HW2 Model Selection

Huasong Zhang
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
huasongz@uw.edu

Abstract

In this homework assignment, there are two parts. For the first part, we need to deal with the data on hare and lynx population. There is some relationship between them, and our task is to find out different dynamical systems describing the population of these two. In order to find the good model describing the dynamical system, I used SINDY and did least square, LASSO, and ridge regression on that to find the best model. For part two, it's very similar to part one. Instead, we have a large dataset about the chemistry reaction. Our task is to track the movement and find out the model. I used the same SINDY and regression methods.

1 Introduction and Overview

There are a lot of things going on in the real world that we know there has to be some relationships, but we do not actually know the model describing that. In order to better understand the relationship, we need to find the model from the given data. One of the method is SINDY which stands for Sparse Identification of Nonlinear Dynamics. This method works based on the fact that most dynamical systems have only a few nonlinear terms out of the whole family of possible terms. Therefore, we can build a large library, do regression on that and pick out the ones that has large values. Then, we have our model.

2 Theoretical Background

For a dynamical system, we can assume it as a linear system, which is $A * x = b$. In this linear system, the A is the library we have, which contains all the possible terms. b is the derivative term usually on the left hand side. x is the one we are trying to find; it represents the weight on each term in library A. However, in real life problem, we only have the real data; the first step we need to do is to build it's derivative. After we have that, we can build library A. In terms of possible terms, we usually have polynomials, trigonometry terms, exponential, and so on. After getting A and b, we are able to solve the $A * x = b$ using different methods to get different models.

In order to promote sparsity, there are two ways to do that. The first way is to use LASSO regression, which automatically make unimportant coefficients equal to 0. Another way is to use other regression methods, such as linear regression and ridge regression. These methods cannot make coefficients equal to absolute zeros; they can only make them small. In order to promote sparsity, we could set some thresholds and get rid of those small coefficients.

For this assignment, I used three regression methods to solve this problem, which are linear regression, LASSO, and ridge regression. The principle of linear regression is to find:

$$argmin_x ||A * x - b||^2$$

For LASSO, which is similar to linear regression, but including an extra penalty term. So for LASSO regression, the objective function is:

$$argmin_x ||A * x - b||^2 + \lambda * |x|$$

And LASSO will make the unnecessary terms equal to 0, which promote sparsity. For ridge regression, the only difference comparing to LASSO is that the penalty term is the l2 norm of x instead of l1 norm. So the objective function now becomes:

$$argmin_x ||A * x - b||^2 + \lambda * |x|_2$$

After did the regression on the data, by eyeballing the coefficients on a bar plot, we can set an appropriate threshold to get rid of the unimportant coefficients. Then, we need to redo regression on the remaining terms in A to get updated coefficients for them in order to get the accurate model.

Since the model we get is for derivatives; in order to get the true value, we need to solve the ODEs. We can solve the equations directly; however, the number of data points we have this this assignment is too small, we cannot get a good result. I used forward Euler method to calculate the data. Forward Euler method uses the definition of derivative and use the current value to predict the next value. It leaves x_{n+1} term on the left side and moves all the other terms on the right side:

$$x' = \frac{x_{n+1} - x_n}{\Delta t}$$

$$x_{n+1} = x' * \Delta t + x_n$$

KL divergence is a way to evaluate how one distribution is different from another distribution. The equation for KL divergence is:

$$\int p(x, \beta) * log(\frac{p(x, \beta)}{q(x, \beta)})$$

Since we only have the prediction and true value of the data, we need to find out the distribution of them. In order to do that, I put hare and lynx population into histogram using 4 bins and get the probability distribution. After that, I can plug in the numbers in the equation and find out the KL divergence. If the number for KL divergence is small, it means these two distributions are very similar.

AIC and BIC are another two ways to evaluate two distributions. The general formula is:

$$-2log(L) + k * p$$

L is the likelihood function, which is calculated as:

$$SSE = (y - \hat{y})^2$$

p is the number of parameters in the model and k is the only difference distinguishes between AIC and BIC. The k is 2 for AIC and $log(n)$ for BIC where n is the length of predictions. So the equations for AIC and BIC are:

$$AIC = -2log(L) + 2 * p$$

$$bic = -2log(L) + log(n) * p$$

AIC and BIC would also give us how two distribution are different from each other. AIC estimates the distance between the true likelihood function of the data and the likelihood function of the model; while BIC is an estimate of a posterior probability of a model being true. Therefore, for both AIC and BIC, the smaller in absolute value, the better [AIC].

For Part One question 4, in order to find are there other latent variables in this dynamical system, we need to use the time-delay embedding method to build a big matrix and then do SVD on it to pick out the important components. The big matrix is called Hankel matrix, and it is build in this way:

$$X = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_{n-k} \\ x_2 & x_3 & x_4 & \dots & x_{n-k+1} \\ \dots & \dots & \dots & \dots & \dots \\ x_k & x_{k+1} & x_{k+2} & \dots & x_n \end{bmatrix}$$

The matrix X is a tall and skinny matrix, which means we can reduce the dimension of it by doing SVD. Then we can plot the singular values to find out the number of important components.

