AMATH 582 Homework Five: Background Subtraction

Daniel W. Crews

March 10, 2021

Abstract

The dynamic mode decomposition (DMD) analyzes data into spatiotemporally-correlated modes of varying complex frequencies. This work looks at a simple example of the DMD by solving the problem of video background separation. This is accomplished by subtracting the lowest frequency modes from a video clip, with the wrinkle of negative pixel values dealt with in video-dependent ways. Generally negative values can simply be subtracted out. Yet in a video of a skiier, the snow is much brighter than the skiier so subtracting out negative values removes the skiier too. For this clip, the negative pixels are flipped to their absolute values.

1 Introduction and Overview

This work separates the background from two short video clips, one of race cars and the other of a skiier. This is done using the dynamic mode decomposition (DMD) method, which approximates the Koopman operator that evolves the video as a dynamical system. Importantly, both images have little camera wobble, so the singular value decomposition (SVD) is able to identify orthogonal modes of spatiotemporal correlation and isolate the background. The DMD then approximates the Koopman operator on these modes according to a multistep process outlined in Section 2. Having singled out the lowest frequency modes for a pure background reconstruction, this is then subtracted from the video clips to accomplish background separation.

2 Theoretical Background

2.1 Dynamic mode decomposition

In this class, previous works studied the decomposition of a data matrix into its proper modes using the SVD operation. These frames, such as handwriting examples, were not time-dependent. Yet in a video clip (or time series data) each realization is separated from its neighbors by a time difference Δt. If the video is collected into an array $X \in \mathbb{R}^{n\times m}$ of dimension (pixels×time), and a second array X' is formed one difference Δt into the future, then the two may be supposed to be related by a Koopman evolution operator,

$$X' = AX \tag{1}$$

where A is taken to solve the problem in the least-squares Moore-Penrose pseudoinverse Y^+ sense [1]. However, this makes $A = X'X^+$ a (pixels×pixels) operator. Generally speaking this matrix A will be too large to perform an eigenvalue decomposition on directly with any efficiency. On the other hand, dynamic mode decomposition tackles this problem by projecting into a low-rank truncation of X obtained via SVD,

$$X_r = U_r \Sigma_r V_r^\dagger \tag{2}$$

with r the truncation order. Therefore the Koopman operator A may be represented in this subspace,

$$\widetilde{A} \equiv U_r^\dagger A_r U_r = U^\dagger X' V_r \Sigma_r^{-1} \tag{3}$$

with $\widetilde{A} \in \mathbb{C}^{r\times r}$. Having identified an approximate A, a more tractable eigendecomposition is performed

$$\widetilde{A}W = W\Lambda. \tag{4}$$

The orthogonal modes of the low-dimensional evolution operator are then lifted to the full-rank space by

$$\Phi \equiv X' V_r \Sigma_r^{-1} W \tag{5}$$

and the fundamental frequencies computed via $\Omega \equiv \log(\Lambda)/\Delta t$. By this method the DMD has been accomplished according to the approximate continuous-time reconstruction

$$\widetilde{X}(t) = \Phi \exp(\Omega t) b \tag{6}$$

where b is the vector of initial conditions obtained via $b = \Phi^+ X(0)$. This reconstruction is approximate in the sense that high mode dynamics have been disregarded [2].

2.2 Background subtraction method

The above is the standard DMD method. In this work, a slight variation is used to perform the background subtraction. By examination of the eigenvalue spectrum Λ of the low-dimensional operator of Eqn. 4, the lowest frequency modes Λ_L (usually just one or two) are extracted and its reduced eigenvector W_L used to construct the low-frequency DMD operator Φ_L [2]. The method then proceeds as normal. A slight variation on the method presented in Ref. [2] is used, wherein the best-fit background $b(t)$ is identified for each time increment by calculating $b(t_i) = X(t_i)\Phi^+$ rather than exponentially evolving the frequencies. This was found to work better, but they are perhaps equivalent and the author may simply have had a bug.

3 Algorithm Implementation and Development

Each video consists of a number of frames at sixty frames-per-second. These are loaded in, converted to grayscale, and then analyzed by the DMD algorithm of Section 2. For each video the low-rank truncation of X_r used to construct $\widetilde{A}$ was chosen to be 100 modes. The resulting eigenvalue spectrum of of $\widetilde{A}$ is shown in Fig. 1 and 2 for the race cars and skiier respectively. It was found that four race car modes and five skiier ones were sufficient for background reconstruction. Experimentation on the precise number of low-frequency modes was found not to be significant provided that there were enough of them, *but* not too many.

