Amath482 Homework5

Kexin Jiao

Abstract:

The homework mainly focuses on the practice of Singular Value Decomposition(SVD) and Dynamic Mode Decomposition(DMD) to solve the actual problem: separating the given videos to the foreground and background videos.

Introduction and overview:

The goal of the assignment is to separate the video stream to both foreground and background videos by DMD. To be specific, I first get the original foreground and background videos by getting the sparse matrix and DMD matrix. Then I removed the negative residual values, put them into R matrix, added and subtracted R from DMD and sparse matrices, making sure the approximate low-rank and sparse DMD reconstruction are real-valued.

Theoretical background:

The Singular Value Decomposition

$A = U\Sigma V^*$ gives a decomposition of the matrix A through the unitary matrices U and V and a diagonal matrix Σ. The U and V matrix are rotation matrices while the Σ is the stretching matrix. The values σ on the diagonal of Σ are singular values; the vectors u(columns of U) are left singular vectors; the vectors v(columns of V) are right singular vectors. When handling non-square matrices of arbitrary size, m*n matrix(m>=n), m-n columns that are orthogonal are added to matrix U to make the decomposition work. m-n rows of zeros is added to the matrix Σ as well. However, the Matlab has a built-in function [U,S,V] = svd(A,'econ') to do the singular value decomposition and produces an economy-size decomposition of m*n matrix A.

Computing the SVD

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

$$= V\Sigma U^*U\Sigma V^* = V\Sigma^2 V^*$$

Multiplying on the right by V gives $A^*AV = V\Sigma^2$ so that the columns of V are the eigenvectors of A^*A with eigenvalues given by the square of the singular values. Multiplying on the right by U gives $A^*AU = U\Sigma^2$ to solve U.

The Dynamic Mode Decomposition

The DMD method approximates the modes of the Koopman Operator. The Koopman operator A is a linear time-independent operator

$$X_{j+1} = AX_j$$

Where the j indicates the specific data collection and A is the linear operator that maps the data from time t_j to t_{j+1}. Applying A to a snapshot of data will advance it forward in time by Δt. The DMD algorithm takes advantage of low dimensionality in the data to create a low-rank approximation of the linear mapping that best iterates through the snapshots of data. To summarize the algorithm:

1. Sample data at N locations M times whose snapshots should be evenly spaced by a fixed Δt and use the snapshots to create the matrix X

2. Construct two submatrices X_1^{M-1} and X_2^M

3. Compute the SVD of X_1^{M-1}

4. Create the matrix $S = U * X_2^M V \Sigma^{-1}$ and find its eigenvalues and eigenvectors

5. Use the initial snapshot X1 and the pseudoinverse of ψ to find the initial condition

6. Compute the solution using DMD modes along with their projection to the initial conditions and the time dynamics computed using eigenvalues of S

(Reference: Amath482 lecture notes from Professor Jason Bramburger)

Algorithm implementation and development:

To set up the data, I first loaded the data by VideoReader method and read the video frames from the file by the read command. Since the video takes too long to run, I resized it to 20% with the use of imresize command. I also got the number of the video frames and time of the video by the NumberOfFrames and Duration commands respectively. At each shift by making a loop, I converted them into a gray scale and double form by rgb2gray and im2double commands and reshaped each frame(image) into a column of a matrix. I found my dt by subtracting the number of video frames from the time(duration) of the video to set the time up. After contracting two submatrices X1 and X2, I applied the singular value decomposition by svd command on X1 to get the U,sigma,V matrices. I then plotted the energy of each singular values so that I can find the significant modes(rank) to represent the image. With the rank, I got my S vector and computed the eigenvalues and eigenvectors by the eig command on S. I extracted the eigenvalues, put it

in log scale, and devided by dt to get the omega so that the DMD solution look more like a solution to a continuous time ODE. I got my phi by multiplying the eigenvectors of S against U. To create the DMD solution, I used the pseudoinverse to get the initial condition, calculated the modes by iterating exponentials y0.*exp(omega*t(iter)), and found the DMD by multiplying phi and modes. To separate the foreground and background videos, I first found the sparse matrix by subtracting the absolute of DMD solutions from X1 and the R matrix of residual negative values by filtering the negative values. Then, I got the reconstructed foreground video by subtracting R from the sparse matrix, the reconstructed background video by adding R to the absolute DMD, and the reconstructed matrix by adding the foreground and the background videos back together. Finally, I reshaped the matrices back to the 3D original form, picked a frame number, and displayed the separated foreground and background videos by imshow command at the picked point to display. I did the same procedure for all videos.

