Homework 6: Background Subtraction in Video Streams

Anamol Pundle

March 19, 2015

Abstract

In this work, Dynamic Mode Decomposition, along with robust Principal Component analysis, is used to take a video clip containing a foreground and background object and separate the video stream to both the foreground video and a background. The two methods are tested with several videos. It is found that the robust PCA algorithm gives consistently better results than the DMD method. It is also observed that continuously moving objects can be filtered out more easily than objects that do not move.

1 Introduction and Overview

Exploiting low-dimensionality in complex systems has already been demonstrated to be an effective method for either computationally or theoretically reducing a given system to a more tractable form. In what has been proposed so far with POD reductions, we have used model-based algorithms. Thus a given set of governing equations dictate the dynamics of the underlying system. These equations construe the model. However, an alternative to this approach exists when either the model equations are beyond their range of validity or governing equations simply are not known or well-formulated. In this alternative approach, experimental data itself drives the understanding of the system, without model equations being prescribed. Thus data-based algorithms can be constructed to understand, mimic and control the complex system. Dynamic Mode Decomposition (DMD) is a relatively new data-based algorithm that requires no underlying governing equations, rather snapshots of experimental measurements are used to predict and control a given system.

In this assignment, two videos are considered. The videos contain a clearly defined background (not moving) and a foreground (moving). The aim is to separate the two. This is done using two methods: the robust Principal Component Analysis algorithm, and the Dynamic Mode Decomposition Method. Both methods are evaluated using the videos generated by the algorithms.

2 Theoretical Background

2.1 Robust Principal Component Analysis

Robust Principal Component Analysis (RPCA) is a modification of the widely used statistical procedure Principal component analysis (PCA) which works well with respect to grossly corrupted observations. A number of different approaches exist for Robust PCA, including an idealized version of Robust PCA, which aims to recover a low-rank matrix L_0 from highly corrupted measurements $M = L_0 + S_0$. This decomposition in low-rank and sparse matrices can be achieved by techniques such as Principal Component Pursuit method (PCP), Stable PCP, Quantized PCP , Block based PCP, and Local PCP. Then, optimization methods are used such as the Augmented Lagrange Multiplier Method (ALM), Alternating Direction Method (ADM), Fast Alternating Minimization (FAM) or Iteratively Reweighted Least Squares (IRLS). RPCA has many real life important applications particularly when the data under study can naturally be modeled as a low-rank plus a sparse contribution, such as Video Surveillance andFace Recognition.

2.2 Dynamic Mode Decomposition

Physical systems, such as fluid flow or mechanical vibrations, behave in characteristic patterns, known as modes. In a recirculating flow, for example, one may think of a hierarchy of vortices, a big main vortex driving smaller secondary ones and so on. Most of the motion of such a system can be faithfully described using only a few of those patterns. The dynamic mode decomposition (DMD) provides a means of extracting these modes from numerical and experimental pairs of time-shifted snapshots. Each of the modes identified by DMD is associated with a fixed oscillation frequency and growth/decay rate, determined by DMD without requiring knowledge of the governing equations. This is to be contrasted with methods, such as the proper orthogonal decomposition, which produce a set of modes without the associated temporal information. The DMD method provides a decomposition of experimental data into a set of dynamic modes that are derived from snap shots of the data in time. The mathematics underlying the extraction of dynamic information from time- resolved snapshots is closely related to the idea of the Arnoldi algorithm, one of the workhorses of fast computational solvers.

3 Algorithm Implementation and Development

- Import video into matlab using the function VideoReader.
- Convert the video into a 4D matrix. Find the size of each frame and the number of frames of each video.
- Convert each frame of the video from RGB to grayscale using the function rgb2gray.

- Restrict the video to 60 frames to capture movement and reduce computational time and memory required.
- Reshape each frame to a one column, and construct a matrix with each column being one reshaped frame.
- Define X_1^{M-1} and X_2^M.
- Perform 'economy' singular value decomposition on X_1^{M-1}.
- Find the matrix $\widetilde{S}$ from U, X_1^{M-1}, $V$and Sigma, obtained from the SVD of X_2^M.
- Find the eigenvalues of the $\widetilde{S}$ matrix.
- Find the DMD modes, ϕ.
- Plot the modes and find the mode closest to 0 on the plot. This is the background.
- Reconstruct the X matrix using only the background mode to get the background, and the rest of the modes to reconstruct the foreground.
- Reshape and take absolute values of both matrices. Convert both matrices to integer values to get video.
- For the robust PCA method, define value of lamba.
- Perform the robust PCA to obtain the low rank (background) and the sparse (foreground) matrix.
- Heuristically calculate the value of lambda that gives the best reconstruction.
- Profit.