3 Algorithm Implementation and Development

3.1 Part 1

Since the original data only has 30 elements for each animal; that is not enough for building a model, the model will not perfectly describe the relationship. Therefore, the first thing we need to do is to interpolate the data so that we have have more information and we can get a more accurate model. I interpolated the data and extended it from 30 to 290 elements. Next, we need to construct the derivative matrix of the data to build b vector. I used the central difference method to calculate the derivative for hare and lynx population which is:

$$x' = \frac{x_{n+1} - x_{n-1}}{2\Delta t}$$

and saved as x1dot and x2dot. Now we need to build library A so that we we can do sparse regression. For my library A, I used polynomial terms up to degree 3; I also included trigonometry terms. After getting A and b, we can solve $Ax = b$ problem.

The first method I used is the basic linear regression, which is the "pinv" command in python. The A matrix is tall and skinny, therefore, we would want to use the pseudo-inverse of A. Since "pinv" will not make some of the coefficients equal to absolute 0, I set the threshold of 0.5. Any coefficients have absolute value less than 0.5 will be considered as 0. In this way, we could promote sparsity and only keep the important terms in A. The same strategy was also used for ridge regression to promo sparsity.

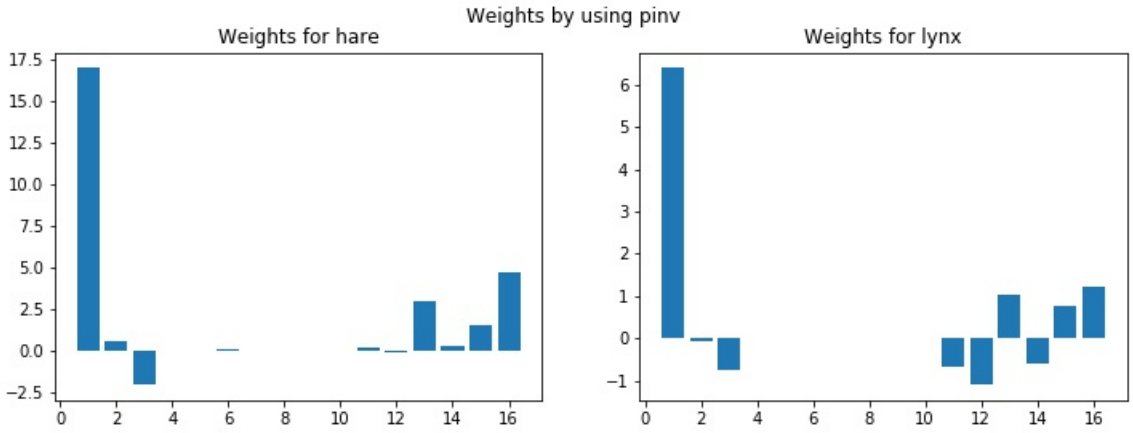


Figure 1: Coefficients for pinv

After obtaining the terms in A, I did regression again to get the updated coefficients of these terms. This is the process of getting the model.

For LASSO regression, this method already promoted sparsity by forcing some coefficients equal to 0. However, there were still a lot of terms left with very small coefficients. Therefore, I set threshold on LASSO as well to make it more sparse. Then I re-regressed the model and got the coefficients of my model.

In order to find the best model among these three, I calculated the prediction for the derivatives and compare with the real derivatives. Then, the model with the smallest difference is the best model I want.

The model is for the dynamical system, which is the derivative. In order to see how the model fits the true value, we need to solve the equation and get the true value. I tried to solve the ODEs, however, the population blew up and went to infinity due the small amount of data points. Therefore, I used the forward Euler method to solve for it. After getting the true value from the model, I compared it with the real data, and calculated the KL divergence, AIC, and BIC for all three models I got.

Lastly, since an ecology system is complicated; but we only have information of two animal population. There may be other latent variables that may affect their population, such as weather, food resources, and so on. Therefore, I used time-delay embedding method to construct a big matrix of these two populations and did SVD on it to find out the important number of components by plotting singular values. If there are more than two significant singular values, it means there are other factors that can affect the population.

3.2 Part 2

For part two, it is very similar to the question 1 from part one. This time, we would like to get the movement trajectory of a chemistry reaction. Instead of tracking the movement from the entire data frame, I only picked out the pixel has maximum value and noted down x and y coordinates. Also, I did not build the derivative matrix as b for this case; I used the time embedding idea and built the matrix with time moving forward by one step. Then, I used the same library A in Part One and used the same three regression methods to find the coefficients.

4 Computational Results

4.1 Part I

Figure 1 Figure 2 and Figure 3 shows the coefficients from three regression methods. As we can see from the plot that pinv takes the constant terms as significant, however, LASSO and ridge have more weights on the trigonometry terms. And the predictions for derivative are shown in Figure 4 Figure 5 and Figure 6. By calculated the error, pinv has the smallest error which means it is the best model. And also from the comparison between true value with prediction in Figure 7 we can see that pinv matches with the true value more comparing to other methods. By calculating the KL divergence, AIC, and BIC for three methods, the results are shown in Table 1: Actually from the KL divergence we can see that for real value prediction, ridge regression did the best. And pinv performs the best from AIC and BIC.

