Homework 6: Background Subtraction in Video Streams

Karl Marrett (kdmarrett@gmail.com)

Abstract

The task in this report is to separate foreground and background objects in a video stream. To do this we will compare two strategies: the Dynamic Mode Decomposition and the robust Principal Component Analysis code to compare their quality and performance in this task. The goals of the report are two separate the objects with these two methods and compare them qualitatively.

1 Introduction

We have thus far used several methods for taking high dimensional systems and projecting them into a lower perhaps more tractable form. The Dynamic Mode Decomposition takes a similar inspiration and returns the *dynamic* modes that comes from data that changes with respect to time.[2] Compared to our previous techniques, this basically provides a formal way to analyze linear structure in time as opposed to single snapshots in a data set. Given the problem of separating two objects with different linear dynamics in video frames, the motivation for this technique should be clear. This algorithm is defined more rigorously in the next section.

Finding the right basis and the methods to transform data into such a basis are central problems in machine learning. In the past reports, we have mostly used certain assumptions about our data to decide a reasonable basis to transform our data. For example, in the analysis of sound files we used the convenient decomposition into the Fourier domain to create a low dimensional picture of the dynamics of the data.[2] However, when we instead don't know anything about the underlying structure of mechanism of the data, we need a systematic approach. For example, given neural data of a spike train response of several neurons we may be able to explain structure in the data in a low dimensional Fourier space but the simplest model may in fact be described in an entirely different space. For problems of this nature, we want a data-driven approach for choosing the right basis to describe our data. Having a generic to high dimensional data of this sort allows equation-free modeling while also sidestepping our potentially misleading initial assumptions about the data.

These concerns illustrate some of the motivation for new methods for choosing a lower dimensional space for the data. Further details of how the rPCA and the DMD technique might describe a low dimensional bases for our data is described in detail in the next section.

2 Theoretical Background

The DMD technique requires certain assumptions about our underlying system and the data being processed. The data must be collected at regularly spaced intervals of time, this essentially builds the N dimensional matrix we define as $\mathbf{X}$ to compute future time points. Although the sampling must be linear, the amount of sampling will have particular effects on the analysis which we will discuss more in the Section Algorithm and Implementation. We build the data matrix X of N data points and M time points in the following way:

$$\mathbf{X} = [\mathbf{U}(\mathbf{x}, \mathbf{t_1}), \mathbf{U}(\mathbf{x}, \mathbf{t_1}), \mathbf{U}(\mathbf{x}, \mathbf{t_1}), \dots \mathbf{U}(\mathbf{x}, \mathbf{t_M})] \tag{1}$$

where $\mathbf{x}$ is the original data matrix of N channels. We must further define another matrix that is a subset of the columns of X as shown below:

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

This is just the subset of columns j through k of the original X matrix.

2.1 The Koopman Operator

The Koopman Operator relates the dynamics of one time slice of our original m channeled $\mathbf{X}$ matrix to the $t+1$ time point. A few things to note about the Koopman operator relevant to the DMD are:

1. it is a linear and time-independent operator
2. it represents nonlinear, infinite dimensional dynamics without linearization
3. it can be viewed as the low dimensional modes of the dynamics of the system

The dynamic modes of the decomposition are those of the Koopman operator. Since the Koopman operator describes the dynamics of A we can relate it to the data matrix X with

$$\mathbf{x_{j+1}} = \mathbf{A}\mathbf{x_j} \tag{3}$$

where j, as before, represents a time point or a column U of the X matrix. Using our definition in Equation 2 we can build the matrix from the first through $M-1$ time points

$$\mathbf{X_1^{M-1}} = [\mathbf{x_1}\mathbf{x_2}\mathbf{x_3} \dots \mathbf{x_{M-1}})] \tag{4}$$

which can be substituted using Equation 3 to

$$\mathbf{X_1^{M-1}} = [\mathbf{x_1}\mathbf{A}\mathbf{x_1}\mathbf{A^2}\mathbf{x_1} \dots \mathbf{A^{M-1}}\mathbf{x_1})] \tag{5}$$

We will also need one other sub-matrix of X for the analysis: X_2^M which is constructed in the same way as X_1^{M-1} but is advanced one time point for all columns. Having these matrices we can compute the Koopman operator A, this computation is detailed further in the Section Algorithm and Implementation.