4 Computational Results

Two videos are examined in this assignment. The results from these follow.

A very simple case is considered to begin with. The first video is a $320x640$ video shot from an iPhone of a pen swinging in front of a wall. Figure 1 shows a still from the video, after being converted to grayscale. As seen in the picture, the gray wall is the background and the moving pen is the foreground.

Figure 2 shows the spectral distributions of the eigenvalues μ as well as the modes ω associated with those eigenvalues. The mode closest to zero is the background. The eigenvalues are seen to be in a circle, which means an oscillatory solution.

Figure 3 shows images generated after DMD. The background is generated using only the first mode, whereas the foreground is generated using the rest

Figure 1: Still from Video 1. The moving pen is in the foreground, while the wall is the background.

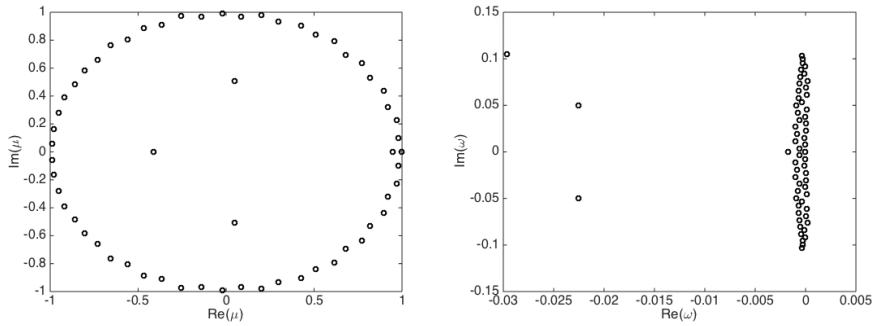

Figure 2: (a) spectral distributions of the eigenvalues μ (b) modes ω associated with eigenvalues μ

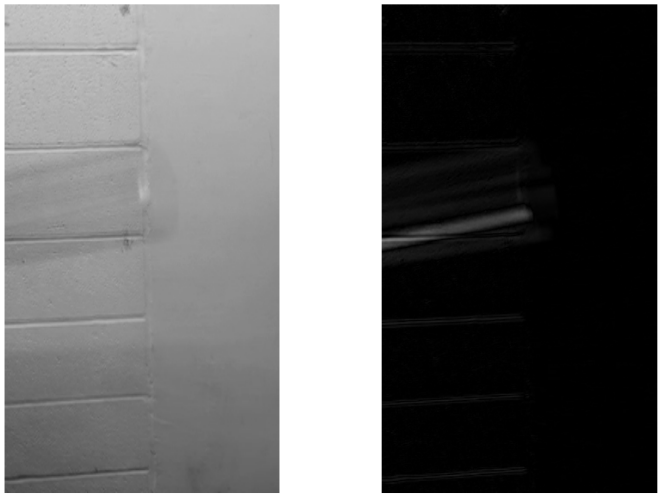

Figure 3: Image reconstructions of Video 1 using DMD method (a) Low-rank reconstruction (background) (b) Sparse reconstruction(foreground)

of the modes. It is seen that the DMD does a good job of separating the background and the foreground, though it is not perfect. A shadow of the moving pen can be observed in the low-rank reconstruction.

Figure 4 shows images generated using the robust PCA algorithm,with a heuristically calculated value of lambda, equal to 0.004. Near-perfect resolution of the background and the foreground is observed.

The second case considered is that of a swinging pendulum. The video is a $640x360$ video downloaded from youtube. Figure 5 shows a still from the video, after being converted to grayscale. As seen in the picture, the desk, bed, floor and wall are the background, while the swinging pendulum is the foreground.

Figure 6 shows the spectral distributions of the eigenvalues μ as well as the modes ω associated with those eigenvalues. The mode closest to zero is the background. The eigenvalues are seen to be in a circle, which means an oscillatory solution.

Figure 7 shows images generated after DMD. The background is generated using only the first mode, whereas the foreground is generated using the rest of the modes. It is seen that the DMD does a good job of separating the background and the foreground, though it is not perfect. A shadow of the swinging pendulum can be observed in the low-rank reconstruction.

