Model Discovery for Nonlinear Ordinary Differential Equations and Partial Differential Equations

Ashley Batchelor

Abstract

For the first part, I used linear fitting to a library of functions to derive a system of two first order differential equations that characterize snowshoe hare and lynx population data over a period of 30 years. I compared various models of linear fittings using Kullback–Leibler (KL) Divergence, the Akaike Information Criterion (AIC), and the Bayesian Information Criterion (BIC). For the second part, I used linear fitting to a library of functions to derive a partial differential equation that characterizes video of a Belousov-Zhabotinsky Chemical oscillator. I compared various models using KL Divergence.

Sec. I. Introduction and Overview

The primary food source for the Canadian lynx is the snowshoe hare. In 1921, Charles Gordon Hewitt tracked the relative populations of snowshoe hares and lynxes based on the number of pelts taken each year by the Hudson Bay Company[1]. An early model for predator and prey population relationships is characterized by the Lotka-Volterra (LV) equations for the respective hare and lynx populations x_1 and x_2, which include four parameters α, β, γ, and δ: [2]

$$\frac{dx_1}{dt} = \alpha x_1 - \beta x_1 x_2 \,, \frac{dx_2}{dt} = -\gamma x_2 + \delta x_1 x_2 \qquad (1,2)$$

I used model discovery for ordinary differential equations (ODE) to determine similar models to the LV equations.

In a mixture of potassium bromate, cerium(IV) sulfate, malonic acid, and citric acid in dilute sulfuric acid, the ratio of concentration of the cerium(IV) and cerium(III) ions will oscillate, causing an oscillation of the color of the solution. This process is known as a Belousov–Zhabotinsky (BZ) Chemical Oscillator. I used a video of this solution for model discovery for partial differential equations (PDE) in space and time.

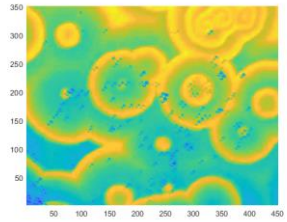

Figure 1 - Video frame of BZ Chemical Oscillator

Sec. II. Theoretical Background

[1] Hewitt, C. G. (1921) The Conservation of the Wild Life of Canada. Charles Scribner's Sons.
[2] Volterra, V. (1926). Fluctuations in the Abundance of a Species Considered Mathematically. Nature, 118(2972), 558-560.
Lotka, A. J. (1925). Principles of physical Biology. Baltimore: Waverly.

A system of ODEs (e.g. varying in time t) for a set of n observables $\boldsymbol{X} = \mathrm{x}_n$, may be modeled as a product of a library of functions and a set of weightings.

$$\frac{d\mathbf{x}_n}{dt} = \Theta\xi = B \tag{3}$$

The library Θ may include a variety of functions of each x_n where each column is the data points of the respective function. The rows of the weightings ξ correspond to the weightings of each function of x_n that are equal to $\dot{\mathrm{x}}_n$ For example:

$$\Theta = \begin{pmatrix} | & | & | & | & | & | \\ x_1 & x_2 & x_1^2 & x_2^2 & x_1x_2 & \dots \\ | & | & | & | & | & | \end{pmatrix}, \xi = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \dots \end{pmatrix}, \mathrm{B}=\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dots \end{pmatrix} \tag{4,5,6}$$

This provides an over-determined linear system. With a set of measurements for x_n, the weightings ξ may be determined by a linear fit using such methods as QR Decomposition, Moore Pseudo-Inverse, Lasso, or others

Two models $f(\boldsymbol{X}, \beta)$ and $g(\boldsymbol{X}, \mu)$ (expressed as probabilities) may be compared to one another using Kullback–Leibler Divergence:

$$KL = \int f(\boldsymbol{X}, \beta)\, log\, \frac{f(\boldsymbol{X},\beta)}{g(\boldsymbol{X},\mu)}\, d\boldsymbol{X} \tag{7}$$

A fit $f(\boldsymbol{X}, \beta)$ may be compared to data $g(\boldsymbol{X}, \mu)$ to determine a relative strength of that fit. Two fits for $f(\boldsymbol{X}, \beta)$ may compared and the fit with a lower KL divergence may be understood as a better fit.

