Amath563: Finding Governing Equations from Data

Helena Liu, May 1st, 2020

Abstract:

This work finds a dynamical model given some data using sparse regression and dynamical mode decomposition. The framework is implanted in MATLAB and applied to two data sets. The results demonstrate the potential and limitation of these methods.

I. Introduction and Overview

Extracting governing equations to explain observable data is fundamental in understanding dynamical systems. Hence, techniques must be developed to extract meaningful information from data. Two such techniques are the dynamical mode decomposition (DMD) and sparse regression. This work outlines these two techniques, implement them in MATLAB and apply these methods to a predator-prey dataset as well as a chemical reaction dataset.

II. Theoretical Background

A) Regression for Dynamical Systems

Suppose we have m datapoints $x \in \mathbb{R}^d$ and would like to construct a nonlinear dynamical system, $\dot{x} = N(x,t)$, one approach is to construct a library of n candidate functions $f_i(x) \in \mathbb{R}^m$, *i=1,2,...n,* such that $\dot{x}$ can be written as a linear combination of these functions:

$$\begin{pmatrix} | & | & \dots & | \\ f_1 & f_2 & \dots & f_n \\ | & | & \dots & | \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} \dot{x_1} \\ \vdots \\ \dot{x_m} \end{pmatrix}.$$

Hence, finding a model in such setup would be to find the coefficients c. This becomes a problem of solving for the system the system $Ac = b$, where A is the function library matrix and b consists of the data derivative estimate. Because m is usually much larger than n, the above system is overdetermined and there is usually not an x that satisfies all m equations. This can be addressed using a least square criterion, where c is picked to minimize $J = \|Ac - b\|_2^2$.

Using as few terms as possible in a model is often desirable due to better interpretability and generalization. In general, these methods involve an additional L1-term in the cost function,

$$J = \frac{1}{2}\|Ac - b\|_2^2 + \lambda\|x\|_1 , \quad (1)$$

which punishes dense solution. λ is a parameter that allows one to control the importance of sparsity promotion relative to small least squares error.

B) Dynamical Mode Decomposition

Another system identification tool is dynamical mode decomposition (DMD) [1]. Its formulation assumes linear dependence between datapoints. Given a collection of data snapshots, DMD seeks for a linear operator A that relates k[th] data point, x_k, to the next:

$$x_{k+1} = Ax_k, \textit{for } k = 1, 2, \dots, m\text{-}1,$$

where m is the total number of snapshots collected. These snapshots are then organized into matrices X and X':

$$X = \begin{pmatrix} | & | & ... & | \\ x_1 & x_2 & ... & x_{m-1} \\ | & | & ... & | \end{pmatrix}, X' = \begin{pmatrix} | & | & ... & | \\ x_2 & x_3 & ... & x_m \\ | & | & ... & | \end{pmatrix},$$

$$\text{with } X' = AX.$$

Since X is usually high dimensional, A would be too expensive to compute using pseudo-inverse. Instead, the following steps using SVD are used [1]:

1. Compute the economy sized SVD of X:
$$X = U\Sigma V^*$$
2. Obtain the reduced rank matrix $\tilde{A}$:
$$A = X'X^+ = X'V\Sigma^{-1}U^*$$
$$\tilde{A} = U^*AU = U^*X'V\Sigma^{-1}$$
3. Compute the eigendecomposition of $\tilde{A}$:
$$\widetilde{AW} = W\Lambda$$
4. Project back to the original high-dimensional system and obtain the DMD modes:

$\phi = X'V\Sigma^{-1}W$, which gives $X = \phi e^{\Omega t}b$, where $X(0) = \phi b$.

One major caveat is the linearity assumption. This can be addressed by first form a linear embedding of the data from the time-delay embedded Hankel matrix, *H*, which does row-wise stacking of *d* time delayed snapshots as follows:

$$H = \begin{pmatrix} x_1 & x_3 & ... & x_{m-d+1} \\ x_2 & x_3 & ... & x_{m-d+2} \\ \vdots & \vdots & ... & \vdots \\ x_d & x_{d+1} & ... & x_m \end{pmatrix}.$$

C) KL Divergence, AIC, BIC

To measure how well the model matches the actual data, KL divergence, AIC and BIC scores are used. KL divergence, which is given as follows, measures how much information is lost about the distribution *P* when it is being approximated by distribution *Q*:

$$KL(P,Q) = \sum_j P(j)\log\left(\frac{P(j)}{Q(j)}\right).$$

AIC and BIC scores are given as:

$$AIC = 2n - 2lnL, \quad BIC = nlnm - 2lnL,$$

which penalizes models with large number of parameters, n. Here, m is the sample size and lnL is the log likelihood, which is given by

$$lnL = -\frac{m}{2}ln2\pi - \frac{m}{2}ln\sigma^2 - \frac{m}{2},$$

where σ^2 is the variance of residual between the model and actual data. The log likelihood assumes that the residuals are i.i.d. normal random variables with zero-mean.

III. Algorithm Implementation and Development

This section outlines the practical implementation of the theoretical methods discussed in the previous section. All MATLAB modules are provided in **Appendix A**. A local cubic spline interpolation was applied to predator-prey data in order to yield additional datapoints and reduce the step size of finite difference method for derivative approximation.

A) Dynamical Mode Decomposition

Another major caveat of DMD, in addition to its linear assumption, is that it can only be used for short term prediction because modes with large positive/negative real values will diverge/decay quickly. A workaround of this shortcoming is to use a sliding window approach. For the predator-prey data, the last two years of each sliding window was saved for testing. Two was chosen heuristically from the rate of expansion/decay of the modes. It is important to note that for time delay embedding, those two years were not included in the Hankel matrix for training. The first 20 years were used for training, and this number was chosen to ensure there were enough points for the Hankel matrix. The window then slides forward by the segment used for testing each time. Similar methods were used to construct the sliding window DMD for the BZ reaction data. For time-delay embedding, the number of delay points, d, was chosen such that it was enough to create SVD modes that resembled sinusoids, since the goal is to find a linear embedding of the data. For the predator-prey data, d was chosen to be 8 years. Similar methods were used in setting up DMD for BZ reaction data.

