AMATH 582 Homework 5: Background Subtraction in Video Streams

Brady Griffith

Abstract

In this project, an attempt is made to use DMD to separate the background in two videos. This is done by taking the lowest frequency DMD mode to be the background and leaving all others as the foreground.

Introduction and Overview

I've been working hard on the final, so I'll keep this write up succinct. In this project, I will apply DMD to separate the foreground from the background in two videos.

Theoretical Background

The full derivation is laid out in chapter 20 of the textbook of this course [1], so I will only repeat the points critical to implementation.

$$\mathbf{X}_1^{M-1} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{1}$$

Then find Schmid's approximation for the Koopman operator

$$\tilde{\mathbf{S}} = \mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\mathbf{\Sigma}^{-1} \tag{2}$$

And find its eigenvector and eigenvalues.

$$\tilde{\mathbf{S}}\mathbf{y}_k = \mu_k\mathbf{y}_k \tag{3}$$

And from that, get the DMD modes

$$\psi_k = \mathbf{U}\mathbf{y}_k \tag{4}$$

Which can approximate vector of each frame as

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k(0)\psi_k(\mathbf{x})\exp(\omega t) \tag{5}$$

Then to separate out the background, I find the DMD mode p with the smallest $\|\omega_p\|$.

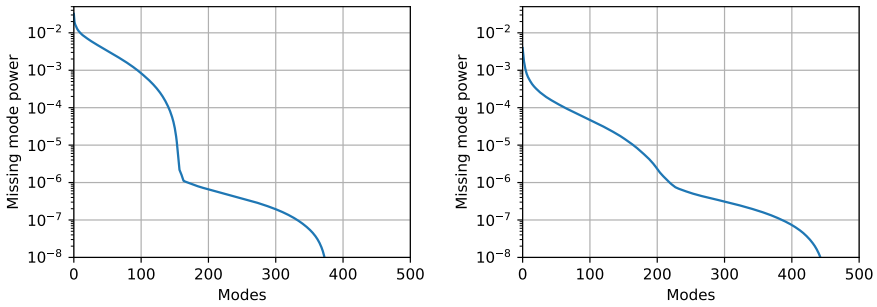

Figure 1: The fraction of the total covariance for the modes calculated that is encompassed with n modes. Left for the Monte Carlo video and right for the ski drop video.

Algorithm Implementation and Development

In order to make the problem fit into the memory available on my computer, I only load in every third pixel in both the x and y direction. I also average the color channels to product a monochrome image.

When calculating the SVD, I make use of the full_matrices=False argument in the scipy SVD function. The full SVD would swamp my computer by a factor of a few thousand. I make sure this substantially reduced dimensionality still accurately represents the data.

Computational Results

First, I consider the effects of reducing the dimensionality of the data. I only find the first M (Where M is the number of pixels) modes. In fig 1 and 2 you can see that the projection is more than suitable for this analysis.

I then preformed the DMD as described in the book. There seems to be some persistent bug I couldn't track down. In figure 3, I show a view of the output that should be background. It's seems to be captured correctly by the result is scalled by a factor of around a million. This breaks the calculation to find the foreground described in the assignment prompt. The S also ends up being singular, so several of the eigenvalues are 0. Everything seems to be implemented the same as the example code, so the issue could be in a subtle quirk of the python packages I used that I'm not familiar with. But debugging had to halt on time grounds.

The code that would generate the split videos is written but commented out, since I know the input would be nonsense.

Figure 2: For each video I take one frame and project it into the reduced dimensionality SVD modes and then back. The top is the original for reference.

Summary and Conclusions

In this project, I tried to use DMD to separate a foreground from a background in a video. I wouldn't classify this as a success, but with more time for debugging, I'm sure I could have had this working. This technique seems like it would be applicable to research I've done, specifically when I could have used a way to interpolate between weather radar images.

References

[1] J. N. Kutz, *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford: Oxford University Press, first edition ed., 2013. OCLC: ocn858608087.

Figure 3: The background calculated, scaled so the range of values matches a black to white scale.

Here is a link to the Github repository for this project.

Python Functions

dmd

dmd

`dmd(X2, u, s, vh)`

Returns the DMD modes and complex frequencies for the system.

Arguments:

- `X2` *array-like* - The X_2^M matrix
- `u` *array-like* - The U matrix of the X_1^M-1 SVD
- `s` *array-like* - The s array of the X_1^M-1 SVD
- `vh` *array-like* - The vh matrix of the X_1^M-1 SVD

Returns:

- `ndarray` - Array of complex frequencies for DMD modes
- `ndarray` - Matrix with rows of the DMD modes

x_dmd

`x_dmd(t, psi, w, b)`

The DMD approximation of x(t).

