Homework 5: Background Subtraction in Video Streams

Emma Nuss, github: emmashie

Abstract

Through the use of the theory of dynamic mode decomposition (DMD), moving foregrounds were separated from stationary backgrounds in two videos. The DMD reconstruction proved to be more difficult for the video of a skier going down a mountainside, but was quite successful at separating a race car from the stationary race track.

1. Introduction

In this project we have two video clips, one of race cars driving around a track and one of a skier skiing down a mountain slope. The goal of this project is to apply the theory of dynamic mode decomposition to decompose the videos into a foreground (the moving component) and a background (the stationary component). The theory of dynamic mode decomposition and its application to this problem is described in the following sections.

2. Theoretical Background

2.1. Dynamic Mode Decomposition

Dynamic mode decomposition (DMD) is a data analysis technique that relies on reducing the dimensionality of data. DMD reconstruction takes a series of data images, evenly spaced through time, and decomposes the data into dynamic modes, or foreground and background modes (Kutz, 2013). These images can be compiled into a data matrix (X), where each column is a 1-D array of the spatial information from a single image. Subsets of the matrix X are then constructed where:

$$X_1^{M-1} = [x_1\ x_2\ ...\ x_{M-1}] \tag{2.1}$$

$$X_2^{M} = [x_2\ x_3\ ...\ x_{M}] \tag{2.2}$$

where M is the number of images. A low-rank matrix ($\tilde{A}$) is computed by using a finite number of modes by:

$$\tilde{A} = U^T X_2^M V \Sigma^{-1} \tag{2.3}$$

where U, V, and Σ are from the singular value decomposition of X_1^{M-1}. The eigenvectors and eigenvalues of $\tilde{A}$ are computed and φ is constructed by:

$$\varphi = X_2^M V \Sigma^{-1} eV \tag{2.4}$$

where eV are the eigenvectors. Using φ, the initial conditions, and the modes of the DMD, you can decompose the system into background and foreground components by:

$$X_{background} = b_p \varphi_p e^{\omega_p \mathbf{t}} \tag{2.5}$$

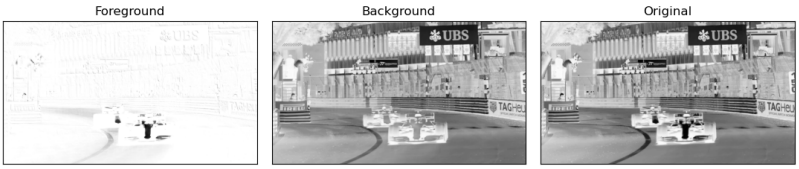

Figure 1: A random snapshot of the foreground, background, and orginial frame from the DMD reconstruction for the Monte Carlo Race video.

$$X_{foreground} = X - |X_{background}| \tag{2.6}$$

where b_p is computed by the least-squares system with φ and X_1^{M-1}, ω_p are the frequencies present in the system computed from the eigenvalues of φ, and $\mathbf{t}$ is the time vector.

3. Algorithm Implementation and Development

Videos were loaded into Python and converted into greyscale. The video matrices were then reshaped so that each column of the data matrix was pixel intensity information for each image. The reshaped video matrix was split into the two sub-matrices described in Equations 2.1 and 2.2.

Next the singular value decomposition was computed for the data matrix X_1^{M-1}. The spectrum of the singular value decomposition was examined to determine the number of modes to use in the DMD. For both videos, the first three modes were chosen since they made up a significant amount of the variance. The output of the singular value decomposition was then used to compute $\tilde{A}$ using Equation 2.3, using only the first three modes.

The eigenvalues and eigenvectors of $\tilde{A}$ were computed and then φ was calculated using Equation 2.4. The vector b_p was computed from the least-squares problem with φ and X_1^{M-1}. The frequencies were computed as the log of the eigenvalues divided by the timestep between video frames. The last step was to compute the background and foreground matrices using Equations 2.5 and 2.6.

Lastly, to deal with negative values in the foreground matrix, a residual matrix was calculated. The residual matrix consisted of the negative values in the foreground matrix with zeros set for all positive values. This residual was then added to the absolute value of the background matrix and subtracted from the foreground matrix to ensure that there were no negative values.

4. Computational Results

4.1. Monte Carlo Race Video

The DMD reconstruction of the Monte Carlo Race video was relatively successful at decomposing the race cars from the stationary background (Fig. 1). In the foreground component of the DMD reconstruction the race car can be seen visually and the race car appears to be muted than compared with the original (Fig. 1).