By using time-delay embedding method to find the latent variables, the singular value plot is shown in Figure 8 As we can see in the plot that there are definitely more than two singular values that are important. Therefore, it means there are many more possible factors in this dynamical system that may affect the population of hare and lynx.

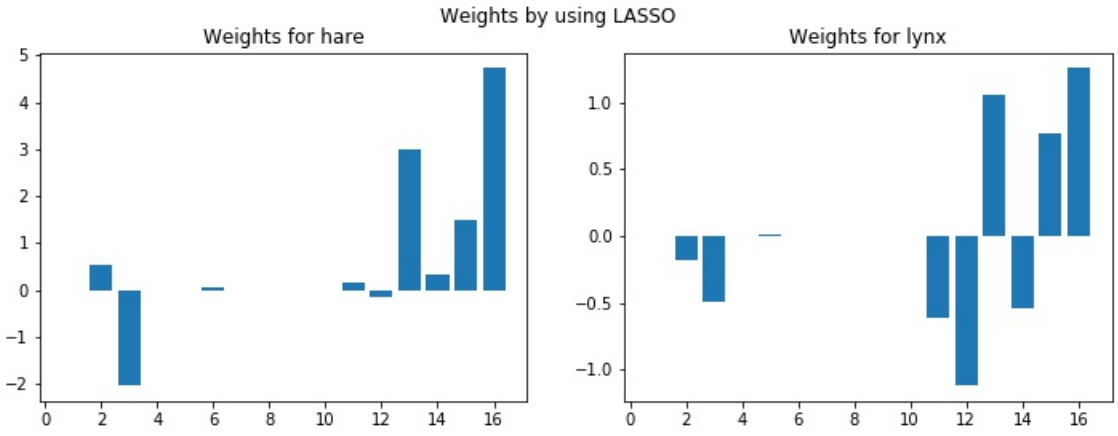