B) Regression for Dynamical Systems

Three different libraries of functions were used in this work. The first library consists of only terms in a Lotka-Volterra dynamical model. The second library consists of polynomials up to order 4. The third library consists of sinusoidal functions of time with periods going from 0.2 to 4.7 in 0.5 increments. Other libraries were tried but those three were included in the report because they offered the best KL divergence. To approximate the derivative, a two-step leapfrog finite difference method was used. Total variation regularization [2] to denoise the derivative was tried but offered no improvement in accuracy, so it is not included in the report. Regression was performed using the QR-based backslash and *lasso()* in MATLAB. The model points were then obtained by simulating the identified system using *ode45()* in MATLAB.

C) Cross-Validation

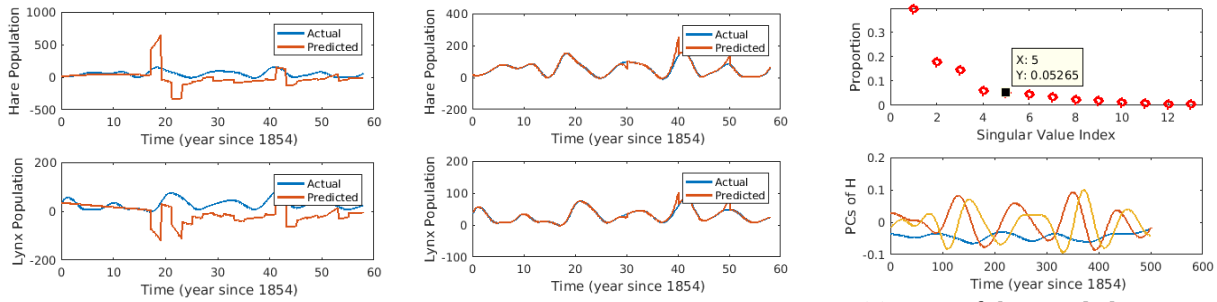
Due to the limited sample size, cross-validation is skipped in this work. Moreover, the entire dataset was used in sparse regression due to the data shortage.

IV. Computational Results

A) Dynamical Mode Decomposition for the Population Data

The Sliding window DMD approach is applied to the Hare and Lynx population and the prediction is compared against the actual data in *Figure 1*. *Figure 1a* shows the DMD result

without time-delay embedding and the zigzags comes from the windowing effect. The prediction also contains negative values, which are not physically possible. *Figure 1b* shows the prediction with DMD on time-delay embedded matrix. The prediction occasionally begins to diverge from the actual data at the end of a sliding window, due to modes with positive real eigenvalues. However, it achieves a much better match with the actual data compare to *Figure 1a*, as expected. This is because DMD assumes linear dynamics and a linear embedding of the dynamics can be obtained using time-delay embedding. The first three principal components of the Hankel matrix are plotted in the bottom row of *Figure 1c*, and these components appears similar to sinusoidal functions, which is a signature of linearity.



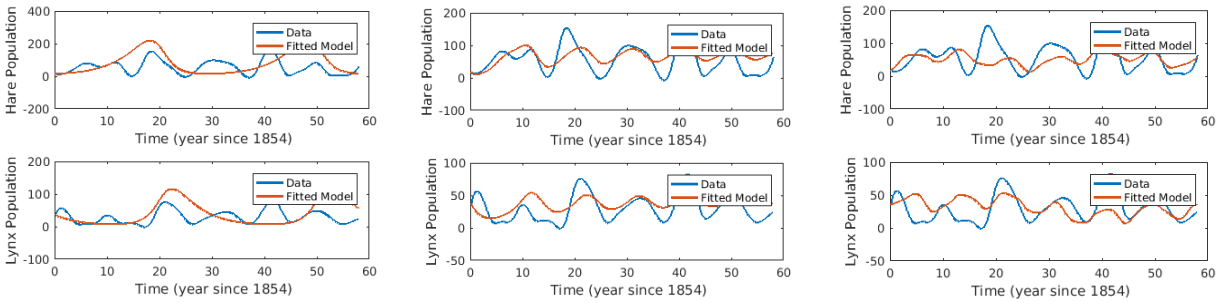
(a) DMD without Time-Delay Embedding *(b) DMD with Time-Delay Embedding* *(c) SVD of the Hankel Matrix for Time-Delay Embedding*

***Figure 1**: Sliding Window DMD for the Hare and Lynx Population*

The relative contribution of singular values of the Hankel matrix is plotted in the top row of *Figure 1c*, and there are five components that account for 95% of the total variance. Thus, there are likely more variables than the two observable ones that is driving the population data. The possible existence of latent variable limits the performance of fitting a dynamical system based only on two observable variables, as shown in the next Section. In fact, the dataset used in this study appears to be much noisier than the predator-prey dataset used in [1].

B) Inferring Dynamical Systems from Population Data

Three different libraries of functions, described in in *Section III*, are applied to approximate systems of ordinary differential equations from the population data. The fitted model is compared with the actual data in *Figure 2*. All three models can generate oscillations. However, Lotka-Volterra model oscillate much slower than the actual data, whereas models fitted using libraries of higher order polynomials and sinusoids match the period of the actual data better.



