AMATH482 HW06

Joshua Borgman

University of Washington

Josh Borgman

Friday, March 20, 2015

Contents

1. Introduction and Overview

Exploiting low dimensionality effective method for computational or theoretical reduction of a given system to a more tractable form. rPCA is model based algorithm with a given set of governing equations dictating the dynamics of the underlying system. Alternatively data based algorithms can be used to understand, mimic, and control complex systems. Data-based algorithms are used when either the model equations are beyond their range of validity or their governing equations simply are not known or well-formulated. Thus experimental data is used to drive our understanding. Dynamic mode decomposition is such a data based algorithm that uses no underlying governing equation, but relies on experimental "snapshots" used to predict and control a system of interest. Thus, with DMD methods, the dynamics are assumed to be linear so that an exact solution can be constructed and future predictions do not require any additional work.

Alternatively in POD methods, a low dimensional basis is selected from the data matrix **X** by used of the SVD. The dynamics of the governing equations are projected onto this reduced basis and then evolved forward in time. While this is computationally more expensive, the POD method retains the nonlinear dynamics of the system then its projection onto the governing equations.

Both methods seem to have advantages and pitfalls, thus it is up to the user to exploit the particularities of a given system of interest when applying these methods.

2. Theoretical Background

Dynamic Mode Decomposition (DMD)

Dynamic mode decomposition begins with the data collection process. Namely two parameters are used to begin the analysis. *N* is the number of spatial points saved per time snapshot (in our assignment it will be the pixel matrices at a given time). *M* is the number of snapshots taken (in our case it will be the number of frames taken by our cameras). For our application and implementation it will be a necessity that collected data is at regularly spaced intervals of time. Thus we can arrange the data into an *N x M* matrix

$$\boldsymbol{X} = [U(\boldsymbol{x}, t_1)\ U(\boldsymbol{x}, t_2) \ldots U(\boldsymbol{x}, t_M)]$$

And matrix **x** is a vector of data collection points of length *N*. The DMD method seeks to approximate the modes of the *Koopman operator.* The Koopman operator is a linear, infinite dimensional operator that represents nonlinear, infinite dimensional dynamics without linearization, and is the adjoint of the Perron-Frobenius operator. Thus we can view the DMD method as computing the eigenvalues and eigenvectors of experimental data, which represent a linear model that approximates the underlying dynamics of the system, even if the dynamics are nonlinear. *The underlying model assumes linearity*, so the decomposition gives the growth rates and frequencies associate with each mode. The computation of the Koopman operator is at the heart of the DMD method, representing a linear, time independent operator **A** such that

$$\boldsymbol{x}_{j+1} = \boldsymbol{A}\boldsymbol{x}_j$$

Representing a mapping of the data from time t_j to time t_{j+1} .

In order to construct the Koopman operator, we must consider the foreword stepped matrix,

$$\boldsymbol{X}_1^{M-1} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_{M-1}}] = [\boldsymbol{x_1}, \boldsymbol{A}\boldsymbol{x_1}, \boldsymbol{A^2}\boldsymbol{x_1}, \dots, \boldsymbol{A^{M-1}}\boldsymbol{x_1}]$$

Formally what the DMD algorithm seeks to do is fit the first *M-1* data collection points using the Koopman Operator. Thus the final data point $\mathbf{x}_\mathbf{M}$ is represented as best as possible in terms of a Krylov basis

$$\boldsymbol{x_M} = \sum_{m=1}^{M-1} b_m \boldsymbol{x}_m + \boldsymbol{r}$$

With b_M being the coefficients of the Krylov space vectors and $\boldsymbol{r}$ the residual that lies outside the Krylov space. This best fit the data using the DMD procedure will be computed in an L^2 sense using the pseudo-inverse. We will seek to take advantage of any low dimensional structures in the data by exploiting the SVD. It is useful to be reminded that if the data matrix is full rank and the data have no suitable low dimensional structure then the DMD method fails immediately. However, often times the low-dimensionality allows DMD to take advantage of the structure to project a future state of the system. Generalizing the approach above we see

$$\boldsymbol{A}\boldsymbol{X_1^{M-1}} = \boldsymbol{X_2^M} = \boldsymbol{X_1^{M-1}}\boldsymbol{S} + \boldsymbol{r}^*_{M-1}$$

Where e_{M-1} is the (M-1)th unit vector and

$$\boldsymbol{S} = \begin{bmatrix} 0 & \cdots & & \cdots & 0 & b_1 \\ 1 & \ddots & & & 0 & b_2 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & & 0 & b_{M-2} \\ 0 & \cdots & 0 & & 1 & b_{M-1} \end{bmatrix}$$

Rather than specifically computing the matrix **S** we calculate

$$\tilde{\boldsymbol{S}} = \boldsymbol{U}^*\boldsymbol{X_2^M}\boldsymbol{V}\boldsymbol{\Sigma^{-1}}$$

Which is related by a similarity transformation. We then consider the eigenvalue problem

$$\tilde{\boldsymbol{S}}\boldsymbol{y_k} = \mu_k \boldsymbol{y_k} \qquad k = 1,2,..,K$$

Where *K* is the rank of the approximation we are making, and the eigenvalues μ_k capture the time dynamics of the discrete Koopman map as a setep is taken forward in time.

