AMATH 482 Assignment 5

Megan Freer

March 17, 2021

Abstract

Dynamic Mode Decomposition (DMD) is a powerful mathematical tool that allows us to produce a linear model that governs our data snapshots, even if the system is nonlinear, thus making it easier to perform computations. DMD allows us to predict data into the future, as well as to separate out the background and foreground of videos. An interesting test case for the application of DMD is separating out the background and foregrounds of videos with a constant background and movement in the foreground, such as a skier going down a slope and two cars going around a racetrack.

1 Introduction and Overview

Dynamic Mode Decomposition (DMD) is a useful mathematical tool for dimensionality reduction. The DMD algorithm takes advantage of low dimensionality in the data to create a low-rank approximation of the linear mapping that best iterates you through the snapshots of the data. DMD is completely data-driven, meaning that it does not require governing equations. DMD is particularly useful for allowing us to forecast our data, or predict what it will be in the future.

As an example of using DMD, we will be separating out the foreground and background of two videos: one of a skier skiing down a slope and one of cars moving around a race track. The DMD can be used to recreate the videos with one version with just the foreground present (either the skier or cars) or a new video with just the background present (ski slopes or race track, which do not change). This example allows us to explore how DMD can be used to separate out the foreground and background of videos by reducing dimensionality.

We will be using MATLAB to perform the Dynamic Mode Decomposition on the video data. This then allows us to create low rank and sparse matrices that represent the background and foreground of the videos. This will allow us to explore the use of DMD through this example.

2 Theoretical Background

Dynamic Mode Decomposition (DMD) is a relatively new technique that aims to take advantage of low-dimensionality in experimental data without having to rely on a given set of governing equations, and is especially useful for forecasting our data. DMD finds a basis of spatial modes (not necessarily orthogonal like Proper Orthogonal Decomposition, or POD) for which the time dynamics are just exponential functions (with potentially complex exponents). Therefore the only things happening in time are oscillations, decay, and/or growth. DMD essentially turns a problem into a linear systems of differential equations. DMD is completely data-driven, which means that it does not require governing equations, but rather only data snapshots to apply the algorithm.

Setting up DMD, we are going to assume that we have snapshots of spatio-temporal data, meaning that we have data that evolves in both space and time. We will take N to be the number of spatial points saved per unit time snapshot, and M to be the number of snapshots taken. One important thing to note is that the time data should be collected at regularly spaced intervals:

$$t_{m+1} = t_m + \Delta t, \quad m = 1, ..., M-1, \quad \Delta t > 0 \tag{1}$$

The snapshots are then denoted as:

$$U(x,t_m) = \begin{bmatrix} U(x_1,t_m) \\ U(x_2,t_m) \\ \vdots \\ U(x_n,t_m) \end{bmatrix} \tag{2}$$

for each m=1,...,M. We can use these snapshots to form columns of data matrices:

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

which can also be shortened to be just columns j through k of the full snapshot matrix X, as denoted by X_j^k.

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

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

where the j indicates the specific data collection time and A is the linear operator that maps the data from t_j to t_{j+1}. The vector x_j is an N-dimensional vector of the data points collected at time j. In other words, applying A to a snapshot of data will advance it forward in time by Δt. This is essentially linearly mapping from one time step to the next, even if the dynamics of the system are nonlinear.

The algorithm for DMD involves first sampling data at N prescribed locations M times. The data snapshots should be evenly spaced in time by a fixed Δt. These snapshots can then be used to create the matrix X. From the data matrix X, we can next construct the two submatrices X_1^{M-1} and X_2^M, which can be defined as:

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

$$X_2^M = AX_1^{M-1} + re_{M-1}^T \tag{6}$$

where e_{M-1} is the vector with all zeros except a 1 at the (M-1)st component and r is a residual (or error) vector to account for the fact that the final point x_M was not included in our Krylov basis (which is defined by the columns of X). Also worth noting is that A is unknown at this point and our goal is to find it. The next step is to compute the SVD of X_1^{M-1}. The SVD is defined as:

$$X_1^{M-1} = U\Sigma V^* \tag{7}$$

Next, create the matrix $\tilde{S}$ and find its eigenvalues and eigenvectors. $\tilde{S}$ is defined as:

$$\tilde{S} = U^* X_2^M V \Sigma^{-1} \tag{8}$$

Next, use the initial snapshot x_1 and the pseudoinverse of Ψ to find the coefficients b_k, as defined by:

$$b = \Psi^\dagger x_1 \tag{9}$$

where $\Psi^\dagger$ is the pseudoinverse of the matrix Ψ. Finally, compute the solution at any future time using the DMD modes along with their projection to the initial conditions and the time dynamics computed using the eigenvalues of $\tilde{S}$. In order to describe continual multiplications by A, we can expand our eigenbasis to get:

