator simply is a linear, time-independent operator that offers control of nonlinear dynamics without linearization. Assuming the data model is linear, we can apply the Koopman operator A as:

$$x_{j+1} = Ax_j \tag{4}$$

where x_j is the jth column of the data matrix X. The operator simply maps the data from t_j to t_{j+1}. For our assignment, we particularly use the sub-matrices X_1^{M-1} and X_2^M. Then we can re-write X_1^{M-1} using equation 4 as:

$$X_1^{M-1} = \begin{bmatrix} x_1 & Ax_1 & A^2x_1 & \cdots & A^{M-2}x_1 \end{bmatrix} \tag{5}$$

where the columns space is what called the Krylov space. Then, we get X_2^M as:

$$X_2^M = Ax_1^{M-1} \tag{6}$$

Although there is a residual vector to take account of how the last element x_M not being mapped onto Krylov space, we can neglect the vector as we perform SVD since the vector is orthogonal to the principal component basis. For convenience, we notate the two sub-matrices as X_1 and X_2 respectively from now on. Since we are performing data analysis, dimensionality reduction plays a critical role. Hence, we apply SVD on X_1 to make a lower rank approximation. Then substituting X_1 with SVD matrices, we get:

$$\begin{aligned} AX_1 &= X_2 \\ AU\Sigma V^* &= X_2 \\ U^*AU\Sigma V^* &= U^*X_2 \\ U^*AU &= U^*X_2V\Sigma^{-1} =: \tilde{S} \end{aligned} \tag{7}$$

We assume that Σ is invertible by using the economy SVD that removes all the zero singular values. Further, matrices with the same eigenvalues are considered similar matrices because the eigenvalues define the behavior of a matrix. Hence from equation 7, $\tilde{S}$ is a similar matrix of $U * AU$ sharing the same eigenvalues. The eigenvectors will represent the spatial dynamics and the eigenvalues will represent the time dynamics of the spatial-temporal data collected. Then we determine the eigenmodes ψ, by projecting the eigenvectors y_k onto the left singular modes:

$$\begin{aligned} \tilde{S}y_k &= \mu_k y_k \\ \psi_k &= Uy_k \end{aligned} \tag{8}$$

Since we decomposed the data by spatial and temporal dynamics, we now need to reconstruct the data based on DMD eigenmodes and eigenvalues. The DMD solution X_{dmd} at each time frame is given by the following equation:

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k} t = \Psi diag(e^{\omega_k t})b \tag{9}$$

where $\omega_k = ln(\mu_k)/\Delta t$, just another representation of eigenvalues and Ψ is the DMD eigenmode matrix with columns of ψ_k. b is the weight coefficient to match the first time measure. Hence, the equation 9 is simply a multiplication of space and time dynamics. The DMD reconstructed data X_{DMD} will have columns of x_{DMD} for all time frames. For our assignment, such X_{DMD} with the eigenmode corresponding to close to zero eigenvalues will be the background video. This is because the eigenvalue represents the time dynamics. Since the background features don't change over time, there will be no dynamics over time meaning the eigenvalue equal to zero. Hence, X_{DMD} for our assignment will be:

$$X_{DMD} = b_p \psi_p e^{\omega_p t} \tag{10}$$

with zero value eigenvalue as pth element. Taking out the background from the original video, we are only left with the foreground. Hence, the foreground data X_{sparse} will be given by subtracting background data from the original data. The next section will deliver the process of obtaining and separating the background and foreground matrices using DMD.

3 Algorithm Implementation and Development

This section explains the implementation process in MATLAB. For more details of MATLAB functions, refer to Appendix A.

1. Load the video and read in the frames. This gives the 4-D data where the first 2 dimensions represent the pixels of an image, 3rd dimension represent the RGB color plane, and all three by the number of frames.

2. For simplicity, we convert each frame to grayscale. This is to take account of any color reflections or changes throughout the video. For the ski-drop video, crop borders only focus on the landscape.

3. Reshape each frame as a column vector. Define the data matrix with the columns of reshaped frames. This results in a matrix of height$\times$width of the frame by the number of frame size.

4. Define two sub-matrices: X_1, X_2. They are the time-shifted matrices where the column dimension of the sub-matrices will be one less than that of the original matrix.