Computational results:

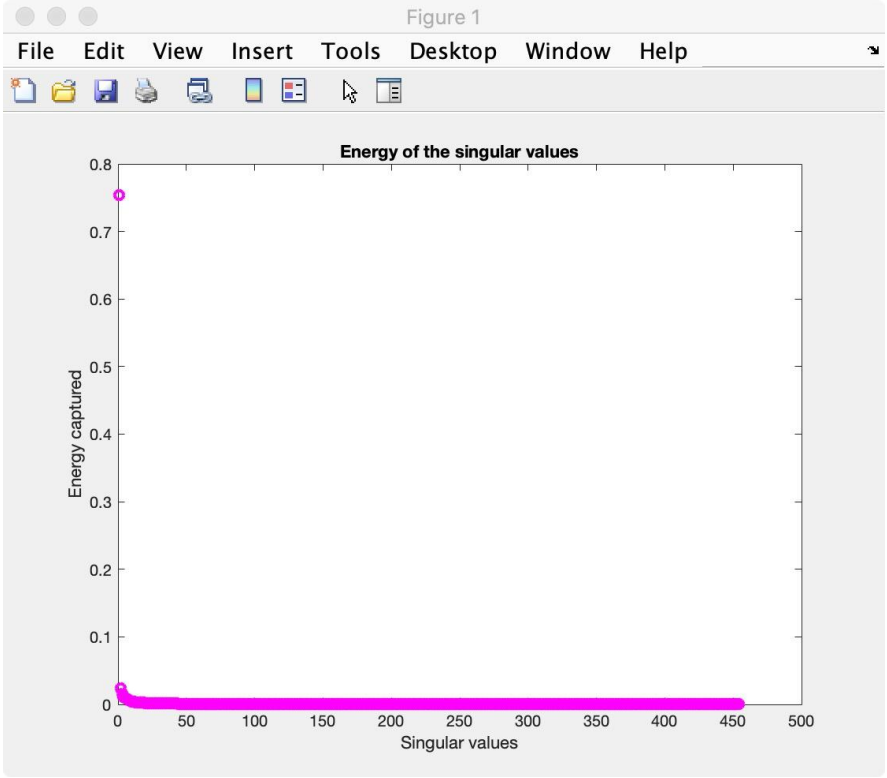

Figure 1: This figure displays the energy of the singular values of the ski drop video.
We can see that the first rank is significant that already captured more than 70 percent of the energy, so I use rank 1 as significant mode to represent the image.