Relating back to the similarity transformed original eigenvalues and vectors of **S** in order to construct the DMD modes we have

$$\psi_k = \boldsymbol{U}\boldsymbol{y}_k$$

And thus the approximate solution at all future times $\mathbf{x}_{\mathrm{DMD}}(t)$, is given by

$$\boldsymbol{x}_{DMD}(t) = \sum_{k=1}^{K} b_k(0)\psi_k(\boldsymbol{x})\exp(\omega t) = \Psi diag(\exp(\omega t))\boldsymbol{b}$$

Where b_k (0) is the initial amplitude of each mode Ψ is the matrix whose columns are the eigenvectors ψ_k, and $\boldsymbol{b}$ is the vector of coefficients b_k. We solve for **b** using the pseudo inverse.

Unlike the POD method, which requires solving a low-rank set of dynamical quantities to predict the future states, there is no additional work required for its future state predictions. Thus the advantages of DMD center around the requirement that no equations are needed and the future state is known for all time, provided the DMD approximation holds. The approximation will likely diverge at some point though given the exponential growth experienced in the exponential nature of the real parts of our eigenvalues. Any noise in the system can push the eigenvalue outside the unit circle and limit the range of the DMD prediction. Sampling rate has a noticeable effect on the strength of the predictions, with greater sampling rates producing DMD predictions that are valid for longer windows in time.

Robust PCA/POD/SVD

The main concept differentiating robust PCA/POD/SVD from the standardized versions previously discussed is the measure of fit being performed. Traditionally a L^2 norm is considered when implementing these procedures. While the L^2 norm is highly favored for its centrality and physical interpretation in most applications as an energy, such a fitting schemes is highly susceptible to the effects of outliers and sparse noise fluctuations. These fluctuations apply severe costs as the higher dimensional least-square fitting amplifies this error. This amplification of error leads to significant deformation of the singular values and PCA/POD modes describing the data. The robust PCA technique provides an algorithm capable of separating low-rank data with corrupted (sparse) measurement/matrix error. More specifically we are to consider a matrix which is composed of these two matrices

$$\boldsymbol{M} = \boldsymbol{L} + \boldsymbol{S}$$

Where **M** is the data matrix of interest, **L** is the low rank structure matrix and **S** some sparse data **S**. The difficulty in separation into these submatrices includes the fact that neither the rank of the matrix **L**, nor the number of nonzero elements of the matrix **S** is known. It is worth noting that in most practical applications, the matrix decomposition often suffers from small perturbations accounting for the fact that the low-rank component is only approximately low rank and that small errors can be added to all the entries. This yields a more generalized decomposition form

$$\boldsymbol{M} = \boldsymbol{L} + \boldsymbol{S} + \boldsymbol{N}$$

Where **N** is this set of dense, small perturbations.

3. Algorithm Implementation and Development

Dynamic Mode Decomposition (DMD)

The generalized algorithm proceeds as follows:

i. Sample data at N prescribed locations M times. The data slices should be evenly spaced with regard to time. This produces our data matrix X.
ii. From the data matrix X, construct the two submatrices X_1^{M-1} and X_2^M.
iii. Compute the SVD decomposition of X_1^{M-1}.
iv. The matrix similarity matrix $\tilde{\boldsymbol{S}}$ can then be computed and its eigenvalues and eigenvectors found.
v. Project the initial state of the system onto the DMD modes using the pseudo-inverse.

vi. Using these results compute the solution at any future time using the DMD modes along with their projection to the initial conditions and the time dynamics computed using the eigenvalue of $\widetilde{\boldsymbol{S}}$.

Robust PCA/POD

In this work, the robust PCA algorithm **inexact_alm_rpca** was used, which implements the inexact augmented Lagrange multiplier method for Robust PCA [2]. It should be noted that the use of **inexact_alm_**rpca requires the data be real. Further specifics of the code implementation can be viewed in the author commented sections in Appendix A. Selection of the lambda value input was subjective and seemed to perform well around .0012.