5. Perform SVD on X_1. Then reduce the dimensionality by mode of choosing. For this assignment, we arbitrarily determine the mode for dimensionality reduction by checking the outcomes with different modes. The reduction mode for Monte Carlo was 3 and for Ski Drop was 20. Figure 1 show the singular values of both Monte Carlo and Ski Drop video matrices.

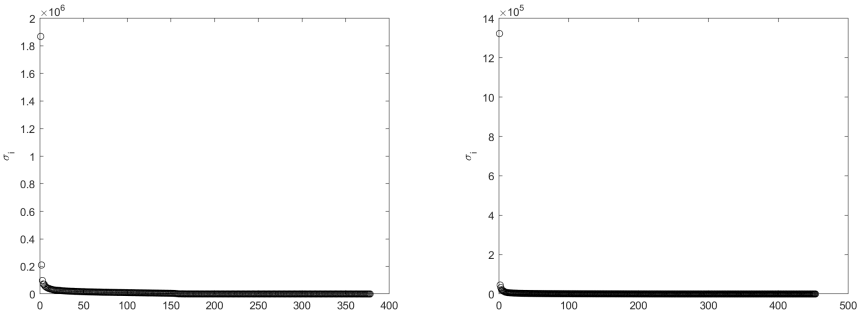
6. Define similar Koopman operator $\tilde{S}$ as from the equation 7.

7. Eigendecompose the matrix $\tilde{S}$. Then find the ω with the eigenvalues.

8. For our assignment, recall that we defined background to be the landscape that doesn't vary much over time. So the eigenvalues corresponding to the background mode are close to zero. Plot the ω values in the complex plane to check the eigenvalues close to zero. Figure 2 shows the ω plots for two videos.

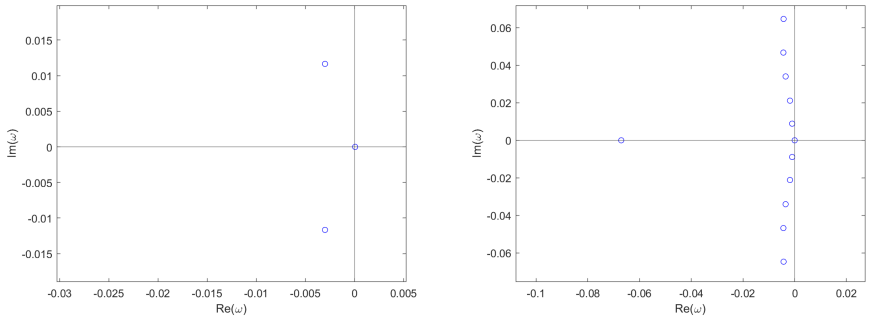
9. Construct the eigenmode Ψ of the corresponding close to zero ω. Multiply Ψ with the pseudo inverse of the first column of X_1 to get the initial condition.

10. For each time frame, calculate equation 9 to get the DMD solution at that time. The overall DMD solution will then be the projection of the solutions onto Φ. The DMD matrix will be the background video.



(a) Monte Carlo X_1 matrix singular values (b) Ski Drop X_1 matrix singular values

Figure 1: Singular value plots



(a) Monte Carlo $\tilde{S}$ matrix ω values (b) Ski drop $\tilde{S}$ matrix ω values

Figure 2: ω complex plane plots

11. Recall the sparse matrix, the foreground data, is computed by subtracting the background data from the full video. To calculate the sparse matrix using only the real values, we take out the absolute value of the DMD matrix. So $X_{sparse} = X - |X_{DMD}|$ However, this may result in some negative values in the sparse matrix which doesn't make sense in the context of video frames. To counter, we simply take out the negative values in the sparse matrix: $X_{sparse} = X_{sparse} - R$. R is the negative value matrix. Then the DMD matrix and the nonnegative sparse matrix will represent the background and foreground of the video respectively.

12. With the foreground and background video matrices, reconstruct the original video by adding them.

4 Computational Results

This section will deliver the computational results obtained after the implementation process.

4.1 Monte Carlo

For the "Monte Carlo" video, we reduced the data to 3 modes and selected the DMD eigenmode that corresponds to the eigenvalue closest to zero. Figure 3 show the snapshot of all original, background, foreground, and reconstructed Monte Carlo video in three different time frame. We can see that the cars are removed in the background videos and that they are shown as the white forms in the foreground video. Further, the background of the foreground videos is black by subtracting the background video which makes the grayscale value to be nonpositive. We compensate with the negative grayscale values by subtracting them out to set the grayscale value to be zeros. The addition of the two background and foreground videos are conveyed in the last row of the figure and we can see the reconstructed video is the same as the original.