Figure 2: Coefficients for LASSO

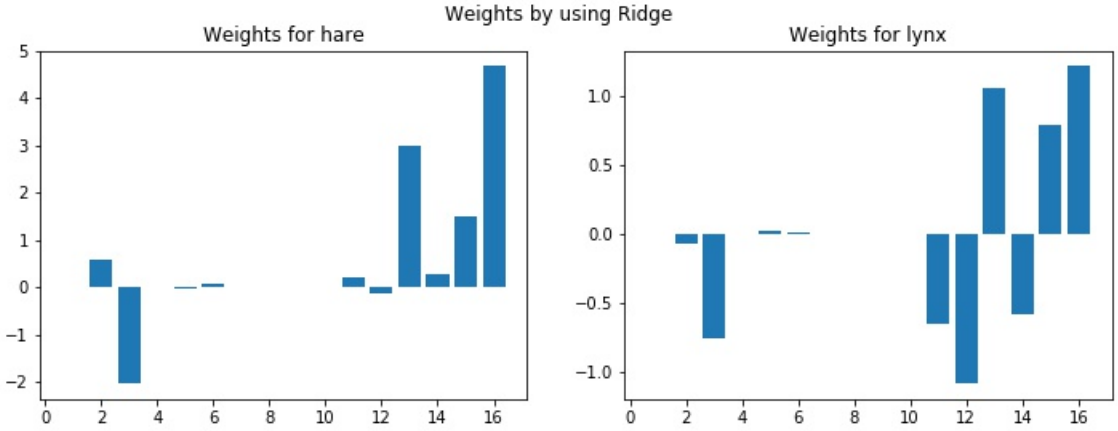


Figure 3: Coefficients for Ridge

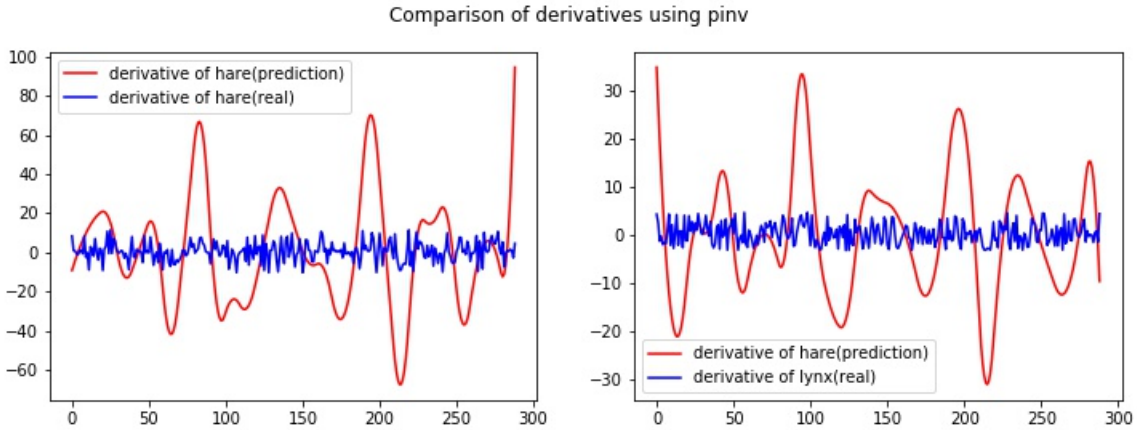


Figure 4: Derivative prediction for pinv

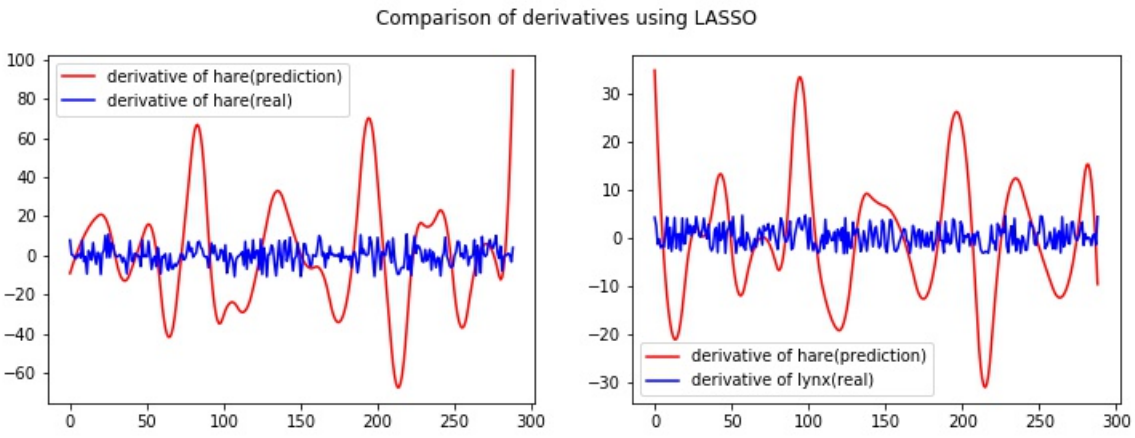


Figure 5: Derivative prediction for LASSO

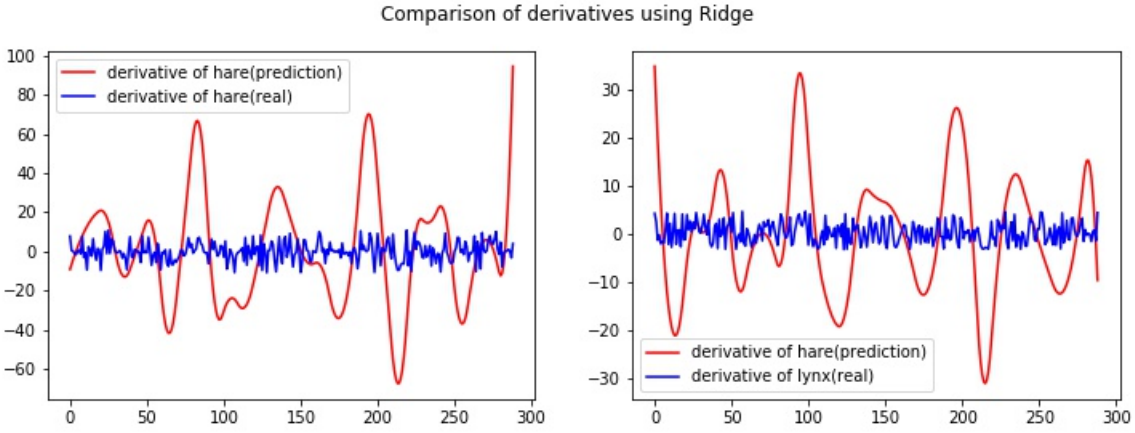
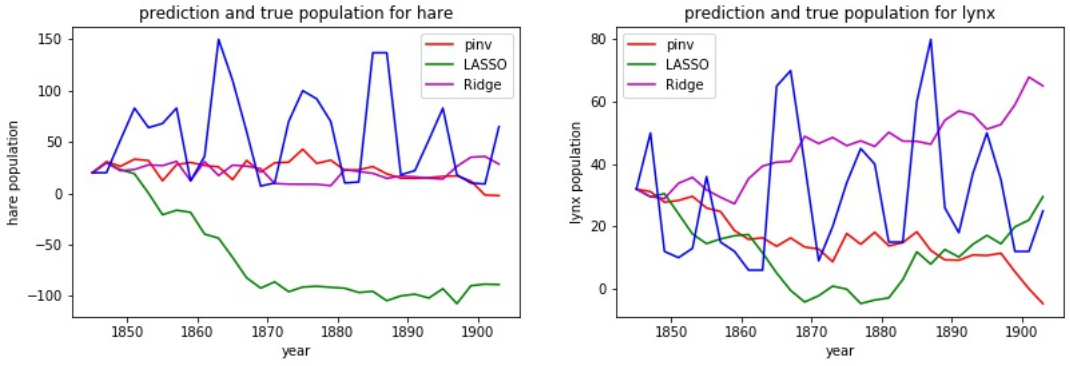