Another metric for the relative strength of a model is the Akaike Information Criterion (AIC) which is based on the maximum value of a likelihood function $\mathcal{L}$.

$$AIC = K - 2\ln[\mathcal{L}(\hat{\boldsymbol{\mu}}|x)] \tag{8}$$

where K is the number of parameters in the model, $\hat{\boldsymbol{\mu}}$ is an estimate of the best parameters used (e.g. with the lowest KL divergence), and x is the data.[3] Assuming a Gaussian variance between the data and the true values, the log likelihood function $\ln[\mathcal{L}(\hat{\boldsymbol{\mu}}|x)]$ for n samples of x with a variance $\sigma^2 = \frac{1}{n}\|f_i - g_i\|_2^2$ may be expressed as[4]:

$$\ln[\mathcal{L}(\hat{\boldsymbol{\mu}}|x)] = -\frac{n}{2}(\ln 2\pi + \ln\sigma^2 + 1) \tag{9}$$

Additionally, one may use a Bayesian Information Criterion (BIC), which is similar to the AIC.

$$BIC = \ln(n)\, K - 2\ln[\mathcal{L}(\hat{\boldsymbol{\mu}}|x)] \tag{10}$$

where n is the number of data points.

Time delay embedding provides a means to check how the variance of a system evolving in time is spread across multiple degrees of freedom, i.e. multiple variables. The first step is to construct a Hankel matrix which contains columns constructed from subsets of time in the system, each incremented forward by one time step Δt. The next step is to take a singular value decomposition (SVD) of the

[3] Class notes

[4] http://mathworld.wolfram.com/MaximumLikelihood.html

Hankel matrix (described in Kutz[5]) and look at the diagonal entries of the $\mathbf{\Sigma}$ matrix. Normalizing these diagonal values gives a representation of the fraction of the total energy of the system given by a respective variable (projected onto a new common basis).

A system of PDEs (e.g. varying in time t) for a set of observables $U = \mathrm{x}_n$, may be modeled as a product of a library of functions and a set of weightings.

$$\frac{\partial \boldsymbol{U}}{\partial t} = \Theta\xi = B \tag{11}$$

In this case, the library Θ may include spatial partial derivatives, e.g. Ux, Uy, Uxx, Uxy, Uyy, etc. for a function $U(x, y, t)$.

Sec. III. Algorithm Implementation and Development

For the Hare and Lynx data, for each of the measurements of x_n indexed by i, I computed finite difference derivatives using central difference conventions:

$$\dot{x}_{n,i} = \frac{x_{n,i+1} - x_{n,i-1}}{2(t_i - t_{i-1})} \tag{12}$$

I used a library of functions:

$$\Theta = \begin{pmatrix} | & | & | & | & | & | & | & | & | & | & | & | & | & | \\ x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 & x_1^2x_2 & x_2^2x_1 & x_2^3 & x_1^4 & x_2^4 & x_1^5 & x_2^5 & 1 \\ | & | & | & | & | & | & | & | & | & | & | & | & | & | \end{pmatrix} \tag{13}$$

I applied cubic spline interpolation with a step size of 0.1 years to the data x_1 and x_2 and then determined a fit to the library of functions of x_1 and x_2 using equation 3 with QR Regression.

For the KL Divergence, I assigned four bins to the phase space of hare and lynx population values x_1 and x_2 defined by quadrants above and below the mean values of the actual data for hare and lynx populations. For the predicted and actual data f and g, I made a histogram of those four bins to define probability distribution functions (PDF), and computed the KL Divergence using equation 7.