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t}) b \tag{10}$$

Specifically in our assignment, the DMD spectrum of frequencies can be used to subtract background modes. We are assuming that ω_p, where $p \in \{1, 2, \dots, l\}$ satisfies $||\omega_p|| \approx 0$, and that $||\omega_j|| \forall j \neq p$ is bounded away from zero. Thus,

$$X_{DMD} = b_p \varphi_p e^{\omega_p t} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t} \tag{11}$$

where the left side of the sum represents the background video and the right side of the sum represents the foreground video. It should also be true that:

$$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse} \tag{12}$$

where $X_{DMD}^{Low-Rank}$ represents the background video and X_{DMD}^{Sparse} represents the foreground video. This equation rearranged gives us:

$$X_{DMD}^{Sparse} = X - \left|X_{DMD}^{Low-Rank}\right| \tag{13}$$

However, this result may give some negative values in X_{DMD}^{Sparse}, which is not a realistic number for a pixel intensity. Instead, these residual negative values can be put into a $n \times m$ matrix R and then be added back into $X_{DMD}^{Low-Rank}$ as follows:

$$X_{DMD}^{Low-Rank} \leftarrow R + \left|X_{DMD}^{Low-Rank}\right| \tag{14}$$

$$X_{DMD}^{Sparse} \leftarrow X_{DMD}^{Sparse} - R \tag{15}$$

This ensures that the approximate low-rank and sparse DMD reconstructions are real-valued.

3 Algorithm Implementation and Development

Algorithms implementing DMD were used to create videos of the foreground and background of videos of a skier going down a slope, as well as cars going around a race track. To start, the code loads in the specific video that is being analyzed. The user can choose which video they want to analyze and uncomment the corresponding line reading in the video. Next, the code defines the height and width of each frame, as well as the number of frames to be analyzed. Since my computer does not have enough storage/RAM to run the program with even the low quality videos, my program only analyzes the first 1/4 of each video but still produces high-quality results. The next step is to make each of the frames into a grey scale image, allowing the video to be narrowed from 4 dimensions to 3 dimensions. Next, the black and white 3D video data is reshaped into a 2D matrix, with each column representing each frame. Next, the 2D black and white video data is converted to double precision.

The next step is to find dt and the time vector, with dt defined as 1 over the frame rate that was read in when reading in the video. The time vector goes from 0 to the number of frames being analyzed - 1, in intervals of dt. The algorithm then creates the DMD matrices X1 and X2 (or X_1^{M-1} and X_2^M) from the 2D black and white video matrix. Next, a Singular Value Decomposition (SVD) is performed on X1/X_1^{M-1}. The code then creates a scatter plot representing the singular value spectrum and a bar chart representing the rank energies for modes 1 to 20, as the vast majority of the the information in the system is contained in the early modes. The code then narrows down U, Σ, and V to only include 2 modes, based on the fact that the first 2 modes contain the overwhelming majority of the energy in the system.

Next, the algorithm computes $\tilde{S}$ according to Equation 8, then computes the eigenvalues and eigenvectors of $\tilde{S}$. The algorithm extracts a vector of the eigenvalues that can be used to convert to the `omega` vector, which is just the eigenvalues in another form. The reason for the different form is that we want to make the DMD solution look more like a solution to a continuous time ODE. With that in mind, we find the `Phi` vector, which contains the eigenvectors of the matrix `A`, which we obtain by multiplying the eigenvectors of $\tilde{S}$ against `U`.