Figure 6: Derivative prediction for ridge



(a) Prediction for hare population

(b) Prediction for lynx population

Figure 7: Comparison between true value and predictions

Col1	KL_l	KL_l	AIC_h	AIC_l	BIC_h	BIC_l
pinv	0.37	0.47	-10.68	-3.83	-2.28	7.38
LASSO	0.16	0.18	-16.40	-6.52	-9.40	3.29
Ridge	0.11	0.09	-12.97	-5.97	-5.96	3.84

Table 1: KL divergence, AIC, and BIC

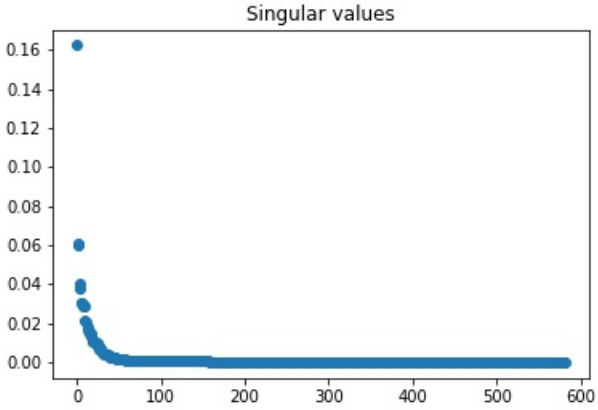


Figure 8: Singular values

4.2 Part II

Figure 9 Figure 10 and Figure 11 shows the coefficients from three methods on Part Two of this assignment. As we can see from the bar plot, this time three methods give us three totally different models.

5 Summary and Conclusions

From this project we can see how messy the data from real life can be. It is impossible for us to find a perfect model to describe the data in real life because there are so many latent factors affecting it, which add so much noise. Only a small amount of noise can affect the whole dynamical system enormously. And also, the way to find the model can also influence the model we get. For instance, the least square approach gives me a constant term, while the LASSO and ridge only gives me model trigonometry terms. For this assignment, I set the alpha value manually for LASSO and ridge. Hopefully I will get better model by changing value of alpha and did cross validation during model selection process.

