Title: Background Subtraction in Video Streams

Author: Wendy Jiang

Abstract: There are two video clips with a foreground and background object, and we use the Dynamic Mode Decomposition (DMD) method on the clips. The aim of the DMD method is to take advantage of low dimensionality in experimental data without having to rely on a given set of governing equations. The first video is a video of skiing. The second video is a video of F1 cars taking a corner in a race. The components of DMD highlight the spatiotemporal properties of the given system, allowing us to study its dynamic over time and make predictions about the future state of the system. Hence, in this assignment, we will use DMD's properties to separate the videos into high frequency moving subject and stationary backgrounds.

Section I. Introduction and Overview

Of the video, there is a high-dimensional nonlinearly system on both space and time. To analysis the video, the dimensionality reduction is we really care about. The Dynamic Mode Decomposition (DMD) allows us to take advantages of low dimensionality in experimental data without having to rely on a given set of governing equations. DMD allows to forecast the data, that is predict what will be in the future. It's output, the decomposition of the complex data into spatio-temporal coherent structures, the basis of the spatial modes is not necessary the orthogonal, for which the time dynamics are just exponential functions.

Here we have two videos to analyze. When we set up, the data involves in both space and time. The dynamic mode decomposition relies on having snapshots of data at N prescribed locations M times. The data snapshots should be evenly spaced in time by a fixed Δt. Use the snapshots to create the matrix X and then compute the SVD to find the eigenvalue and eigenvectors. After that, we use the initial snapshots and the pseudoinverse of ψ to find the coefficients b_k, and then compute the future solution.

Theoretical Background

1. To do the snapshots of spatio-temporal data, we define:

   N = number of spatial points saved per unit time snapshot.

   M = number of snapshots taken.

   The most important thing is that the time data is collected at regularly spaced intervals:

$$t_{m+1} = t_m + \Delta t.\ \ m = 1, \ldots M-1,\ \ \Delta t > 0 \ \ldots (1)$$

   The snapshots are denoted $U(x, t_m) = \begin{pmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \vdots \\ U(x_n, t_m) \end{pmatrix}. \ \ \ldots (2)$

From each m = 1,…,M, we can use the snapshots from the columns of data matrices.
$X \ = \ [U(x,t_1) \ U(x,t_2) \ \cdots U(x,t_M)]. \ \ldots(3)$

$X_j^k \ = \ [U(x,t_j) \ U(x,t_{j\,+\,1}) \ \cdots \ U(x,t_k)]. \ \ldots(4)$

The matrix X_j^k is just the columns j through k of the full snapshot matrix X.

2. The DMD method approximates the modes of the Koopman operators. The Koopman operator A is a linear, time-independent operator such that $x_{j\,+1} \ = \ Ax_j.\ldots(5)$. Where j is 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.
3. Dynamic Mode Decomposition: To construct the appropriate Koopman operator that best represent the data collected, we will consider the matrix $X_1^{M-1} = [x_1 x_2 \cdots \ x_{M-1}]. \ \ldots(6)$, 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 as $X_1^{M-1} \ = \ [x_1 \ \ Ax_1 \ \ A^2 x_1 \ \ A^{M-2}x_1].\ldots(7)$

   The columns are formed by applying powers of A to the vector x_1, and are said to form the basis for the Krylov space. We can get the way of relating the first M-1 snapshots to x_1 using just the Koopman matrix, to get $X_2^M \ = \ AX_1^{M-1} \ + \ re_{M-1}^T. \ \ldots(8)$. Where e_{M-1} is the vector with all zeros except a 1 at the (M-1)st component. A is the unknow and it's our goal to find it. We are going to circumvent finding A directly by finding other matrices with the same eigenvalues. First use SVD to write $X_2^M \ = \ AU\Sigma V^* \ + \ re_{M-1}^T. \ \ldots(8)$

   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. Hence, the residual vector r must be orthogonal to the POD basis, giving that $U * r \ = \ 0. \ \ldots(9)$.

   Multiply to get $U * X_2^M \ = \ U * AU\Sigma V^*.\ldots(10)$

   Then isolate $U * AU$: $U * AU \ = \ U * X_2^M V\Sigma^{-1}. \ \ldots(11)$

   In equation (11), everything on the right sided is known from the input data, we called it $\tilde{S}$ to keep convention.