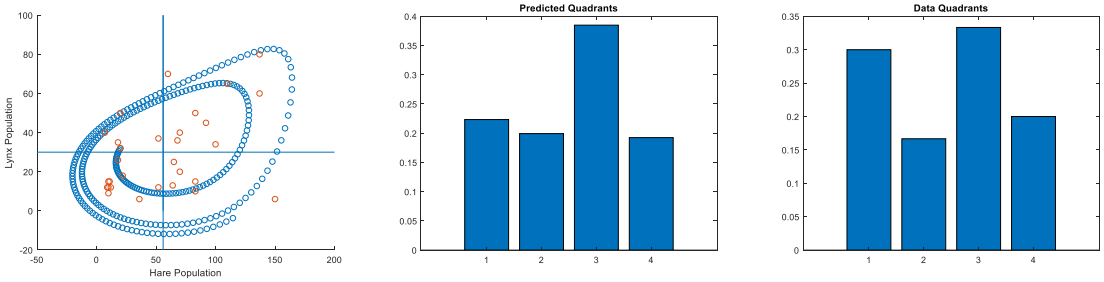

Figure 2 - Phase space diagram of quantic predicted model (blue) and actual data (red) divided into four quadrants.

[5] Kutz, J. Nathan. Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data, Oxford University Press, 2018

In order to check for latent variables, I generated a Hankel matrix with 8 columns for both x_1 and x_2 stepped in increments of 0.5 years and spanning 22 years, performed SVD and plotted the normalized diagonals of the Σ matrix.

For the BZ data, for each of the measurements $x_{m,n}$ corresponding to pixels (m,n) indexed by k in time, I computed finite difference derivatives using central difference conventions:

$$\dot{x}_{m,n,k} = \frac{x_{m,n,k+1} - x_{m,n,k-1}}{2(t_k - t_{k-1})} \tag{14}$$

Because the BZ_tensor data was so big (351x451x1200), for less time consuming computation, I used a smaller 50x50 pixel region of interest for fitting data.

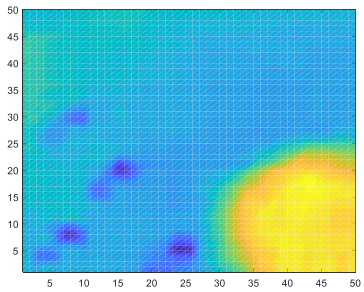

Figure 3 First frame of the 50x50 pixel Region of Interest

I generated a library of functions using finite difference formulas with central difference conventions, where a partial derivative is indicated below by a subscript, e.g. U_x.

$$\Theta = \begin{pmatrix} | & | & | & | & | & | & | & | & | & | & | & | & | & | & | \\ U & U^2 & UU^3 & U_x & U_{xx} & U_xU & U_x^2 & U_xU_{xx} & U_y & U_{yy} & U_yU & U_y^2 & U_yU_{yy} & U_{xy} & U_{xy}U \\ | & | & | & | & | & | & | & | & | & | & | & | & | & | & | \end{pmatrix} \tag{15}$$

I then determined a fit to the library of functions of U using equation 3 with QR Regression.

Sec. IV. Computational Results

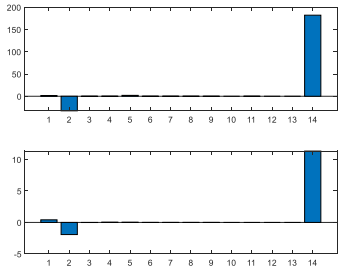

Figure 4 - Weightings of each parameter of Θ for Hare and Lynx data.

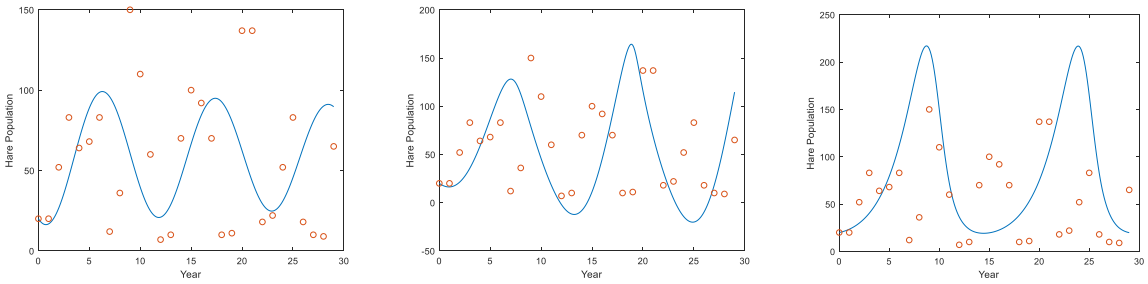

Figure 5 – Hare population fit trajectories (blue) with data (red) for first order, quintic, and Lotka-Volterra models.