2.2 SVD

The SVD has a particular role in the DMD algorithm. The main means of compressing and characterizing the high dimensional video data is via the SVD technique. The theorem introducing the SVD essentially asserts that *any* matrix can be decomposed as follows:

$$A_m n = U_m m S_m n V_n n^T$$

The decomposition has deep connections to the Eigen faces of A, namely in that the columns of U are orthonormal eigenvectors of AA^T and that the columns of V are also the orthonormal eigenvectors of $A^T A$. Another point to note is that the diagonal matrix S holds the square roots of the eigenvalues of U or V. These relations allow us to analytically solve for the u, s, or v matrices although in this report we will only be leveraging the Matlab command *svd*. Since U represents the new modes of A we can treat SV^T as the projection on to the modes.

When implementing the SVD in the script we need to use the reduced SVD to the size of the data. The major advantage of the SVD is the ordered components of the weights of the S matrix, which gives a criterion for deciding which dimensions describe meaningful structure in the data, and those which are redundant. By viewing the eigen values and only including the features that are needed we save processing time and memory.

2.3 Robust Principal Component Analysis

To test the relative performance of the DMD, robust we will use a technique that builds on the previous techniques we have used in the class known as the Robust Principal Component Analysis (rPCA). This technique has been advocated by Candes et al. as a means to recover a low-rank and or sparse components of an evolution matrix.[1] This is computed via a relatively expensive convex optimization procedure and has been shown to be state of the art over similar matrix separation methods. Since the focus of this paper is primarily on the implementation of the DMD algorithm, we will use the rPCA function provided in the cited paper and we will not go into this algorithm in more detail.[1]

3 Algorithm Implementation and Development

To test the performance of the robust PCA technique and the DMD three video clips were used. One was of my hand in the foreground moving in circles in a still frame. The next was a butterfly taking off in slow motion. The butterfly video, due to being in slow motion was especially noisy and so served as a test of the conditions. The final video clip tested was the paint can used in the previous report. This was used because the camera was very stable in these videos and thus served as a measure of perhaps the highest ability of the methods to classify foreground and background video.

Once these video files were loaded into the Matlab script they were passed to my function *getX*. This function reshapes the video data into a double, grayscale matrix where each image is a vector column. Arranging the data in this way facilitates the analogy we have

been building where the rows of X represent our channels our measurements and where the columns represent the time, dt, or dynamics of the system. While reading in the video, the frame rate (in this case 30 frames per sec for each video) was used to calculate dt, the timestep which is crucial for the DMD analysis.

The data was then passed to the function *dmd* which begins by constructing the matrix $X1$ and $X2$. These matrices differ by one time point and are both one shorter than the original X data. Next the Singular Value Decomposition of $X1$ was computed via the Matlab embedded *svd* command. I then constructed the similarity matrix s by projecting X2 onto the modes of X1. Then I found the eigen vectors (ev) and values (d) of the similarity matrix. By taking the log of the eigen values divided by dt, I was able to find the weighting vector *omega* for the final computation of the reconstruction. From the eigen vectors I computed *phi* which represent the DMD modes from which to compute the future state of the system. I then computed the initial state of the system y_0 via the pseudoinverse and then created a reconstruction of video for each frame.

To take advantage of the sparsity in the data, after the DMD is performed I separated the reconstruction into foreground video and background video. These are the $\mathbf{X_{Low-Rank}}$ and $\mathbf{X_{Sparse}}$ matrices respectively. This is done by taking the eigenvalue associated with the near zero component of the omega values. The function *getBackground* returns an index to this omega. The mode associated with this index is used to generate the background video whereas the this mode was removed for the generation of the foreground image. After generating both, the foreground was further distinguished from the background via subtraction of the the final DMD reconstruction for each frame.