4.2 Ski Drop

For the "Ski Drop" video, we reduced the data to 20 modes and again selected the DMD modes that correspond to the DMD eigenvalue closest to zero. The data reduction modes are determined arbitrarily after experimenting with a different number of modes to which gives the best background result. Figure 4 shows the 3 frame snapshots of the original, background, foreground, and reconstructed videos. The foreground of this video is the person skiing and falling snow. For this video, in particular, the foreground features are hard to locate. Hence, we cropped the unnecessary background border and adjusted the contrast to better depict the features. Further, the orange circles point out the location of the person depicted in the original, foreground, and reconstructed video. We can follow the orange marks and see that the background indeed took out the foreground features and the reconstructed video is the same as the original.

5 Summary and Conclusions

For this assignment, we explored one use case of Dynamic Mode Decomposition of video separation. Throughout the process, we were able to sense the prominence of the unique decomposition method of Dynamic Mode Decomposition taking account of both spatial and temporal space. Unlike the other decomposition methods such as SVD and POD, we learned that DMD incorporates the idea of Koopman operator and Krylov space to obtain information about space and time from eigenspace. After comparing the background and foreground frames, we conclude that DMD successfully separated the background/foreground videos in real-time. Likewise proved the solid performance even without the governing equation modeling the movement of foreground objects.

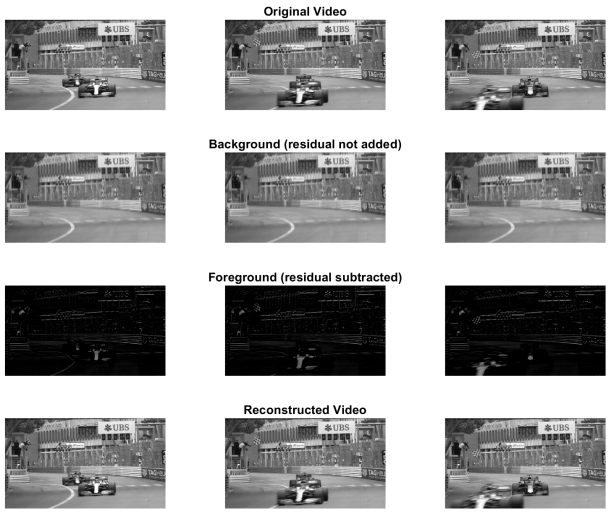


Figure 3: 3 frame snapshots of DMD applied Monte Carlo video

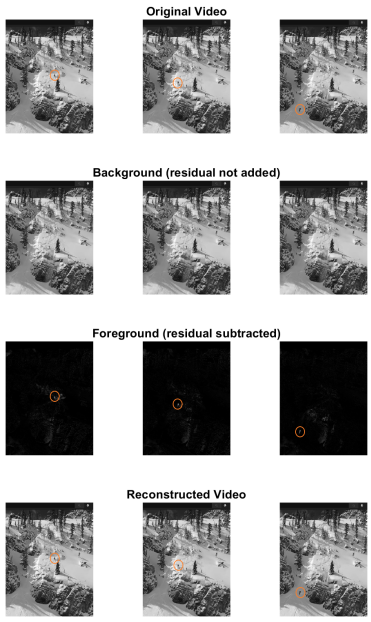


Figure 4: 3 frame snapshots of DMD applied Ski Drop video

Appendix A MATLAB Functions

This section lists all the important MATLAB Functions used in the process of submarine tracking.

