Amath 482 Homework 5

USE OF DYNAMIC MODE DECOMPOSITION IN VIDEO PROCESSING

MICHAEL GABALIS

Abstract: This paper aims to demonstrate the usefulness of Dynamic Mode Decomposition (DMD) for decomposing a system into a more fundamental set of components. The components od DMD allow us to study the dynamics of a system over time and make predictions about its future state. In this paper, we will use these properties of the DMD to separate a video into two components: a high frequency moving subject and a low frequency stationary background. The case study will be a video of two cars racing in the Monte Carlo, and a skier going down a mountain slope at high speed.

University of Washington

March 2021

1 INTRODUCTION AND OVERVIEW

In our definition, a dynamical system is one in which the dynamics are dictated by some fundamental set of equations, often including nonlinearities, that rely on some subset of the variables in the system. While quite useful, there are many systems where the governing equations are unknown. For these, a data-driven approximation of the dynamics is required to further explore the system and infer upon its fundamental structure. To begin, the Singular Value Decomposition (SVD) can provide a set of principal components that describe the patterns in the data. However, it does so without spatial-temporal patterns, thus the components found in the returned basis may not be the exact individual signals the make up the input rather some combination of them. Additionally, SVD is also restricted by the fact that these components must be orthogonal. To get around these flows we can use the Dynamic Mode Decomposition (DMD), that seeks to capture the low-rank structure of a given system. By providing a new, chosen coordinate system to map onto, the DMD breaks down a system into a set of fundamental components that provide the underlying spatial-temporal properties of the system. In addition, the DMD doesn't require the modes to be orthogonal, so it may extract more information about the fundamental signals. Overall, the DMD is a perfect tool for performing a linear approximation to the dynamics of the system. In this paper we will use the DMD to extract two spatial-temporal components of a video: the foreground and the stationary background.

2 THEORETICAL BACKGROUND

2.1 SINGULAR VALUE DECOMPOSITION

The Singular Value Decomposition is a diagonalization of a given matrix A that focuses on the rotations or reflections and stretching a vector undergoes when it is transformed by A. For any $A \in \mathbb{C}^{m\times n}$ there exists a diagonalization of the form:

$$A = U\Sigma V^* \tag{1}$$

Where $U \in C^{m\times n}$ and $V \in C^{n\times n}$ are unitary matrices, $\Sigma \in R_{\geq 0}^{m\times n}$ is a diagonal matrix, and V^* is the complex conjugate transpose of V. As U and V are unitary matrices, their columns form an orthonormal basis for their respective space and the matrices have the properties of $UU^* = I_m$ and $VV^* = V^*V = I_n$. The diagonal entries of Σ are called the singular values of A. These singular values and their corresponding columns in U are the left singular vectors and the columns in V are the right singular vectors. These are often arranged within Σ such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. This breaks down the transformation of AX into three different fundamental transformations: a rotation or reflection of V transpose within the domain of X, a scaling of the components the space X by Σ , and another rotation or reflection of the space within the codomain of A.

$$X = \sum_{i=1}^{min(m,n)} u_j \sigma_j v_j^* \tag{2}$$

where u_j , σ_j , v_j^* are the jth column of each SVD component. Since the singular values are a non-increasing sequence, we see that each successive term is increasingly less important to the structure of X. Thus, if the singular values decrease rapidly, we can get an accurate representation of X with only the first few terms of the sum. This is a low-rank approximation of the matrix X. The idea of PCA is that these first few terms form a good approximation to the data set

because they represent the fundamental dynamics of the system. These principal components are the columns of U with the largest singular value and the direction U in which our data has the highest variance. This is likely to still capture the dynamics of our system while leaving out the noise or redundancy.