Similarly, X was also passed into the *inexactrpca* method. Both techniques were timed for comparison of their efficiencies using the commands *tic* and *toc*. As a diagnostic the ability to take a subset of features was added to see the effect on the quality of separation. In addition, I added the functionality to compute the residual errors between the actual video data and the reconstruction as a measure of the accuracy of the two techniques. The results of both separations were extensively plotted as described in the next section.

4 Computational Results

By examining the sigma values of all video data, we can certainly see that all of X matrices we computed were low rank, this gave our DMD technique an advantage in efficiency by excluding the redundant features. We can also see, for instance in the hand movie in Figure 8, the moving parts of the image were easily identified and targeted in either of the techniques. Figures including the separation results for every technique and video are at the end of this report. In general the rPCA, tended to have more robust selection of the moving objects whereas the DMD could be offput by noise. These findings varied over the videos and frames however.

One of the most notable difference between the rPCA and DMD techniques was the amount of time they took relatively. On average, the DMD method would report the time of about 10 seconds whereas the rPCA was around 200 seconds.

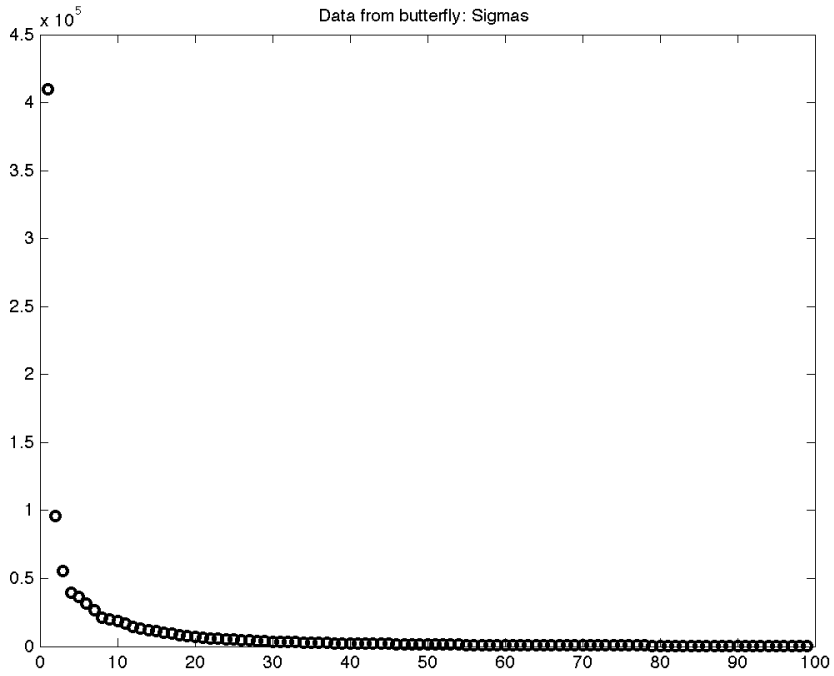

Figure 1: Shown are the sigma values of the Singular Value Decomposition of the X1 matrix of the butterfly video data. The dominant first mode, indicates the matrix is sparse and can be represented in a lower dimensional space.

Table 1: Table showing the error r between the foreground background object distinction tasks for each video. The number of residuals error of pixels summed over the entire video is given

	Case Butterfly	**Case Hand**	**Case Paint**
DMD **r**	45546	-9880	56451
rPCA **r**	2268	8272	4772

5 Conclusion

From our analysis we can see that Dynamic Mode Decomposition is an incredibly powerful tool for doing equation free-modeling.[2] With this analysis, we were able to essentially train a matrix operator to predict the state of one time point to the next, thus creating a linear approximation of a high-dimensional nonlinear system. In this analysis, predicting the future dynamics of the video data by constructing future time points from the DMD modes would not work. This is due to the noisiness and low structure of the videos recorded. However, under analysis of stable equations the DMD analysis is robust enough to reproduce the dynamics of a system, with of course some cumulative error.[2]

