Model Discovery Applications in Dynamical Systems

Maple Tan
AMATH563 Homework 2

May 3, 2019

1 Abstract

Explore how to use model discovery methods to fit data.

2 Introduction and Overview

We will look at two different sets of data. One is the amount of Snowshoe Hare and Snowshoe Lynx pelts collected from the years 1845 to 1903 and the second being a video snippet of a Belousov-Zhabotinsky chemical oscillator. With both sets of data, we will use sparse regression to find the best nonlinear, dynamical systems model that will fit the data and calculate the KL divergence. For only the first set of data, we will also calculate the AIC and BIC scores for a couple of models and time-delay embed the system and determine if there are latent variables.

3 Theoretical Background

When it comes to model discovery, it starts by assuming that our data is governed by the differential equation:

$$\frac{dy}{dt} = f(y, t) \tag{1}$$

Where f is currently unknown. First that the above differential equation can be generalized into an $Ax = b$ problem, where $Ax = f(y, t)$ and $b = \frac{dy}{dt}$. To set up the problem, we begin by calculating the derivative of our data at our time shots to serve as our b. The differentiation schema is important and in this case, we experiment with a central difference schema and the built in method from the python library `scipy`.

Next, we note that the potential forms of f could be a polynomial, a trigonometric function, exponential, etc. So, let's suppose that f is governed by a library of these possible functions. We then create a library of all possible functions based on our data.

As an example, suppose that we have a set of discrete time series which serves as our data:

$$\mathbf{y} = \begin{bmatrix} y(t_1), & y(t_2), & y(t_3), & \dots, & y(t_n) \end{bmatrix}$$

Next, we create a library based on our y:

$$A = \begin{bmatrix} | & | & | & & | & | & \\ \mathbf{y} & \mathbf{y}^2 & \mathbf{y}^3 & \cdots & \sin(\mathbf{y}) & \cos(\mathbf{y}) & \cdots \\ | & | & | & & | & | & \end{bmatrix}$$

Where the library should have a variety of different functions that are not limited to the ones listed above. Now we solve our $Ax = b$ problem, promoting sparsity since we will have an over-determined system. In this case, the different x values are the coefficients associated with the each function in our library. The larger the magnitude of the coefficient, the more important that function plays in the system. We can then isolate those functions to recreate our system.

When it comes to evaluating the viability of our model, we have the Kullback-Leibler (KL) divergence and it measures the distance between probability densities. In our case, the probability densities will be two data sets, one representing the model and the other representing the truth. If we let $f(X, \beta)$ be one model/data set and $g(X, \mu)$ be the other, the KL divergence is defined as:

$$I(f, g) = \int f(X, \beta) \log \left[\frac{f(X, \beta)}{g(X, \mu)} \right] dX \tag{2}$$

Where β and μ are parameterizations of the model f and g respectively. This quantity measures the amount of information lost and note that if $f = g$ then $I = 0$ and no information was lost. In practice, f will represent the truth, or measurements of an experiment, while g will be a model proposed to model f.

Building on this, we also have the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) which provides an information score when not all the information is available to calculate the KL divergence. These values instead provide a way to estimate KL divergence based upon the empirical log likelihood function at its maximum point.

$$AIC = 2K - 2\log[\mathcal{L}(\hat{\mu}|x)] \tag{3}$$

Where K is the number of parameters used in the model, $\hat{\mu}$ is an estimate of the best parameters used (lowest KL divergence) in $g(X, \mu)$ computed from a maximum likelihood estimate(MLE), and x are independent samples of the data to be fit. Note that the AIC

penalizes the model for adding extra terms. Then the BIC is the same as AIC but does not penalize for adding extra terms.

$$BIC = \log(n)K - 2\log[\mathcal{L}(\hat{u}|x)] \tag{4}$$

Where n is the number of data points. Note that in practice, in both AIC and BIC, we can write:

$$\mathcal{L}(\hat{u}|x)1 = \text{SSE}(y, \hat{y}) = \left(\sum_{i=1}^{n}(y - \hat{y})^2\right)^{1/2}$$

Where y is the recorded data and $\hat{y}$ is the model predicted data.