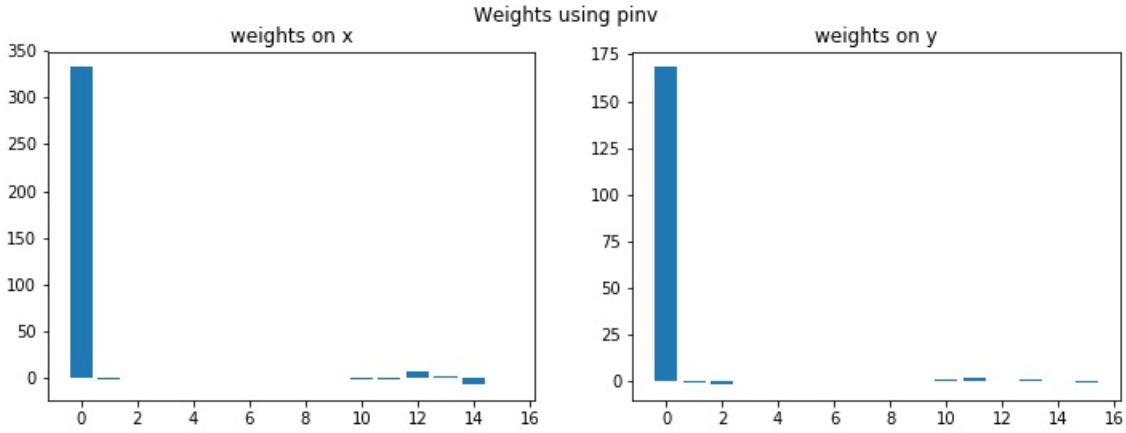


Figure 9: Coefficients for pinv

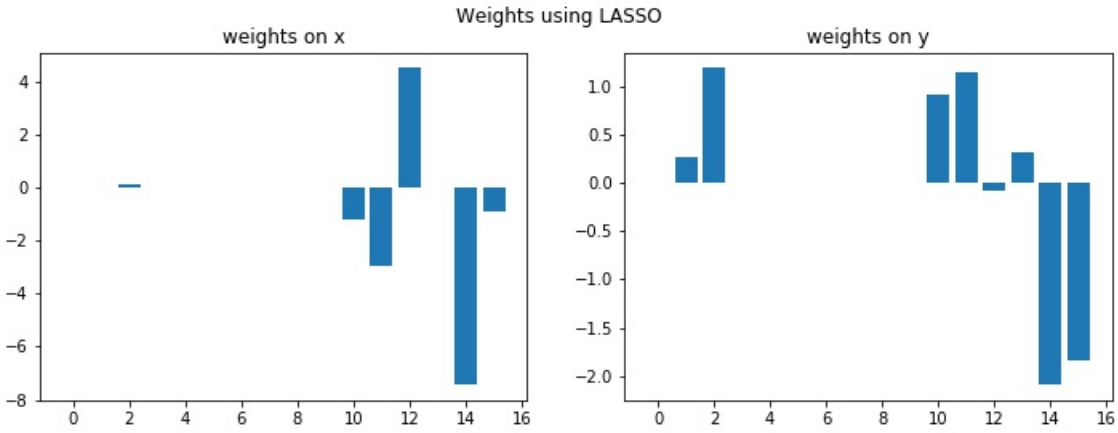


Figure 10: Coefficients for LASSO

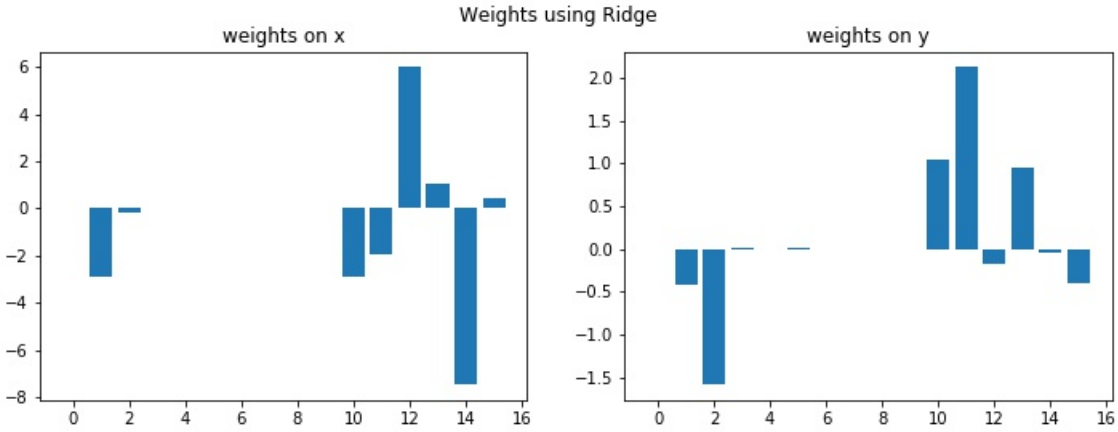


Figure 11: Coefficients for Ridge

Appendix Codes

Part I

```
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
from scipy.interpolate import UnivariateSpline
from sklearn import linear_model
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LassoCV, LassoLarsCV, LassoLarsIC
from scipy import integrate
from scipy.integrate import solve_ivp
from scipy.linalg import hankel
#from collections import Counter

population = scipy.io.loadmat('population.mat')
x1 = population['hare'].ravel()
x2 = population['lynx'].ravel()
t = population['year'].ravel()

# interpolate data
spl1 = UnivariateSpline(t, x1)
spl2 = UnivariateSpline(t, x2)
```

```
t_new = np.arange(1845,1903.1,0.2)
x1_new = spl1(t_new)
x2_new = spl2(t_new)
dt = t_new[1]-t_new[0]

# compute the derivative
n = len(t_new);
x1dot = np.zeros(n-2)
x2dot = np.zeros(n-2)
for j in range(1,n-1):
    x1dot[j-1] = (x1_new[j+1]-x1_new[j-1])/(2*dt)
    x2dot[j-1] = (x2_new[j+1]-x2_new[j-1])/(2*dt)

# make the measurement the same size of derivative
x1s = x1_new[1:n-1]
x2s = x2_new[1:n-1]

# build library
def LAB(x1,x2):
    M = np.array([x1,x1,x2,x1**2,x1*x2,x2**2,x1**3,x1*(x2**2),(x1**2)*x2,
                  x2**3,np.sin(x1),np.cos(x1),np.sin(x2),np.cos(x2),
                  np.sin(x1)*np.cos(x2),np.cos(x1)*np.sin(x2)])
    if isinstance(x1, int) or isinstance(x1, float):
        M[0] = 1
    else:
        M[0] = np.ones(len(x1))
    return M.T

A = LAB(x1s,x2s)
[g,h] = A.shape

# least square fit
xi11 = np.dot(np.linalg.pinv(A),x1dot)
xi12 = np.dot(np.linalg.pinv(A),x2dot)
f, (ax1, ax2) = plt.subplots(1,2,figsize=(12, 4))
ax1.bar(range(1,1+1),xi11)
ax1.set_title('Weights for hare')
ax2.bar(range(1,1+1),xi12)
ax2.set_title('Weights for lynx')
f.suptitle('Weights by using pinv')
f.savefig('pinv.jpg')

a1 = np.where(abs(xi11)>0.5)
b1 = np.where(abs(xi12)>0.5)
r1 = np.zeros(h)
r2 = np.zeros(h)
r1[a1] = 1
r2[b1] = 1
l1 = len(a1[0])
l2 = len(b1[0])
for i in range(5):
    xi11_new = np.dot(np.linalg.pinv(A*r1),x1dot)
    xi12_new = np.dot(np.linalg.pinv(A*r2),x2dot)
    a11 = np.where(abs(xi11_new)>0.5)
    b11 = np.where(abs(xi12_new)>0.5)
    if len(a11[0]) == l1 and len(b11[0]) == l2:
        break
    else:
        r1 = np.zeros(h)
        r2 = np.zeros(h)
        r1[a11] = 1
        r2[b11] = 1
```

```
r11 = r1
r12 = r2
x1_pre11 = np.sum(A*r11*xi11_new,axis = 1)
x2_pre11 = np.sum(A*r12*xi12_new,axis = 1)
err11 = np.sum((x1_pre11-x1dot)**2)
err12 = np.sum((x2_pre11-x2dot)**2)

f, (ax1, ax2) = plt.subplots(1,2,figsize=(12, 4))
ax1.plot(x1dot,'r',label="derivative of hare(prediction)")
ax1.plot(x1_pre11,'b',label="derivative of hare(real)")
ax1.legend()
ax2.plot(x2dot,'r',label="derivative of hare(prediction)")