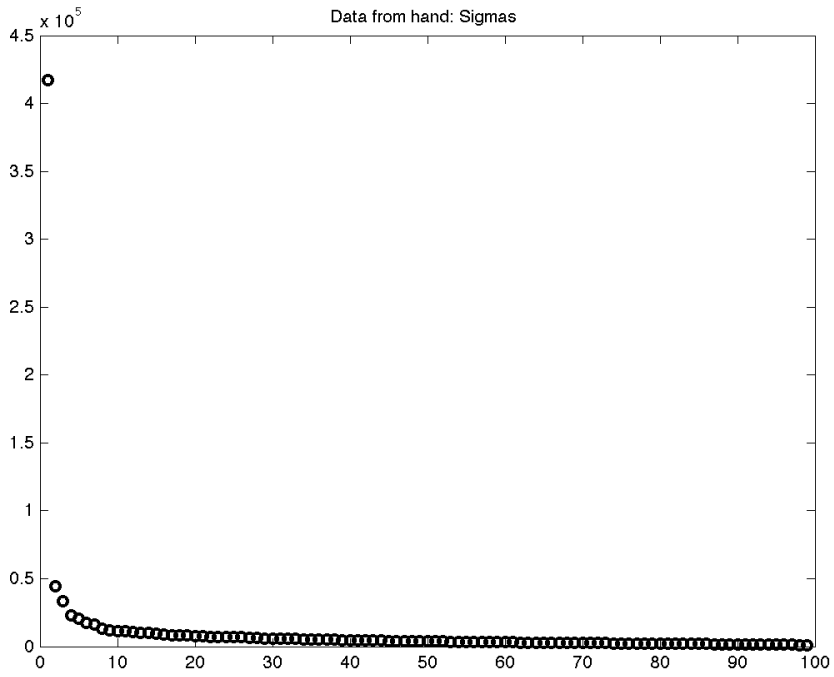

Figure 2: Shown are the sigma values of the Singular Value Decomposition of the X1 matrix of the hand motion video data. The dominant first mode, indicates the matrix is sparse and can be represented in a lower dimensional space.

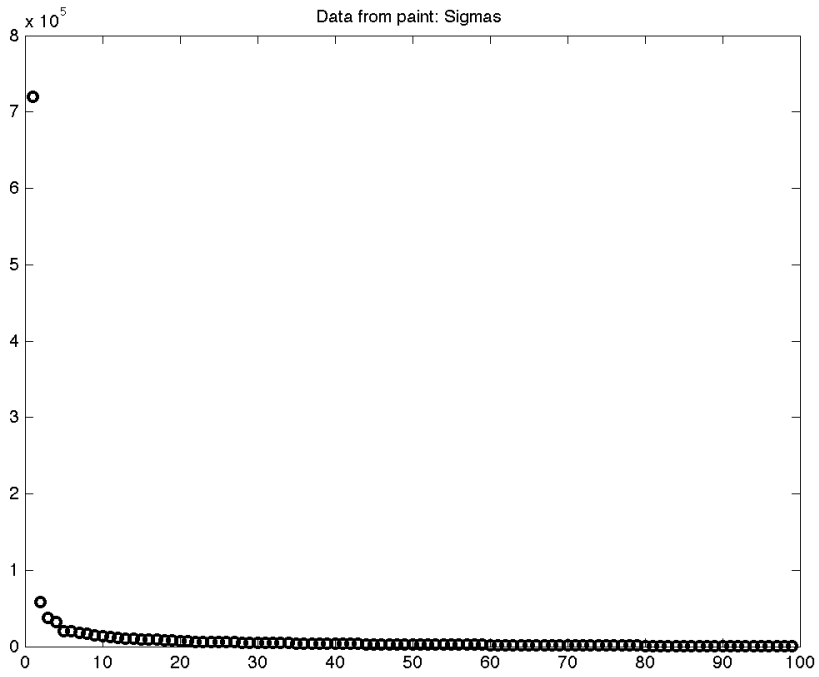

Figure 3: Shown are the sigma values of the Singular Value Decomposition of the X1 matrix of the paint can video data. The dominant first mode, indicates the matrix is sparse and can be represented in a lower dimensional space.