Another valuable tool to look at is the energy of our system. When finding the principal components, we can measure how much energy our low-rank approximation yields. This can measure whether or not we are accurately approximating the data of dimension n with a rank r matrix:

$$energy = \frac{\sigma_1^2 + \cdots + \sigma_{\mathrm{n}}^2}{\sigma_1^2 + \cdots + \sigma_{\mathrm{r}}^2} \tag{3}$$

2.2 Dynamic Mode Decomposition

The goal of DMD is to describe a dynamical system in a data-driven way where the governing equations are often unknown and likely to contain nonlinearities. Our goal is to reconstruct the dynamical system by finding parameters that provide the best linear approximation of the system. We can begin by writing our function as a function of the state variable x_k:

$$\mathrm{y\_k} = \mathrm{g(x\_k)} \tag{4}$$

x_k is our state variable of interest and our measurements y_k measures these states. We can rewrite this as a set of linear equations when y_k = x_k. Defining them as:

$$\mathrm{X} = [\mathrm{x\_1}, \mathrm{x\_2}, \ldots, \mathrm{x\_\{n\text{-}1\}}, \mathrm{X'} = [\mathrm{x\_2}, \mathrm{x\_3}, \ldots, \mathrm{x\_n}] \tag{5}$$

$$\frac{dx}{dt} = Ax \rightarrow AX \approx X' \rightarrow A \approx X'X^{\dagger} \tag{6}$$

Dynamic Mode Decomposition begins by developing the best linear approximation to the system, as we see in **equation 5 and 6 above.** By arranging the state of our system at each point in time as the columns of X and X' , we can stagger the two matrices to encode the transition time between the states $\mathrm{x\_i \text{ and } x\_\{k+1\}}$. The matrix A will be the best linear approximation to the dynamics of our system as it explains as much of the transition to the next state as linearly possible. By solving the solution given in **equation 6**, we get:

$$x(t) = \sum_{\mathrm{k=1}}^{\mathrm{n}} b_k \phi_k e^{\omega_k t} \tag{7}$$

where ϕ_k and ω_k are the eigenvectors and eigenvalues of the matrix A, and b_k is the coordinates of x(0) in the eigenvector basis. The DMD approximates modes of the Koopman operator, a linear, infinite-dimensional

operator that is a nonlinear dynamical system on the Hilbert space of measurement functions of the states. This Koopman operator advances measurements along with the flow map as shown below:

$$Kg = g \circ F \rightarrow K\mathrm{g}(\boldsymbol{x_k}) = g\big(\boldsymbol{F}(\boldsymbol{x_k})\big) = g(\boldsymbol{x_{k+1}}) \tag{8}$$

To work in a better coordinate system, we can project our data onto the principal components obtained by the SVD. From the normal SVD we write:

$$A = X'X^{\dagger} = X'V\Sigma^{-1}U^{*} \tag{9}$$

A has the potential to be a very large matrix from the construction so we take the low rank approximation of X, projecting A onto the first r principal components to make a new matrix called $\tilde{A}$.

$$\tilde{A} = U_r^{*}X'V_r\Sigma_r^{-1} \tag{10}$$

We can now construct the eigen decomposition by creating a matrix where the columns represent exact DMD modes with our low-rank approximation:

$$\widetilde{\Phi} = X'V_r\Sigma_r^{-1}\mathrm{W} \tag{11}$$

As we can see from **equation 7,** eigenvalues with positive real parts cause the system to drastically change in time. In these situations, the model poorly predicts future events outside the time-frame of the data and some kind of data transformation is required.

We can use the DMD spectrum of frequencies to subtract background modes. We create two data frames from our initial: One from the beginning to the last -1 frame, and one from the second to last frame. Each of these represent a data snapshot: the data matrix X, and the shifted data matrix X'. We use the first data frame of our SVD and corresponding analysis. We assume that the norm of ω_p where p is a positive integer has a second norm close to zero giving us the following:

$$\mathrm{X\_\{DMD\}} = b_{\mathrm{p}}\phi_{\mathrm{p}}e^{\omega_{\mathrm{p}}t} + \sum_{\mathrm{j}\neq\mathrm{p}}^{\mathrm{n}} b_{\mathrm{j}}\phi_{\mathrm{j}}e^{\omega_{\mathrm{j}}t} \tag{12}$$

This will produce a DMD matrix that is the same dimensions as our original X matrix, although each term is complex. We will solve for each of the parameters using the same process as before. The first part is the background video or low-rank DMD approximation, and the second part will be the foreground video or the sparse DMD approximation. We know that X will be the summation of the low-rank DMD matrix and the sparse DMD matrix, so we can solve for the sparse matrix. Since we subtracted the low-rank approximation from our initial data to obtain our sparse approximation, we may obtain negative values in the sparse matrix. To counteract this we will put the negative values into a residual matrix R so our matrices look as follows:

$$X_{DMD}^{Low-rank} \leftarrow R + \left|X_{DMD}^{Low-rank}\right| \quad (13)$$

$$X_{DMD}^{sparse} \leftarrow X_{DMD}^{sparse} - R$$

This allows us to account for the magnitudes of complex values from the DMD reconstruction, while preventing pixel intensities from being below zero, making sure that both approximate low-rank and sparse DMD reconstruction are real-valued and valid.