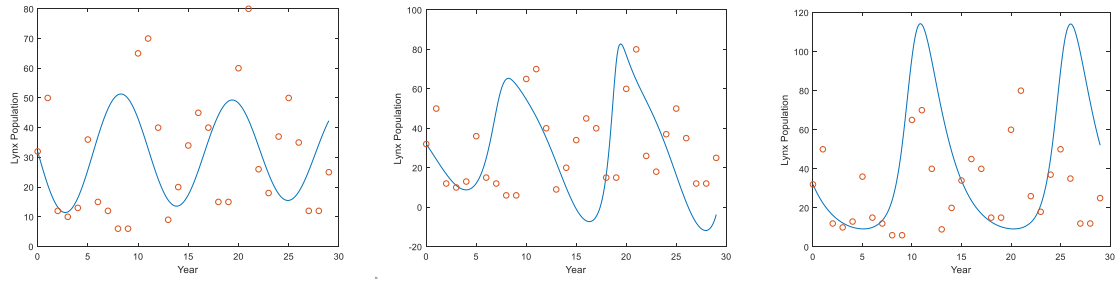

Figure 6 – Lynx population fit trajectories (blue) with data (red) for first order, quintic, and Lotka-Volterra models.

Model	KL	AIC (Lynx)	BIC (Lynx)	AIC (Hare)	BIC (Hare)	Parameters used
First Order	0.0022	2666	2668	3096	3103	1,14; 1,14
Quintic	0.0177	2795	2803	3243	3250	1,2,12,13,14;1,2,12,13,14
Lotka-Volterra	0.0577	2954	2961	3346	3353	1,4; 2,4

Figure 7 – Table of metrics for each fit, parameters used correspond to columns in equation 11 for Θ used for sparse fit for Lynx and Hare data.

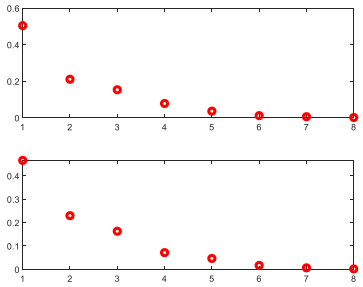

Figure 8 – Diagonals corresponding to time embedding of x_1 and x_2

The Σ matrix for each variable shows about 90% of the energy in the first three modes, but the fourth mode is still fairly large. Perhaps there are latent variables as my model tended to show two or three modes dominating, especially in the constant term which will not give rise to oscillations.

BZ Data

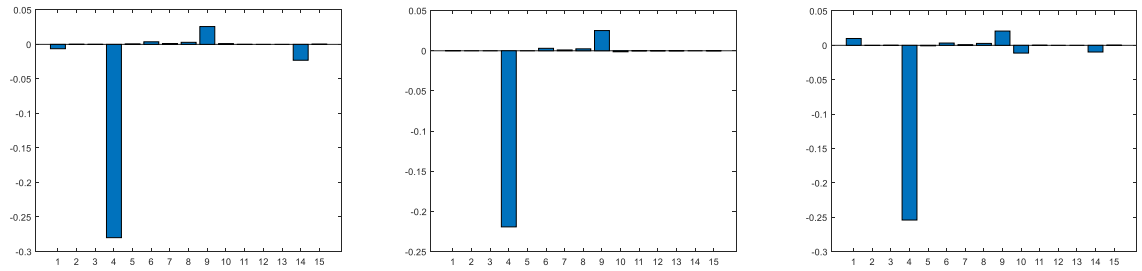

Figure 9 - Coefficients for ξ for QR Regression, Lasso (λ=0.002), and robustfit. For equation 11, I did regression using coefficients (4,9,14), (4,9), and (1,4,9,10,14) for the respective models..