- `abs(X)`: returns the absolute value of each element in array `X`.
- `diag(A)`: returns the diagonal entries in matrix `A`.
- `I = find(X)`: returns the indices vector of elements of array `X` that matches given logical argument.
- `length(X)`: returns the length of an array `X`.
- `sum(X)`: returns the sum of the elements in array `X`.
- `cumsum(X)`: returns a vector of the cumulative sum of the elements in array `X`.
- `log(X)`: returns the natural log value of each element in array `X`.
- `exp(X)`: returns the exponential e^x of all the elements in array `X`.
- `X = zeros(sz1,sz2)`: returns all-zero matrix with size of `sz1` by `sz2`.
- `real(z)`: returns the real part of complex number `z`.
- `imag(z)`: returns the imaginary part f complex number `z`.
- `[U,S,V] = svd(A, 'econ')`: performs economy singular value decomposition on matrix `A`. Returns the U,Σ,V matrices.
- `[V, D] = eig(A, B)`: returns the D diagonal matrix of eigenvalues and V with columns of eigenvectors. The results satisfy the equation $A * V = B * V * D$.
- `imshow(I)`: displays grayscale image `I`.
- `centerCropWindow2d(sz, cropsz)`: creates rectancular center cropping window of size `cropsz`.
- `imcrop(I,sz)` crops the images by specified size.
- `v = VideoReader('video file')`: creates object v to read the video data from video file.
- `frame = read(v)`: reads all video frames from the video object v.
- `get(v,'numberOfFrames')`: returns the total number of frames in the video.
- `get(v,'Duration')`: returns the runtime of the video.
- `I = rgb2gray(RGB)`: converts the colored image to grayscale image.
- `reshape(A,sz1,sz2)`: reshapes `A` into `sz1-by-sz2` array.
- `double(s)`: converts the values in s to double precision values.
- `uint8(X)`: converts the values in array `X` to type uint8.