(a) Lotka-Volterra Model *(b) Polynomial Functions* *(c) Sinusoidal Functions*

***Figure 2**: Fitting a Dynamical System to the Population Data using Libraries of Functions*

The solutions in *Figure 2* were obtained using the QR-based backslash. Lasso has also been applied but not included in the result. This is because for every α used, the solution either becomes unstable or decays to a constant. This could that although some higher order polynomial, although with small coefficients, can have significant impact on the result. When those terms are removed and regression is performed again, the performance diminishes. This suggests that the least squares solutions are already parsimonious.

Interestingly, when the library contains a constant and both first order term of lynx and hare data, least-squares put a lot of weight on those lower order terms. Their weights remain high even when lasso was applied. When those terms were included, the solution were unstable. The solution becomes stable when the constant and one of the first order term were removed. All of these suggests that coefficient magnitude is not the sole indicator of term importance.

KL divergence, AIC and BIC scores were computed for each population of the models above and summarized in *Table I*. Bin size is chosen to be 10% of the data range, and the general trends do not seem to be sensitive to the bin size choice. Comparing the first two rows shows that DMD with time-delay achieves better scores than without for all three measures, since DMD with time-delay embedding captures the actual data distribution better, as observed in *Figure 1*. By comparing the last three rows, Lotka-Volterra achieves better KL divergence than the other two models. This is probably because Lotka-Volterra spans the entire data range as seen in *Figure 2a*, so its distribution should resemble that of the actual data. Although the other two methods match the data oscillations much better, it is not reflected in the measure, because those measures do not take timing into account. On the other hand, AIC and BIC scores penalize the variance of residuals, and the library of polynomials, which appears in *Figure 2* to have slightly lower residuals than the other two methods, achieve slightly better scores. In addition, DMD requires a larger number of parameters for the fit, which got penalized heavily by the AIC and BIC scores.

Model	KL (H)	KL (L)	AIC (H)	AIC (L)	BIC (H)	BIC (L)
DMD	1.7126	2.1134	8344	7112	10875	9500
Time-delay DMD	1.1823	0.2018	5634	4728	8165	7259
Lotka-Volterra	0.5959	0.2907	6594	5873	6612	5890
Polynomials	1.2006	2.8289	5809	5007	5923	5120
Sinusoids	1.7342	0.8699	5986	5305	6169	5489

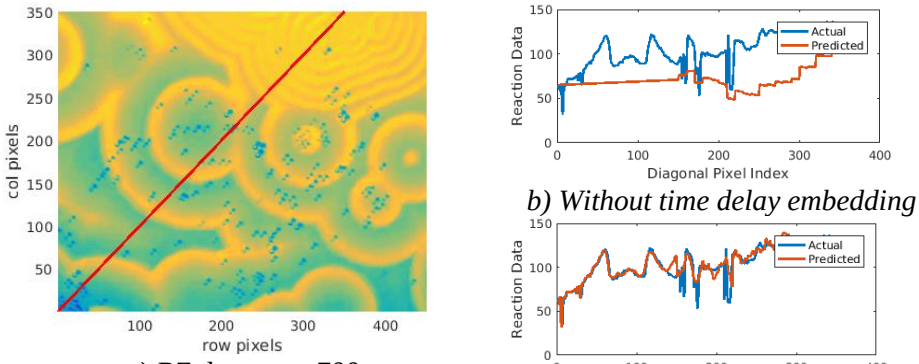
Table I: KL, AIC and BIC Scores for Different Models of Hare (H) and Lynx (L) Populations

C) Dynamical Mode Decomposition for the BZ Reaction Data

The BZ Reaction data consists of 1200 frames. For simplicity, only the frame at t=700 is selected for illustration in *Figure 3*. Similar trends were observed for other frames. To further simplify the analysis, only the data long the diagonal pixels (denoted by the red line in *Figure 3a*) were analyzed. Diagonal pixels were picked to catch the diagonal spread.

Comparing *Figure 3b* and *3c*, DMD performed on time-delay embedded matrix achieves a much better match between the actual and predicted data, as previously seen in *Figure 1*. In fact, the prediction with time-delay embedding was able to follow the general trend throughout the diagonal. Repeating the analysis in *Figure 1c* shows only one mode capturing 95% of the total variance, suggesting there are likely little effect from latent variables. KL divergence, AIC and

BIC scores were also computed, but they are not meaningful because there are no other models for comparison. Hence, they are not included in the report.



a) BZ data at t=700

b) Without time delay embedding

c) With time delay embedding

Figure 3: DMD for BZ Reaction Data. The red line in a) indicates the subset of data on which DMD was applied.

V. Summary and Conclusions

In this project, DMD and sparse regression were implemented to discover model from two different datasets. While the theory behind these methods were straightforward, careful implementation was needed in order to obtain satisfactory results.

First, DMD has limited time span for which it can predict accurately, so the sliding window trick was used such that only the immediate future was predicted. Second, DMD assumes linear dynamics, so time-delay embedding with enough points, which finds a linear basis, was used. With those two implementations, the prediction matches the general trend of the two datasets.

Moreover, limited performance was achieved by sparse regression due to the possible existence of latent variables. Terms with large coefficients may not imply importance and can sometimes cause instability, which makes promoting sparsity less straightforward to implement. Better results were attained by removing a few problematic terms from the library on a trial and error basis. The models were evaluated by KL divergence, which conveys how well data distribution was retained by the model but neglects timing information. AIC and BIC scores are better suited to quantify the pointwise errors between the model and data.

Overall, this project demonstrates the potential and limitation of sparse regression and DMD. This inspires future work to improve upon those techniques for identifying model from data.

VI. References:

[1] Brunton SL, Proctor JL, Kutz JN. 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl Acad. Sci*. 113, 3932–3937.