Arguments:

- `t` *float* - The time in frames
- `psi` *array-like* - Matrix with rows of the DMD modes
- `w` *array-like* - Array of complex frequencies for DMD modes
- `b` *array-like* - Array of initial values of the DMD modes

Returns:

- `ndarray` - DMD approximation of pixels at t

frame_bg_sep

`frame_bg_sep(t, X, psi, w, b)`

Separate the forground and background of frame t

Arguments:

- `t` *int* - The time in frames
- `psi` *array-like* - Matrix with rows of the DMD modes
- `w` *array-like* - Array of complex frequencies for DMD modes
- `b` *array-like* - Array of initial values of the DMD modes

Returns:

- `ndarray` - Foreground array
- `ndarray` - Background array

show_frame

```
show_frame(frame, shape, path_out)
```

Plot the frame provided.

Arguments:

- `frame` *array-like* - 1 D array of pixels
- `shape` *tuple* - The shape of the image (pixels_y, pixels_x)
- `path_out` *str* - Path to save figure to

svd

plot_n_modes

```
plot_n_modes(X, V, n, shape, path_out)
```

Shows the numbers represented with the selected number of SVD modes

Arguments:

- `X` *array_like* - Data matrix with rows of images
- `V` *array_like* - Matrix with mode vectors as columns
- `n` *int* - Number of modes to use in the representation
- `shape` *tuple* - The shape of the image (pixels_y, pixels_x)
- `path_out` *str* - Path to save figure to

plot_mode_fraction

```
plot_mode_fraction(s, path_out)
```

Plots the fraction of power represented with n modes

Arguments:

- `s` *array-like* - 1D arrray of the variances of the principal components.
- `path_out` *str* - Path to save figure to

loadVid

open_video

```
open_video(vid_path)
```

Loads the video matrices

Arguments:

- `vid_path` *str* - Path to the video file

Returns:

- `ndarray` - X_1^M-1
- `ndarray` - X_2^M

Python Code

main.py

```
import loadVid
import svd
import dmd
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt


def run_analysis():
    vids = ['monte_carlo.mov', 'ski_drop.mov']
    shapes = [(370, 624), (534, 488)]

    for vid, shape in zip(vids, shapes):
        X1, X2 = loadVid.open_video('HW5/data/' + vid)

        u, s, vh = linalg.svd(X1, full_matrices=False)
        svd.plot_n_modes(X1.T, u.T, vh.shape[0], shape,
                         'HW5/figures/' + vid[:-4] + '-rd.png')

        svd.plot_mode_fraction(s, 'HW5/figures/' + vid[:-4] + '-modes.pdf')

        w, psi = dmd.dmd(X2, u, s, vh)
        b, _, _, _ = np.linalg.lstsq(psi, X1[:, 0])

        fg, bg = dmd.frame_bg_sep(5, X1, psi, w, b)
        dmd.show_frame(fg, shape, 'HW5/figures/' + vid[:-4] + '-fg.pdf')
        dmd.show_frame(bg, shape, 'HW5/figures/' + vid[:-4] + '-bg.pdf')
        # dmd.save_fgbg_videos(X1, psi, w, b,
        #                      'HW5/figures/' + vids[:-4] + '-vid.avi')


if __name__ == '__main__':
    run_analysis()
```

loadVid.py

```
import numpy as np
import av
from sklearn.utils import extmath
import matplotlib.pyplot as plt


def open_video(vid_path):
    """Loads the video matrices

    Args:
        vid_path (str): Path to the video file

    Returns:
        ndarray: X_1^M-1
        ndarray: X_2^M
    """
    v = av.open(vid_path)
    total_frames = v.streams.video[0].frames
    pixels = (v.streams.video[0].format.height // 3) * \
             (v.streams.video[0].format.width // 3)

    X = np.zeros((total_frames, pixels))
    i = 0
    for packet in v.demux():
        for frame in packet.decode():
            img = np.array(frame.to_image(), dtype=np.float64)[2::3, 2::3].mean(2)
            X[i] = img.flat
            i += 1

    return X[:-1].T.copy(), X[1:].T.copy()
```

