AMATH 482 - Assignment 5

Clarisa Leu-Rodriguez

Winter 2021 - March 17th, 2021

Abstract

In this paper, we look at two different videos - where the first video is an eight-second long clip of a skier dropping down a mountain and the second video is a six-second long clip of the Monte Carlo race for Formula 1. Our goal when analyzing these two videos is to apply the concepts of dynamic mode decomposition (DMD) in order to separate the background of these videos from the foreground - where DMD is a very powerful tool/algorithm when it comes to the analysis of nonlinear systems. We explore the ideas of video processing, singular value decomposition (SVD), and DMD in this paper - and provide directions for future work on this problem as well.

Section I. Introduction and Overview

We are given two different videos, one being an eight-second long clip of a skier dropping down a mountain and the second being a six-second long clip of the Monte Carlo race for Formula 1. Our goal when analyzing these two videos is to apply the dynamic mode decomposition (DMD) algorithm and isolate the background from the foreground in each video - where the background in the ski drop clip is the white snow and the foreground is the moving skier. The background in the Monte Carlo clip is the stadium/race track and the foreground is the racing cars. DMD relies on having snapshots of spatio-temporal data (i.e. - data which evolves in both space and time) which is collected at regularly spaced intervals. Given this spatio-temporal data (which is not necessarily linear), DMD gives us a low-rank approximation of the linear mapping which best iterates through the snapshots of data. For us, these snapshots of spatio-temporal data will be each frame in each of the respective videos - where in order to separate the background from these videos we apply DMD to give us a low-rank approximation (this is the background of the video) and to get the foreground we subtract the background from the original video. Below in Figure 1, we show two frames from the videos (original) we will be analyzing.

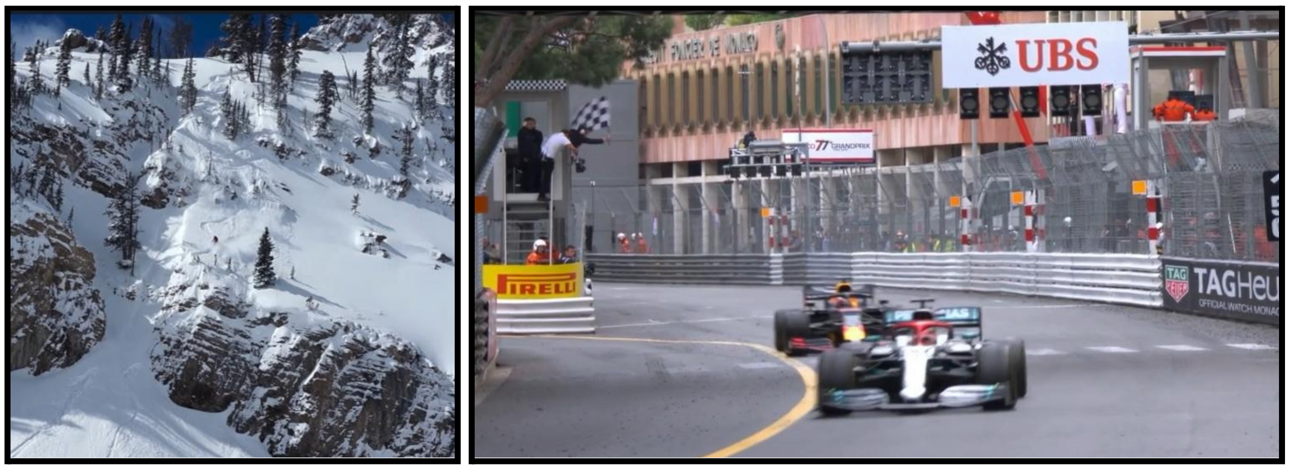

Figure 1: On the left (Figure 1a) we show a frame from the ski drop video where the skier (foreground) is dropping down the mountain. On the right (Figure 1b) we show a frame from the Monte Carlo video where the cars (foreground) are racing down the track in the stadium (background).

Section II. Theoretical Background

To successfully isolate the background and foreground from our videos - we use the DMD algorithm which takes advantage of low dimensionality in the given spatio-temporal data to create a linear mapping which best iterates through the snapshots of data. In this section, I will discuss the mathematics and theory behind our solution - particularly touching base on Singular Value Decomposition (which is used in the DMD algorithm) and DMD.

Section II.I. Singular Value Decomposition (SVD)

SVD is the factorization of a matrix which is a generalization of the eigendecomposition of a square normal matrix to any m x n matrix - where every m x n matrix A has a SVD. The SVD of a matrix A is shown below.

$$A = U\Sigma V^*$$

Equation 1: The SVD of a Matrix A where A is m x n.