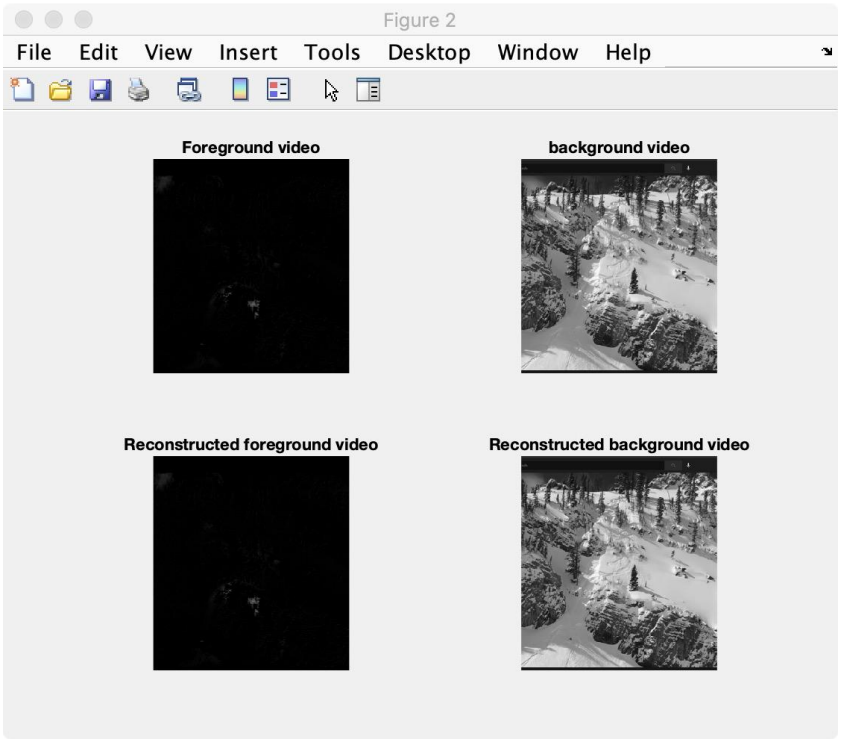

Figure 2: This figure displays the separation and reconstruction of the foreground and background videos at the 320th video frame of ski drop video.

To reconstruct, I extracted the residual negative values from R. I subtracted R from the sparse matrix to get the reconstructed foreground video, while I added R to the absolute DMD to get the reconstructed background video.

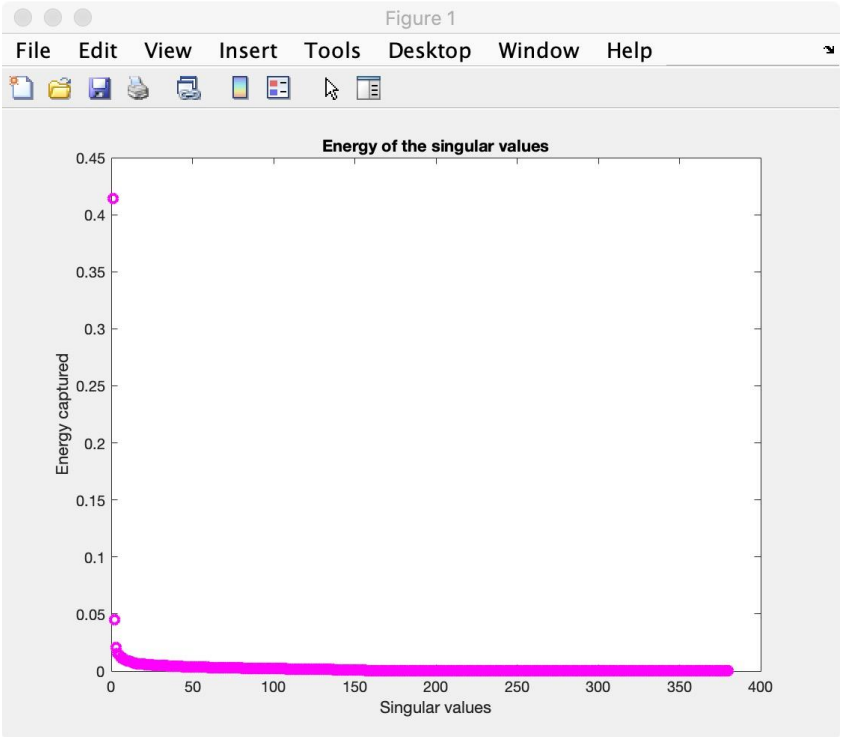

Figure 3: This figure displays the energy of the singular values of monte carlo video.

I set a threshold at 80% to get the number of modes that are necessary for a good image reconstruction. The rank turns out to 60.