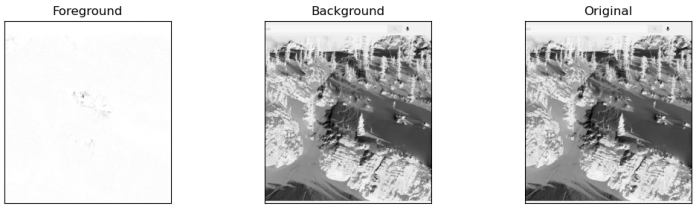

Figure 2: A random snapshot of the foreground, background, and orginial frame from the DMD reconstruction for the ski drop video.

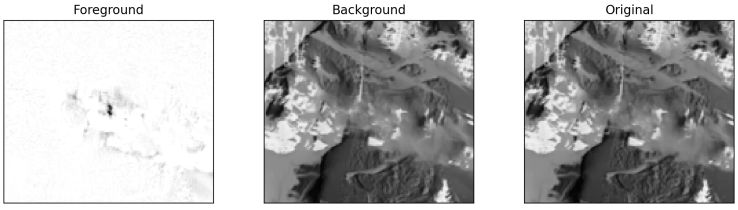

Figure 3: A random snapshot of the foreground, background, and orginial frame from the DMD reconstruction for the ski drop video zoomed into the region where the skier is located.

4.2. Ski Drop Video

The DMD reconstruction of the ski drop video was not as successful as the Monte Carlo Race video at decomposing the skier from the mountainside. The background is generally removed well from the foreground (Fig. 2); however the skier is hard to identify. Zooming into a smaller region around where the skier is located allows us to see how well or not the foreground captures the skier. The zoomed in figures shows that the foreground appears to have identified something that may be the skier (Fig. 3); however, when examining all frames, this is not true at all time steps.

5. Summary and Conclusions

DMD reconstruction is a data analysis tool that can be used to decompose a series of videos into a moving, foreground component, and a stationary, background component. This tool was applied to two videos, a video of race cars around a track and a skier skiing down a slope. The DMD reconstruction worked better with the race care video than the skier video. The skier may be harder to isolate from the background due to the fact that it is very small compared to the background and the snow around the skier also moves a bit from frame to frame. Despite this difficulty, this method shows that it can be useful in similar data analysis scenarios.

References

J. N. Kutz, Data-driven modeling & scientific computation: methods for complex systems & big data, Oxford University Press, 2013.

Appendix A Python Functions

Relevant Python functions:

- `[U, Sdiag, VH] = np.linalg.svd(X)`: Computes the SVD of matrix X where Sdiag is the diagonal components of Σ and VH is the transpose of $\mathbf{V}$
- `[D, eV] = scipy.linalg.eig(A)`: Computes the eigenvalues (D) and eigenvectors (eV) of matrix A
- `b = np.linalg.lstsq(A, x)[0]`: Equivalent to b = A\ x in Matlab
- `x = np.arange(0, n)`: Creates a 1D array of integers starting at 0 up to `n-1`
- `z = np.append(x, y)`: Appends array y to the end of array x and stores it in variable z
- `ind = np.argwhere(D == condition) or np.where(D == condition)`: Finds the indices of an array where a logical condition is satisfied
- `np.asarray(D)`: Takes a list or some non-array and converts it to an array type
- `L = [i for i in range(n)]`: (List comprehension) Creates a list of `0` to `n-1`, can be used for high dimensions of lists and with if conditional statements
- `os.path.join(path, filename)`: Creates a path object based on the `path` and `filename` given