In Equation 2, if A is complex - U is an m x m complex unitary matrix (i.e. its conjugate transpose is also it's inverse - $U^*U = UU^* = I$), Σ is an m x n rectangular diagonal matrix with non-negative, real number along its diagonal, and V^* is an n x n complex unitary matrix. If A is real, then U and $V^T = V^*$ are real orthogonal matrices (i.e. their columns and rows are orthonormal vectors - $U^T U = UU^T = I$ and $V^T V = V V^T = I$).

U and *V* are rotational matrices and Σis a stretching matrix - where the diagonal entries along Σ are the uniquely determined singular values of *A*, the columns of *U* are left-singular vectors, and the columns of *V* are right-singular vectors of *A*. The total number of non-zero singular values gives us the rank of *A* - where the SVD is often used as the preferred method for matrix rank calculations in mathematical software, as often it is more reliable due to approximation errors in real numbers. Additionally, other properties of the SVD include:

- The singular values of *A* are always ordered from largest to smallest (i.e. $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$) along the diagonals of Σ.
- Letting r = rank(A), we have that $\{u_1, u_2,..., u_r\}$ is a basis for the range of *A* and $\{v_r + 1,..., v_n\}$ is a basis for the null space of *A*.

The SVD of a matrix tells us the dimension and the direction of the principal components of the matrix. It reveals information as to how much data is in each dimension/rank approximation (via the singular values of *A*). In our solution, after determining the principal components in our system - we then determine which ones contribute the most energy to our system through looking at the energy captured in each dimension/rank through the following equation:

$$energy_n = \frac{\sigma_1 + ... + \sigma_n}{\sigma_1 + ... + \sigma_r}$$

Equation 2: The energy contained in the rank-*n* approximation of *A* (where *A* is of rank *r*) captured by the singular values (i.e. the $\boldsymbol{\sigma}_i$'s) of *A*.

When analyzing which principal components contribute the most to our system and determining the rank at which to truncate *U*, *V*, and Σ in the DMD algorithm - we see which modes are needed to summarize a threshold of 90% energy in the system. The number of modes needed for 90% energy determine the rank at which to truncate *U*, *V*, and Σ for the DMD algorithm.

Section II.II. Dynamic Mode Decomposition (DMD)

As stated previously, DMD allows us to analyze nonlinear or complex dynamical systems which vary in both space and time. That is, DMD aims to describe the dynamical system (where the governing equations *f* are unknown):

$$\frac{dx}{dt} = f(x, t; \mu)$$

Equation 3: The dynamical system $\frac{dx}{dt}$ which varies in both space (x) and time *(t)*.

DMD is particularly powerful in that it is data driven, equation free, and doesn't make any underlying assumptions of the data (i.e. linearity). It is also simple to understand and fast to compute. The DMD algorithm is as follows:

1. Sample data at N prescribed locations M times. The data snapshots should be evenly spaced in time by a fixed Δt. We use the snapshots to create the matrix **X** (the snapshot/data matrix).
2. From the data matrix **X**, construct two submatrices $\mathbf{X}_1^{M-1}$ and $\mathbf{X}_2^{M}$ - where the subscripts refer to the columns of the original data matrix.
3. Compute the SVD of $\mathbf{X}_1^{M-1}$.
4. Create the matrix $\tilde{\mathbf{S}} = \mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\Sigma^{-1}$ and find its eigenvalues and eigenvectors.
5. Use the initial snapshot $\mathbf{x}_1$ and the pseudoinverse of Ψ to find the coefficients b_k.
6. 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 $\mathbf{S}^{\sim}$.

That is, the DMD algorithm relies on having snapshots of spatio-temporal data which evolves in both space and time. In the first step of the DMD algorithm, N refers to the number of spatial points saved per unit time snapshot and M refers to the number of snapshots taken. Additionally, the time data must be collected over regularly spaced intervals with $t_{m+1} = t_m + \Delta t$, $m = 1, \ldots, M - 1$ with $\Delta t > 0$. These snapshots are denoted as:

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

For each $m = 1, \ldots, M$. We can use these snapshots to form columns of data matrices:

$$\mathbf{X} = \begin{bmatrix} U(\mathbf{x}, t_1) & U(\mathbf{x}, t_2) & \cdots & U(\mathbf{x}, t_M) \end{bmatrix}, \quad \mathbf{X}_j^k = \begin{bmatrix} U(\mathbf{x}, t_j) & U(\mathbf{x}, t_{j+1}) & \cdots & U(\mathbf{x}, t_k) \end{bmatrix}$$

Where the matrix $\mathbf{X}_j^k$ is just columns j through k of the full snapshot/data matrix **X**. The DMD method approximates the modes of the Koopman operator, which is a linear, time-independent operator such that:

$$\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j$$

Equation 4: The Koopman operator **A** which gives us a linear mapping from time t_j to t_{j+1} - where the Koopman operator is global.