According to the identified eigenvalues, the background was then reconstructed using $X_L(t_i) = \Phi_L b(t_i)$ with each coefficient $b(t_i) = \Phi_L^+ X(t_i)$ found via pseudoinverse. As X_L is complex, one then subtracts the background using the element-wise absolute value of the low-frequency modes per timestep

$$X_S(t_i) = X(t_i) - |X_L(t_i)|. \tag{7}$$

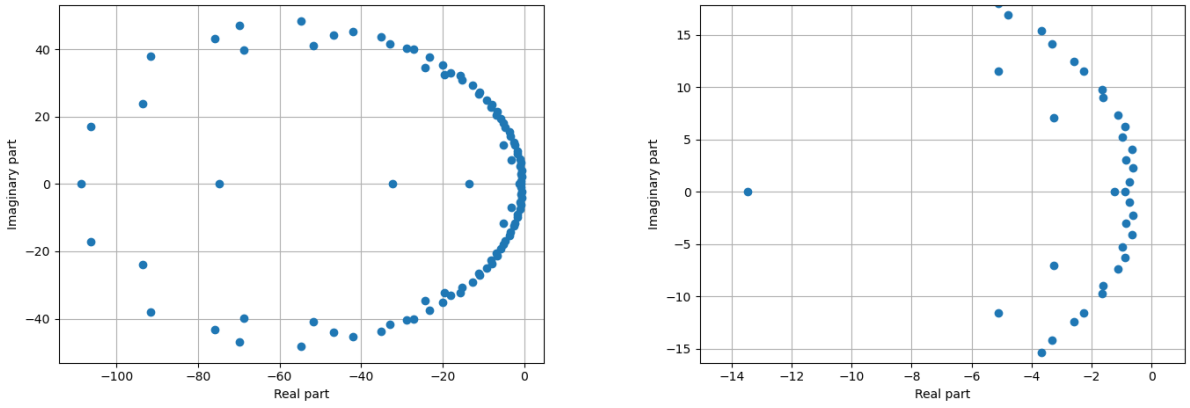

Figure 1: Spectrum of the low-rank approximate race car Koopman operator $\widetilde{A}$ using an $r = 100$ truncation. Evidently the the low-frequency activity is clustered around a cloud of modes. The four lowest modes were found to provide a good low-frequency reconstruction according to the method of Section 2.2.

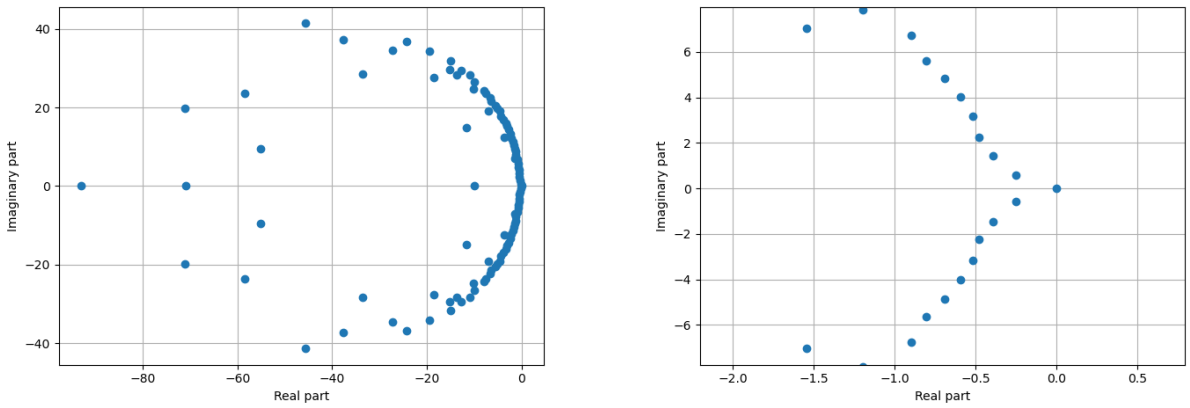

Figure 2: Eigenvalues of the skiier video operator approximated with one hundred modes. Here, the sparse background is observed to occupy mostly one mode, with a far sparser low-frequency spectrum than the race car video. The five smallest modes are used to account for slight variations and background wobble.

4 Computational Results

With the reconstructed backgrounds (seen at $t = 0$ in Fig. 6) the $X_S(t_i)$ were calculated for each video. Different methods were then used to deal with the resulting negative values in the sparse reconstruction X_S. For example, in the skiier video the person skiing is dark compared to the bright snow, so the skiier pixels in Eqn. 7 are very negative. Then to really pull out the skiier against the mostly-zero subtracted field, the absolute value of the pixels are taken and the result plotted on a $[0, 255]$ grayscale. A still is seen in Fig. 3.