Another thing about model discovery is that we often do not know if the variables we have in our measured data is coupled with other variables not recorded. One way to discover these latent variables is through a method called Time Delay Embedding. For Time Delay Embedding, we first construct a matrix known as the Hankel Matrix. The Hankel Matrix takes slice of the data for one row and then steps that entire slice one time step ahead for the next row. If we let our data be x, then, the Hankel Matrix H will be constructed as so:

$$x = [x_0, x_1, x_2, \ldots, x_{n-1}, x_n]$$
$$H = \begin{bmatrix} x_0 & x_1 & \ldots & x_m \\ x_1 & x_2 & \ldots & x_{m+1} \\ x_2 & x_3 & \ldots & x_{m+1} \\ \vdots & & & \end{bmatrix}, \quad m < n$$

Using this, we can apply the SVD to the Hankel Matrix and look at the singular values to find latent variables. This works because if our data is coupled with other variables, then the data will inherently have information stored within it to indicate that other variables exist.

4 Algorithm Implementation and Development

We start by loading out Python libraries and our data which comes in a `.mat` file. Starting with the pelt data, we first load our data using the `loadmat` method from the `scipy.io` library. This method returns a dictionary that contains our data sets in separate 3D matrices. We then extract the data and use the `squeeze` method from `numpy`s library to collapse the matrices into a 1-D vectors. Note that there are only around 30 data points in our data set, which may not be enough. So, we use `UnivariateSpline` from the `scipy.interpolate` library to fill in points between our time steps so that we have a data set of 1,000 measurements instead.

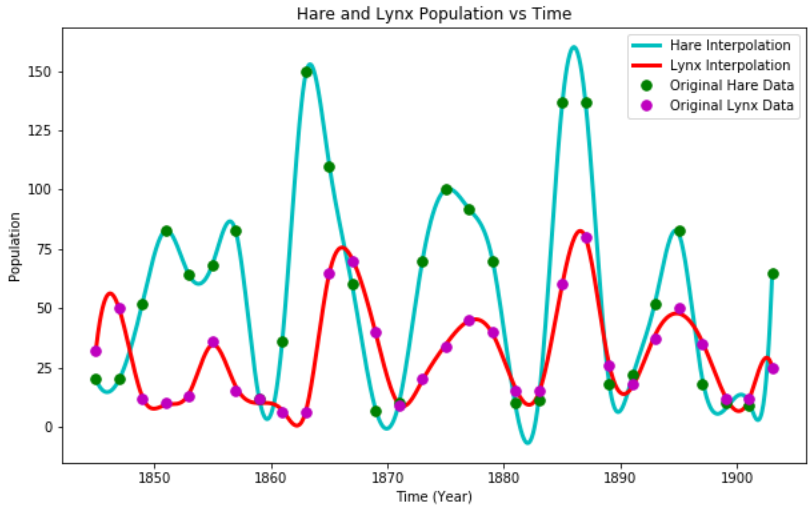

Figure 1: Interpolation of the original hare and lynx pelt data

After interpolating the data, we next take the derivative the both the lynx and hare data. We used two different methods to achieve this: one being the built-in derivative method in `UnivariateSpline` library and the other being a central difference schema:

$$F'(x_n) = \frac{F(x_{n+1} - F(x_{n-1})}{2 * dt} \tag{5}$$

After creating our derivative vector, we next need to create a library of functions. We begin by filling our library a combination of polynomials and trigonometric functions. Then for different trials, we swap up the type of functions in the library. With the library created, we simply need to solve our $Ax = b$ problem. We use the `lasso` method from `scipy`, which will promote a sparse solution that can help us determine which functions in our library is most prominent in our data. Note that the Lasso method in this case will include an intercept along with the coefficients, so we must keep that in account.

We then use a bar graph to plot the different coefficients and to see which coefficients are most important. We the top two or three to recreate our derivative and then using that derivative to create our simulation. Since the derivation method does not make much of a difference in the coefficients we obtain, we simply use definition of the derivative to backwards solve for our simulations and plot them against our original observations.