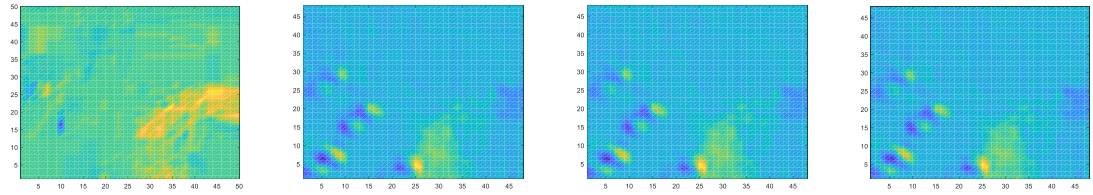

Figure 10 – Plot of Ut and fits for QR Regression, Lasso (λ =0.002), and robustfit.

Sec. V. Summary and Conclusions

For the hare and lynx data, I chose three models that provided positive values throughout the 30 years, and some sort of oscillatory behavior. However, none of the fits showed the same oscillatory behavior as the actual data. Many models that I tried gave results that diverged very quickly when fitting the trajectories with an ODE solver. The best fit was not the LV model, although it did provide an oscillating trajectory that did not diverge. With this set of data, the δ term of the LV model always fit to zero. Generally, the first order and the constant terms provided the best results.

For each model of the BZ reaction, the first spatial derivatives dominated. Lasso promoted the most sparsity. Robustfit gave stronger response for second spatial derivatives.

Appendix A MATLAB functions used and brief implementation explanations

A\B – This applies a QR decomposition fit.

lasso() – This applies a lasso fit, with specified parameters.

ode45() – This solves a system of ordinary differential equations.

pinv() – This applies a Moore-Penrose pseudoinverse fit, with specified parameters.

robustfit() - This applies a least squares fit.

spline() – This applies cubic spline interpolation to the data.

svd() – This applies singular value decomposition to a matrix.