In contrast, the race car video had more camera drift so that the background reconstruction was not perfect every time. The difference was typically negative, so adding these back in via the method described in the assignment (*i.e.* setting them to zero) and was found to work better than the above approach for the skiier video. This is shown in Fig. 4. However, the background subtracted video $|X_S|$ still more clearly contained the race cars. Here there were other artifacts, such as the yellow road marker appearing on top of the reconstructed cars. A sample of these more clearly visible cars with other artifacts is shown in Fig. 5.

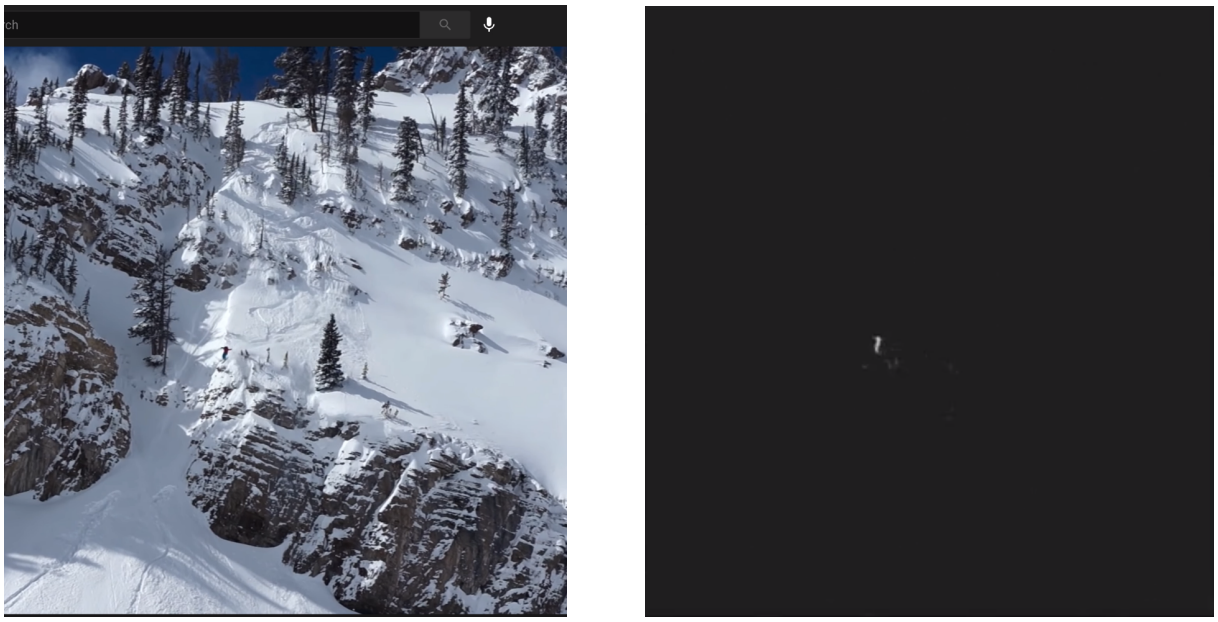

Figure 3: The skiier against a beautiful mountain scene (left) and with the background subtracted (right).

Figure 4: Race cars in a fancy European city (left) and without their background (right).

5 Summary and Conclusions

This numerical experiment looked at the very versatile and downright amazing dynamic mode decomposition (DMD) method, which can identify the temporal behavior of spatiotemporally-correlated modes identified by the SVD. This is effectively an approximation to the Koopman operator describing the possibly infinite-dimensional linear evolution of some difficult-to-describe nonlinear dynamics. While this Koopman operator can in principle be directly approximated using the pseudoinverse of a single time differenced data matrix X', the resulting matrix A is generally far too large for eigendecomposition. The DMD first projects into a low-rank subspace containing the most important modes, as seen by the largest singular values of the data matrix X. A low-dimensional Koopman operator is then eigendecomposed and lifted back into the full data space using the SVD basis operators. While not orthogonal, these modes are in a sense the best ones to represent the general dynamics of the data.

The biggest advantage of the DMD is that it provides a linear time-dependent representation of the dynamics behind data. This work used the lowest frequency modes for a simple example of the DMD's utility, namely in image background separation. However, there is far more potential in the DMD than this application. By obtaining a linear representation for system dynamics control theory can be applied for a great variety of engineering tasks. Most of these control applications involve something akin to the inverted pendulum problem; that is, sufficient data collection, DMD processing, and control may be used to stabilize generally unstable nonlinear systems such as magnetic confinement systems. Thank you all for teaching this! Code and background-subtracted videos are available on the author's Github here.