1. Computational Results

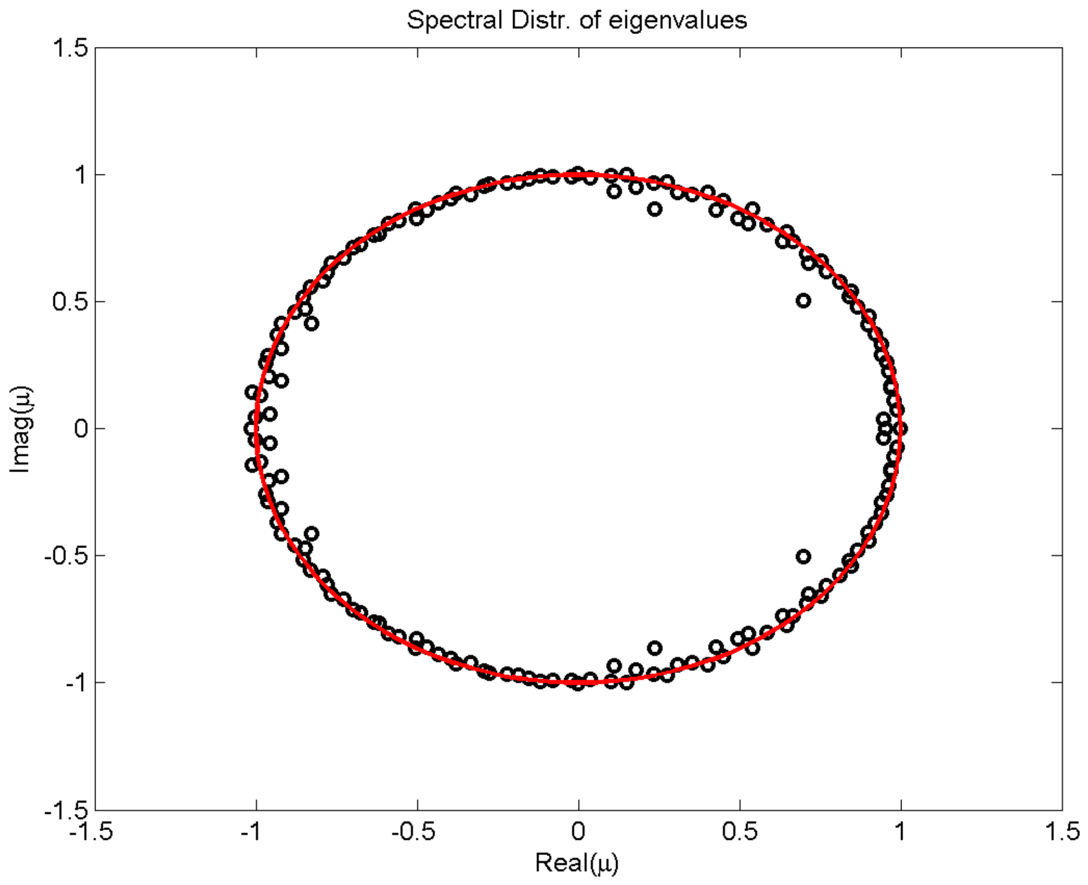


Figure 1: This image shows the eigenvalues computed as part of the DMD method. Note that eigenvalues that lie outside the unit circle (in red) make the approximation unstable, yielding a limited window of accuracy before the solution deteriorates.