svd.py

```
import numpy as np
import matplotlib.pyplot as plt


def plot_n_modes(X, V, n, shape, path_out):
    """Shows the numbers represented with the selected number of SVD modes

    Args:
        X (array_like): Data matrix with rows of images
        V (array_like): Matrix with mode vectors as columns
        n (int): Number of modes to use in the representation
```

```
        shape (tuple): The shape of the image (pixels_y, pixels_x)
        path_out (str): Path to save figure to
    """
    fig, axs = plt.subplots(2, figsize=(3, 3))

    selected = 10
    Y = np.dot(V, X[selected].T)
    Z = np.dot(V[:n].T, Y[:n]).T

    axs[0].axis('equal')
    axs[0].axis('off')
    axs[0].imshow(np.reshape(X[selected], shape), cmap='Greys_r')

    axs[1].axis('equal')
    axs[1].axis('off')
    axs[1].imshow(np.reshape(Z, shape), cmap='Greys_r')

    fig.savefig(path_out, bbox_inches='tight', dpi=300)


def plot_mode_fraction(s, path_out):
    """Plots the fraction of power represented with n modes

    Args:
        s (array-like): 1D arrray of the variances of the principal components.
        path_out (str): Path to save figure to
    """
    fig, ax = plt.subplots(figsize=(4, 3))
    ax.set_xlim(0, 500)
    ax.set_ylim(1e-8, 5e-2)
    ax.set_yscale('log')
    ax.grid()
    f = 1 - np.cumsum(s**2) / np.sum(s**2)

    ax.set_xlabel('Modes')
    ax.set_ylabel('Missing mode power')

    ax.plot(f)
    fig.savefig(path_out, bbox_inches='tight')
```

dmd.py

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```
def dmd(X2, u, s, vh):
    """Returns the DMD modes and complex frequencies for the system.

    Args:
        X2 (array-like): The X_2^M matrix
        u (array-like): The U matrix of the X_1^M-1 SVD
        s (array-like): The s array of the X_1^M-1 SVD
        vh (array-like): The vh matrix of the X_1^M-1 SVD
    Returns:
        ndarray: Array of complex frequencies for DMD modes
        ndarray: Matrix with rows of the DMD modes
    """
    s_i = np.diagflat(1/s)
    S = np.dot(u.T, np.dot(X2, np.dot(vh.T, s_i)))
    mu, y = np.linalg.eig(S)
    psi = np.dot(u, y)
    w = np.log(mu)
    return w, psi


def x_dmd(t, psi, w, b):
    """The DMD approximation of x(t).

    Args:
        t (float): The time in frames
        psi (array-like): Matrix with rows of the DMD modes
        w (array-like): Array of complex frequencies for DMD modes
        b (array-like): Array of initial values of the DMD modes
    Returns:
        ndarray: DMD approximation of pixels at t
    """
    return np.dot(psi, np.exp(w*t)*b)


def frame_bg_sep(t, X, psi, w, b):
    """Separate the forground and background of frame t

    Args:
        t (int): The time in frames
        psi (array-like): Matrix with rows of the DMD modes
        w (array-like): Array of complex frequencies for DMD modes
        b (array-like): Array of initial values of the DMD modes
    Returns:
        ndarray: Foreground array
        ndarray: Background array
    """
```

```
    k = np.argmin(np.abs(w))
    bg = x_dmd(t, psi[:, k], w[k], b[k])
    bg = np.abs(bg)
    fg = X[:, t] - bg
    # r = np.where(fg < 0, fg, np.zeros_like(fg))
    # fg -= r
    # bg += r
    return fg, bg


def show_frame(frame, shape, path_out):
    """Plot the frame provided.

    Args:
        frame (array-like): 1 D array of pixels
        shape (tuple): The shape of the image (pixels_y, pixels_x)
        path_out (str): Path to save figure to
    """
    fig, ax = plt.subplots()
    ax.axis('off')
    ax.imshow(np.reshape(frame, shape), cmap='Greys_r')
    fig.savefig(path_out,  bbox_inches='tight')


def save_fgbg_videos(X, psi, w, b, shape, path_out):

    fg_out = cv2.VideoWriter(path_out[:-4] + '-fg' + path_out[-4:],
                cv2.VideoWriter_fourcc(*'DIVX'), 30, shape)

    bg_out = cv2.VideoWriter(path_out[:-4] + '-bg' + path_out[-4:],
                cv2.VideoWriter_fourcc(*'DIVX'), 30, shape)

    for i in range(0, X.shape[1]):
        fg, bg = frame_bg_sep(t, X, psi, w, b)
        fg_out.write(np.uint8(fg))
        bg_out.write(np.uint8(bg))

    fg_out.release()
    bg_out.release()
```