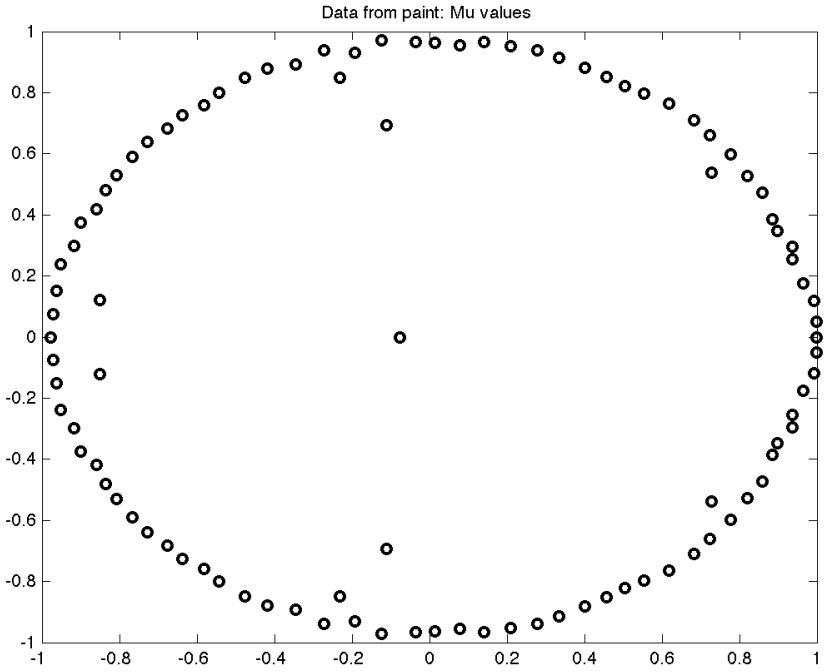

Figure 4: Shown are the mu values, the complex representation of the omega modes representing the dynamics captured in the data. By lying almost completely on a circle, these dynamics are nearly ideal. All points lying outside of the circle are the modes that will have runaway error after enough time steps beyond the given data.

Figure 5: An example background image from the paint can video data. The image appears almost as an averaged smeared image of the entire video, the paintcan can not be seen since it is moving for most of the frames.

Figure 6: An example background image from the butterfly video data. The image appears almost as an averaged smeared image of the entire video, the butterfly can not be seen since it is moving for most of the frames.

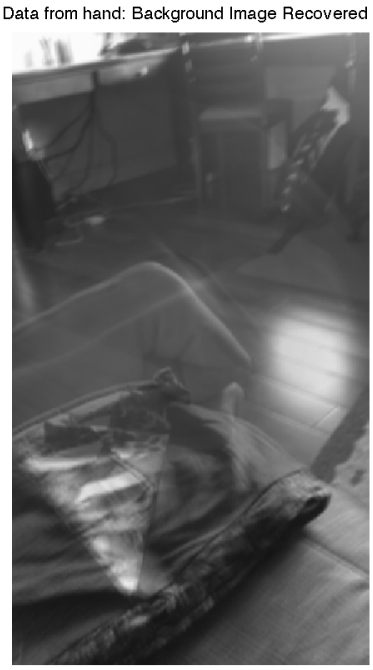

Figure 7: An example background image from the hand video data. The image appears almost as an averaged smeared image of the entire video, the hand can not be seen since it is moving for most of the frames.

Figure 8: An example foreground image from the hand video data.

Figure 9: An example foreground image from the butterfly video data.

Figure 10: An example full DMD reconstruction of the image from the butterfly video data. The image appears identical to the original image in the X data matrix.

Figure 11: An example foreground image from the paint video data. The image appears .

Figure 12: An example of the Aht background image returned from the rPCA method. The image appears as an averaged blur.

Figure 13: An example of the Eht foreground image from the paint video data. The image accentuates all motion (including that of the camera) in the frame.

A

[u,s,v] = svd(A, OPTION) Computes the singular value decomposition for the given matrix. In this report we use the option flag '0' to specify the reduced SVD.