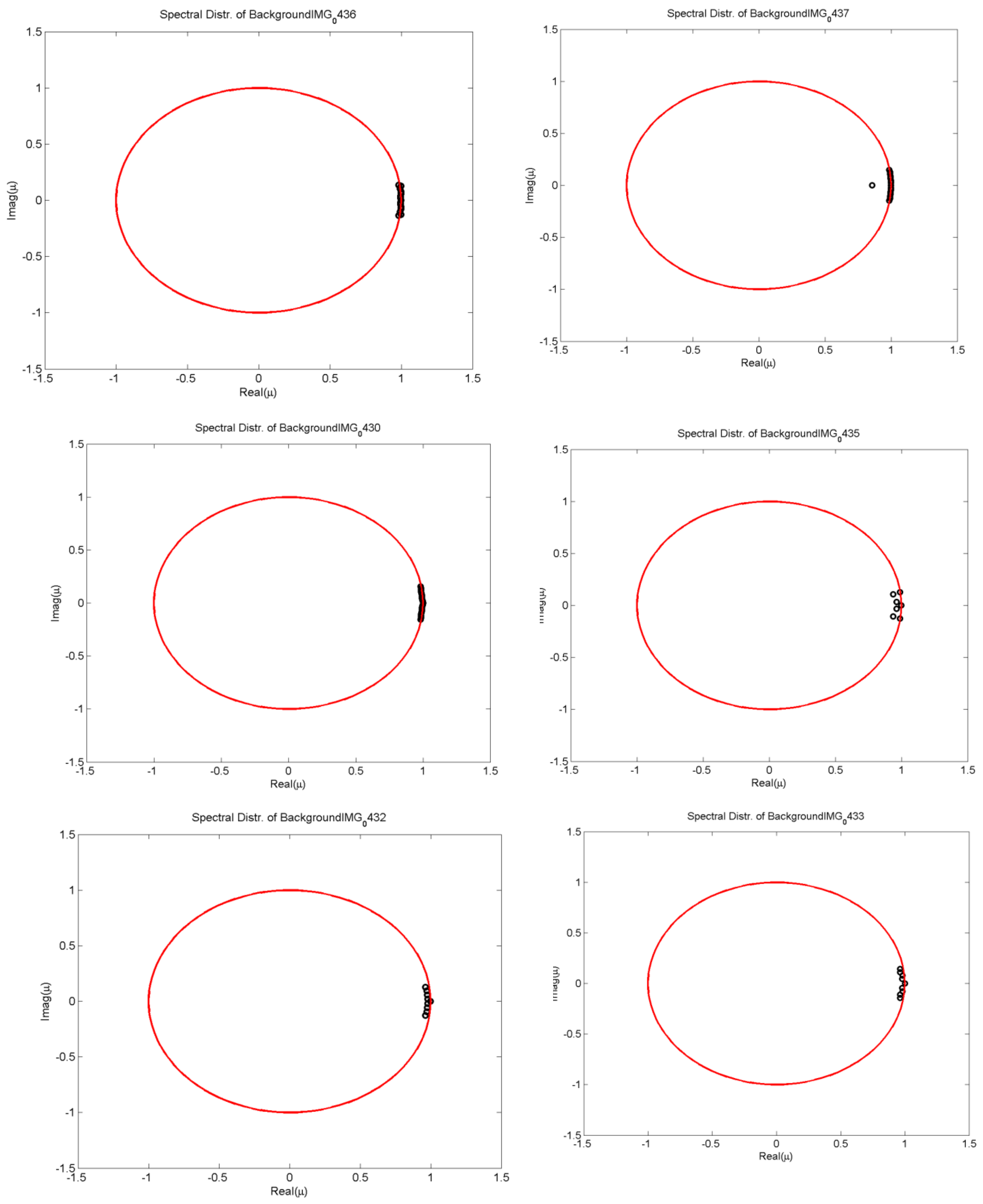


Figure 2: This figure shows the extracted background eigenvalues for the clips.