[2] Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variational based noise removal algorithms. *Physica D*, 60(1), 259-268.

Appendix A: MATLAB Functions Used

```
function KL1 = get_KL(edges, X, y_pred)
% The function calculates the KL divergence between
% X and y_pred, whose distributions are binned using the specfied edges
% The code is modified from page 174 of the textbook

f=hist(X(1,:),edges)+0.01; % generate PDFs
g=hist(y_pred,edges)+0.01;

f=f/trapz(edges,f); % normalize data
g=g/trapz(edges,g);

% compute integrand
Int1=f.*log(f./g);

% use if needed
Int1(isinf(Int1))=0; Int1(isnan(Int1))=0;

% KL divergence
KL1=trapz(edges,Int1);

end

function rhs=rhs_lv(t,x,dummy,b,p,r,d)
% An ODE function of the lotka-volterra model to be
% solved by ode45()

% Lotka-volterra model:
rhs=[(b-p*x(2))*x(1); (r*x(1)-d)*x(2)];

function rhs=rhs_poly(t,x,dummy,xi,yi,indx,indy)
% An ODE function of the 4th order polynomial library to be
% solved by ode45()

xs=x(1); ys=x(2);

% Build the library
A=[xs ys xs.^2 xs.*ys ys.^2 xs.^3 (ys.^2).*xs (xs.^2).*ys ys.^3 ...
    xs.^4 (xs.^3).*ys (xs.^2).*(ys.^2) (ys.^3).*xs ys.^4];
Ax=A; Ax(:,indx)=[]; Ax=Ax';
Ay=A; Ay(:,indy)=[]; Ay=Ay';

% The dynamical system x_dot=Ax
rhs=[xi'*Ax; yi'*Ay];

function rhs=rhs_sin(t,x,dummy,xi,yi,indx,indy)
% An ODE function of the sinusoidal library to be
% solved by ode45()

xs=x(1); ys=x(2);

% Build the library
A=[xs ys];
for om=0.2:0.5:5
    A=[A sin(om*t) cos(om*t)];
end
Ax=A; Ax(:,indx)=[]; Ax=Ax';
Ay=A; Ay(:,indy)=[]; Ay=Ay';

% The dynamical system x_dot=Ax
rhs=[xi'*Ax; yi'*Ay];
```

The following modules and documentations are provided in MATLAB R2016b by MathWorks.

```
function [aic,bic] = aicbic(logL,numParam,numObs)
%AICBIC Akaike and Bayesian information criteria
%
% Syntax:
%
%   [aic,bic] = aicbic(logL,numParam,numObs)
%
% Description:
%
%   Given optimized log-likelihood function values logL obtained by fitting
%   a model to data, compute the Akaike (AIC) and Bayesian (BIC)
%   information criteria. Since information criteria penalize models with
%   additional parameters, aic and bic select models based on both goodness
%   of fit and parsimony. When using either AIC or BIC, models that
%   minimize the criteria are preferred.
%
% Input Arguments:
%
%   logL - Vector of optimized log-likelihood objective function values
%     associated with parameter estimates of various models.
%
%   numParam - Number of estimated parameters associated with each value
%     in logL. numParam may be a scalar applied to all values in logL, or a
%     vector the same length as logL. All elements of numParam must be
%     positive integers.
%
% Optional Input Argument:
%
%   numObs - Sample sizes of the observed data associated with each
%     value of logL. numObs is required for computing BIC, but not AIC.
%     numObs may be a scalar applied to all values in logL, or a vector the
%     same length as logL. All elements numObs must be positive integers.
%
% Output Arguments:
%
%   aic - Vector of AIC statistics associated with each logL objective
%     function value. The AIC statistic is defined as:
%
%       aic = -2*logL + 2*numParam
%
%   bic - Vector of BIC statistics associated with each logL objective
%     function value. The BIC statistic is defined as:
%
%       bic = -2*logL + numParam*log(numObs)
%
% Reference:
%
%   [1] Box, G.E.P., Jenkins, G.M., Reinsel, G.C., "Time Series Analysis:
%       Forecasting and Control", 3rd edition, Prentice Hall, 1994.
%
% See also ARIMA, GARCH, VGXVARX, VGXINFER, VGXCOUNT.

% Copyright 1999-2010 The MathWorks, Inc.

% Ensure the optimized logL is a vector:
```

Appendix B – MATLAB CODE

The MATLAB code used in this assignment have been uploaded to GitHub: https://github.com/Helena-Yuhan-Liu/Inferring-Structure-of-Complex-Systems.