Figure 8 shows images generated using the robust PCA algorithm,with a heuristically calculated value of lambda, equal to 0.004. Near-perfect resolution of the background and the foreground is observed.

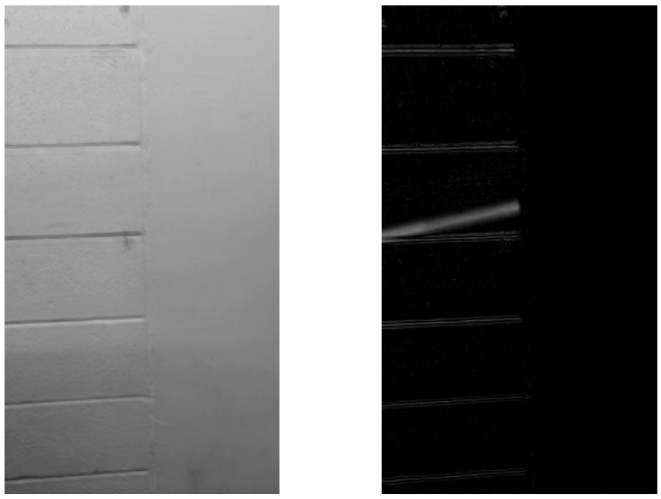

Figure 4: Image reconstructions of Video 1 using robust PCA algorithm (a) Low-rank reconstruction (background) (b) Sparse reconstruction(foreground)

Figure 5: Still from Video 2. The swinging pendulum is in the foreground, while the wall is the background.

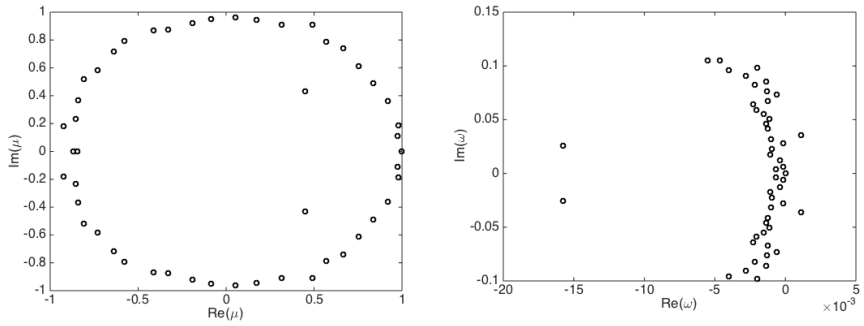

Figure 6: (a) spectral distributions of the eigenvalues μ (b) modes ω associated with eigenvalues μ

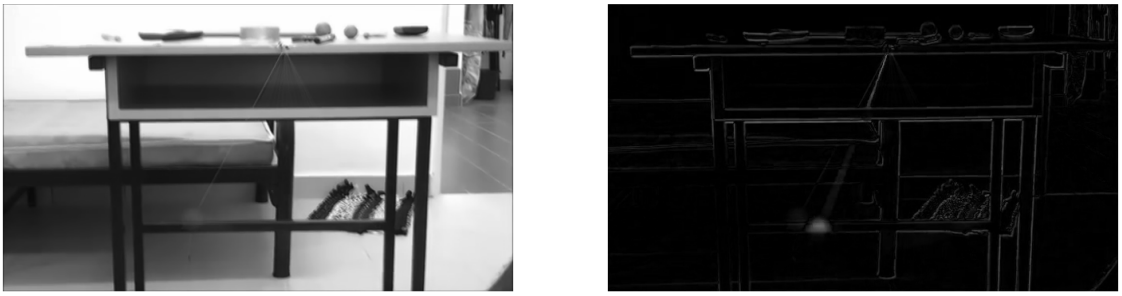

Figure 7: Image reconstructions of Video 2 using DMD method (a) Low-rank reconstruction (background) (b) Sparse reconstruction(foreground)