   Note: The goal of this data-driven method is dimensionality reduction. 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 M and N. Precisely, $U \ \in \ \mathbb{C}^{N\times K}, \ \Sigma \ \in \ \mathbb{R}^{K\times K}, \ V \ \in \ \mathbb{C}^{M-1\,\times\,K}. \ \ldots(12)$

   Notice that $\tilde{S}$ and A are related by applying a matrix on one side and its inverse on the other. So, they have the same eigenvalues. Let's write the eigenvector/eigenvalues pairs of $\tilde{S}$ as $\tilde{S}y_k \ = \ \mu_k y_k.\ldots(13)$ thus giving the eigenvectors of A, called DMD modes, by $\psi_k \ = \ Uy_k.\ldots(14)$

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

$$X_{DMD} = b_p\varphi_p e^{\omega_p t} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t} \qquad \dots(15)$$

$b_p\varphi_p e^{\omega_p t}$ is the background of the video and $\sum_{j \neq p} b_j\varphi_j e^{\omega_j t}$ is the foreground of the video.

Then, a proper DMD reconstruction should also produce $X_{DMD} \in \mathbb{R}^{n \times m}$. However, each term of the DMD reconstruction is complex: $b_j\psi_p exp(\omega_j t) \in \mathbb{C}^{n \times m} \;\forall\, j.\;\; \dots(16)$, though they sum to a real-valued matrix. Consider calculating the DMD's approximate low-rank reconstruction according to $X_{DMD}^{Low-Rank} = b_p\varphi_p e^{\omega_p t}.\;\; \dots(17)$. Since it should be true that $X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}.\;\; \dots(18)$, can be calculated with real-valued elements only as follows $X_{DMD}^{Sparse} = X - \left|X_{DMD}^{Low-Rank}\right|.\;\; \dots(18)$

Finally, with these low-rank and sparse DMD approximations, of the image, since we subtracted the low-rank approximation from our initial data obtain the sparse approximation reconstruction, we may obtain negative values in our sparse matrix. However, there's negative values in sparse matrix. We put these negative values into a residual $n \times m$ matrix R and then create a low-rank and sparse reconstruction as follows:

$$X_{DMD}^{Low-Rank} \leftarrow R + \left|X_{DMD}^{Low-Rank}\right|.\;\; \dots(19)$$

$$X_{DMD}^{Sparse} \leftarrow X_{DMD}^{Sparse} - R.\;\; \dots(20)$$

This way the magnitudes of the complex values form the values are from the DMD reconstruction are accounted for, while maintaining the important constraints that $X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$, so that none of the pixel intensities are below zero and ensuring that the approximate low-rank and sparse DMD reconstructions are real-values.

Algorithm Development

We use the VideoReader command to load the `ski_drop_low.mp4` and `monte_carlo_low.mp4` file. We create the dt of the video by the function 1/vidFrames, and a time vector by spacing from 0 to 1 to the Length of the video (number of frames) of the video intervals of dt. Using a loop with numFrames times to separate each frame and convert the image to grayscales by rgb2gray command. Then we use `im2double` command to convert the data from unit8 to double. To reduce the computational time, we reshape the data to convert it into columns. After that, we split the data matrix into two separate matrices, we called it X1 and X2. X1 represented the snapshots of data from the first to the last one. X2 represented the data from snapshots frame 2 to the last frame. Our goal is to determine how to get from one snapshots(X1) to the other(X2) by linear transformation. We get rank by using svd to plot the singular value to see how much significant energy it can capture in the system. After find the rank, we set it called r and then truncated by svd function to find the s, v, u matrices contain only 1:r of the original columns. Then we transformed to the low-rank domain by constructing a new variable called S,

S equals to `U' * X2 * V * diag(1./diag(S))`. Hence, we can find eigenvalue and eigenvectors by eig[eV,D] command. Taking the approximate low-rank subspace and do eigen decomposition to the diagonal frequencies. We created DMD modes using `phi = U * eV`. We are able to obtain the `omega` values by taking the log of the eigen values, and then divided it by dt. Calculated the DMD solution, we set the initial condition and solved y0 using `y0 = phi \ X1(:,1)` function. Then set initial values with all zeros, and crete a time dynamics matrix using y0 multiply be the exp(omega*iter). Multiply phi with u_modes to get DMD approximation Xdmd.
Our original data is the sum of low-dank and sparse matrix approximation, which called X1. Then we could subtract the Xdmd from X1 to get X_sprase matrix. In oreder to avoid negative values, we set R as the filter of sparse matrix. We add R to the absolute value of low-rank matrix to get more accurate low-rank matrix. Then the sparse of DMD equals to the original sparse matrix minus R.