A) DMD for predator-prey data

```
%% Setup up data

% X is a 2 by T matrix
%   X(1,:) contains the Hare population over time
%   X(2,:) contains the Lynx population over time

clear all; close all; clc;
format compact;

% Construct data matrix X
PH = [20,20,52,83,64,68,83,12,36,150,110,60,7,...   % Hare data
    10,70,100,92, 70,10,11,137,137,18,22,52,83,18,10,9,65];
PL = [32,50,12,10,13,36,15,12,6,6,65,70,40,9,...    % Lynx data
    20,34,45,40,15, 15,60,80,26,18,37,50,35,12,12,25];
P = [PH; PL];

% Spline interpolation for smoother data and more data points
% The performance is better than without it
year = 0:2:58; % Year since 1854
dt=0.1; t=dt:dt:58;
X = spline(year,P,t);

% Visualize data
figure; plot(t,X(1,:), 'r'); hold on; plot(t,X(2,:),'b'); hold off;
xlabel('Year since 1854'); ylabel('population'); legend('Hare','Lynx');

%% DMD sliding window
% Note, I am using the sliding window approach where within each window,
% parts of the data is allocated for training and the rest is used for
% prediction. This gives me better result, perhaps due to the short
% prediction window.

% Need overlapping window, because without it the prediction is
% very discontinuous

wstep= round(2/dt); % small test window
ws=wstep+round(20/dt); % window size
wstart = 0:wstep:(round(58/dt)-ws); % window starting indices

r=2;    % number of modes, at most 2 b/c U is 2 by 2
u_dmd = zeros(r,round(58/dt));

for ii = 1:length(wstart)
    % data length used for training vs data length used for prediction
    if ii==length(wstart) % last window
        ws2 = round(58/dt)-wstart(ii);
        wtrain = ws - wstep;
    else
        ws2 = ws;
        wtrain = ws2 - wstep;
    end
    wtest = ws2-wtrain;
    X1 = X(:,(wstart(ii)+1):(wstart(ii)+wtrain));
    X2 = X(:,(wstart(ii)+2):(wstart(ii)+wtrain+1));
    [U2,Sigma2,V2] = svd(X1,'econ'); U=U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);

    % DMD code from the lecture
```

```
    Atilde = U'*X2*V/Sigma;
    [W,D] = eig(Atilde);
    Phi = X2*V/Sigma*W;

    mu = diag(D);
    omega = log(mu)/dt;
    omega(real(omega)>0.5)=0.5+...
        1i*imag(omega(real(omega)>0.5)); % clip e-val with large real part
%     disp('The eigen values of the modes are:');
%     disp(omega);

    u0=X1(:,1);     % Now the new y0 has larger dimension
    y0 = Phi\u0;  % pseudo-inverse initial conditions
    u_modes = zeros(r,wtrain);
    for iter = 1:wtrain
         u_modes(:,iter) =(y0.*exp(omega*t(iter)));
    end
    u_dmd_train = Phi*u_modes;

    % forecast
    u_modes2 = zeros(r,wtest);
    for iter = 1:wtest
         u_modes2(:,iter) =(y0.*exp(omega*t(iter+wtrain)));
    end
    u_dmd_test = Phi*u_modes2;

    % append the window result to the final result
    if ii ==1 % first window
        u_dmd(:,(wstart(ii)+1):(wstart(ii)+ws2)) = [u_dmd_train u_dmd_test];
    else
        u_dmd(:,(wstart(ii)+wtrain+1):(wstart(ii)+ws2)) = u_dmd_test;
    end
end

% Comparing the actual data with fitted & predicted modes
figure;
subplot(2,1,1), plot(t,X(1,:),t,u_dmd(1,:),'Linewidth',[2]);
legend('Actual', 'Predicted');
xlabel(' Time (year since 1854)'); ylabel('Hare Population');
subplot(2,1,2), plot(t,X(2,:),t,u_dmd(2,:),'Linewidth',[2]);
legend('Actual', 'Predicted');
xlabel(' Time (year since 1854)'); ylabel('Lynx Population');
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');

%% Repeat for Time delay embedding
% first, specify number of delay points, which are set so that modes look
% somewhat sinusoidal,

dly=8; % number of years to be delayed
wstep= round((dly+2)/dt); % test window slightly higher than dly
ws=wstep+round(20/dt); % window size

% Construct the Hankel matrix
H=[];
for j=1:round(dly/dt)
  H=[H; X(:,j:round((58-dly)/dt)+j)];
end

wstart = 0:wstep:(round((58-dly)/dt)-ws+wstep);
r=2*round(dly/dt);    % two variables per delayed points
u_dmd_td = zeros(r,round(58/dt));

for ii = 1:length(wstart)