The next step is to create the DMD solution. The algorithm then finds `y0` and computes x_{DMD} according to Equation 10. Next, we create the sparse and low-rank matrices, with the sparse matrix defined by Equation 13. We then found the R matrix for the residuals negative values. Next, we found the minimum of the sparse data, which will be used later to make the foreground stand out more, particularly in the skiing video. We then adjusted the low rank matrix to add in the residual matrix, as defined by Equation 14. Next, we add the minimum value previously calculated onto the sparse matrix. The 2D sparse and low rank matrices are then reshaped back into 3D matrices for the frames over time that make up each video. The background is represented by the low-rank matrix, while the foreground is represented by the sparse matrix. Finally, the algorithm plots frame number 35 for the background and the foreground, as well as videos looping through how the background and foreground change from frame to frame.

4 Computational Results

We performed our Dynamic Mode Decomposition (DMD) algorithm on both the video of the person skiing, and the video of the car race. One interesting and informative result was looking at the singular value spectrum for each video, as seen in Figure 1 for the skiing video and Figure 2 for the car race video. For both videos, there is a sharp drop-off in singular values after the first mode, meaning that the majority of the important information is stored in the first mode. Looking at the energies in these same figures, the energy values represents how much energy from the full system is contained in the n-th mode/the energy contained in the rank-n approximation and provides another visualization of how many modes are needed to represent our data well, but also reduce dimensionality. It appears that only 1 to 2 modes is necessary to represent the system well. For the skiing video, we went from an energy of 99.91% to an energy of 99.94% from rank 1 to rank 2. For the car video, we went from an energy of 98.55% to an energy of 99.08% from rank 1 to rank 2. Therefore we decided to use 2 modes when analyzing our videos. While the rank can be approximated to be 1, performing analysis with 2 modes allows for a slightly higher amount of information contained, without a large increase in computations needed.

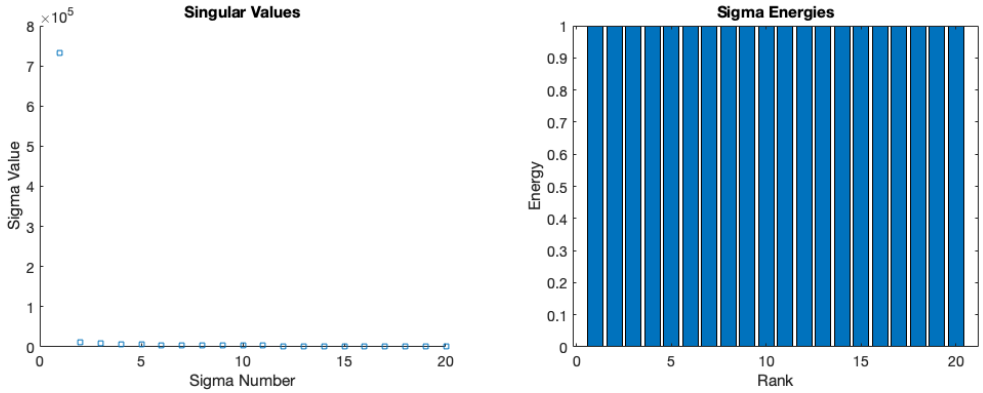

Figure 1: Skiing Video: The figure on the left is the σ values (nonzero singular values) for each σ_n. The figure on the right is the cumulative energy accounted for by each rank. The higher the energy, the more of the energy of the system that is accounted for by that rank approximation. We are estimating our system to have rank 1.

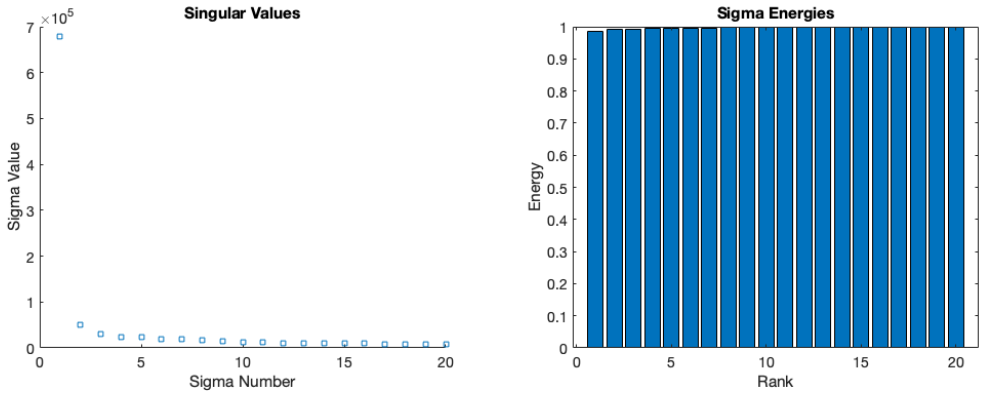

Figure 2: Cars Video: The figure on the left is the σ values (nonzero singular values) for each σ_n. The figure on the right is the cumulative energy accounted for by each rank. The higher the energy, the more of the energy of the system that is accounted for by that rank approximation. We are estimating our system to have rank 1.