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 $\mathbf{x}_j$ is an N-dimensional vector of the data points collected at time j. To construct the appropriate Koopman operator which best represents the data collected, we consider the matrix:

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_2 & \cdots & \mathbf{x}_{M-1} \end{bmatrix}$$

Where $\mathbf{x}_j$ denotes a snapshot of the data at time t_j. The Koopman operator allows us to rewrite this as:

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{A}\mathbf{x}_1 & \mathbf{A}^2\mathbf{x}_1 & \cdots & \mathbf{A}^{M-2}\mathbf{x}_1 \end{bmatrix}$$

Where the columns are formed by applying powers of **A** to the vector $\mathbf{x}_1$, and form the basis for the Krylov subspace. We are able to rewrite this matrix as:

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T$$

Where e_{M-1} is the vector with all zeros except a 1 at the (M - 1)st component. The residual vector **r** is added to account for the final point $\mathbf{x}_M$ not being included in the Krylov basis. To find **A**, recall that we can use the SVD to write

$\mathbf{X}_1^{M-1} = \mathbf{U}\Sigma\mathbf{V}^*$ - which gives us $\mathbf{X}_2^M = \mathbf{A}\mathbf{U}\Sigma\mathbf{V}^* + \mathbf{r}e_{M-1}^T$. We choose **A** in such a way that the columns in $\mathbf{X}_2^M$ can be written as the linear combination of the columns of **U** (i.e. - linear combinations of the POD modes). Giving us the following equation for **S**˜:

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \underbrace{\mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\Sigma^{-1}}_{=:\tilde{\mathbf{S}}}$$

Equation 5: The equation for **S**˜.

We note that **S**˜ and **A** are similar and share the same eigenvalues and eigenvectors. That is, if **y** is an eigenvalue of **S**˜, then $\mathbf{U}_y$ is an eigenvector of **A**. The eigenvector/eigenvalue pairs of **S**˜ can be written as $\tilde{\mathbf{S}}\mathbf{y}_k = \mu_k\mathbf{y}_k$ - thus giving the eigenvectors of **A** which are referred to as the DMD modes:

$$\psi_k = \mathbf{U}\mathbf{y}_k$$

Equation 6: The DMD modes/eigenvectors of **A**.

Then, to describe the continual multiplications of **A**, we can just expand in our eigenbasis to get:

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k\psi_k e^{\omega_k t} = \Psi \text{diag}(e^{\omega_k t})\mathbf{b}.$$

Equation 7: The solution for $\mathbf{x}_{DMD}(t)$ where K is the rank of $\mathbf{X}_1^{M-1}$. The b_k are the initial amplitude of each mode and the matrix Ψ contain the eigenvectors of Ψ_k as its columns. Additionally, $\omega_k = ln(\mu_k)/\Delta t$.

Interpreting b_k, we know that at time $t = 0$ in Equation 7 we get $\mathbf{x}_1$ because this was our initial condition to generate the other $\mathbf{x}_m$. Therefore, taking $t = 0$ in the above gives us $\mathbf{x}_1 = \Psi\mathbf{b} \implies \mathbf{b} = \Psi^\dagger\mathbf{x}_1$ where $\Psi^\dagger$ is the pseudoinverse of the matrix Ψ.

Section III. Algorithm Implementation and Development

To start, we first import and load the two videos into MATLAB - where the ski drop video consists of 454 frames for an eight-second long clip and the dimensions of each frame is 960x540 pixels & the Monte Carlo video consists of 379 frames for a six-second long clip and has the same dimensions. To construct our data matrix **X** for DMD - we iterate through each frame and convert it to a black and white image and flip the contrast of the image so black pixels are white and white pixels are black. Flipping the contrast of each frame was done as visually it made our results for DMD look nicer - where the process described here was also repeated when not flipping the contrast of the pixels in the video frames. Additionally, we convert each image to a double for computation and add each frame to our data matrix as a column matrix of size 960x540.

Next, we construct our DMD matrices $\mathbf{X}_1^{M-1}$ and $\mathbf{X}_2^M$ and perform SVD on $\mathbf{X}_1^{M-1}$ and get our matrices U, Σ, an V. To determine a rank at which to truncate our matrices from SVD - we look at the threshold for modes to include for 90% energy. For the ski drop video - this was 12 modes and for the Monte Carlo video this was 99 modes. Additionally, we plot the singular value spectrum and associated energies using Equation 2. After finding a low rank approximation for $\mathbf{X}_1^{M-1}$ through the SVD - we use Equation 5 to compute **S**˜ and solve for **S**˜'s associated eigenvalues and eigenvectors. Additionally, we use Equation 6 to solve for Phi, solve for ω, and solve for the initial condition y_0. From there, we look at the ω values on the real and complex plane. Next, we pick the smallest ω to summarize the background and use Equation 7 to calculate $\mathbf{X}_{Low\text{-}Rank}$ - which gives us the background of our video. Next, we calculate $\mathbf{X}_{Sparse}$ and additionally construct our matrix **R** which contains the negative residual values in $\mathbf{X}_{Sparse}$. We then add these negative residuals to the absolute value of $\mathbf{X}_{Low\text{-}Rank}$ to construct the background and subtract these negative residuals from $\mathbf{X}_{Sparse}$ to construct the foreground as negative pixel intensities do not make sense. From there, we add the constructured matrices for the foreground and background to look at our reconstructed video as well. Next, we reconstruct the video from the frames and look at how well we did in isolating the background and foreground from one another in each video. The above process was repeated for each video.