ax2.plot(x2_pre11,'b',label="derivative of lynx(real)")
ax2.legend()
f.suptitle('Comparison of derivatives using pinv')
f.savefig('derivative_pinv.jpg')

# LASSO
clf_l1 = linear_model.Lasso(alpha=0.01)
clf_l1.fit(A, x1dot)
xi21 = clf_l1.coef_
clf_l2 = linear_model.Lasso(alpha=0.01)
clf_l2.fit(A, x2dot)
xi22 = clf_l2.coef_
f, (ax1, ax2) = plt.subplots(1,2,figsize=(12, 4))
ax1.bar(range(1,l+1),xi21)
ax1.set_title('Weights for hare')
ax2.bar(range(1,l+1),xi22)
ax2.set_title('Weights for lynx')
f.suptitle('Weights by using LASSO')
f.savefig('LASSO.jpg')

a2 = np.where(abs(xi21)>0.5)
b2 = np.where(abs(xi22)>0.2)
r1 = np.zeros(h)
r2 = np.zeros(h)
r1[a2] = 1
r2[b2] = 1
l1 = len(a2[0])
l2 = len(b2[0])
for i in range(5):
    clf_l11 = linear_model.Lasso(alpha=0.01)
    clf_l11.fit(A*r1, x1dot)
    xi21_new = clf_l11.coef_
    clf_l22 = linear_model.Lasso(alpha=0.01)
    clf_l22.fit(A*r2, x2dot)
    xi22_new = clf_l22.coef_
    a22 = np.where(abs(xi21_new)>0.5)
    b22 = np.where(abs(xi22_new)>0.5)
    if len(a22[0]) == l1 and len(b22[0]) == l2:
        break
    else:
        r1 = np.zeros(h)
        r2 = np.zeros(h)
        r1[a22] = 1
        r2[b22] = 1
r21 = r1
r22 = r2
x1_pre2 = np.sum(A*r21*xi21_new,axis = 1)
x2_pre2 = np.sum(A*r22*xi22_new,axis = 1)
err21 = np.sum((x1_pre2-x1dot)**2)
err22 = np.sum((x2_pre2-x2dot)**2)
```

```
f, (ax1, ax2) = plt.subplots(1,2,figsize=(12, 4))
ax1.plot(x1dot,'r',label="derivative of hare(prediction)")
ax1.plot(x1_pre2,'b',label="derivative of hare(real)")
ax1.legend()
ax2.plot(x2dot,'r',label="derivative of hare(prediction)")
ax2.plot(x2_pre2,'b',label="derivative of lynx(real)")
ax2.legend()
f.suptitle('Comparison of derivatives using LASSO')
f.savefig('derivative_pinv.jpg')

# Ridge
clf_r1 = Ridge(alpha=0.05)
clf_r1.fit(A, x1dot)
xi31 = clf_r1.coef_
clf_r2 = Ridge(alpha=0.05)
clf_r2.fit(A, x2dot)
xi32 = clf_r2.coef_
f, (ax1, ax2) = plt.subplots(1,2,figsize=(12, 4))
ax1.bar(range(1,l+1),xi31)
ax1.set_title('Weights for hare')
ax2.bar(range(1,l+1),xi32)
ax2.set_title('Weights for lynx')
f.suptitle('Weights by using Ridge')
f.savefig('Ridge.jpg')

a3 = np.where(abs(xi31)>0.5)
b3 = np.where(abs(xi32)>0.2)
r1 = np.zeros(h)
r2 = np.zeros(h)
r1[a3] = 1
r2[b3] = 1
l1 = len(a3[0])
l2 = len(b3[0])
for i in range(5):
    clf_r11 = Ridge(alpha=0.01)
    clf_r11.fit(A*r1, x1dot)
    xi31_new = clf_r11.coef_
    clf_r22 = Ridge(alpha=0.01)
    clf_r22.fit(A*r2, x2dot)
    xi32_new = clf_r22.coef_
    a33 = np.where(abs(xi31_new)>0.5)
    b33 = np.where(abs(xi32_new)>0.5)
    if len(a33[0]) == l1 and len(b33[0]) == l2:
        break
    else:
        r1 = np.zeros(h)
        r2 = np.zeros(h)
        r1[a33] = 1
        r2[b33] = 1
r31 = r1
r32 = r2
x1_pre3 = np.sum(A*r31*xi31_new,axis = 1)
x2_pre3 = np.sum(A*r31*xi31_new,axis = 1)
err31 = np.sum((x1_pre3-x1dot)**2)
err32 = np.sum((x2_pre3-x2dot)**2)