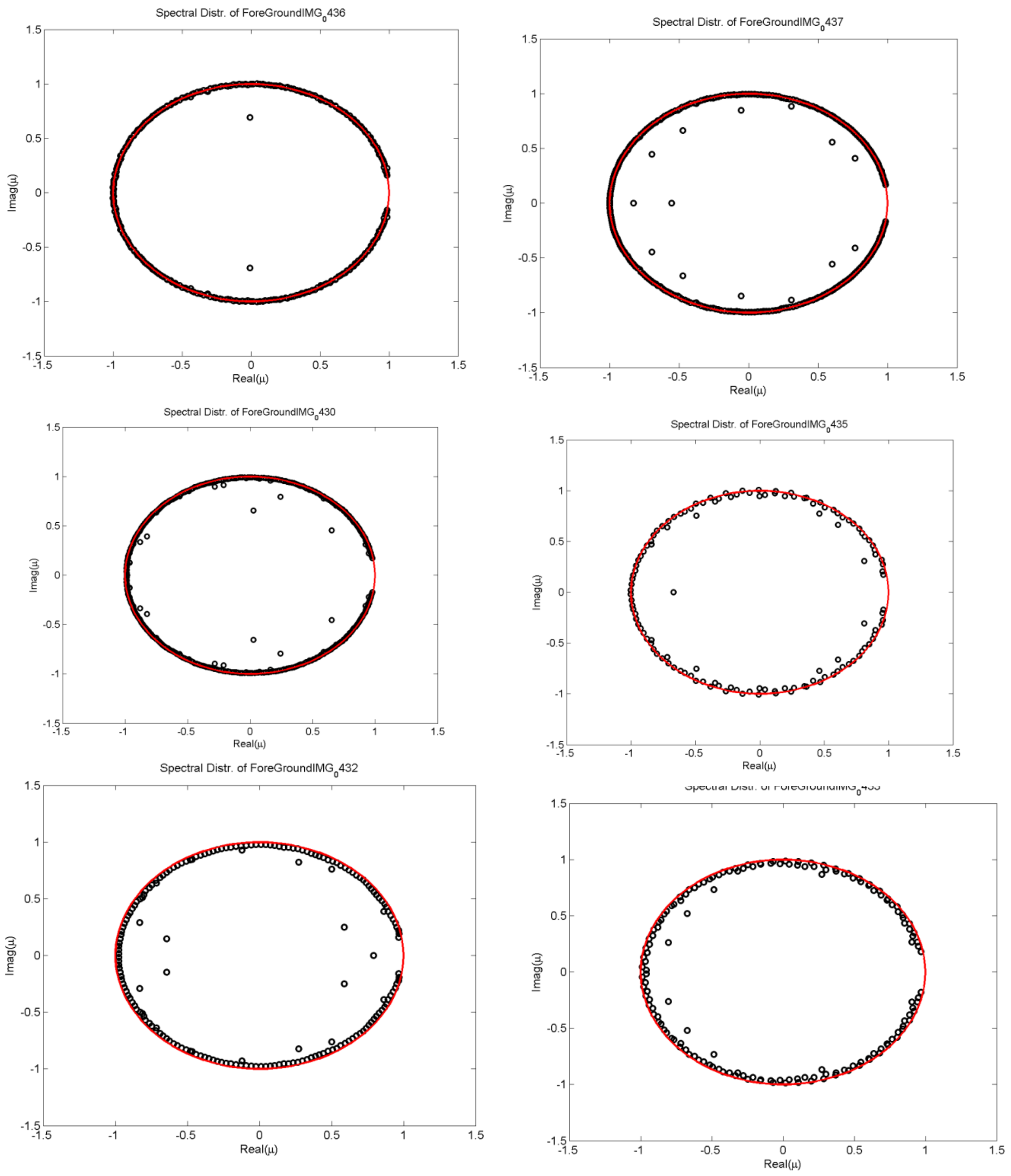
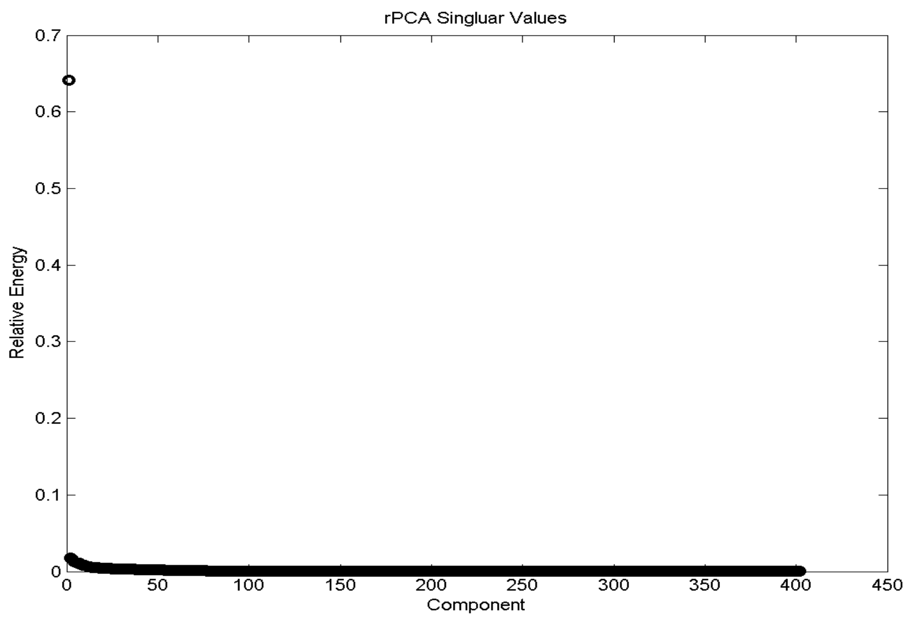
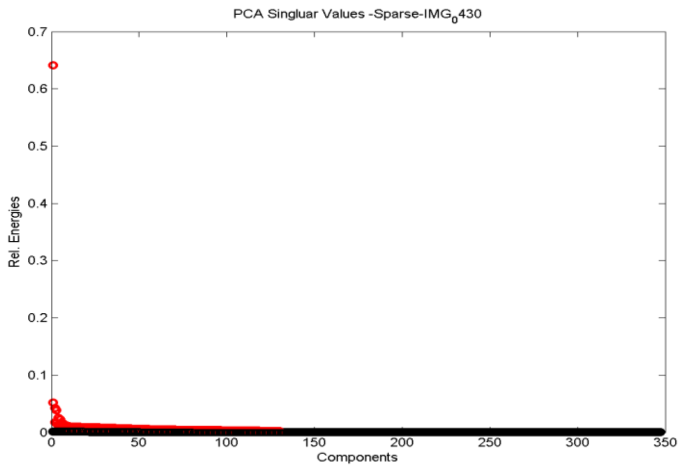


Figure 3: This Figure Shows the foreground eigen components extracted from the clips.





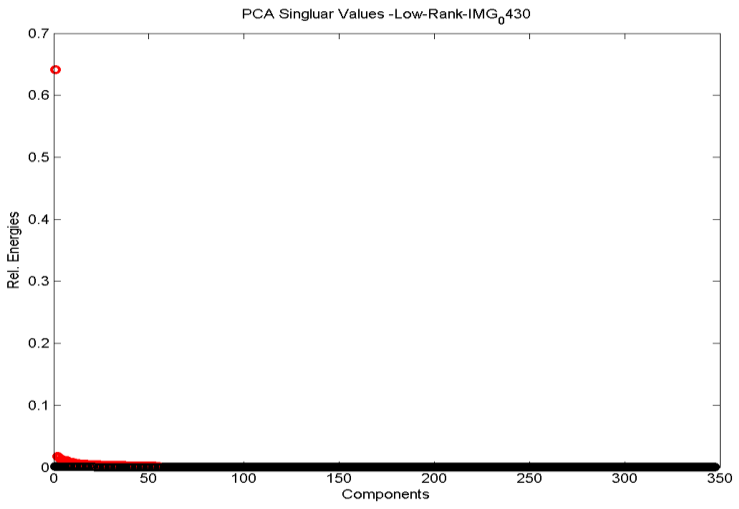


Figure 4: This is the principle components of our robust PCA algorithms. Above is the overall image, the other two are the sparse and low rank approximations respectively.