3 Algorithm Implementation and Development

The code is broken up into two major parts: applying the SVD and analyzing the principal components, then using different classifying techniques on the test data.

3.1 Preprocessing

1. First we will load the video chosen and pre-allocate memory for storing frame (Appendix B: code 9-10).
2. From here we will reduce the size of the matrix by reducing the resolution and skipping every other frame. Then iterate through each frame of the video, converting each frame to black and white, reshaping them into column vectors, then storing the frames into our matrix X (Appendix B: 12-34).
3. The pixel values are inverted to provide better performance (Appendix B: 28).

3.2 DMD

4. DMD will be performed on the data matrix X by first creating the staggered matrices X1 and X2 where the first iteration of columns from X1 equal the next iteration of columns from X2. We will then take the SVD of X1 (Appendix B: 36-42)
5. The singular values of X1 are then plotted, where we will choose a low-rank approximation that captures a good percentage of the energy in the system (Appendix B: 43-52).
6. Using the low-rank approximation we will calculate $\tilde{A}$. From $\tilde{A}$ we will solve for the relevant eigenvalues and eigenvectors of the matrix as well as the initial conditions and the frequencies (Appendix B: 55-64).
7. We then need to minimize the frequency to find the low frequency for the stationary background (Appendix B: 66-67).
8. Calculate the low-rank DMD matrix and the sparse DMD matrix using the parameters we solved for (Appendix: 73-74).
9. The sparse DMD matrix may have negative entries, which correspond to unrealistic pixel values, so we generate a matrix R to contain these negative entries. We will then add this matrix R to the absolute value of the low-rank DMD matrix and subtract it off the sparse DMD matrix (Appendix B: 78-79).

3.3 Generating the Plots

10. Invert the pixel values of the low rank and sparse DMD matrices to get back into our original space (Appendix B: 87-89)
11. Plot the corresponding low-rank and sparse DMD deconstructions of the data matrix X (Appendix B: 91-99).

4 COMPUTATIONAL RESULTS

The first video that we tried the DMD technique on was a clip of multiple formula one cars driving towards the camera on the Monte Carlo racetrack. We found that inverting the pixel values produced a much better separation for the foreground and background, because there was so little differentiation in the coloring of the picture. Once loading in the video and placing each frame into a column vector of our data matrix, we took the SVD to obtain the singular values of the matrix. By looking at the normalized singular values in **figure 1,** it is obvious that thee is one dominant singular value with the next few being somewhat relevant. It seems that about 75 singular values capture about 90% of the energy of the system.

In **figure 2,** we see the results of our DMD decomposition and reconstruction of the background and foreground images. Interestingly, the sparse reconstruction not only captures the movement of the cars, but it also captures many of the edges in the picture. This may just be an inaccuracy in trying to separate out the high and low frequencies, as the edges can cause some overlap. Furthermore, the R matrix we used to store the negative entries in the sparse DMD matrix also appears to capture the cars quite well. It almost looks like we can see a shimmer of the cars in the background, and it is possible that this is the result of adding the R matrix back to the low-rank DMD matrix. When I set R equal to 0 this shimmer seems to vanish.