Appendix B Python Code

Code can be found at: https://github.com/emmashie/amath-582

```
#!/usr/bin/env python
""" functions used for AMATH 582 hw3
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy import linalg

def norm(x):
        return x/np.nanmax(x)

def rgb2grey(rgb):
    """ convert rgb data to greyscale
    """
    grey = 0.2989 * rgb[:,:,0] + 0.5870 * rgb[:,:,1] + 0.1140 * rgb[:,:,2]
    return norm(grey)

def video_svd(video, duration):
        time = np.linspace(0, duration, len(video))
        dt = np.diff(time)[0]
        video_grey = np.asarray([rgb2grey(video[i,:,:,:]) for i in range(len(video))])
        [nimgs, nx, ny] = np.shape(video_grey)
        X = np.reshape(video_grey, (nimgs, nx*ny), order='F').T
        X1 = X[:,:-1]
        X2 = X[:,1:]
        U, Sdiag, VH = np.linalg.svd(X1, full_matrices=False)
        V = VH.T
        return X, X1, X2, U, Sdiag, V, nimgs, nx, ny

def mode_appoximation(X2, U, Sdiag, V, modes):
        Atilde = U[:,:modes].T@X2@V[:,:modes]@np.diag(1/Sdiag[:modes])
        [D, eV] = linalg.eig(Atilde)
        Phi = X2@V[:,:modes]@np.diag(1/Sdiag[:modes])@eV
        return D, Phi

def reconstruct_DMD(D, X, X1, Phi, time, nx, ny, nimgs):
        omega = np.log(D)/np.diff(time)[0]
        b = np.linalg.lstsq(Phi, X1[:,0], rcond=None)[0]
        xmodes = np.asarray([b*np.exp(omega*time[i]) for i in range(len(time))])
        X_lr = Phi@xmodes.T
        X_sp = X - np.abs(X_lr)
        R = X - np.abs(X_lr)
        R[R>=0] = 0
        X_lr_R = R + np.abs(X_lr)
        X_sp_R = X_sp - R
        foreground = np.reshape(X_sp_R, (nx, ny, nimgs), order='F')
        background = np.reshape(X_lr_R, (nx, ny, nimgs), order='F')
        return foreground, background

import numpy as np
import matplotlib.pyplot as plt
import skvideo.io
import os
import hw5_functions as f
from scipy import linalg
plt.ion()
#plt.style.use('ggplot')
```

```
### race video
filepath = os.path.join('../data','monte_carlo_low.mp4')
mc = skvideo.io.vread(filepath)
meta_mc = skvideo.io.ffprobe(filepath)
T = 6.322983
time = np.linspace(0, T, len(mc))

X, X1, X2, U, Sdiag, V, nimgs, nx, ny = f.video_svd(mc, T)
D, Phi = f.mode_appoximation(X2, U, Sdiag, V, modes=3)
foreground, background = f.reconstruct_DMD(D, X, X1, Phi, time, nx, ny, nimgs)

mc_grey = np.reshape(X, (nx,ny,nimgs), order='F')

n = 50
fig, ax = plt.subplots(ncols=3, figsize=(11,3))
ax[0].imshow(foreground[:,:,n], cmap='Greys')
ax[0].get_xaxis().set_visible(False)
ax[0].get_yaxis().set_visible(False)
ax[0].set_title('Foreground')

ax[1].imshow(background[:,:,n], cmap='Greys')
ax[1].get_xaxis().set_visible(False)
ax[1].get_yaxis().set_visible(False)
ax[1].set_title('Background')

ax[2].imshow(mc_grey[:,:,n], cmap='Greys')
ax[2].get_xaxis().set_visible(False)
ax[2].get_yaxis().set_visible(False)
ax[2].set_title('Original')
fig.tight_layout()
fig.savefig('monte_carlo.png')

### ski video
filepath = os.path.join('../data','ski_drop_low.mp4')
ski = skvideo.io.vread(filepath)
meta_ski = skvideo.io.ffprobe(filepath)
T = 7.574233
time = np.linspace(0, T, len(ski))

X, X1, X2, U, Sdiag, V, nimgs, nx, ny = f.video_svd(ski, T)
D, Phi = f.mode_appoximation(X2, U, Sdiag, V, modes=3)
foreground, background = f.reconstruct_DMD(D, X, X1, Phi, time, nx, ny, nimgs)

ski_grey = np.reshape(X, (nx,ny,nimgs), order='F')

n = 50
fig, ax = plt.subplots(ncols=3, figsize=(11,3), sharex=True, sharey=True)
ax[0].imshow(foreground[:,:,n], cmap='Greys')
ax[0].get_xaxis().set_visible(False)
ax[0].get_yaxis().set_visible(False)
ax[0].set_title('Foreground')
ax[0].set_xlim(232,725)

ax[1].imshow(background[:,:,n], cmap='Greys')
ax[1].get_xaxis().set_visible(False)
ax[1].get_yaxis().set_visible(False)
ax[1].set_title('Background')
ax[1].set_xlim(232,725)
```

```
ax[2].imshow(ski_grey[:,:,n], cmap='Greys')
ax[2].get_xaxis().set_visible(False)
ax[2].get_yaxis().set_visible(False)
ax[2].set_title('Original')
ax[2].set_xlim(232,725)
fig.tight_layout()
fig.savefig('ski.png')
```