Section IV. Computational Results

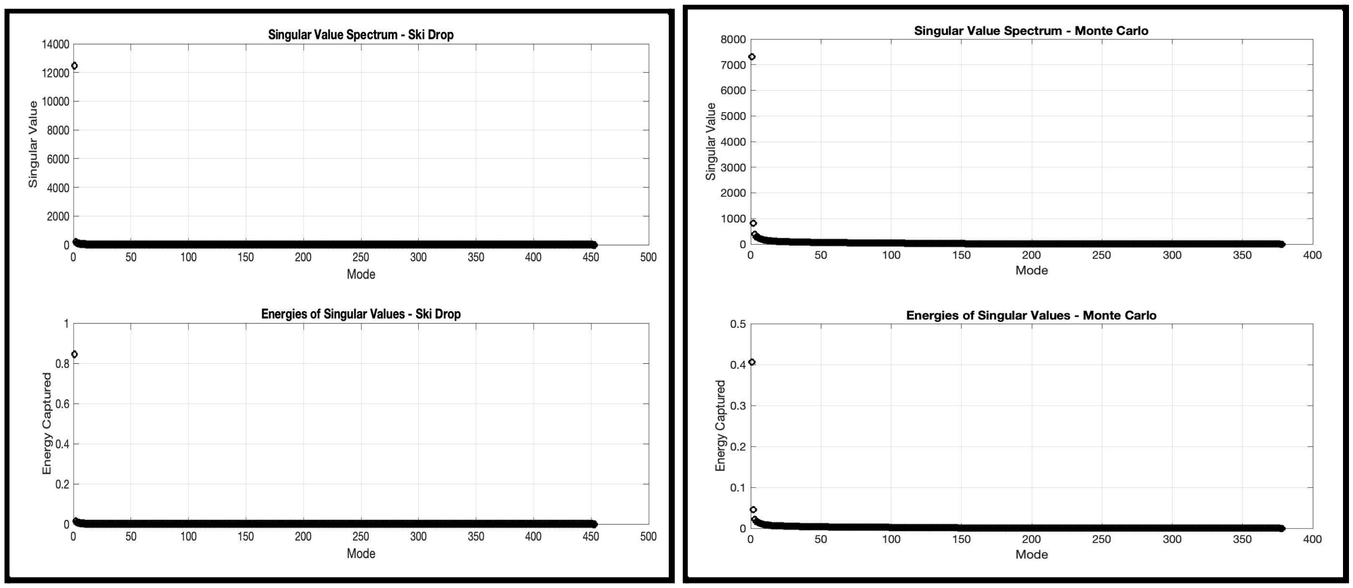

Figure 2: Above, we plot the singular values for each video's $\mathbf{X}_1^{M\text{-}1}$ respectively and the energies using Equation 2. On the left (Figure 2a) we plot the ski drop video's and on the right (Figure 2b) we plot the Monte Carlo video's.