2. Summary and Conclusion

The exploitation of low dimensional algorithms has great power, yielding opportunities for insight even when confronted with an unknown system. This gives the user an awesome amount of power provided they have ample computational power or low-dimensional systems. Using these approximations we can view the dynamics of the system locating relatively time invariant components (i.e. abs(omega) ~0) and separate them.

It is again worth noting the strengths and weaknesses of both methods.

In DMD methods, the dynamics are assumed to be linear so that an exact solution can be constructed and future predictions do not require any additional work. This comes at the cost of losing the future system dynamics and having limited confidence past a specified window.

In comparison, the POD method retains the nonlinear dynamics of the system then its projection onto the governing equations but is computationally more expensive and complex.

The main point is that application of either must be tailored to the characteristics of the given system, exploiting unique features of the system.

Appendix A:

```
function [A_hat E_hat iter] = inexact_alm_rpca(D, lambda, tol, maxIter)

% Oct 2009
% This matlab code implements the inexact augmented Lagrange multiplier
% method for Robust PCA.
%
% D - m x n matrix of observations/data (required input)
%
% lambda - weight on sparse error term in the cost function
%
% tol - tolerance for stopping criterion.
%     - DEFAULT 1e-7 if omitted or -1.
%
% maxIter - maximum number of iterations
%         - DEFAULT 1000, if omitted or -1.
%
% Initialize A,E,Y,u
% while ~converged
%   minimize (inexactly, update A and E only once)
%     L(A,E,Y,u) = |A|_* + lambda * |E|_1 + <Y,D-A-E> + mu/2 * |D-A-E|_F^2;
%   Y = Y + \mu * (D - A - E);
%   \mu = \rho * \mu;
% end
%
% Minming Chen, October 2009. Questions? v-minmch@microsoft.com ;
% Arvind Ganesh (abalasu2@illinois.edu)
%
% Copyright: Perception and Decision Laboratory, University of Illinois,
Urbana-Champaign
%            Microsoft Research Asia, Beijing

addpath PROPACK;

[m n] = size(D);

if nargin < 2
    lambda = 1 / sqrt(m);
end

if nargin < 3
    tol = 1e-7;
elseif tol == -1
    tol = 1e-7;
end

if nargin < 4
    maxIter = 1000;
elseif maxIter == -1
    maxIter = 1000;
end

% initialize
Y = D;
```

```
norm_two = lansvd(Y, 1, 'L');
norm_inf = norm( Y(:), inf) / lambda;
dual_norm = max(norm_two, norm_inf);
Y = Y / dual_norm;

A_hat = zeros( m, n);
E_hat = zeros( m, n);
mu = 1.25/norm_two % this one can be tuned
mu_bar = mu * 1e7
rho = 1.5          % this one can be tuned
d_norm = norm(D, 'fro');

iter = 0;
total_svd = 0;
converged = false;
stopCriterion = 1;
sv = 10;
while ~converged
    iter = iter + 1;

    temp_T = D - A_hat + (1/mu)*Y;
    E_hat = max(temp_T - lambda/mu, 0);
    E_hat = E_hat+min(temp_T + lambda/mu, 0);

    if choosvd(n, sv) == 1
        [U S V] = lansvd(D - E_hat + (1/mu)*Y, sv, 'L');
    else
        [U S V] = svd(D - E_hat + (1/mu)*Y, 'econ');
    end
    diagS = diag(S);
    svp = length(find(diagS > 1/mu));
    if svp < sv
        sv = min(svp + 1, n);
    else
        sv = min(svp + round(0.05*n), n);
    end

    A_hat = U(:, 1:svp) * diag(diagS(1:svp) - 1/mu) * V(:, 1:svp)';

    total_svd = total_svd + 1;

    Z = D - A_hat - E_hat;

    Y = Y + mu*Z;
    mu = min(mu*rho, mu_bar);

    %% stop Criterion
    stopCriterion = norm(Z, 'fro') / d_norm;
    if stopCriterion < tol
        converged = true;
    end

    if mod( total_svd, 10) == 0
        disp(['#svd ' num2str(total_svd) ' r(A) ' num2str(rank(A_hat))...
            ' |E|_0 ' num2str(length(find(abs(E_hat)>0)))...
```

```
                ' stopCriterion ' num2str(stopCriterion)]);
    end

    if ~converged && iter >= maxIter
        disp('Maximum iterations reached') ;
        converged = 1 ;
    end
end
```