Appendix B MATLAB codes

```
clear all; close all; clc;

[~,~,data] = xlsread('HareLynx.csv');
hare = cell2mat(data(:,2));
lynx = cell2mat(data(:,3));

ti=(0:1:29)';
dt=0.1;
t=(0:dt:29)';



x1=hare;
x2=lynx;
x1=spline(ti,hare,t);
x2=spline(ti,lynx,t);

x0=[hare(1) lynx(1)];



n=length(t);
for j=2:n-1
  x1dot(j-1)=(x1(j+1)-x1(j-1))/(2*dt);
  x2dot(j-1)=(x2(j+1)-x2(j-1))/(2*dt);
end

one=ones(289,1);
zilch=zeros(289,1);
x1s=x1(2:n-1);
x2s=x2(2:n-1);

%A=[x1s x2s];
A=[x1s x2s x1s.^2 x1s.*x2s x2s.^2 x1s.^3 (x1s.^2).*x2s (x2s.^2).*x1s x2s.^3
x1s.^4 x2s.^4 x1s.^5 x2s.^5 one];
%A=[x1s x2s zilch zilch zilch zilch zilch zilch zilch zilch zilch zilch zilch
one];
```

```
%A1=[x1s x2s zilch zilch zilch zilch zilch zilch zilch zilch zilch zilch
zilch one];
%A2=[x1s x2s zilch zilch zilch zilch zilch zilch zilch zilch zilch zilch
zilch one];

xi1=A\x1dot.';
xi2=A\x2dot.';
%lambda=0.002;
%xi1=lasso(A,x1dot.','Lambda',lambda);
%xi2=lasso(A,x2dot.','Lambda',lambda);
%xi1=robustfit(A,x1dot,[],[],'off');
%xi2=robustfit(A,x2dot,[],[],'off');

%xi1=pinv(A)*x1dot.';
%xi2=pinv(A)*x2dot.';
subplot(2,1,1), bar(xi1)
subplot(2,1,2), bar(xi2)

%Calculate Hankel Matrices
H1=[x1(1:100).'
   x1(6:105).'
   x1(11:110).'
   x1(16:115).'
   x1(21:120).'
   x1(26:125).'
   x1(31:130).'
   x1(36:135).'];

[u1,s1,v1]=svd(H1,'econ');
figure(2), subplot(2,1,1),
plot(diag(s1)/(sum(diag(s1))),'ro','Linewidth',[3])

H2=[x2(1:100).'
   x2(6:105).'
   x2(11:110).'
   x2(16:115).'
   x2(21:120).'
   x2(26:125).'
   x2(31:130).'
   x2(36:135).'];

[u2,s2,v2]=svd(H2,'econ');
subplot(2,1,2), plot(diag(s2)/(sum(diag(s2))),'ro','Linewidth',[3])

sol = ode45(@(t,y) odefun(t,y,xi1,xi2),[t(1) t(numel(t))],[hare(1),
lynx(1)]);
 x1predict=deval(sol,t,1);
 x2predict=deval(sol,t,2);

f=zeros(4,1);
lynxmean=mean(lynx);
haremean = mean(hare);
for i=1:numel(x1predict)
    if (x2predict(i) > lynxmean) && (x1predict(i) > haremean)
        f(1) = f(1) + 1;
```

```
    elseif (x2predict(i) > lynxmean) && (x1predict(i) < haremean)
        f(2) = f(2) + 1;
    elseif (x2predict(i) < lynxmean) && (x1predict(i) < haremean)
        f(3) = f(3) + 1;
    elseif (x2predict(i) < lynxmean) && (x1predict(i) > haremean)
        f(4) = f(4) + 1;
    end
end

g=zeros(4,1);

for i=1:numel(lynx)
    if ((lynx(i) > lynxmean) && (hare(i) > haremean))
        g(1) = g(1) + 1;
    elseif ((lynx(i) > lynxmean) && (hare(i) < haremean))
        g(2) = g(2) + 1;
    elseif ((lynx(i) < lynxmean) && (hare(i) < haremean))
        g(3) = g(3) + 1;
    elseif ((lynx(i) < lynxmean) && (hare(i) > haremean))
        g(4) = g(4) + 1;
    end
end

f=f/sum(f);
g=g/sum(g);
Int=f.*log(f./g); % compute integrand
Int(isinf(Int))=0; Int(isnan(Int))=0;
KL=sum(Int); % KL divergence

parameters=2;
n=numel(x1);
varianceLynx=(1/n)*(sum((x2predict(:)-x2(:)).^2))
varianceHare=(1/n)*(sum((x1predict(:)-x1(:)).^2))
AICLynx=2*parameters+n*log(2*pi)+n*log(varianceLynx)+n
BICLynx=log(n)*parameters+n*log(2*pi)+n*log(varianceLynx)+n
AICHare=2*parameters+n*log(2*pi)+n*log(varianceHare)+n
BICHare=log(n)*parameters+n*log(2*pi)+n*log(varianceHare)+n



 function odes = odefun(t,y,xi1,xi2)

 ode1 = xi1(1)*y(1) + xi1(2)*y(2) + xi1(3)*y(1).^2 + xi1(4)*y(1)*y(2) +
xi1(5)*y(2).^2 + xi1(6)*y(1).^3 + xi1(7)*(y(1).^2)*y(2) +
xi1(8)*y(1)*(y(2).^2) + xi1(9)*y(2).^3 + xi1(10)*y(1).^4 + xi1(11)*y(2).^4 +
xi1(12)*y(1).^5 + xi1(13)*y(2).^5 + xi1(14);
 ode2 = xi2(1)*y(1) + xi2(2)*y(2) + xi2(3)*y(1).^2 + xi2(4)*y(1)*y(2) +
xi2(5)*y(2).^2 + xi2(6)*y(1).^3 + xi2(7)*(y(1).^2)*y(2) +
xi2(8)*y(1)*(y(2).^2) + xi2(9)*y(2).^3 + xi2(10)*y(1).^4 + xi2(11)*y(2).^4 +
xi2(12)*y(1).^5 + xi2(13)*y(2).^5 + xi2(14);
 odes = [ode1; ode2];
 end
```