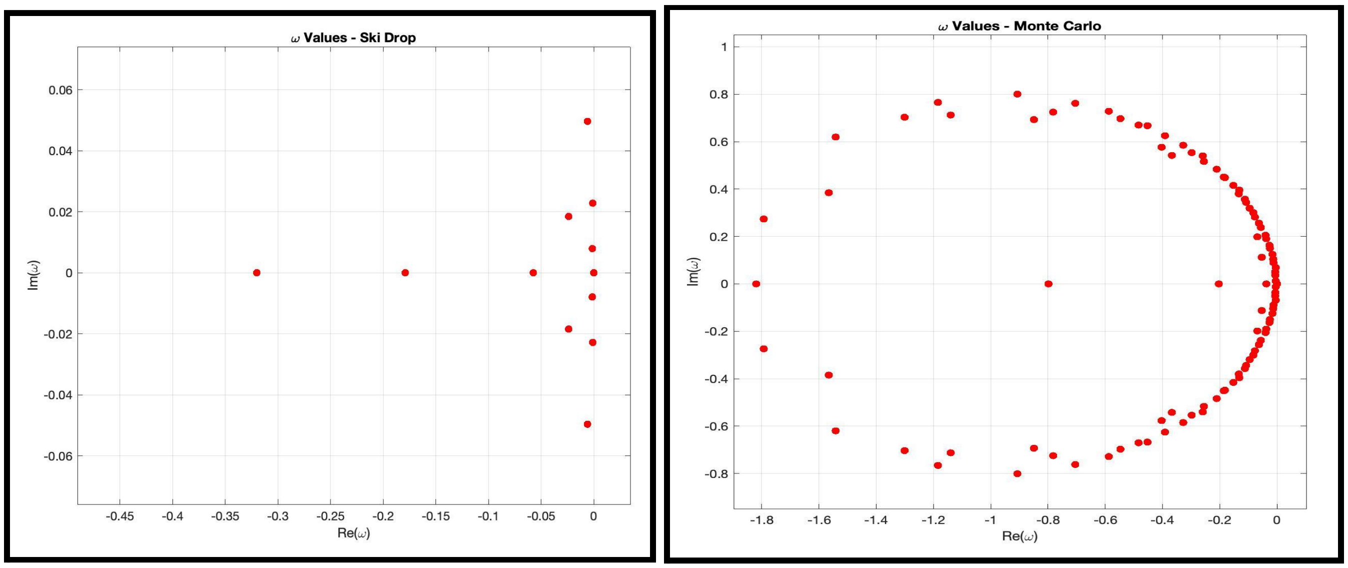

Figure 3: Above, we plot the ω values for each video which represent the time dynamics of our DMD modes - where on the left (Figure 3a) we plot the ski drop video's and on the right (Figure 3b) we plot the Monte Carlo video's.

Section V. Summary and Conclusions

Analyzing our results after applying SVD to $\mathbf{X}_1^{M-1}$ in Figure 2, we see that the first mode for the ski drop video makes up most of the energy in the system and for the Monte Carlo video - the first few modes makeup most of the energy in the system. As stated previously, to summarize 90% of the total energy in the ski drop video - 12 modes were needed. To summarize 90% of the total energy in the Monte Carlo video, 99 modes were needed. Comparing and contrasting the different backgrounds in each video, the Monte Carlo one had much more noise present in the background as shown in Figure 1 with the people in the background waving the flag, etc. so this makes sense.

Analyzing our ω values in Figure 3, recall that the ω values describe the time dynamics for our DMD modes where the DMD modes summarize spatial patterns in the frames. If ω is real and positive, these modes blow up in time and become larger. If ω is real and negative, they end up disappearing. If ω is complex then it is oscillating in time, etc. We see that in the ski drop video, one ω value is needed to represent the background which is shown with the red dot at the origin. In the Monte Carlo video, the ω values have a more interesting spread and more modes would be needed to describe the background. Due to time constraints, we only used the smallest ω value to represent the background for the Monte Carlo video - similar to the ski drop video. However, future work to improve our results from DMD on this video would be to use a larger range of ω values to represent the background.

Looking at Figure 4 in Appendix C. Additional Figures, we see that we were able to successfully isolate the background from the foreground fairly well for the ski video, where if you look closely in the foreground image the black spec is the skier moving down the mountain and in the background video - the skier is not present at all. In the reconstructed frame for the ski video we see that the frame includes both the foreground and background as expected. Looking at Figure 5 in Appendix C. Additional Figures, we see that we weren't as successful in completely isolating the background for the Monte Carlo video. If you look closely - you can still see some of the outline left by the car in the background frame. However, the foreground frame looks very nice and we can clearly see the moving car. The reconstructed frame consists of both the foreground and background as we expect for the Monte Carlo video. As stated previously, the next step here would be to include a large range of ω values close to zero to improve our results for DMD on the Monte Carlo video. That is, future work for this problem would also be to experiment more with how changing the number of modes used in DMD improves the results for background separation, as well as exploring different parameters as they relate to DMD. However, DMD proved to be very successful in isolating the background from the foreground in the ski drop video.

Section VI. References

- https://en.wikipedia.org/wiki/Dynamic_mode_decomposition - I used this source to better understand the dynamic mode decomposition.
- In addition, the course notes provided on the course website as well as lecture material was used.

Appendix A. MATLAB Functions Used and Brief Implementation Explanation

This appendix contains a list of the MATLAB functions used and additional information regarding how it was used to solve our submarine problem. Additional information can also be found in Section II. Theoretical Background and also Section III. Algorithm Implementation and Development. Note, descriptions of functions were sourced from the MATLAB documentation which can be found here online: https://www.mathworks.com/help/matlab/.