Computational Result

Of video ski

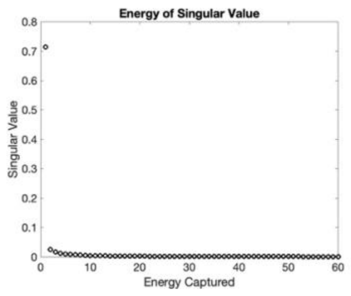

Figure1. Singular Value Spectrum

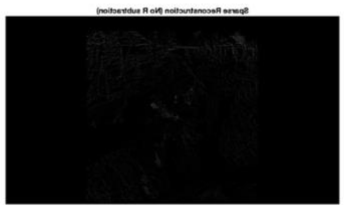

Figure 2. Sparse Reconstruction (No R subtraction)

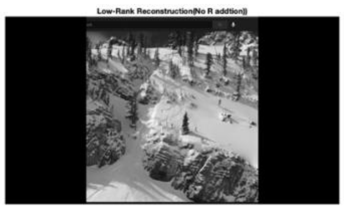

Figure 3. Low-Rank Reconstruction(No R addition)

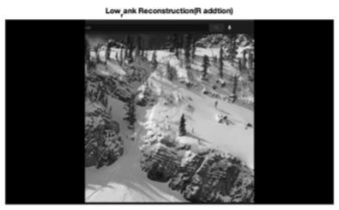

Figure 4. Low-rank Reconstruction(R addition)

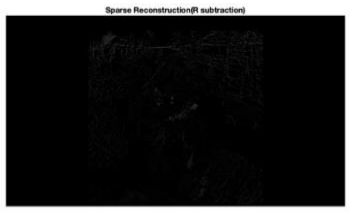

Figure 5. Sparse Reconstruction(R subtraction)

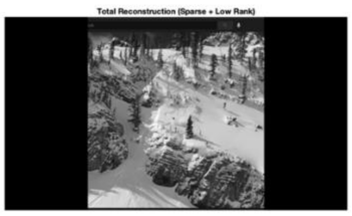

Figure 6. Total Reconstruction (Sparse + Low Rank)

Of video monte_caro

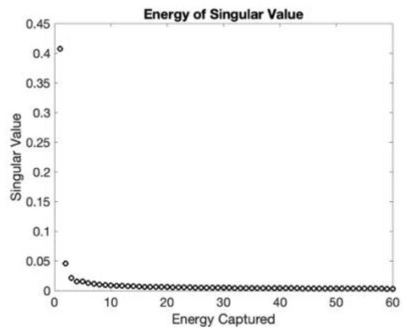

Figure7. Singular Value Spectrum

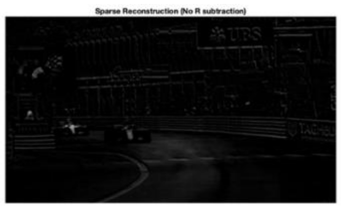

Figure 8. Sparse Reconstruction (No R subtraction)

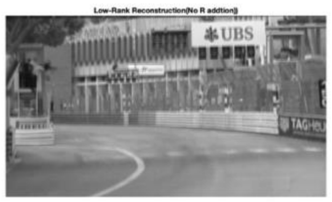

Figure 9. Low-Rank Reconstruction(No R addition)

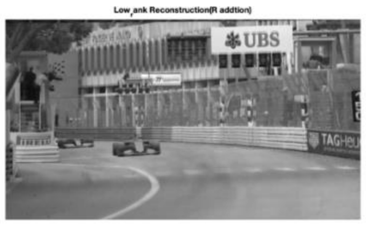

Figure 10. Low-rank Reconstruction(R addition)

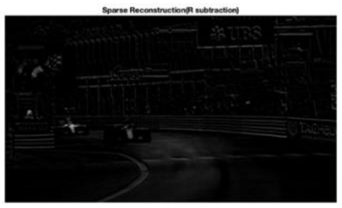

Figure 11. Sparse Reconstruction(R subtraction)

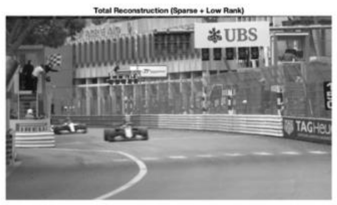

Figure 12. Total Reconstruction (Sparse + Low Rank)

We use almost the same process to analysis these two videos here. First, we consider about the video of ski. For the rank, as the singular value spectrum, we all set rank equals to 1. After that, we set r = 1. And then truncated our u, s, and v by svd command, and set X1 and X2 matrix. Use svd(X1) to get the calculation of low-rank approximation. We see the absolute value of omega value is around 6.56e-4, which is a very small value. It indicates that this is a background mode. We assume there is only 1 significant modes of the background DMD low-rank matrix. For figure 2, we plot the sparse reconstruction, which also represents the foreground. It is similar with the plot of the sparse reconstruction after the R subtraction. Of low rank reconstruction, if we add R, it would not cause too much difference. And for the total reconstruction, it would be more clearly than the tow low-rank reconstruction plots. Same as the plots in `monte_carlo_low.mp4` file.