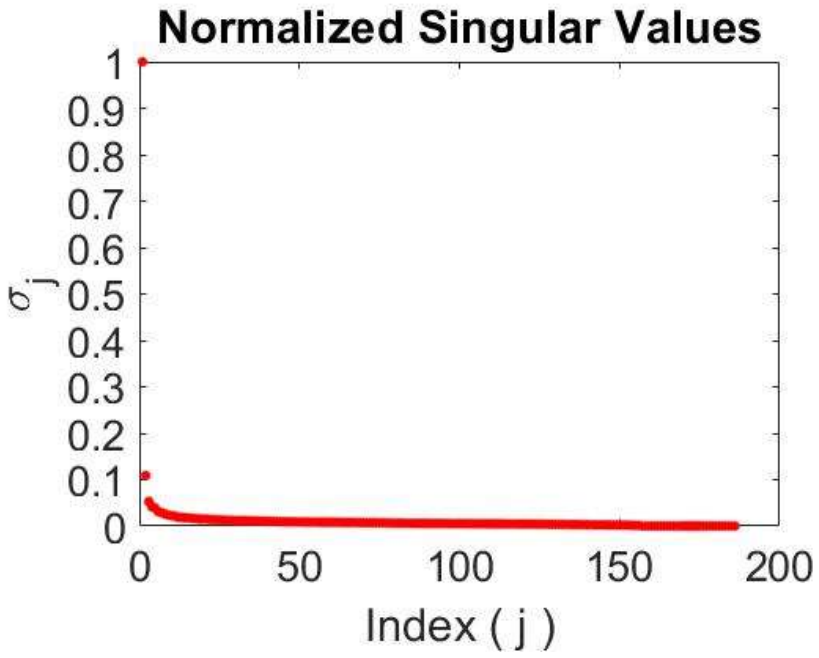

Figure 1: Normalized Singular Values and Variance obtained from taking SVD of the Monte Carlo video.

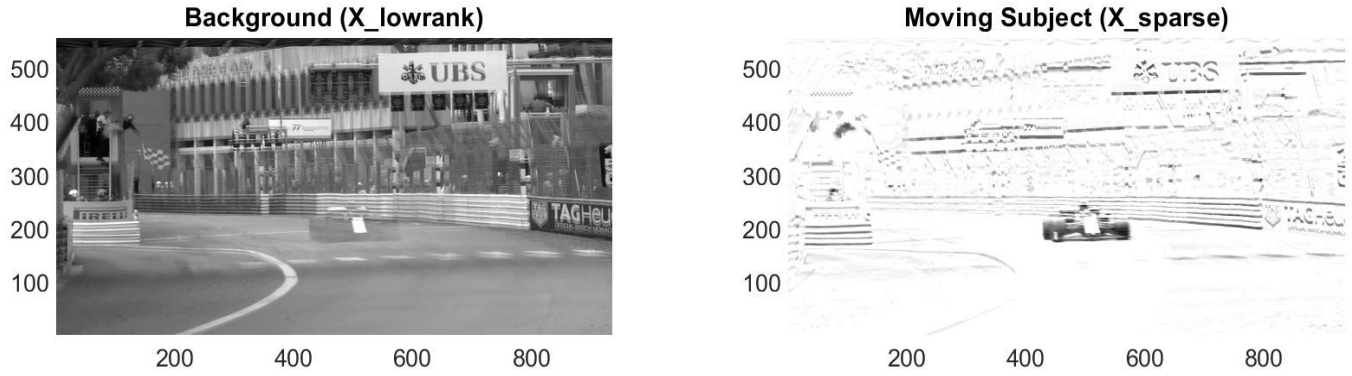

Figure 1: DMD decomposition of the Monte Carlo video split into two parts: the high frequency areas of the moving objects and the low frequency areas of the background.

The next video follows a skier going down a mountain slope with snow falling with them. We didn't show the plot of the singular values this time because they looked almost identical to the singular values of the Monte Carlo racetrack video. There is a fairly dominant first singular value followed by a very quick decline in magnitude. A low-rank approximation with about 75 modes captured at least 90% of the energy in the system, greatly reducing the size of the video data. As we can see in **figure 3,** the low rank approximation does a great job at separating the background with the moving skier. There are a lot fewer defined edges in this system, except where the snow creates a weird dark spot in the middle of the frame. This is likely because the snow is falling as the skier goes down the cliff, creating more moving parts even if it is not as drastic as the moving skier. It is difficult to see, but it looks like the R matrix is capturing the moving object just like last time.