- `OBJ =` **`VideoReader`**`(FILENAME)` constructs a multimedia reader object, OBJ, that can read in video data from a multimedia file. FILENAME looks for the file on the MATLAB path. We used this to read in our video data into MATLAB, and also to grab corresponding features of the video such as frame number, frame height, and frame width.
- `[X, MAP] =` **`frame2im`**`(F)` returns the indexed image X and associated colormap MAP from the single movie frame F. We used this when looping through each frame in our video and converting it to an image.
- `IM2 =` **`imcomplement`**`(IM)` computes the complement of the image IM. In the complement of the image, black becomes white and white becomes black. We used this to flip the contrast of each image in our videos before DMD.
- `IM2 =` **`im2double`**`(I1)` convert the image I1 to double precision. We used this to convert each frame in our video to double precision for computation.
- **`zeros`**`(M, N, P, ...)` generates an M-by-N-by-P-by-... array of zeros. We used this to create a zero-initialized array to be used to store our transformed data. We use the `zeros` function in MATLAB to preallocate memory before computation - which saves the computer time as it doesn't need to grab more buckets during computation as the space to store the result has already been preallocated.
- **`reshape`**`(X, M, N, P, ...)` returns an N-D array with the same elements as X but reshaped to have the size M-by-N-by-P-by-.... We used reshape to our video frame data either for computation or visualization.
- `[U, S, V] =` **`svd`**`(X, 'econ')` produces the "economy size" singular value decomposition of X. We used this to compute the svd of $\mathbf{X}_1^{M-1}$ in the DMD algorithm.
- **`diag`**`(X)` computes the main diagonal of X. We used this to grab the singular values along the diagonal of S after SVD.
- `[V, D] =` **`eig`**`(A)` computes the eigenvectors V and eigenvalues D of the matrix A.
- `[M, I] =` **`min`**`(X)` returns the smallest element in the vector X (M) and also the index I corresponding to its position in X. We used this to find the smallest omega value to represent the background in the DMD algorithm.
- **`abs`**`(X)` is the absolute value of the elements of X. When X is complex, it computes the complex modulus (magnitude) of the elements of X. We used this in our code when we wanted to find the minimum value in absolute value in our omega values (summarizes background).
- `IM2 =` **`im2uint8`**`(I1)` convert the image I1 to uint8 precision. We used this to convert each frame in our video to from double precision to uint8 precision after computation when trying to visualize them.
- **`imshow`**`(I)` displays the image I. We used this when plotting our background, foreground, and reconstructed videos.
- `drawnow` updates figure windows and process callbacks.

Appendix B. MATLAB Codes

This appendix contains the MATLAB code used in this paper. Please refer to Appendix A. MATLAB Functions Used and Brief Implementation Explanation for more information regarding how each the MATLAB functions used in our code below works.

```
%{
    Assignment #5 - Background Subtraction in Video Streams
    AMATH482 - Computational Methods For Data Science - Mar. 17th, 2021
    Taught by Professor Jason J. Bramburger (Winter 2021)
    Written By: Clarisa Leu-Rodriguez - email: cleu@uw.edu
%}


%% Import Videos
clear all; close all; clc;  % Clear and refresh workspace.

% We use the low resolution versions for ease of computation.
% Uncomment depending on which video you want to perform DMD on.