    % data length used for training vs data length used for prediction
    if ii==length(wstart) % last window
```

```
        ws2 = round(58/dt)-wstart(ii);
        wtrain = ws - wstep;
        wtest = ws2 - wtrain;   % test on wstep + dly points
    else
        ws2 = ws;
        wtrain = ws2 - wstep;
        wtest = wstep;
    end
    X1 = H(:,(wstart(ii)+1):(wstart(ii)+wtrain));
    X2 = H(:,(wstart(ii)+2):(wstart(ii)+wtrain+1));
    [U2,Sigma2,V2] = svd(X1,'econ'); U=U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);

    % DMD code from the lecture
    Atilde = U'*X2*V/Sigma;
    [W,D] = eig(Atilde);
    Phi = X2*V/Sigma*W;

    mu = diag(D);
    omega = log(mu)/dt;
    omega(real(omega)>0.5)=0.5+...
        1i*imag(omega(real(omega)>0.5)); % clip e-val with large real part
%     disp('The eigen values of the modes are:');
%     disp(omega);

    u0=X1(:,1);     % Now the new y0 has larger dimension
    y0 = Phi\u0;  % pseudo-inverse initial conditions
    u_modes = zeros(r,wtrain);
    for iter = 1:wtrain
         u_modes(:,iter) =(y0.*exp(omega*t(iter)));
    end
    u_dmd_train = Phi*u_modes;

    % forecast
    u_modes2 = zeros(r,wtest);
    for iter = 1:wtest
         u_modes2(:,iter) =(y0.*exp(omega*t(iter+wtrain)));
    end
    u_dmd_test = Phi*u_modes2;

    if ii ==1 % first window
        u_dmd_td(:,(wstart(ii)+1):(wstart(ii)+ws2)) = [u_dmd_train u_dmd_test];
    else
        u_dmd_td(:,(wstart(ii)+wtrain+1):(wstart(ii)+ws2)) = u_dmd_test;
    end
end

% Compare the actual data with fitted and predicted modes
figure;
subplot(2,1,1), plot(t,X(1,:),t,u_dmd_td(1,:),'Linewidth',[2]);
legend('Actual', 'Predicted');
xlabel(' Time (year since 1854)'); ylabel('Hare Population');
subplot(2,1,2), plot(t,X(2,:),t,u_dmd_td(2,:),'Linewidth',[2]);
legend('Actual', 'Predicted');
xlabel(' Time (year since 1854)'); ylabel('Lynx Population');
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');

% SVD on matrix H to see dimensionality
[u,s,v]=svd(H,'econ');
figure; subplot(2,1,1), plot(diag(s)/(sum(diag(s))),'ro','Linewidth',[3])
xlabel('Singular Value Index'); ylabel('Proportion');
subplot(2,1,2), plot(v(:,1:3),'Linewidth',[2]);
xlabel(' Time (year since 1854)'); ylabel('PCs of H');
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');
% Turns out that five modes have relative energy above 0.05
% this implies there may be hidden factors influencing the
```

```
% population dynamics, making low dim model hard to attain

% Bottom line: with enough time delayed points (so the modes look
% somewhat linear), the performance is much better than without
% time delay embedding

%% KL, AIC BIC
% For hare, KL does not capture the striking improvement from TD,
% probably b/c KL has the huge caveat of not taken timing into account
% For lynx, KL does, b/c lots of data outside of the bin

% First, for the Hare population
edges=0:15:150;     % bin width 10% of the data range
KL_H = get_KL(edges, X(1,:), real(u_dmd(:,1)));
KL_H_td = get_KL(edges, X(1,:), real(u_dmd_td(:,1)));

% Then Lynx
edges=0:8:80;
KL_L = get_KL(edges, X(2,:), real(u_dmd(:,2)));
KL_L_td = get_KL(edges, X(2,:), real(u_dmd_td(:,2)));

% AIC BIC for without time-delay embedding
n=length(t);

% First, compute AIC BIC for the Hare population
logL=n/2*(-log(2*pi)-log(var(X(1,:)-real(u_dmd(1,:))))-1);
[aicH, bicH] = aicbic(logL, length(t), length(t));

% Then, compute AIC BIC for the Lynx population
logL=n/2*(-log(2*pi)-log(var(X(2,:)-real(u_dmd(2,:))))-1);
[aicL, bicL] = aicbic(logL, length(t), length(t));

% AIC BIC for time delayed embedding

% First, compute AIC BIC for the Hare population
logL=n/2*(-log(2*pi)-log(var(X(1,:)-real(u_dmd_td(1,:))))-1);
[aicHtd, bicHtd] = aicbic(logL, length(t), length(t));

% Then, compute AIC BIC for the Lynx population
logL=n/2*(-log(2*pi)-log(var(X(2,:)-real(u_dmd_td(2,:))))-1);
[aicLtd, bicLtd] = aicbic(logL, length(t), length(t));
```