f, (ax1, ax2) = plt.subplots(1,2,figsize=(12, 4))
ax1.plot(x1dot,'r',label="derivative of hare(prediction)")
ax1.plot(x1_pre2,'b',label="derivative of hare(real)")
ax1.legend()
ax2.plot(x2dot,'r',label="derivative of hare(prediction)")
```

```
ax2.plot(x2_pre2,'b',label="derivative of lynx(real)")
ax2.legend()
f.suptitle('Comparison of derivatives using Ridge')
f.savefig('derivative_ridge.jpg')

def predict(x,y,n,r1,r2,xi,yi,dt):
    x_pre = np.zeros(n)
    y_pre = np.zeros(n)
    x_pre[0] = x
    y_pre[0] = y
    for i in range(n-1):
        x = x+sum(LAB(x,y)*r1*xi)*dt
        y = y+sum(LAB(x,y)*r2*yi)*dt
        x_pre[i+1] = x
        y_pre[i+1] = y
    return (x_pre,y_pre)

x1_pre_1,x2_pre_1 = predict(20,32,30,r11,r12,xi11_new,xi12_new,2)
x1_pre_2,x2_pre_2 = predict(20,32,30,r21,r22,xi21_new,xi22_new,2)
x1_pre_3,x2_pre_3 = predict(20,32,30,r31,r32,xi31_new,xi32_new,2)
plt.figure()
plt.plot(t,x1_pre_1,'r',label='pinv')
plt.plot(t,x1_pre_2,'g',label='LASSO')
plt.plot(t,x1_pre_3,'m',label='Ridge')
plt.plot(t,x1,'b')
plt.legend()
plt.xlabel('year')
plt.ylabel('hare population')
plt.title('prediction and true population for hare')
plt.savefig('pre_h.jpg')
plt.figure()
plt.plot(t,x2_pre_1,'r',label='pinv')
plt.plot(t,x2_pre_2,'g',label='LASSO')
plt.plot(t,x2_pre_3,'m',label='Ridge')
plt.plot(t,x2,'b')
plt.legend()
plt.xlabel('year')
plt.ylabel('lynx population')
plt.title('prediction and true population for lynx')
plt.savefig('pre_l.jpg')

# KL divergence
hist_h = np.histogram(x1,4)
hist_l = np.histogram(x2,4)
hist1 = np.histogram(x1_pre_1,4)
hist2 = np.histogram(x2_pre_1,4)
kl1 = scipy.stats.entropy(hist1[0], hist_h[0])
kl2 = scipy.stats.entropy(hist2[0], hist_l[0])

# AIC
def AIC(y,y_pred,k):
    res = y - y_pred
    sse = sum(res**2)
    a = 2*k-2*np.log(sse)
    return a
# k is the number of predictors

AIC11 = AIC(x1_pre_1,x1,len(a1[0]))
AIC12 = AIC(x2_pre_1,x2,len(b1[0]))
AIC21 = AIC(x1_pre_2,x1,len(a2[0]))
AIC22 = AIC(x2_pre_2,x2,len(b2[0]))
AIC31 = AIC(x1_pre_3,x1,len(a3[0]))
```

```
AIC32 = AIC(x2_pre_3,x2,len(b3[0]))

# BIC
def BIC(y,y_pred,k,n):
    res = y - y_pred
    sse = sum(res**2)
    b = np.log(n)*k - 2*np.log(sse)
    return b
# n is the number of data
BIC11 = BIC(x1_pre_1,x1,len(a1[0]),len(x1_pre_1))
BIC12 = BIC(x2_pre_1,x2,len(b1[0]),len(x2_pre_1))
BIC21 = BIC(x1_pre_2,x1,len(a2[0]),len(x1_pre_2))
BIC22 = BIC(x2_pre_2,x2,len(b2[0]),len(x2_pre_2))
BIC31 = BIC(x1_pre_3,x1,len(a3[0]),len(x1_pre_3))
BIC32 = BIC(x2_pre_3,x2,len(b3[0]),len(x2_pre_3))

# Delay time embeding
X = np.array([x1_new,x2_new])
h = hankel(X)
[u,s,v] = np.linalg.svd(h)
plt.figure()
plt.plot(s/sum(s),'o')
plt.title('Singular values')
plt.savefig('sig.jpg')
```

Part II

The codes for problem 1 in Part One can be used as references.

Reference

"AIC vs. BIC." The Methodology Center, www.methodology.psu.edu/resources/aic-vs-bic/.