%video = VideoReader('ski_drop_low.mp4'); vid_title = "Ski Drop";
video = VideoReader('monte_carlo_low.mp4'); vid_title = "Monte Carlo";


%% Extract Data for DMD
video_frames = read(video);
num_frames = video.NumFrames;
frame_width = video.Width;
frame_height = video.Height;
dt = 1; t = 1:num_frames;

% Construct struct needed to use frame2im() function.
for j = 1:num_frames
    vid_dat(j).cdata = video_frames(:, :, :, j);
    vid_dat(j).colormap = [];
end

X = zeros(frame_height * frame_width, num_frames);  % Each frame is a column in
X.
for j = 1:num_frames
    x = frame2im(vid_dat(j));
    % Show original video
    % imshow(x);
    % Show original video
    % imshow(x);
    x = rgb2gray(x);
    % Flip contrast of the image (visually works better with DMD).
    x = imcomplement(x);
    x = im2double(x);
    X(: , j) = reshape(x, [frame_height * frame_width, 1]);
end
```

```matlab
% Create DMD Matrices from video data.
X1 = X(:, 1:end-1); X2 = X(:, 2:end);


%% DMD: Perform SVD & Truncate U, Sigma, and V to Lower Rank
[U, Sigma, V] = svd(X1, 'econ');  % SVD of X1

% Plot Singular Value Spectrum & Energy From SVD
figure();
subplot(2, 1, 1);
plot(diag(Sigma), 'ko', 'LineWidth', 2);
title('Singular Value Spectrum - ' + vid_title); xlabel('Mode');
ylabel('Singular Value'); grid on; set(gca, 'fontsize', 12);
subplot(2, 1, 2);
plot(diag(Sigma) / sum(diag(Sigma)), 'ko', 'LineWidth', 2);
title('Energies of Singular Values - ' + vid_title); xlabel('Mode');
ylabel('Energy Captured'); grid on; set(gca, 'fontsize', 12);

% Truncate to rank with 90% energy
energy = 0;
total_energy = sum(diag(Sigma));
threshold = 0.9;
r = 0;
while energy <= threshold
    r = r + 1;
    energy = energy + Sigma(r,r) / total_energy;
end

% Low-rank approximation of U, Sigma, and V
U_r = U(:, 1:r);
Sigma_r = Sigma(1:r, 1:r);
V_r = V(:, 1:r);


%% DMD: Computation of ~S & DMD Modes
S = U_r' * X2 * V_r / Sigma_r;
[eV, D] = eig(S);  % Compute eigenvalues and eigenvectors.
mu = diag(D);
omega = log(mu) / dt;
Phi = U_r * eV;

% Plot Omegas
figure();
plot(real(omega), imag(omega), 'r.', 'Markersize', 20);
title("\omega Values - " + vid_title);
xlabel('Re(\omega)'); ylabel('Im(\omega)');
grid on; set(gca, 'fontsize', 12);
```

```matlab
%% DMD: Create DMD Solution
y0 = Phi\X1(:, 1);  % Pseudoinverse to get initial conditions.
% We pick the smallest omega which summarizes the background.
[min_omega, min_index] = min(abs(omega));
X_dmd = y0(min_index).*Phi(:, min_index).*exp(omega(min_index).*t);


%% DMD: Create Sparse and Non Sparse
X_sparse = X - abs(X_dmd);
R = X_sparse.*(X_sparse < 0);
X_background = R + abs(X_dmd);
X_foreground = X_sparse - R;
X_reconstructed = X_foreground + X_background;


%% DMD: Playback video frame by frame
figure();
for j = 1:num_frames
    % Reshape & flip contrast back.
    background = reshape(X_background(:, j), [frame_height, frame_width]);
    background = imcomplement(background);
    foreground = reshape(X_foreground(:, j), [frame_height, frame_width]);
    foreground = imcomplement(foreground);
    reconstructed = reshape(X_reconstructed(:, j), [frame_height,
frame_width]);
    reconstructed = imcomplement(reconstructed);

    % Plot Background, Foreground, and Reconstructed
    subplot(3, 1, 1);
    imshow(im2uint8(background));
    title('Background - ' + vid_title); set(gca, 'fontsize', 12);

    subplot(3, 1, 2);
    imshow(im2uint8(foreground));
    title('Foreground - ' + vid_title); set(gca, 'fontsize', 12);

    subplot(3, 1, 3);
    imshow(im2uint8(reconstructed));
    title('Sum of Background & Foreground - ' + vid_title);
    set(gca, 'fontsize', 12);

    drawnow;
end
```