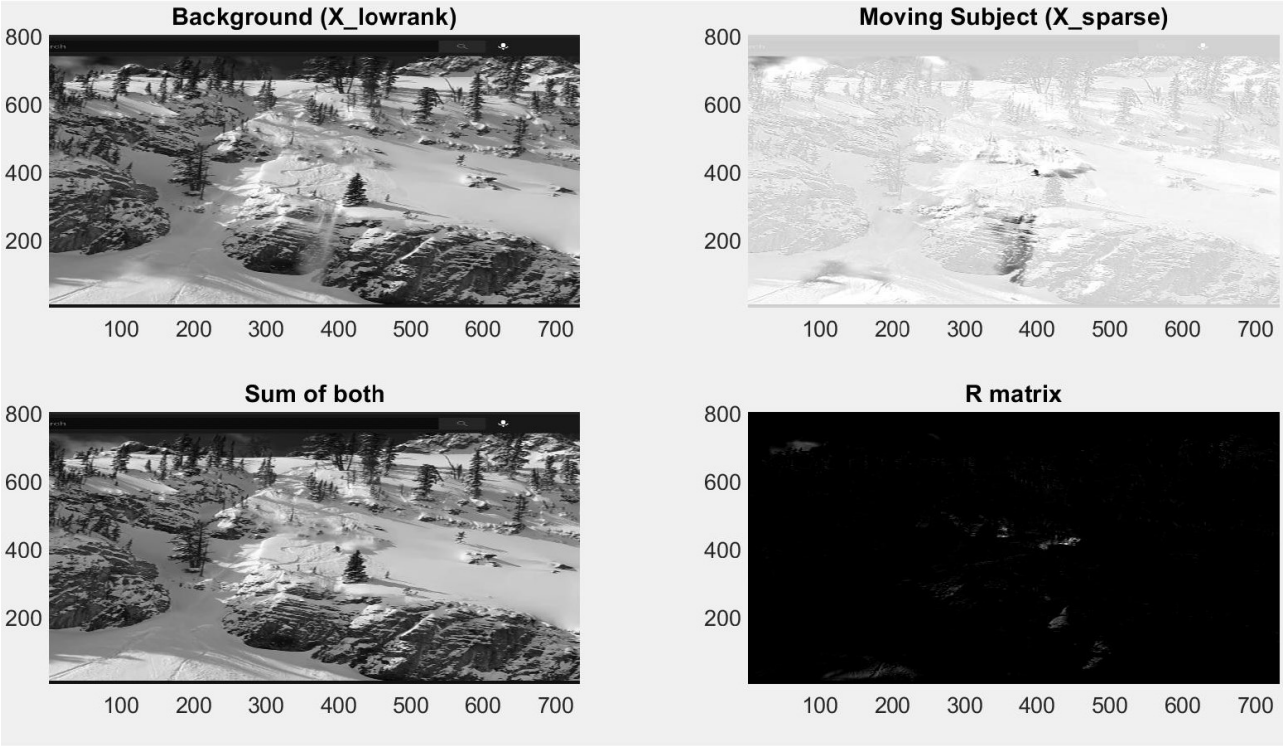

Figure 2:Figure 2: DMD decomposition of the Skiing video split into two parts: the high frequency areas of the moving objects and the low frequency areas of the background.

5 SUMMARY AND CONCLUSIONS

We saw that the SVD was very efficient at breaking down the system and creating a low rank approximation that lowered the computation time needed to perform the DMD. The first singular value seemed to consistently capture almost all the important data, with the rest being much less significant albeit still important. However, while the SVD is a powerful tool, the principal components in SVD must be orthogonal, placing restrictions on the types of modes that can occur together, while also ignoring some of the spatial-temporal information in the system. The DMD performs much better, by encoding each mode with a specific frequency, the DMD can capture spatial-temporal information a lot better. This phenomenon is important when we are trying to model dynamical systems in a data-driven way. As seen, the parameters we solve for can be very useful when trying to separate a moving object from a stationary background. That said, DMD struggles to differentiate one type of movement from the other, causing the falling snow to pop up quite visibly in the second video. To counteract this, other preprocessing steps may be necessary to improve the quality of the input before DMD can be used or we may need to be more selective in the choosing of the DMD modes. So, even though the DMD can take a long time to run for high-resolution data and struggles to differentiate multiple movements, it is still a powerful tool in image processing.

6 APPENDIX A

Below are the following MATLAB functions I used and a brief description of their functions:
data.

[**W,D] = eig(**A) : Returns a diagonal matrix D of eigenvalues and the eigenvectors as column vectors in W.

imcomplement(I): Given a black & white image matrix I, this returns a new image matrix with every pixel value inverted.

flipud(X): Flips a matrix top to bottom to be compatible with pcolor.

pcolor(X): plots a colored representation of the given matrix.