Figure 3: Skiing Video: Reconstructed background and foreground at frame 35.

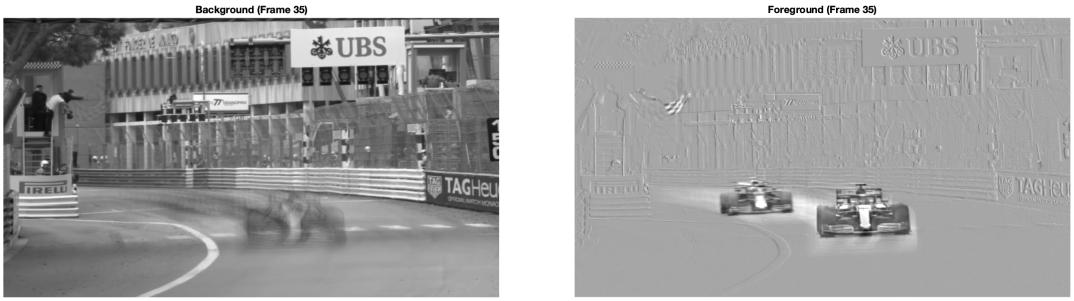

Figure 4: Cars Video: Reconstructed background and foreground at frame 35.

Next, we computed the background and foreground for each video, as can be seen in Figure 3 for the skiing video and Figure 4 for the car race video. These figures contain what the video looks like at Frame 35, shortly after the video has started. In the skiing background video, we can see that the background contains the snowy scenery, but the position of the skier is not obvious, especially when watching the background in video format. In the skiing foreground video, we can see the skier in black standing out from the plain grey background. While the skier is small and only appears as a small black figure in the frame represented here, the movement of the skier in the foreground is obvious when played as a video. The car video produces similar results, with the background as the raceway, with only a slight leftover blurred part of the car. The car race foreground is very clearly the moving cars in black, with the rest of the background in a light grey.

Overall, Dynamic Mode Decomposition is able to accurately separate out the foreground and background of videos with movement.

5 Summary and Conclusions

In conclusion, Dynamic Mode Composition (DMD) is a useful tool, particularly for separating out the background and foreground of a video. This mathematical tool was able to help us accurately distinguish and create videos of the background and foreground of videos of a skier going down a slope and cars going around a racetrack. We were also able to demonstrate how MATLAB can be used as a way to implement these algorithms. The ideas behind DMD can be applied to many other applications in order to predict future data and distinguish between the background and foreground of videos with reduced dimensionality.

Appendix A MATLAB Functions

The following MATLAB functions are implemented in this code:

- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `video = read(v)` reads all video frames from the file associated with v.
- `szdim = size(A,dim)` returns the length of dimension dim in the matrix A.
- `Y = round(X)` rounds each element of X to the nearest integer. In the case of a tie, the round function rounds away from zero to the integer with larger magnitude.
- `X = zeros(sz1,...,szN)` returns an sz1-by-...-by-szN matrix of zeros.
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I.
- `B = reshape(A, [sz1,...,szN])` reshapes A into a sz1-by-...-by-szN array where szN indicates the size of each dimension.
- `I2 = im2double(I)` converts the image I to double precision.
- `[U,S,V] = svd(X, 'econ')` produces an economy-size singular value decomposition of matrix X in order to find the matrices that evaluate to X=U*S*V'.
- `sig = diag(S)` returns a column vector of the main diagonal elements of S.
- `scatter(x,y)` creates a scatter plot with circles at the locations specified by the vectors x and y.
- `sum(A)` returns the sum of the elements of A along the first array dimension whose size does not equal 1.
- `bar(x,y)` creates a bar graph with one bar for each element in y, with the bars drawn at the locations specified by x.
- `[V, D] = eig(A)` returns a diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that A*V=V*D.
- `length(obj)` returns the length of an object, such as the length of an array.
- `abs(X)` returns the absolute value of each element in array X.
- `min(A)` returns the minimum elements of an aray. If A is a matrix, then min(A) is a row vector containing the minimum value of each column. If A is a vector, then min(A) returns the minimum of A.
- `imshow(I)` displays the grayscale image I in a figure.