BZ Data

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

t=1:1:1200;
dt=t(2)-t(1);
U=BZ_tensor(100:149,200:249,:); %Take  ROI for calculation
[m,n,k]=size(U); % x vs y vs time data

Udot=zeros(m,n,k-2);
for jj=1:m  % walk through x (space)
for jjj=1:n %walk through y (space)
for j=2:k-1  % walk through time
   Udot(jj,jjj,j-1)=( U(jj,jjj,j+1)-U(jj,jjj,j-1) )/(2*dt);
end
end
end

%Derivatives
dx=1;
dy=1;

%calculate Ux
Ux=zeros(m-2,n-2,k);
for time=1:k
for i=2:n-1
    for j=2:m-1
        Ux(j-1,i-1,time)=(U(j,i+1,time)-U(j,i-1,time))/2*dx;
    end
end
end

%calculate Uy
Uy=zeros(m-2,n-2,k);
for time=1:k
for i=2:n-1
    for j=2:m-1
        Uy(j-1,i-1,time)=(U(j+1,i,time)-U(j-1,i,time))/2*dy;
    end
end
end

%calculate Uxx
Uxx=zeros(m-2,n-2,k);
for time=1:k
for i=2:n-1
    for j=2:m-1
        Uxx(j-1,i-1,time)=(U(j,i+1,time)-2*U(j,i,time)+U(j,i-1,time))/dx.^2;
    end
end
end
```

```
%calculate Uyy
Uyy=zeros(m-2,n-2,k);
for time=1:k
for i=2:n-1
    for j=2:m-1
        Uyy(j-1,i-1,time)=(U(j+1,i,time)-2*U(j,i,time)+U(j-1,i,time))/dy.^2;
    end
end
end

%calculate Uxy
Uxy=zeros(m-2,n-2,k);
for time=1:k
for i=2:n-1
    for j=2:m-1
        Uxy(j-1,i-1,time)=(U(j+1,i+1,time)-U(j-1,i+1,time)-U(j+1,i-
1,time)+U(j-1,i-1,time))/4*dx*dy;
    end
end
end

%Reshape all the function matrices, truncate where needed
u=reshape(U(2:49,2:49,2:1199),(n-2)*(m-2)*(k-2),1);
udot=reshape(Udot(2:49,2:49,:),(n-2)*(m-2)*(k-2),1);
ux=reshape(Ux(:,:,2:1199),(n-2)*(m-2)*(k-2),1);
uy=reshape(Uy(:,:,2:1199),(n-2)*(m-2)*(k-2),1);
uxx=reshape(Uxx(:,:,2:1199),(n-2)*(m-2)*(k-2),1);
uxy=reshape(Uxy(:,:,2:1199),(n-2)*(m-2)*(k-2),1);
uyy=reshape(Uyy(:,:,2:1199),(n-2)*(m-2)*(k-2),1);

%Define library of functions theta (noted here as "A")
zilch=zeros(2760192,1);
%A=[u u.^2 u.^3 ux uxx ux.*u ux.*ux ux.*uxx uy uyy uyy.*u uy.*uy uy.*uyy uxy
uxy.*u];
A=[zilch zilch zilch ux zilch zilch zilch zilch uy zilch zilch zilch zilch
zilch zilch];

%xi=A\udot;
xi=lasso(A,udot,'Lambda',0.002);
%xi=robustfit(A,udot,[],[],'off');
%xi=pinv(A)*udot;
bar(xi)

%Use selected functions from library and generate model prediction
Asparse=zeros(2760192,15);
%Asparse(:,1)=A(:,1);
Asparse(:,4)=A(:,4);
Asparse(:,9)=A(:,9);
%Asparse(:,10)=A(:,10);
Asparse(:,14)=A(:,14);
udotpredict=Asparse*xi;
```