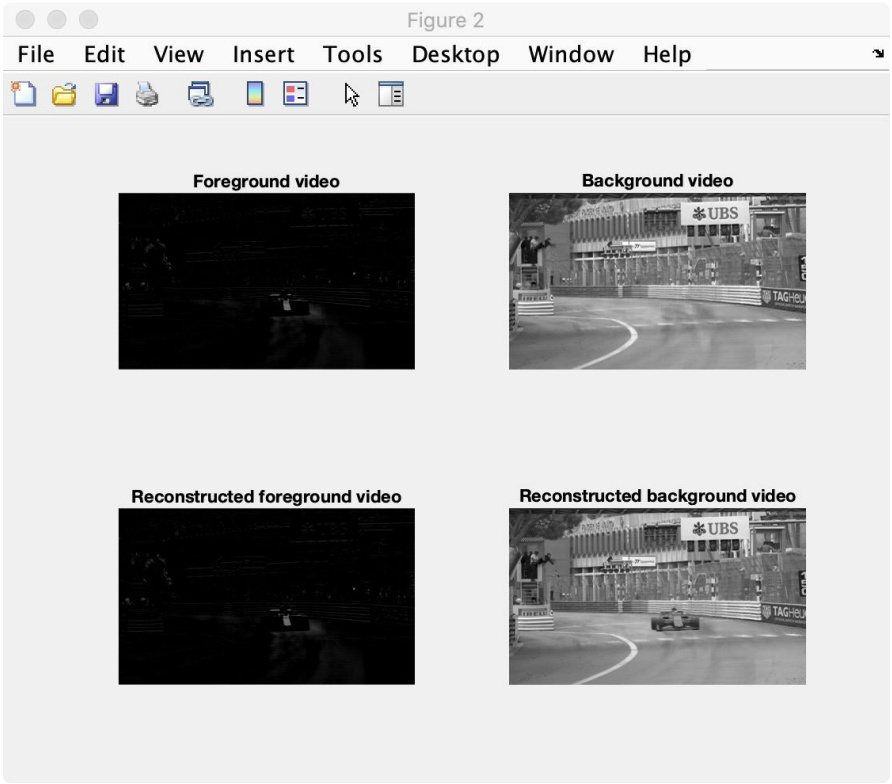

Figure 4: This figure displays the separation and reconstruction of the foreground and background videos at the 380th video frame of monte carlo video.

Similarly, to reconstruct, I eliminated the residual negative values from R. I subtracted R from the sparse matrix to get the reconstructed foreground video, while I added R to the absolute DMD to get the reconstructed background video.

Summary and conclusions:

After doing the singular value decomposition, I plotted the energy of the singular values, found the significant modes, and approximated the low-rank sparse and DMD to represent(separate) the foreground and background videos. I subtracted and added the calculated R matrix which eliminated the negative residual values from the sparse and DMD to get the reconstructed foreground and background videos.

MATLAB function used and brief implementation explanation:

-VideoReader: create object to read video files
-NumberOfFrames: get the number of frames
-read: read one or more video frames
-imresize: rescale/resize the matrix
-rgb2gray: convert RGB image to grayscale
-Duration: get the duration/time of the video
-size: return the length of the dimension
-find: return indices and values of nonzero elements

-im2double: convert image to double precision
-reshape: reshape matrix A into a sz1by...by szN array
-[U,S,V] = svd(A,'econ'): perform a singular value decomposition and produce an economy-size decomposition
-diag: get diagonal elements of matrix
-plot: plot multiple XY pairs using the same axes
-subplot(m,n,p): divide the current figure into m*n grid and create axes in the position p
-imshow: display image
-eig: return a column vector containing the generalized eigenvalues of square matrices A and B.
-set: set graphics object properties
(Reference: MATLAB search helps)