Summary

For the two recording videos, there are both background and foreground. The foregrounds are moving; however, the backgrounds are more stable. We can use DMD algorithm to create a low-rank and sparse approximation of these data after we reshape the data in columns.
After we use svd to get the rank of the video data, we can find the rank, which also the significant energy they can capture. Based on the same rank, the value of omega would represent the eigen vectors, it is quite important here, since it could distinguish the background. The lower value of the omega represents the more capture of the background. The difference of plot with R or not contain R is not significant.

Appendix A.

- D = diag(v) returns a square diagonal matrix with the elements of vector v on the main diagonal.
- [V,D] = eig(A) returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that A*V = V*D.
- I = rgb2gray(RGB) converts the truecolor image RGB to the grayscale image I.
- Y = double(X) converts the values in X to double precision.
- imshow(I) displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.
- J = im2uint8(I) converts the grayscale, RGB, or binary image I to uint8, rescaling or offsetting the data as necessary.
- If the input image is of class uint8, then the output image is identical. If the input image is of class logical, then im2uint8 changes true-valued elements to 255.
- B = reshape(A,sz) reshapes A using the size vector, sz, to define size(B).
- [U,S,V] = svd(A) performs a singular value decomposition of matrix A, such that A = U*S*V'.

Appendix B.

```
clear all; close all; clc;

%% import videos
vid_ski_drop = VideoReader('ski_drop_low.mp4');
% vid_ski_drop = VideoReader('monte_carlo_low.mp4');

dt = 1/vid_ski_drop.Framerate;
t = 0:dt:vid_ski_drop.Duration;
vidFrames_ski = read(vid_ski_drop);
numFrames_ski = get(vid_ski_drop,'numberOfFrames');

frame = im2double(vidFrames_ski(:,:,:,1));
X = zeros(size(frame,1) * size(frame,2), numFrames_ski);

for j = 1:numFrames_ski
    frame = vidFrames_ski(:,:,:,j);
    frame = im2double(rgb2gray(frame));

    X(:,j) = reshape(frame, 540 * 960, []);
    % imshow(frame); drawnow
end

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

[U, Sigma, V] = svd(X1,'econ');
lambda = diag(Sigma).^2;

% Plot SVD Results
% Singular values1
figure(1)
plot(diag(Sigma),'ko','Linewidth',2)
ylabel('\sigmaj')

% figure(2)
% plot(1:453, lambda/sum(lambda), 'mo', 'Linewidth', 2);
% title("Energy of each Diagonal Variance (Ski-Drop)");
% xlabel("Diagonal Variances"); ylabel("Energy Captured");

%%
r = 1;
U = U(:, 1:r);
Sigma = Sigma(1:r, 1:r);
V = V(:, 1:r);

%%
S = U' * X2 * V * diag(1 ./ diag(Sigma));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu) / dt;
Phi = U * eV;

%% DMD solution
y0 = Phi \ X1(:,1); % pseudoinverse to get initial conditions
umodes = zeros(length(y0), length(t) - 1);
for iter = 1:length(t) - 1
    umodes(:,iter) = y0 .* exp(omega * t(iter));
end
udmd = Phi * umodes;

%% Sparse
X_sparse = X1 - abs(udmd(:,size(udmd, 2)));
neg_vals = X_sparse < 0;
R = X_sparse .* neg_vals;

udmd_new = R + abs(udmd(:,size(udmd, 2)));
X_sparse_new = X_sparse - R;

X_reconstructed = X_sparse_new + udmd_new;

%% Show
temp1 =  reshape(X_sparse, [size(frame, 1), size(frame, 2), length(t) - 1]);
imshow(im2uint8(temp1(:,:,150)))
```

```
title("Sparse Reconstruction (No R subtraction)");

%%
temp2 =  reshape(udmd, [size(frame, 1), size(frame, 2), length(t) - 1]);
imshow(im2uint8(temp2(:,:,150)))
title("Low-Rank Reconstruction(No R addtion))");

%%
temp3 =  reshape(udmd_new, [size(frame, 1), size(frame, 2), length(t) - 1]);
imshow(im2uint8(temp3(:,:,150)))
title("Low_rank Reconstruction(R addtion)");

%%
temp4 =  reshape(X_sparse_new, [size(frame, 1), size(frame, 2), length(t) -
1]);
imshow(im2uint8(temp4(:,:,150)))
title("Sparse Reconstruction(R subtraction)");

%%
temp5 =  reshape(X_reconstructed, [size(frame, 1), size(frame, 2), length(t)
- 1]);
imshow(im2uint8(temp5(:,:,150)))
title("Total Reconstruction (Sparse + Low Rank)");
```