Appendix B:

DMD Code

```
%{
Joshua Borgman  - AMATH 482 : Homework 6: DMD wtih robust POD
03/19/2015

%}

clear all; close all; clc

%% Generation of Data Matrix X
%Step 1:
%Sample data at N prescribed locations, M times; Data snapshots
%should be evenly spaced by a fixed time interval. This gives Data Matrix
%X.

fileName = 'IMG_0433.MOV'; %Video File name in directory
obj = VideoReader(fileName);
fps = get(obj, 'FrameRate'); noFrames = get(obj, 'NumberOfFrames');
h = get(obj, 'height'); w = get(obj, 'width');

vid = read(obj);
vidBW = zeros(h,w,noFrames);
for j=1:noFrames
    vidBW(:,:,j) = rgb2gray(vid(:,:,:,j));
end

t = linspace(0,fps*noFrames, noFrames); dt = t(2)-t(1); %M = noFrames
%%
r = [0 2 3 5 6 7];

for j =1: length(r)
    file = strcat('IMG_043', num2str(r(j)));
    fileName = strcat(file,'.MOV'); %Video File name in directory
    obj = VideoReader(fileName);
    fps = get(obj, 'FrameRate'); noFrames = get(obj, 'NumberOfFrames');
    h = get(obj, 'height'); w = get(obj, 'width');

    vid = read(obj);
    vidBW = zeros(h,w,noFrames);
    for j=1:noFrames
        vidBW(:,:,j) = rgb2gray(vid(:,:,:,j));
    end
    save(file, 'vidBW', 'fps', 'noFrames', 'h', 'w', 'vid');
end
%%
for n = [0 2 3 5 6 7]
    clearvars -except n
    %clear all; close all; clc
    file = strcat('IMG_043',num2str(n));
    load(strcat(file, '.mat'));
    t = linspace(0,fps*noFrames, noFrames); dt = t(2)-t(1); %M = noFrames
    X = zeros(h*w, noFrames);
%Xr = zeros(h*w, noFrames); Xg = zeros(h*w, noFrames); Xb = zeros(h*w,
noFrames);

    for j = 1:noFrames
        X(:,j) = reshape(vidBW(:,:,j),h*w,1);
    end

% for j = 1:noFrames
%     Xr(:,j) = reshape(vid(:,:,1,j),h*w,1);
%     Xg(:,j) = reshape(vid(:,:,2,j),h*w,1);
%     Xb(:,j) = reshape(vid(:,:,3,j),h*w,1);
% end


% DMD Algorithm


%Step 2:
%From the data matrix (X), construct two submatrices
%X^(M-1)_1 and X^(M)_2
    X1 = X(:, 1:end -1); X2 = X(:, 2:end);

%Step 3:
%Compute the SVD decomposition of X^(M-1)_1
    [U, Sigma, V] = svd(X1, 'econ');

%Step 4:
%Compute the similarity matrix S\tilda and find its eigenvalues and vectors
    S = U'*X2*V*diag(1./diag(Sigma));
    [eV, D] = eig(S);
    mu = diag(D);
    omega = log(mu)/(dt);  %contains the DMD dynamics.
    Phi = U*eV; %Contains the DMD modes which are the basis functions

%
%Step 5:
%Project the intial state of the system iont o the DMD modes using the
%pseudo-inverse.
    y0 = Phi\X1(:,1); % pseduo-inverse intial conditions
    %
    u_modes = zeros(size(X1,2),length(t));

    for j = 1:length(t)
        u_modes(:, j) = (y0.*exp(omega*t(j)));
```

```
end

u_dmd = Phi*u_modes;

% Reconstruct without fore/background?
thresh = 0.05;
ind = find(abs(omega)<=thresh*max(abs(omega)));
yFore = y0; yFore(ind)=[];
yback = y0(ind);

omegaFore=omega; omegaFore(ind) = [];
omegaback = omega(ind);

u_modesFore = zeros(size(X1,2)-length(ind),length(t));
u_modesBack = zeros(length(ind), length(1));

for j = 1:length(t)
    u_modesFore(:, j) = (yFore.*exp(omegaFore*t(j)));
    u_modesBack(:, j) = (yback.*exp(omegaback*t(j)));

end

PhiFore = Phi; PhiFore(:,ind)=[];
PhiBack = Phi(:, ind);

u_fore = PhiFore*u_modesFore;
u_back = PhiBack*u_modesBack;

u_dmdf = zeros(h,w,noFrames);
u_dmdb = zeros(h,w, noFrames);

for j = 1:noFrames
    u_dmdf(:,:,j) = reshape(u_fore(:,j),h,w,1);
    u_dmdb(:,:,j) = reshape(u_back(:,j),h,w,1);
end

%Step 6:
%Compute the solution at any future time using the DMD modes along witht
%heir projection to the intial conditions and the time dynamics computed
%using the egienvalues of S\tilda

% Plot Spectral distribution of eigenvalues mu_k
plot(real(mu), imag(mu), 'ko', 'Linewidth', [2])
set(gca,'Fontsize', [12]), title('Spectral Distr. of eigenvalues'),
xlabel('Real(\mu)'), ylabel('Imag(\mu)');
axis([-1.5 1.5 -1.5 1.5]);
[xc, yc] = pol2cart(linspace(0,2*pi,100),1);
hold on, plot(xc,yc, 'r-', 'Linewidth', [2]);
close all;

% Plot Spectral dist of Background

plot(real(mu(ind)), imag(mu(ind)), 'ko', 'Linewidth', [2])
```

```
    set(gca,'Fontsize', [12]), title(strcat('Spectral Distr. of Background ',
file)),
    xlabel('Real(\mu)'), ylabel('Imag(\mu)');
    axis([-1.5 1.5 -1.5 1.5]);
    [xc, yc] = pol2cart(linspace(0,2*pi,100),1);
    hold on, plot(xc,yc, 'r-', 'Linewidth', [2]);

    print(strcat('SpectDistBack', file, '.png'), '-dpng');
    close all;

    % Plot Spectral dist of Foreground
    muFore = mu; muFore(ind) = [];
    plot(real(muFore), imag(muFore), 'ko', 'Linewidth', [2])
    set(gca,'Fontsize', [12]), title(strcat('Spectral Distr. of ForeGround ',
file)),
    xlabel('Real(\mu)'), ylabel('Imag(\mu)');
    axis([-1.5 1.5 -1.5 1.5]);
    [xc, yc] = pol2cart(linspace(0,2*pi,100),1);
    hold on, plot(xc,yc, 'r-', 'Linewidth', [2]);

    print(strcat('SpectDistFore', file, '.png'), '-dpng');
    close all;
    save(strcat(file, 'results'), 'vidBW', 'u_dmdb', 'u_dmdf')
end

%%
close all;

for j=1:noFrames
    imshow(rgb2gray(vid(:,:,:,j))); drawnow
end


%%

close all;
u_play = uint8(u_dmdf);
% for j=1:noFrames
%     imshow(uint8(u_play(:,:,j))); drawnow
% end

for k = 1 : noFrames
    mov(k).cdata = u_play(:,:,k);
    mov(k).colormap = gray;
end


vidObj = VideoWriter(strcat(file, '.avi'));
open(vidObj);
for j=1:noFrames
    A=frame2im(mov(j));
    writeVideo(vidObj,A);
end
close(vidObj);
```

```
%% Save the data material to play on comp with imshow
save(strcat(file, 'results'), 'vidBW', 'u_dmdb', 'u_dmdf')
```