B) System identification with regression for predator-prey data

```
%% Discovering model from data
% Instead of using DMD, this file uses approximates dynamical system
% using a library of nonlinear functions and then applying least square
% to find x_dot=Ac, where c consists of the coefficients in front of the
% nonlinear functions.

clc; clear all; close all;

% Construct data matrix X
PH = [20,20,52,83,64,68,83,12,36,150,110,60,7,...   % Hare data
    10,70,100,92, 70,10,11,137,137,18,22,52,83,18,10,9,65];
PL = [32,50,12,10,13,36,15,12,6,6,65,70,40,9,...    % Lynx data
    20,34,45,40,15, 15,60,80,26,18,37,50,35,12,12,25];
P = [PH; PL];

% Spline interpolation for smoother data and more data points
% The performance is better than without it
year = 0:2:58; % Year since 1854
dt=0.1; t=dt:dt:58;
X = spline(year,P,t);
```

```
%% Part 3: Fitting Lotka-Volterra Coefficients
% Tried cross-validation: divided data and averaged, but worsened
% the performance b/c the data size is so small

% Try finite difference method to get the derivative
n=length(t);
for j=2:n-1
  x1dot(j-1)=(X(1,j+1)-X(1,j-1))/(2*dt);
  x2dot(j-1)=(X(2,j+1)-X(2,j-1))/(2*dt);
end
x1dot=x1dot.'; x2dot=x2dot.';
% % Total variational derivative didn't improve substantially
% x1dot = TVRegDiff(X(1,:), 50, 0.85); x1dot=x1dot(2:end-2);
% x2dot = TVRegDiff(X(2,:), 50, 0.85); x2dot=x2dot(2:end-2);

% Construct and solve for Ax=b
xs=X(1,2:n-1)'; ys=X(2,2:n-1)';
Ax=[xs -xs.*ys]; Ay=[xs.*ys -ys];
xi=Ax\x1dot;
yi=Ay\x2dot;
b=xi(1); p=xi(2); r=yi(1); d=yi(2);

% Compare the data with the Lotka-Volterra model
[t,y_lv]=ode45('rhs_lv',t,X(:,1),[],b,p,r,d);
figure;
subplot(2,1,1), plot(t,X(1,:),t,y_lv(:,1),'Linewidth',[2]);
legend('Data', 'Fitted Model');
xlabel(' Time (year since 1854)'); ylabel('Hare Population');
subplot(2,1,2), plot(t,X(2,:),t,y_lv(:,2),'Linewidth',[2]);
legend('Data', 'Fitted Model');
xlabel(' Time (year since 1854)'); ylabel('Lynx Population');
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');

%% Part 4: Data Fit Using a Richer Library

% Construct a library of polynomials and solve Ax=b
% Some polynomial terms were removed b/c they cause instability
A1=[xs ys xs.^2 xs.*ys ys.^2 xs.^3 (ys.^2).*xs (xs.^2).*ys ys.^3 ...
    xs.^4 (xs.^3).*ys (xs.^2).*(ys.^2) (ys.^3).*xs ys.^4];
indx=2; indy=1;  % terms to remove
A1x=A1; A1x(:,indx)=[];
A1y=A1; A1y(:,indy)=[];
xi1=A1x\x1dot;
yi1=A1y\x2dot;

% Visualize the Sparsity, already sparse
figure; subplot(2,1,1), bar(xi1)
subplot(2,1,2), bar(yi1)

% % Lasso for sparsity, removes the small high order coeff
% % hurts performance a lot!
% % Maybe because high order functions with small coefficents do matter
% xi1=lasso(A1x,x1dot,'Lambda',0.02,'Alpha', 0.5);
% yi1=lasso(A1y,x2dot,'Lambda',0.02,'Alpha', 0.5);
% figure; subplot(2,1,1), bar(xi1)
% subplot(2,1,2), bar(yi1)

% Compare the data with the polynomial nolinear model
[t,y_poly]=ode45('rhs_poly',t,X(:,1),[],xi1,yi1,indx,indy);
figure;
subplot(2,1,1), plot(t,X(1,:),t,y_poly(:,1),'Linewidth',[2]);
legend('Data', 'Fitted Model');
xlabel(' Time (year since 1854)'); ylabel('Hare Population');
subplot(2,1,2), plot(t,X(2,:),t,y_poly(:,2),'Linewidth',[2]);
legend('Data', 'Fitted Model');
```

```
xlabel(' Time (year since 1854)'); ylabel('Lynx Population');
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');

% Construct a library of sinusoids
% Note, sinusoid is not necessary for getting limit cycles
% Use sin(t) instead of sin(X), which is hard to control
A2=[xs ys];
for om=0.2:0.5:5
    A2=[A2 sin(om*t(2:n-1)) cos(om*t(2:n-1))];
end
indx=2; indy=1;  % terms to remove
A2x=A2; A2x(:,indx)=[];
A2y=A2; A2y(:,indy)=[];

xi2=A2x\x1dot;
yi2=A2y\x2dot;

% Visualize the Sparsity
figure; subplot(2,1,1), bar(xi2)
subplot(2,1,2), bar(yi2);

% Compare the data with the sinusoidal nolinear model
[t,y_sin]=ode45('rhs_sin',t,X(:,1),[],xi2,yi2,indx,indy);
figure;
subplot(2,1,1), plot(t,X(1,:),t,y_sin(:,1),'Linewidth',[2]);
legend('Data', 'Fitted Model');
xlabel(' Time (year since 1854)'); ylabel('Hare Population');
subplot(2,1,2), plot(t,X(2,:),t,y_sin(:,2),'Linewidth',[2]);
legend('Data', 'Fitted Model');
xlabel(' Time (year since 1854)'); ylabel('Lynx Population');
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');

% Some notes
% - Remove the "problem" terms really helped

%% KL Divergence
% Important caveat: no time matching!
% But gives a general sense if the range and distribution of data
% is kept in the model

% First, compute the KL for the Hare Population:
edges=0:15:150;
KL_H=zeros(3,1);
KL_H(1) = get_KL(edges, X(1,:), y_lv(:,1));
KL_H(2) = get_KL(edges, X(1,:), y_poly(:,1));
KL_H(3) = get_KL(edges, X(1,:), y_sin(:,1));

% Repeat for the Lynx Population:
edges=0:8:80;
KL_L=zeros(3,1);
KL_L(1) = get_KL(edges, X(2,:), y_lv(:,2));
KL_L(2) = get_KL(edges, X(2,:), y_poly(:,2));
KL_L(3) = get_KL(edges, X(2,:), y_sin(:,2));

%% AIC BIC

% First, compute AIC BIC for the Hare population

logL = zeros(3,1); % Preallocate loglikelihood vector
logL(1)=n/2*(-log(2*pi)-log(var(X(1,:)'-y_lv(:,1)))-1);
logL(2)=n/2*(-log(2*pi)-log(var(X(1,:)'-y_poly(:,1)))-1);
logL(3)=n/2*(-log(2*pi)-log(var(X(1,:)'-y_sin(:,1)))-1);
numParam = [4; nnz(xi1)+nnz(yi1); nnz(xi2)+nnz(yi2)]; % Number of param
[aicH, bicH] = aicbic(logL, numParam, n*ones(3,1));
```

```
% Repeat for the Hare population

logL = zeros(3,1); % Preallocate loglikelihood vector
logL(1)=n/2*(-log(2*pi)-log(var(X(2,:)'-y_lv(:,2)))-1);
logL(2)=n/2*(-log(2*pi)-log(var(X(2,:)'-y_poly(:,2)))-1);
logL(3)=n/2*(-log(2*pi)-log(var(X(2,:)'-y_sin(:,2)))-1);
numParam = [4; nnz(xi1)+nnz(yi1); nnz(xi2)+nnz(yi2)];
[aicL, bicL] = aicbic(logL, numParam, n*ones(3,1));
```