Appendix B MATLAB Code

```
% Megan Freer
% AMATH 482
% Assignment 5

clear all; close all; clc;

% Uncomment below to analyze the other video
v = VideoReader('ski_drop_low.mp4');
% v = VideoReader('monte_carlo_low.mp4');

% Read all video frames from file associated with v
frames = read(v);
height = size(frames, 1);
```

```
width = size(frames, 2);
timepoints = round(size(frames, 4) / 4); % downsample bc computer issues

% Make matrix to save black and white frames in
bw_frames = zeros(height, width, timepoints);

% Loop through and make black and white
for i = 1:timepoints
    current_frame = frames(:,:,:,i);
    bw_frame = rgb2gray(current_frame);
    bw_frames(:, :, i) = bw_frame;
end

% Reshape each frame into a column
bw_frames_2d = reshape(bw_frames, [height*width, timepoints]);

% Converts the image to double precision
bw_frames_2d = im2double(bw_frames_2d);

% Find dt and time vector
dt = 1/v.FrameRate;
t = 0:dt:timepoints-1;

% Create DMD matrices
X1 = bw_frames_2d(:,1:end-1);
X2 = bw_frames_2d(:,2:end);

% Perform SVD of X1
[U, Sigma, V] = svd(X1,'econ');

% Finding sigma values from S matrix
sigmas = diag(Sigma);
num_modes_plotting = 20;

% Plot sigmas
figure()
scatter(1:num_modes_plotting, sigmas(1:num_modes_plotting), 's')
title('Singular Values', 'FontSize', 18)
xlabel('Sigma Number', 'FontSize', 18)
ylabel('Sigma Value', 'FontSize', 18)
set(gca, 'FontSize', 14)

% Calculate energy for first 20 rank/sigma
energies = zeros(1, num_modes_plotting);
for i=1:num_modes_plotting
    energies(i) = sum(sigmas(1:i).^2)/sum(sigmas.^2);
end

% Plot energies
figure()
bar(1:num_modes_plotting, energies)
ylim([0 1])
title('Sigma Energies', 'FontSize', 18)
xlabel('Rank', 'FontSize', 18)
```

```
ylabel('Energy', 'FontSize', 18)
set(gca, 'FontSize', 14)

% Choose smaller number of modes
modes = 2;
U_small = U(:, 1:modes);
Sigma_small = Sigma(1:modes, 1:modes);
V_small = V(:, 1:modes);

% Perform computation of S~
S = U_small'*X2*V_small*diag(1./diag(Sigma_small));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues

omega = log(mu)/dt;
Phi = U_small*eV;

% Create DMD Solution
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),timepoints - 1);
for iter = 1:timepoints - 1
   u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;

% Create sparse and low-rank matrixes
X_DMD_sparse = X1 - abs(u_dmd);
R = X_DMD_sparse .* (X_DMD_sparse < 0); % residual negative values matrix

% Find the minimum value in order to add into the sparse data later
min_value = abs(min(min(X_DMD_sparse)));

% Use residual negative values matrix
X_DMD_low_rank = R + abs(u_dmd); % background
X_DMD_sparse = X_DMD_sparse + min_value; % suggestion instead of subtracting R

% Create background and foreground videos
background = reshape(u_dmd, [height, width, timepoints-1]);
foreground = reshape(X_DMD_sparse, [height, width, timepoints-1]);

% Plot a resulting video frame
figure()
imshow(uint8(background(:,:,35)))
title('Background (Frame 35)', 'FontSize', 20)

figure()
imshow(uint8(foreground(:,:,35)))
title('Foreground (Frame 35)', 'FontSize', 20)

% Play videos
figure()
for i = 1:timepoints - 1
    imshow(uint8(foreground(:,:,i)))
    title('Foreground Video', 'FontSize', 20)
```

```
    hold off
    drawnow
end

figure()
for i = 1:timepoints - 1
    imshow(uint8(background(:,:,i)))
    title('Background Video', 'FontSize', 20)
    hold off
    drawnow
end
```