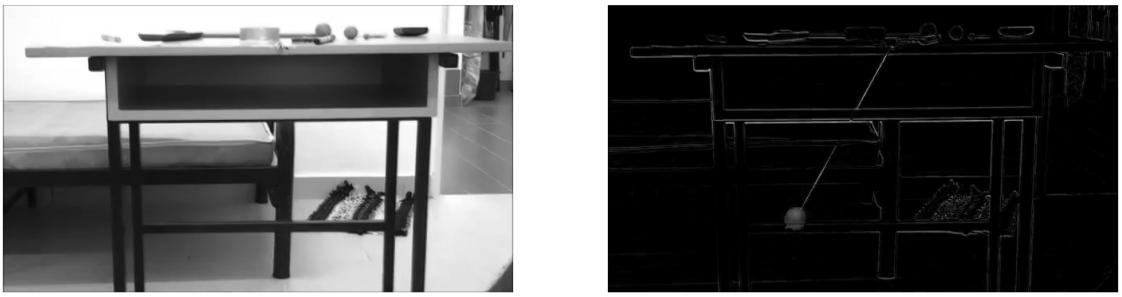

Figure 8: Image reconstructions of Video 2 using robust PCA algorithm (a) Low-rank reconstruction (background) (b) Sparse reconstruction(foreground)

5 Summary and Conclusions

The techniques of robust Principal Component Analysis and Dynamic Mode Decomposition are successfully applied to separating the background and foreground of two videos. It is seen that the robust PCA is better at separating the two, even though the DMD does not do a bad job. The value of lambda used in the rPCA algorithm is found to be 0.004 for maximum effectiveness.

References

1. Kutz, J. Nathan, Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data, September 2013
2. Wright, Ganesh: Low-Rank Matrix Recovery and Completion via Convex Optimization, http://perception.csl.illinois.edu/matrix-rank/sample_code.html
3. Matlab help

Appendix I: MATLAB functions used

`VideoReader`: reads wave file into MATLAB

- `OBJ = VideoReader(FILENAME)` constructs a multimedia reader object, OBJ, that can read in video data from a multimedia file.

`svd`: Singular value decomposition.

- `[U,S,V] = svd(X)` produces a diagonal matrix S, of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that `X = U*S*V'`.

`frame2im`: Return image data associated with movie frame.

- `[X,MAP] = frame2im(F)` returns the indexed image X and associated colormap MAP from the single movie frame F.

`rgb2gray`: Convert RGB image or colormap to grayscale. `rgb2gray` converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale intensity image I.

`imshow`: displays image

- `imshow(I)` displays the image I in a Handle Graphics figure, where I is a grayscale, RGB (truecolor), or binary image. For binary images, `imshow` displays pixels with the value 0 (zero) as black and 1 as white.

`inexact_alm_rpca`: inexact augmented lagrange multiplier method for robust principal component analysis