[Ahat, Ehat] = inexactalmrpca(X, lambda) takes a data matrix X and a weighting lambda and separates the images into a foreground and background image Ahat and Ehat.

tic, toc reports the time to execute the code between tic and toc and print to stdout

B

```
function [r] = getR(original, constructed)
    % return the difference between the two matrices

    [m,n] = size(constructed);
    [o,p] = size(original);

    for k = 1:n
        temp = double(original(:,k)) - double(constructed(:,k));
    end

    r = sum(sum(temp));
end

function [bkd, ind] = getBackground(omega)
    %returns the background mode
    % chooses mode with value closest to 0,0
    [bkd, ind] = min(abs(real(omega)));
end
function [u_dmd, u_modes, bkd, phi, omega, mu, sigma, fgd_u_dmd, bkd_u_dmd, r_dmd] = ...
    dmd(X,dt,frames, features, predict);
    % return DMD of matrix X

    % boolean flag to subtract by modes or by final image
    byModes = 0;
    %create subsets
    % where X2 = AX1
    X1 = X(:,1:(end-1));
    X2 = X(:,2:end);
    %decompose the X1 matrix
    [u,sigma,v] = svd(X1,'econ');

    %build s (similarity matrix)
    s = u(:,1:features)'*X2*v(:,1:features) ...
        *diag(1./diag(sigma(:,1:features)));
    %find the eigen decomposition of s
    [ev, d] = eig(s);
    mu = diag(d); % eigen values of s matrix
    omega = log(mu)/(dt);
    % phi computes DMD modes to predict future state of system
    phi = u*ev;
```

```
% save on memory
clear v
clear u
% return the background mode and it's index
[bkd, bkdInd] = getBackground(omega);
y_0 = phi\X(:,1);
fgd_omega = omega; fgd_omega(bkdInd) = [];
bkd_omega = omega(bkdInd);
bkd_y0 = y_0(bkdInd);
fgd_y0 = y_0; fgd_y0(bkdInd) = [];
bkd_phi = phi(:, bkdInd);
fgd_phi = phi; fgd_phi(:, bkdInd) = [];
for iter=1:frames
    % sum all modes to produce reconstruct
    u_modes(:, iter) = (y_0.*exp(omega*(dt*iter)));
    % sum all non-background modes to produce foreground reconstruct
    bkd_u_modes(:, iter) = (bkd_y0.*exp(bkd_omega*(dt*iter)));
    fgd_u_modes(:, iter) = (fgd_y0.*exp(fgd_omega*(dt*iter)));
    if (byModes)
        fgd_u_modes(:, iter) = fgd_u_modes(:, iter) - ...
            bkd_u_modes(:, iter);
    end
end
% save on memory
u_dmd = uint8(phi*u_modes);
% fprintf('phi');
% size(phi)
% fprintf('umodes');
% size(u_modes)

% fprintf('bkdphi');
% size(bkd_phi)
% fprintf('bkdmodes');
% size(bkd_u_modes)

% fprintf('fphi');
% size(fgd_phi)
% fprintf('fumodes');
% size(fgd_u_modes)

bkd_u_dmd = uint8(bkd_phi*bkd_u_modes); % background
fgd_u_dmd = uint8(fgd_phi*fgd_u_modes);% foreground
```

```
    if (~byModes)
        [row, col, fr] = size(fgd_u_dmd);
        r_dmd = zeros(1, fr);
        % subtract background from foreground
        for kk= 1:fr
            r_dmd(kk) = sum(u_dmd(:,kk) - (fgd_u_dmd(:,kk) + bkd_u_dmd(:,kk)));
            fgd_u_dmd(:,kk) = fgd_u_dmd(:,kk) - bkd_u_dmd(:,kk);
        end
    end

e% Karl Marrett
% Dynamic Mode Decomposition for separating objects in video

close all;
addpath inexact_alm_rpca/
% don't reload mats each run
clearvars -except hand butterfly paint;
if (~exist('hand', 'var'))
    load hand.mat;
end
if (~exist('butterfly', 'var'))
    load butterfly.mat;
end
if (~exist('paint', 'var'))
