AMATH 482 Homework 5

Ariel Luo

Mar 15, 2020

Abstract

We are going to separate 2 videos into foreground videos which contain the foreground object and background videos which contain the background object by performing the Dynamic Mode Decomposition method.

1 Introduction and Overview

In this homework, we are working with 2 videos. From a couple of previous homework, we have been using model-based algorithms to analyze high-dimension data, for example SVD and PCA. For algorithms like these, we have the governing equations which give us the model. For this homework, we are going to use something new called the data-based algorithm. Our goal is to separate the given videos into foreground videos and background videos by tracking the object using the Dynamic Mode Decomposition.

2 Theoretical Background

As we learned from our lectures, when we have problems formulating the governing equations, we can use data-based algorithms. Dynamic mode decomposition is a data-based algorithm that requires no governing equations. It uses snapshots of experiments to predict/control a system. After performing the DMD method, we will get a set of dynamic modes from the snapshots. We are going to write the snapshots as

$$X = [U(x,t_1)U(x,t_2)U(x,t_3)...U(x,t_m)] \tag{1}$$

X is an N x M matrix where N is the number of spatial points saved per time snapshot and M is the number of snapshots taken and

$$X_j^k = [U(x,t_j)U(x,t_j+1)...U(x,t_k)] \tag{2}$$

where j, k are columns in X The DMD method approximates the modes of the Koopman operator. It is a linear, infinite dimensional, time-independent operator A which maps data from t_j to t_{j+1} and such that

$$x_{j+1} = Ax_j \tag{3}$$

when we apply A to a snapshot, we will advance it forward in time by Δt. To get the Koopman operator that represents the data, we will use

$$X_1^{M-1} = [x_1 Ax_1 A^2 x_1 ... A^{M-2} x_1] \tag{4}$$

we can write this as:

$$X_2^M = AX_1^{M-1} + re_{M-1}^T \tag{5}$$

where e_{M-1} is the (M-1)th unit vector. Then we can compute the lower-rank matrix $\tilde{S}$ where

$$\tilde{S} = U * X_2^M V \Sigma^{-1} \tag{6}$$

Since we are going to need the eigenvalues and eigenvectors to construct the DMD modes, we are going to use the fact that $\tilde{S}$ is similar to A which means that A has eigenvectors Uy_k if y_ks are eigenvectors of $\tilde{S}$. Using this, we cna get the approximate solution at all future times which is

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k U y_k e^{\omega_k t} \tag{7}$$

3 Algorithm Implementation and Development

Algorithm 1: Sampling data/Find X and construct X_2^M and X_1^{M-1}

Load data

Algorithm 2: SVD and find $\tilde{S}$ and its eigenvalues and vectors

Algorithm 3: Reconstruct DMD's approximate low-rank and approximate sparse

4 Computational Results

5 Summary and Conclusions

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation.

- `sz = size(A)` returns a row vector whose elements are the lengths of the corresponding dimensions of A. For example, if A is a 3-by-4 matrix, then `size(A)` returns the vector `[3 4]`.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that `A = U*S*V'`.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector v on the main diagonal.
- `Y = fft(X)` computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm.
- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `data = read(ds)` returns data from a datastore. Subsequent calls to the read function continue reading from the endpoint of the previous call.

Appendix B MATLAB Code

```
vidObj=VideoReader('ski_drop_low.mp4')
mov=read(vidObj);

%% Code from lecture 27
% ut=fft(u);
% [t, utsol] = ode45(@(t,y) nls_rhs(t,y,k),t,ut);
% for j = 1:length(t)
%     usol(j,:) = ifft(utsol(j,:)); % back to x-space
% end
% X = usol';
% X1 = X(:,1:end-1);
```

```
% X2 = X(:,2:end);
% [U, Sigma, V] = svd(X1,'econ');
% 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;
```

Code for HW5