- `[A, E] = exact_alm_rpca(D, λ)` gives the low rank and sparse matrix deconstruction of D.

```
% APPENDIX II: MATLAB CODE


clear all; close all; clc;

y = wavread('Allthatjazz');
z = wavread('ClassicWrok');
x = wavread('Grunge');

deodato_rec = 75;
floyd_rec = 75;
y = single(y); z = single(z); x = single(x);

v = y'/2;
w = z'/2;
x = x'/2;
Fs = length(v)/floyd_rec;
Fs2 = length(w)/deodato_rec;
Fs3 = length(x)/deodato_rec;
a = 100;

L = length(v)/Fs;
L2 = length(w)/Fs2;
L3 = length(x)/Fs3;
k=(2*pi/(2*L))*[0:(length(v)-1)/2 -(length(v)-1)/2:-1]; ks=fftshift(k);

if rem(length(k), 2) > 0

    ks = ks(length(ks)/2:end);
else
    ks = ks(length(ks)/2-1:end);
end
k = single(k); ks = single(ks);
tfinal = length(v)/Fs;
t = single(1:length(v))/Fs;

Sgt_spec = []; tslide = 0:0.1:tfinal; tslide = single(tslide);
Sgt_spec2 = []; Sgt_spec3 = []; Spec1 = []; Spec2 = [];
Spec3 = [];

for ii = 1:length(tslide)
    g = exp(-a*(t-tslide(ii)).^2);
    Sg = g.*v; Sg2 = g.*w; Sg3 = g.*x;
    Sgt = fft(Sg); Sgt2 = fft(Sg2); Sgt3 = fft(Sg3);
    Sgt = fftshift(Sgt); Sgt2 = fftshift(Sgt2); Sgt3 = fftshift(Sgt3);
    LSgt = length(Sgt);
    Sgt = Sgt(int64(LSgt/2):end); Sgt2 = Sgt2(int64(LSgt/2):end);
    Sgt3 = Sgt3(int64(LSgt/2):end);
    Sgt = single(Sgt); Sgt2 = single(Sgt2); Sgt3 = single(Sgt3);

    Sgt_spec = [Sgt_spec abs((Sgt))]; Sgt_spec2 = [Sgt_spec2 abs((Sgt2))];
    Sgt_spec3 = [Sgt_spec3 abs((Sgt3))];
```

```
    if rem(ii+50, 50) == 0
        Spec1 = [Spec1; Sgt_spec];
        Spec2 = [Spec2; Sgt_spec2];
        Spec3 = [Spec3; Sgt_spec3];
        Sgt_spec = [];
        Sgt_spec2 = [];
        Sgt_spec3 = [];
    end



end

Sgtot = [Spec1; Spec2; Spec3];
clear Sgt_spec; clear Sgt_spec2; clear Sgt_spec3;
clear Sgt1; clear Sgt2; clear Sgt3;
[u,s,v] = svd(Sgtot',0);

nberr = []; ldaerr = [];
nbstd = []; ldastd = [];
for kkk = 1:1

    clc;
    rowbeg = 1;
    rowend = 4;
    trainp = 10;
    testp = 15 - trainp;
    allanswers = [ones(15,1); 2*ones(15,1); 3*ones(15,1)];
    Efinal = 0; Enbfinal = 0;
    Pf = 0; Col = 0; Deo = 0;
    Errlda = []; Errnb = [];
    for jj = 1:100
        floydtrain = []; coltrain = []; deotrain = [];
        ftest = []; coltest = []; deotest = [];
        fsampl = randsample(15,trainp); ft = setdiff(1:15, fsampl)';
        colsampl = randsample(15,trainp) + 15; colt = setdiff(16:30, colsampl)';
        deosampl = randsample(15,trainp) + 30; deot = setdiff(31:45, deosampl)';

        for kk = 1:trainp
            ff = v(fsampl(kk), rowbeg:rowend);
            cc = v(colsampl(kk), rowbeg:rowend);
            dd = v(deosampl(kk), rowbeg:rowend);
            floydtrain = [floydtrain ; ff];
            coltrain = [coltrain ; cc];
            deotrain = [deotrain ; dd];
        end
        for kk = 1:testp
            ff = v(ft(kk), rowbeg:rowend);
            cc = v(colt(kk), rowbeg:rowend);
            dd = v(deot(kk), rowbeg:rowend);
            ftest = [ftest ; ff];
            coltest = [coltest ; cc];
            deotest = [deotest ; dd];
        end
```

```
    train = [floydtrain; coltrain; deotrain];
    tests = [ftest; coltest; deotest];
    correct = [allanswers(11:15); allanswers(26:30); allanswers(41:45)];
    answer = [ones(trainp,1); 2*ones(trainp,1); 3*ones(trainp, 1)];


    [ind err] = classify(tests, train, answer);

    Err = 0;
    for ii = 1:length(ind)
        if ind(ii) == correct(ii)
            Err = Err + 1;
            if ii <=5
                Pf = Pf +1;
            elseif ii <=10
                Col = Col + 1;
            else
                Deo = Deo + 1;
            end
        end

    end
    Err = Err/15*100;

    Errlda = [Errlda Err];


    figure(2)
    nb = fitNaiveBayes(train, answer);
    prednb = nb.predict(tests);

    figure(2)
    bar(ind);
    set(gca, 'FontSize',16);
    xlabel('Test Songs'); ylabel('Classification');

    Err = 0;
    for ii = 1:length(prednb)
        if prednb(ii) == correct(ii)
            Err = Err + 1;
        end

    end
    Err = Err/15*100;
    Errnb = [Errnb Err];

end

mean(Errnb)
std(Errnb)

mean(Errlda)
```

```
    std(Errlda)


    Pf = Pf/(testp*jj)*100;
    Col = Col/(testp*jj)*100;
    Deo = Deo/(testp*jj)*100;
    Banderr = [Pf Col Deo];
    Bands = ['Floyd', 'COl', 'ddd'];


    nberr = [nberr mean(Errnb)];
    nbstd = [nbstd std(Errnb)];
    ldaerr = [ldaerr mean(Errlda)];
    ldastd = [ldastd std(Errlda)];


end
```

Published with MATLAB® R2014b