C) DMD for BZ reaction data

```
%% DMD for BZ Reaction data
% Follows similar DMD analysis procedure as for the predator-prey data

clc; clear all; close all;
load('BZ.mat');

[m,n,k]=size(BZ_tensor); % x vs y vs time data
% for j=1:k         % visualize data
%     A=BZ_tensor(:,:,j);
%     pcolor(A), shading interp, pause(0.2)
% end

%% Look at a sample frame

% For simplicity, only sample snapshots are analyzed

% Moreover, only diagonal pixels are analyzed. This is not
% only for simplcity, but the video suggests that there is a source
% at the top right corner that's diffusing diagonally

% Example frame at 700
A=BZ_tensor(:,:,700); X=diag(A)'; p=1:length(X); l=length(X);
figure; pcolor(A); shading interp; hold on;
plot(p,p,'r','Linewidth',[2]); hold off;
xlabel('row pixels'); ylabel('col pixels');
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');
% subplot(1,2,2); plot(p,X);

% Again, repeat the sliding window stuff
% Settings to show in the report
wstep=10; ws=wstep+80;
wstart = 0:wstep:(l-ws); % window starting indices

r=1;    % number of modes, at most 1 b/c U is 1 by 1
u_dmd = zeros(r,l);

for ii = 1:length(wstart)
    % data length used for training vs data length used for prediction
    if ii==length(wstart) % last window
        ws2 = l-wstart(ii);
        wtrain = ws - wstep;
    else
        ws2 = ws;
        wtrain = ws2 - wstep;
    end
    wtest = ws2-wtrain;     % save some years for prediction
    X1 = X((wstart(ii)+1):(wstart(ii)+wtrain));
    X2 = X((wstart(ii)+2):(wstart(ii)+wtrain+1));
    [U2,Sigma2,V2] = svd(X1,'econ'); U=U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);

    % DMD code from the lecture
    Atilde = U'*X2*V/Sigma;
    [W,D] = eig(Atilde);
    Phi = X2*V/Sigma*W;
```

```
    mu = diag(D);
    omega = log(mu);
%     omega(real(omega)>0.5)=0.5+...
%         1i*imag(omega(real(omega)>0.5)); % clip e-val with large real part
% %     disp('The eigen values of the modes are:');
% %     disp(omega);

    u0=X1(1);
    y0 = Phi\u0;  % pseudo-inverse initial conditions
    u_modes = zeros(r,wtrain);
    for iter = 1:wtrain
        u_modes(:,iter) =(y0.*exp(omega*p(iter)));
    end
    u_dmd_train = Phi*u_modes;

    % forecast
    u_modes2 = zeros(r,wtest);
    for iter = 1:wtest
        u_modes2(:,iter) =(y0.*exp(omega*p(iter+wtrain)));
    end
    u_dmd_test = Phi*u_modes2;

    % append the window result to the final result
    if ii ==1 % first window
        u_dmd(:,(wstart(ii)+1):(wstart(ii)+ws2)) = [u_dmd_train u_dmd_test];
    else
        u_dmd(:,(wstart(ii)+wtrain+1):(wstart(ii)+ws2)) = u_dmd_test;
    end
end

% Comparing the actual data with fitted & predicted modes
figure; plot(p,X,p,u_dmd,'Linewidth',[2]);
legend('Actual', 'Predicted');
xlabel('Diagonal Pixel Index'); ylabel('Reaction Data');
set(gcf, 'position', [100 100 450 200]); set(gcf,'color','w');

%% Time delay embedding

% Set using similar approach as for the predator-prey data
wstep=70; dly=60; ws=wstep+80;

% Construct the Hankel matrix
H=[];
for j=1:dly
  H=[H; X(:,j:(l-dly)+j)];
end

% Starting point for each sliding window
wstart = 0:wstep:(l-dly-ws+wstep);
r=1*dly;    % one variable per delayed points
u_dmd_td = zeros(r,l);

for ii = 1:length(wstart)
    % data length used for training vs data length used for prediction
    if ii==length(wstart) % last window
        ws2 = l-wstart(ii);
        wtrain = ws - wstep;
        wtest = ws2 - wtrain;   % test on wstep + dly points
    else
        ws2 = ws;
        wtrain = ws2 - wstep;
        wtest = wstep;
    end
    X1 = H(:,(wstart(ii)+1):(wstart(ii)+wtrain));
```

```
    X2 = H(:,(wstart(ii)+2):(wstart(ii)+wtrain+1));
    [U2,Sigma2,V2] = svd(X1,'econ'); U=U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);

    % DMD code from the lecture
    Atilde = U'*X2*V/Sigma;
    [W,D] = eig(Atilde);
    Phi = X2*V/Sigma*W;

    mu = diag(D);
    omega = log(mu);
    omega(real(omega)>0.01)=0.01+...
        1i*imag(omega(real(omega)>0.01)); % clip e-val with large real part
%     disp('The eigen values of the modes are:');
%     disp(omega);

    u0=X1(:,1);     % Now the new y0 has larger dimension
    y0 = Phi\u0;  % pseudo-inverse initial conditions
    u_modes = zeros(r,wtrain);
    for iter = 1:wtrain
         u_modes(:,iter) =(y0.*exp(omega*p(iter)));
    end
    u_dmd_train = Phi*u_modes;

    % forecast
    u_modes2 = zeros(r,wtest);
    for iter = 1:wtest
         u_modes2(:,iter) =(y0.*exp(omega*p(iter+wtrain)));
    end
    u_dmd_test = Phi*u_modes2;

    if ii ==1 % first window
        u_dmd_td(:,(wstart(ii)+1):(wstart(ii)+ws2)) = [u_dmd_train u_dmd_test];
    else
        u_dmd_td(:,(wstart(ii)+wtrain+1):(wstart(ii)+ws2)) = u_dmd_test;
    end
end

% Compare the actual data with fitted and predicted modes
figure; plot(p,X,p,u_dmd_td(1,:),'Linewidth',[2]);
legend('Actual', 'Predicted');
xlabel('Diagonal Pixel Index'); ylabel('Reaction Data');
set(gcf, 'position', [100 100 450 200]); set(gcf,'color','w');

% SVD on matrix H to see dimensionality
[u,s,v]=svd(H,'econ');
figure; subplot(2,1,1), plot(diag(s)/(sum(diag(s))),'ro','Linewidth',[3])
subplot(2,1,2), plot(v(:,1:3),'Linewidth',[2])
% only one sing val with relative energy above 0.05,
% latent variables probably have very little effect
```