Appendix C. Additional Figures

This appendix contains additional figures as they relate to Section V. Summary and Conclusions.

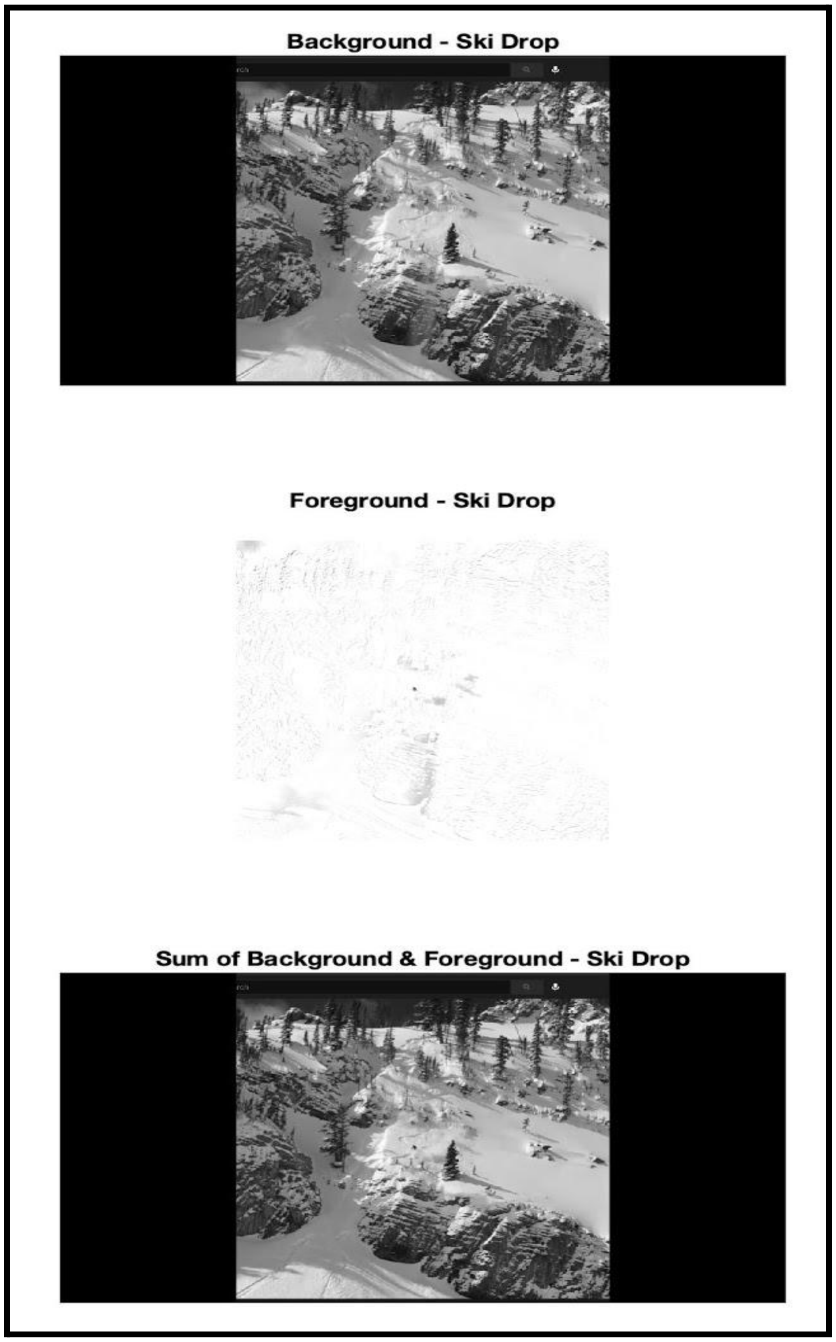

Figure 4: Above, we plot a single frame for the background, foreground, and reconstructed image for the ski drop video.

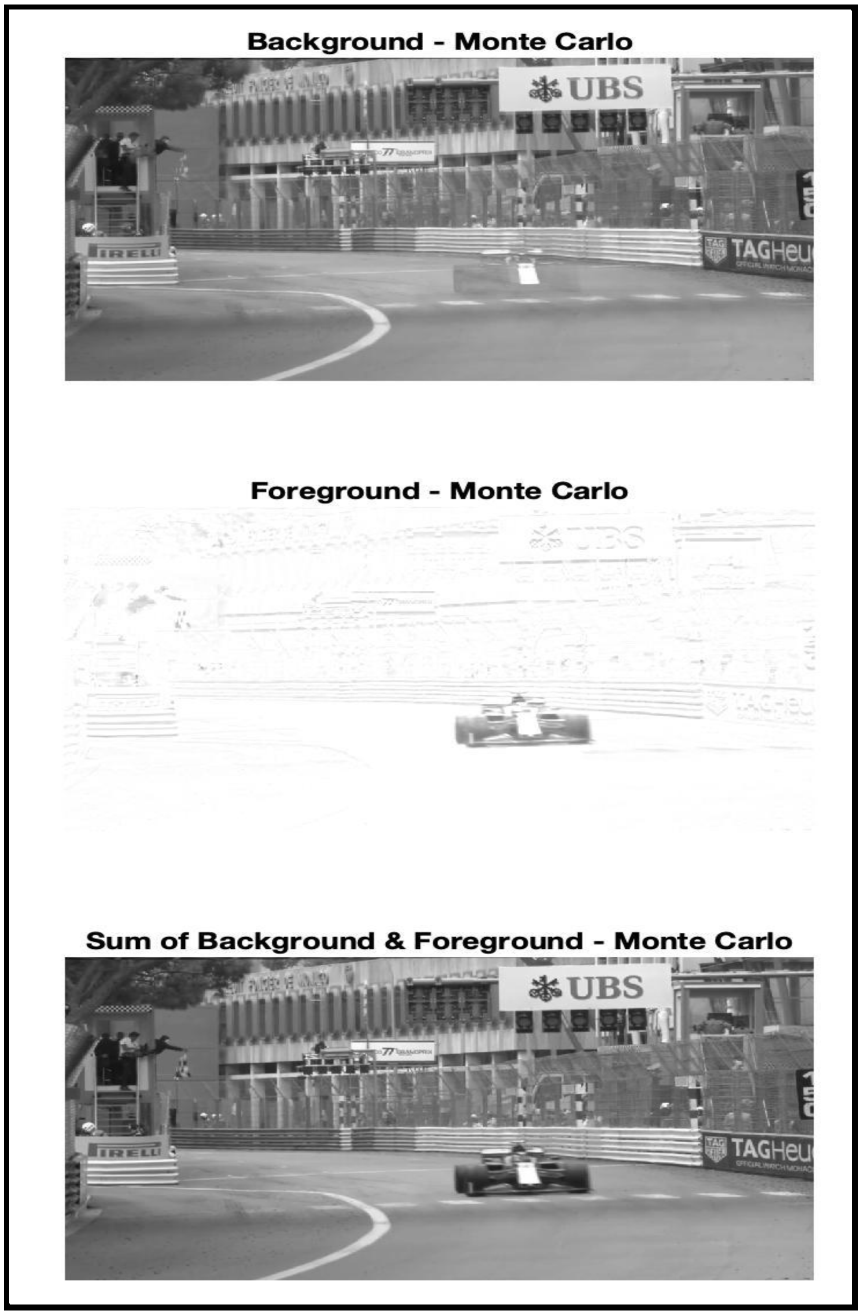

Figure 5: Above, we plot a single frame for the background, foreground, and reconstructed image for the Monte Carlo video.