SVD(X): Calculates the Singular Value Decomposition of the given matrix X, returning U, Σ, V .

7 APPENDIX B

MATLAB Code

```
1  %%Michael Gabalis Code
2  %%Dynamic Mode Decomposition
3
4  clear,clc
5  %% Loading the video clips and processing
6
7  %choose which video to read
8
9  video = VideoReader('monte_carlo.mov');
10 %video = VideoReader('ski_drop.mov');
11
12 skip_frame = 2; % to reduce size of  data
13 res_Red = 0.5; % reduce resolution by this  factor.
14 imheight = video.Height*res_Red;
15 imwidth = video.Width*res_Red;
16 num_frames = ceil(video.Duration * video.FrameRate / skip_frame);
```

```
dt = 1;
t = 1:num_frames;
v = zeros(num_frames, imheight, imwidth); % stores frames as rows of 2D matrices
X = zeros(imheight*imwidth, num_frames); % stores frames as 1D columns

frame = 1;
index = 1;
while hasFrame(video)
    nframe = readFrame(video);
    if mod(index, skip_frame) == 0
        x = imresize(nframe, res_Red);
        v(frame, :, :) = imcomplement(rgb2gray(x)); %  imcomplement
        X(:, frame) = reshape(v(frame,:,:), [imheight*imwidth,  1]);
        frame = frame + 1;
    end
    index = index + 1;
end

X = X(:,1:num_frames);
%% Perform DMD 37
%calculate X1 and X2 matrices
X1 = X(:,1:end-1); X2 = X(:,2:end);

%find SVD and plot singular values
[U2,S2,V2] = svd(X1,  'econ');
threshold = 0.85;
r = find(cumsum(diag(S2) ./ sum(diag(S2)))  > threshold,1);

figure(1)
plot(diag(S2)./max(diag(S2)), 'r.', 'markersize', 15);
title('Normalized Singular Values')
xlabel('Index ( j )')
ylabel('\sigma_j') 51 yticks(0:0.1:1)
set(gca, 'fontsize', 20);

% find constants for DMD reconstruction
U=U2(:,1:r);
Sigma=S2(1:r,1:r);
V=V2(:,1:r);
Atilde = U'*X2*V/Sigma;
[W,D] = eig(Atilde);
Psi=X2*V/Sigma*W;

mu=diag(D);
omega=log(mu)/dt;
y0 =  Psi\X(:, 1);                % pseudo-inverse initial conditions

[omega_min, mindex] =  min(abs(omega));
foreground_modes_indices = find(abs(omega) > omega_min);

%%
clear all
```

```
%find low rank and sparse matrices

X_lowrank = y0(mindex).*Psi(:, mindex).*exp(omega(mindex).*t);
X_sparse = X - abs(X_lowrank);
R = X_sparse .* (X_sparse < 0); % places all negative entries in R


X_lowrank2 = abs(X_lowrank) + R;
X_sparse2 = X_sparse - R;

% Plot the foreground and background plots as well as R and the original
fig = figure;
clc, clear all
for frame = 1:size(X_lowrank2, 2)

    lowrank = imcomplement(reshape(X_lowrank2(:, frame), [imheight, imwidth]));
    sparse = imcomplement(reshape(X_sparse2(:, frame), [imheight, imwidth] ));
    rj = imcomplement(reshape(R(:, frame), [imheight, imwidth]));

    subplot(221)
    pcolor(flipud(lowrank)), shading interp, colormap(gray);
    title('Background (X\_lowrank)')
    set(gca, 'fontsize', 20);

    subplot(222)
    pcolor(flipud(sparse)), shading interp, colormap(gray);
    title('Foreground (X\_sparse)')
    set(gca, 'fontsize', 20);

    subplot(223)
    pcolor(flipud(sparse + lowrank)), shading interp, colormap(gray);
    title('Sum of both')
    set(gca, 'fontsize', 20);

    subplot(224)
    pcolor(flipud(rj)), shading interp, colormap(gray);
    title('R matrix')
    set(gca, 'fontsize', 20);

    drawnow;
    f(frame) = getframe(fig);
end
%%
implay(f)
```