References

[1] J.N. Kutz. *Data-driven Modeling and Scientific Computation.* Oxford University Press, 1st edition, 2013.

[2] J.N. Kutz. *Dynamic Mode Decomposition.* SIAM, 2016.

Figure 5: The absolute value of the sparse reconstruction $|X_S(t)|$ gave better image quality than Fig. 5 yet with more artifacts such as the street appearing on top of the car, seen here on the left side of the first car.

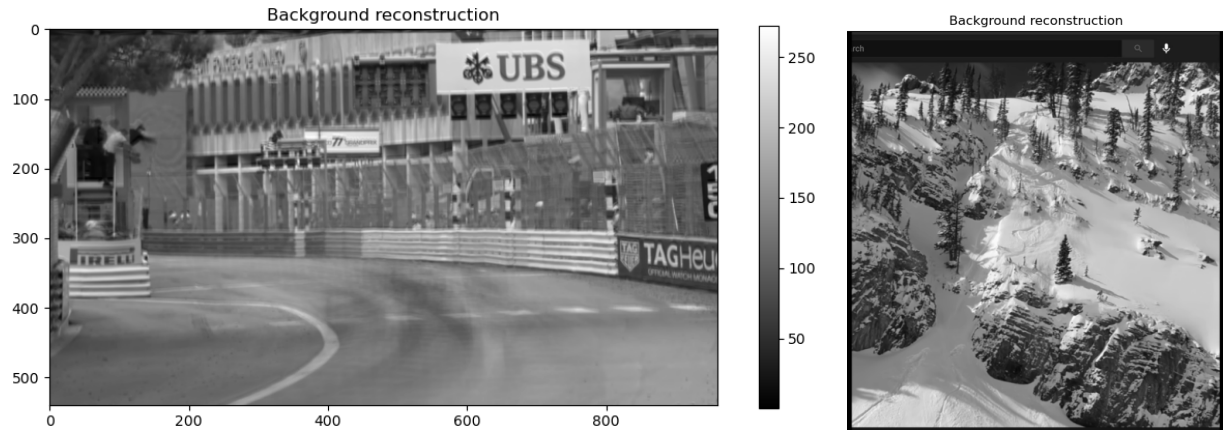

Figure 6: Background reconstructions of race cars (left) and skiier (right) using the low-frequency modes.

Appendix A Python Functions

The following list compiles important Python functions used in implementation:

- The commands `np.linalg.svd(X)` and `np.linalg.eig(X)` are used for big linear algebra operations,
- The `av` package was used to read in the `.mp4` video clips,
- The sequence `np.argsort(np.abs(freqs))` sorts the eigenvalues by their absolute value.

Appendix B Python Implementation

Main implementation file `hw5.py`:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as linalg
import gc # garbage collector to delete big stuff

# To make movies
import matplotlib.animation as animation

# To read movies
import av

def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989*r + 0.5870*g + 0.1140*b

    return gray

### Open, convert, and save (Run once per file)
#Open file
#v = av.open('monte_carlo_low.mp4')
v = av.open('ski_drop_low.mp4')

fps = 60.0
```

```
# Convert frames to images
#vid = np.zeros((380, 1112, 1872)) # monte carlo size
#vid = np.zeros((380, 540, 960)) # monte carlo low size
# vid = np.zeros((454, 540, 960))
# counter = 0
# for packet in v.demux():
#     for frame in packet.decode():
#         #print(frame)
#         img = frame.to_image()
#         vid[counter,:,:] = rgb2gray(np.asarray(img))
#         counter += 1

### Save, faster to save it now and load as np array
#with open('data/montecarlo_low.npy', 'wb') as f:
#   np.save(f, vid)

# with open('data/skidrop_low.npy', 'wb') as f:
#    np.save(f, vid)

### Load
with open('data/montecarlo_low.npy', 'rb') as f:
   vid = np.load(f)

# with open('data/skidrop_low.npy', 'rb') as f:
#    vid = np.load(f)

#vid = vid[:,:,230:730]

### Reshape into data matrix, frames x pixels
X_T = vid[:-1,:,:].reshape(vid.shape[0]-1, vid.shape[1]*vid.shape[2])
XP_T = vid[1:,:,:].reshape(vid.shape[0]-1, vid.shape[1]*vid.shape[2])
# Flip for DMD convention, each frame is a column
X = X_T.T
XP = XP_T.T

del X_T
del XP_T
gc.collect()