MATLAB codes:

```
 clear all; close all; clc

video = VideoReader('ski_drop.mov');
numFrames = video.NumberOfFrames;
vidFrames = read(video);
vidFrames_resize = imresize(vidFrames,0.2);
for j = 1:numFrames
    X = vidFrames_resize(:,:,:,j);
    X_gray = rgb2gray(X);
    X_double = im2double(X_gray);
    X_reshape(:,j) = reshape(X_double,[],1);
%       imshow(X); drawnow
end

duration = video.Duration;
% dt = duration/numFrames;
dt = 1/video.Framerate;
t = 0:dt:duration;

X1 = X_reshape(:,1:end-1);
X2 = X_reshape(:,2:end);
[U, Sigma, V] = svd(X1,'econ');
figure(1)
plot(diag(Sigma)/sum(diag(Sigma)),'mo','Linewidth',2)
ylabel("Energy captured");
xlabel("Singular values");
title("Energy of the singular values");
```

```
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;

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

Xspar = X1 - abs(u_dmd);
trueorfalse = Xspar < 0;
R = Xspar.*trueorfalse;
fore = Xspar - R;
back = R + abs(u_dmd);
Xnew = fore + back;

figure(2)
sizevid = size(X_double);
subplot(2,2,1)
pic_spar = reshape(Xspar,sizevid(1),sizevid(2),454);
imshow(pic_spar(:,:,320))
title("Foreground video");
subplot(2,2,2)
pic_dmd = reshape(u_dmd,sizevid(1),sizevid(2),454);
imshow(pic_dmd(:,:,320))
title("background video");
subplot(2,2,3)
pic_fore = reshape(fore,sizevid(1),sizevid(2),454);
imshow(pic_fore(:,:,320))
title("Reconstructed foreground video");
subplot(2,2,4)
pic_back = reshape(back,sizevid(1),sizevid(2),454);
imshow(pic_back(:,:,320))
title("Reconstructed background video");
```

```
clear all; close all; clc

video = VideoReader('monte_carlo.mov');
numFrames = video.NumberOfFrames;
vidFrames = read(video);
vidFrames_resize = imresize(vidFrames,0.2);
for j = 1:numFrames
    X = vidFrames_resize(:,:,:,j);
    X_gray = rgb2gray(X);
    X_double = im2double(X_gray);
    X_reshape(:,j) = reshape(X_double,[],1);
    imshow(X); drawnow
end

duration = video.Duration;
dt = duration/numFrames;
dt = 1/video.Framerate;
t = 0:dt:duration;

X1 = X_reshape(:,1:end-1);
X2 = X_reshape(:,2:end);
[U, Sigma, V] = svd(X1,'econ');
figure(1)
plot(diag(Sigma)/sum(diag(Sigma)),'mo','Linewidth',2)
ylabel("Energy captured");
xlabel("Singular values");
title("Energy of the singular values");

lambda = diag(Sigma);
threshold = 0.8;
energy = 0;
r = 0;
while energy <= threshold
    r = r+1;
    energy = energy+lambda(r)/sum(lambda);
end

% U = U(:,1:r);
% Sigma = Sigma(1:r,1:r);
% V = V(:,1:r);
% S = U'*X2*V*(1./diag(Sigma));
S = U(:,1:r)'*X2*V(:,1:r)/Sigma(1:r,1:r);
```

```
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U(:,1:r)*eV;

y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(t)-2);
for iter = 1:(length(t)-2)
   u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;

Xspar = X1 - abs(u_dmd);
trueorfalse = Xspar < 0;
R = Xspar.*trueorfalse;
fore = Xspar - R;
back = R + abs(u_dmd);
Xnew = fore + back;

figure(2)
sizevid = size(X_double);
subplot(2,2,1)
pic_spar = reshape(Xspar,sizevid(1),sizevid(2),380);
imshow(pic_spar(:,:,370))
title("Foreground video");
subplot(2,2,2)
pic_dmd = reshape(u_dmd,sizevid(1),sizevid(2),380);
imshow(pic_dmd(:,:,370))
title("Background video");
subplot(2,2,3)
pic_fore = reshape(fore,sizevid(1),sizevid(2),380);
imshow(pic_fore(:,:,370))
title("Reconstructed foreground video");
subplot(2,2,4)
pic_back = reshape(back,sizevid(1),sizevid(2),380);
imshow(pic_back(:,:,370))
title("Reconstructed background video");
```