rPCA Code

```
%{
Joshua Borgman  - AMATH 482 : Homework 6: DMD wtih robust POD
03/19/2015

%}

clear all; close all; clc
%% robust POD and Mode Selection


for n = [0 2 3 5 6 7]
    clearvars -except n; close all;
    file = strcat('IMG_043',num2str(n));
    load(strcat(file, '.mat'));
    t = linspace(0,fps*noFrames, noFrames); dt = t(2)-t(1); %M = noFrames
    X = zeros(h*w, noFrames);
%Xr = zeros(h*w, noFrames); Xg = zeros(h*w, noFrames); Xb = zeros(h*w,
noFrames);

    for j = 1:noFrames
        X(:,j) = reshape(vidBW(:,:,j),h*w,1);
    end
    lambda = 0.0012; thresh = 0.95;
    [R1 R2] = inexact_alm_rpca(X, lambda);
    [U1, S1, V1] = svd(R1.', 'econ');
    [U2, S2, V2] = svd(R2.', 'econ');
    ind1 = find(cumsum(diag(S1))/sum(diag(S1))>=thresh,1);
    ind2 = find(cumsum(diag(S2))/sum(diag(S2))>=thresh,1);
    save(strcat(file, 'rPCAresults'), 'U1', 'U2', 'S1', 'S2', 'V1', 'V2',
'ind1', 'ind2')
end
%% Plot Singular Values

S = S2; spec = 'Sparse';
dS = diag(S); thresh = 0.9;
ind = find(cumsum(diag(S))/sum(diag(S))>=thresh,1);
plot(dS(1:ind)/sum(dS), 'ro', 'Linewidth', [2]), hold on,
plot(dS(ind+1:end)/sum(dS), 'ko', 'Linewidth', [1])
title(strcat('PCA Singluar Values - ',spec, '-', file)),
xlabel('Components'), ylabel('Rel. Energies');
print(strcat('PCA Singluar Values - ',spec, '-', file,'.png'), '-dpng');
```

References

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

ii. Candès, Emmanuel J., et al. "Robust principal component analysis?." *Journal of the ACM (JACM)* 58.3 (2011): 11.