Appendix B MATLAB Code

```
clear; close all; clc;
%% vid 1: monte_carlo
mc_vid = VideoReader('monte_carlo_low.mp4');
mc_fr = read(mc_vid);
numFr = get(mc_vid,'numberOfFrames'); runtime = get(mc_vid,'Duration');

%% convert to grayscale -> video matrx SVD

for i=1:numFr
    mc_gray = rgb2gray(mc_fr(:,:,:,i));
    mc_mat(:,i) = double(reshape(mc_gray, [], 1));
end
dt = runtime/numFr;
%%
% SVD
X1 = mc_mat(:,1:end-1); X2 = mc_mat(:, 2:end);
[U, Sig, V] = svd(X1, 'econ');

%% plot singular values

plot(diag(Sig), 'ko')
ylabel('\sigma_i')

%%
mode = 3;
U = U(:,1:mode); Sig = Sig(1:mode, 1:mode); V = V(:,1:mode);
S = U'*X2*V/Sig;

%% eigendecomp
[eV, D] = eig(S);
mu = diag(D);
omega = log(mu)/dt;

%% plot omega
line = 15:15;
plot(zeros(length(line),1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(length(line),1),'k','Linewidth',2) % real axis
hold on
plot(real(omega)*dt, imag(omega)*dt, 'bo', 'Markersize' ,5)
hold on
yline(0);
hold on
xline(0);
xlim([-1 0.1]); ylim([-0.6 0.6]);
xlabel('Re(\omega)')
ylabel('Im(\omega)')

%% separate foreground and background
thresh = 0.01;
bg_ind = find(abs(omega) < thresh);
omega = omega(bg_ind);
```

```
Phi = U*eV;
Phi = Phi(:,bg_ind);
y0 = Phi\X1(:,1);
%%
sz = size(X1,2);
t = dt*(0:sz-1);

u_modes = zeros(length(y0), length(t));
for i=1:length(t)-1
    u_modes(:,i) = y0.*exp(omega*t(i));
end
Xdmd = Phi*u_modes; % background video

%% sparse, background, foreground matrix
Xsp = X1 - abs(Xdmd);
R = Xsp.*(Xsp<0);
Xbg = R + abs(Xdmd);
Xfg = Xsp - R;

Xre = Xbg + Xfg;

%% plot

figure()
subplot(4,3,1)
imshow(uint8(reshape(X1(:,50), 540,[])))
subplot(4,3,2)
imshow(uint8(reshape(X1(:,65), 540,[])))
title('Original Video', 'FontSize', 11)
subplot(4,3,3)
imshow(uint8(reshape(X1(:,80), 540,[])))
subplot(4,3,4)
imshow(uint8(reshape(Xdmd(:,50), 540,[])))
subplot(4,3,5)
imshow(uint8(reshape(Xdmd(:,65), 540,[])))
title('Background (residual not added)', 'FontSize', 11)
subplot(4,3,6)
imshow(uint8(reshape(Xdmd(:,80), 540,[])))
subplot(4,3,7)
imshow(uint8(reshape(Xfg(:,50), 540,[])))
subplot(4,3,8)
imshow(uint8(reshape(Xfg(:,65), 540,[])))
title('Foreground (residual subtracted)', 'FontSize', 11)
subplot(4,3,9)
imshow(uint8(reshape(Xfg(:,80), 540,[])))
subplot(4,3,10)
imshow(uint8(reshape(Xre(:,50), 540,[])))
subplot(4,3,11)
imshow(uint8(reshape(Xre(:,65), 540,[])))
title('Reconstructed Video', 'FontSize', 11)
subplot(4,3,12)
imshow(uint8(reshape(Xre(:,80), 540,[])))

%%
clear; close all; clc;
```

```
%% vid 2: ski_drop
ski_vid = VideoReader('ski_drop_low.mp4');
ski_fr = read(ski_vid);
numFr = get(ski_vid,'numberOfFrames'); runtime = get(ski_vid,'Duration');

% convert to grayscale -> video matrx SVD
cropsz = [520 400]; % crop size
for i=1:numFr
    ski_gray = rgb2gray(ski_fr(:,:,:,i));
    window = centerCropWindow2d(size(ski_gray), cropsz);
    ski_gray = imcrop(ski_gray,window);
    %imshow(ski_gray)
    ski_mat(:,i) = double(reshape(ski_gray, [], 1));
end
dt = runtime/numFr;

% SVD
X1 = ski_mat(:,1:end-1); X2 = ski_mat(:, 2:end);
[U, Sig, V] = svd(X1, 'econ');
mode = 20;

%% plot singular values

plot(diag(Sig), 'ko')
ylabel('\sigma_i')

%%
U = U(:,1:mode); Sig = Sig(1:mode, 1:mode); V = V(:,1:mode);
S = U'*X2*V*diag(1./diag(Sig));

% eigendecomp
[eV, D] = eig(S);
mu = diag(D);
omega = log(mu)/dt;

%% plot omega
line = 15:15;
plot(zeros(length(line),1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(length(line),1),'k','Linewidth',2) % real axis
hold on
plot(real(omega)*dt, imag(omega)*dt, 'bo', 'Markersize' ,5)
hold on
yline(0);
hold on
xline(0);
xlim([-1 0.1]); ylim([-0.6 0.6]);
xlabel('Re(\omega)')
ylabel('Im(\omega)')

%% separate foreground and background
thresh = 0.01;
bg_ind = find(abs(omega) < thresh);
omega = omega(bg_ind);
Phi = U*eV;
```

```
Phi = Phi(:,bg_ind);
y0 = Phi\X1(:,1);
%%
sz = size(X1,2);
t = dt*(0:sz-1);

u_modes = zeros(length(y0), length(t));
for i=1:length(t)-1
    u_modes(:,i) = y0.*exp(omega*t(i));
end
Xdmd = Phi*u_modes; % background video

%% sparse, background, foreground matrix
Xsp = X1 - abs(Xdmd);
R = Xsp.*(Xsp<0);
Xbg = R + abs(Xdmd);
Xfg = Xsp - R;
Xre = Xbg + Xfg;
Xfg = Xfg - R;
%% plot

figure()
subplot(4,3,1)
imshow(uint8(reshape(ski_mat(:,100), 520,[])))
subplot(4,3,2)
imshow(uint8(reshape(ski_mat(:,200), 520,[])))
title('Original Video', 'FontSize', 11)
subplot(4,3,3)
imshow(uint8(reshape(ski_mat(:,300), 520,[])))
subplot(4,3,4)
imshow(uint8(reshape(Xdmd(:,100), 520,[])))
subplot(4,3,5)
imshow(uint8(reshape(Xdmd(:,200), 520,[])))
title('Background (residual not added)', 'FontSize', 11)
subplot(4,3,6)
imshow(uint8(reshape(Xdmd(:,300), 520,[])))
subplot(4,3,7)
imshow(uint8(reshape(Xfg(:,100), 520,[])))
subplot(4,3,8)
imshow(uint8(reshape(Xfg(:,200), 520,[])))
title('Foreground (residual subtracted)', 'FontSize', 11)
subplot(4,3,9)
imshow(uint8(reshape(Xfg(:,300), 520,[])))
subplot(4,3,10)
imshow(uint8(reshape(Xre(:,100), 520,[])))
subplot(4,3,11)
imshow(uint8(reshape(Xre(:,200), 520,[])))
title('Reconstructed Video', 'FontSize', 11)
subplot(4,3,12)
imshow(uint8(reshape(Xre(:,300), 520,[])))
```