# print('Beginning SVD...')
# u, s, vh = np.linalg.svd(X, full_matrices=False)

# ### Save SVD matrices
# with open('data/skidrop_low_svd_u.npy', 'wb') as f:
#    np.save(f, u)

# with open('data/skidrop_low_svd_s.npy', 'wb') as f:
#    np.save(f, s)

# with open('data/skidrop_low_svd_vh.npy', 'wb') as f:
#    np.save(f, vh)

# quit()
```

```
### Load SVD matrices
# with open('data/skidrop_low_svd_u.npy', 'rb') as f:
#     u = np.load(f)

# with open('data/skidrop_low_svd_s.npy', 'rb') as f:
#     s = np.load(f)

# with open('data/skidrop_low_svd_vh.npy', 'rb') as f:
#     vh = np.load(f)

with open('data/montecarlo_low_svd_u.npy', 'rb') as f:
    u = np.load(f)

with open('data/montecarlo_low_svd_s.npy', 'rb') as f:
    s = np.load(f)

with open('data/montecarlo_low_svd_vh.npy', 'rb') as f:
    vh = np.load(f)

### Truncate modes
print('Files loaded...')

v = vh.T

cutoff = 100#200
ur = u[:, :cutoff]
sr = s[:cutoff]
vr = v[:, :cutoff]

### Koopman operator: low-rank projection
Atilde = np.dot(ur.T, np.dot(XP, np.dot(vr, np.diag(1.0/sr))))

### Eigenvalue problem for Atilde
w, va = np.linalg.eig(Atilde)

### Koopman operator: lift projection to dynamic modes
phi = np.dot(XP, np.dot(vr, np.dot(np.diag(1.0/sr), va)))#np.dot(ur, va)

### Find initial condition among modes
b = np.dot(np.linalg.pinv(phi), X[:,0])

### Full time-zero construction
Xrec = np.dot(phi, b)

# Vid size
size = (vid.shape[1], vid.shape[2])

### DMD frequencies
# Time
t = np.arange(vid.shape[0])/fps
dt = t[1] - t[0]

# Freqs
om = np.log(w)/dt
```

```
### Sort phi, b, and omega
idx = np.argsort(np.abs(om))
Phi = phi[:, idx]
Om = om[idx]

### Get lowest frequencies
sidx = np.where(np.absolute(Om) < 2) # mc, 2 | ski drop 1
Ds = np.diag(Om[sidx])
Phis = Phi[:, sidx][:,0,:]

# Background b
bs = np.zeros((Phis.shape[1], vid.shape[0])) + 0j
bs[:,:-1] = np.dot(np.linalg.pinv(Phis), X)
bs[:,-1] = np.dot(np.linalg.pinv(Phis), XP[:,-1])

print('DMD finished...')

### Delete stuff
del u
del s
del vh
del phi
del Phi
del Atilde
gc.collect()

plt.figure()
plt.plot(np.real(Om), 'o--', label='real')
plt.plot(np.imag(Om), 'o--', label='imag')
plt.plot(np.absolute(om), 'o--', label='abs')
plt.plot(np.absolute(Om), 'o--', label='abs sorted')
plt.xlabel('DMD mode number')
plt.ylabel('Frequency')
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()

plt.figure()
plt.plot(np.real(Om), np.imag(Om), 'o')
plt.grid(True)
plt.xlabel('Real part')
plt.ylabel('Imaginary part')
plt.tight_layout()

# Background recons
Brc = np.dot(Phis, bs)
bkgrnd = Brc.T.reshape(vid.shape)

#Brc = np.array(
plt.figure()
plt.imshow(np.absolute(bkgrnd[200,:,:]), cmap='gray')
plt.title('Background reconstruction')
plt.colorbar()
```

```
plt.show()

print('Reconstruction completed...')

frames = vid - np.absolute(bkgrnd)
#frames[frames < 0] = 0 # add back in zero values
frames[frames < 0] = np.absolute(frames[frames<0]) # absolute value of negatives
frames[frames < 50] = 50 # background filter...

fig, ax = plt.subplots()
anims = []
for i in range(t.shape[0]):
    this_im = ax.imshow(frames[i,:,:], cmap='gray', vmin=0, vmax=np.amax(frames))
    anims.append([this_im])

an = animation.ArtistAnimation(fig, anims, interval=50, blit=True, repeat_delay=1000)
an.save('montecarlo_nbg3.mp4', fps=60)

quit()
```