    load paint.mat;
end

% names of raw video data
raw = {'hand', 'butterfly', 'paint'};
dat.hand = hand;
dat.butterfly = butterfly;
dat.paint = paint;

frm_rate = 30;
dt = 1 / frm_rate; %define timestep
frames = 100; % maximum frames to process
lambda = 0.0012;
predict = 1; % factor to predict future data given dmd modes
features = frames - 1; % rank/features of dmd
%[A_hat E_hat iter] = inexact_alm_rpca(D, lambda, tol, maxIter)

for k = 3:3 %(length(raw) - 1)
```

```
runName = raw{k};
[X, m, n] = getX(dat.(raw{k}), frames);
%where u_dmd is your foreground dmd approximation
% bkd_u_dmd is background
tic
[u_dmd, u_modes, bkd, phi, omega, mu, sigma, fgd_u_dmd,...
    bkd_u_dmd, r_dmd_fgdbkd] = dmd(X, dt, frames, features, predict);
toc
tic
[A_hat, E_hat, iter] = inexact_alm_rpca(X, lambda);
toc
pca_reconstruct = A_hat + E_hat;
[r_pca] = getR(X, pca_reconstruct);
fprintf('Total R error of RPCA approx: %d', r_pca);
[r_dmd] = getR(X, u_dmd);
fprintf('Total R error of DMD approx: %d', r_dmd);

% visualize
runStart = (k - 1)*3;
figure(runStart + 1)
set(gcf, 'visible', 'off');
plot(diag(sigma), 'ko','Linewidth', [2]);
title(strcat({'Data from '}, runName,': Sigmas'));
saveas(runStart + 1, strcat(runName, 'sig'), 'png');

figure(runStart + 2)
set(gcf, 'visible', 'off');
plot(mu, 'ko','Linewidth', [2]);
title(strcat({'Data from '}, runName,': Mu values'));
saveas(runStart + 2, strcat(runName, 'mu'), 'png');

figure(runStart + 3)
set(gcf, 'visible', 'off');
plot(omega, 'ko','Linewidth', [2]);
title(strcat({'Data from '}, runName,': Omega Modes'));
saveas(runStart + 3, strcat(runName, 'omega'), 'png');

vid_reconstruct = get_img(u_dmd, m, n);
vid_foreground = get_img(fgd_u_dmd, m, n);
img_background = get_img(bkd_u_dmd, m, n);
vid_pca_a = get_img(A_hat, m,n);
vid_pca_e = get_img(E_hat, m,n);
```

```
    frameNo = 50;
    figure(runStart + 4)
    set(gcf, 'visible', 'off');
    imshow(img_background(:,:,frameNo));
    title(strcat({'Data from '}, runName,': Background Image Recovered'));
    saveas(runStart + 4, strcat(runName, 'bkd'), 'png');

    figure(runStart + 5)
    set(gcf, 'visible', 'off');
    imshow(vid_foreground(:,:,frameNo));
    title(strcat({'Data from '}, runName,': a foreground image'));
    saveas(runStart + 5, strcat(runName, 'fgd'), 'png');

    figure(runStart + 6)
    set(gcf, 'visible', 'off');
    imshow(vid_reconstruct(:,:,frameNo));
    title(strcat({'Data from '}, runName,': a fully reconstructed image'));
    saveas(runStart + 6, strcat(runName, 'full'), 'png');

    figure(runStart + 7)
    set(gcf, 'visible', 'off');
    imshow(vid_pca_a(:,:,frameNo));
    title(strcat({'Data from '}, runName,': rPCA A_Hat image'));
    saveas(runStart + 7, strcat(runName, 'ahat', int2str(lambda*10000)), 'png');

    figure(runStart + 8)
    set(gcf, 'visible', 'off');
    imshow(vid_pca_e(:,:,frameNo));
    title(strcat({'Data from '}, runName,': rPCA E_Hat image'));
    saveas(runStart + 8, strcat(runName, 'ehat', int2str(lambda*10000)), 'png');
end
```

References

[1] Jacob Grosek and J. Nathan Kutz. Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video. (Dmd):14, April 2014.

[2] J N Kutz. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. OUP Oxford, 2013.