Next we calculate the KL Divergence, AIC and BIC scores of the three different models we created. To calculate the KL Divergence, we use the `entropy` method from the `scipy` library. The `entropy` method works by taking the simulated data and original data as inputs and calculates the KL diversions for us. Then from there, we calculate the sum of squared errors and use that with our definition of AIC and BIC to calculate those values

Next, we use time delay embedding to see if our system may or may not contain some latent variables. Beginning by using a for-loop to build our Hankel Matrix, we then use the SVD functions from the `linalg` library of +`numpy` and then plot the the singular values as extracted from the SVD.

Our next set of data that we will be working with is a snippet from a Belousov-Zhabotinsky chemical oscillator movie. This set of data is a 3-D matrix that contains 1,200 frames of pictures that document the chemical reaction. Unlike the population dynamics, we will be solving a PDE as opposed to ODE. In this case we approach this problem similarly to the previous problem with some changes. Starting off with the data itself, due to memory limitations, we do not take the entire frame of the chemical reaction but rather only a 50 by 50 square of the frame. Next we take the derivative with respect to both time and space, down both X and Y axes. Next we reshape each frame into a 1D vector and basically stack all the time stamps on top of each other. We then create a matrix `D` to multiply against this reshaped matrix so that we can take the partial derivative using the central schema difference. We then create a library based off this reshaped vector and its partial derivatives and similarly, solve the $Ax = b$ problem. Finally, we graph the coefficients.

5 Computational Results

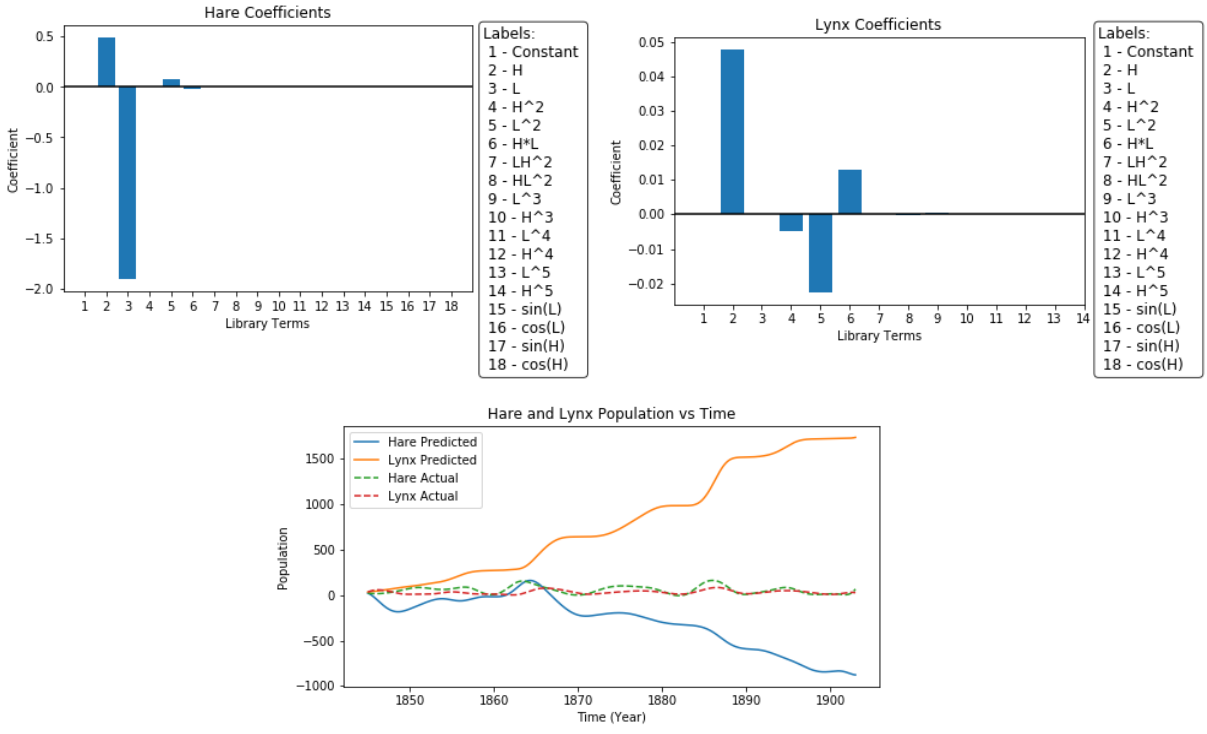

Figure 2: Model 1 for Hare and Lynx pelts

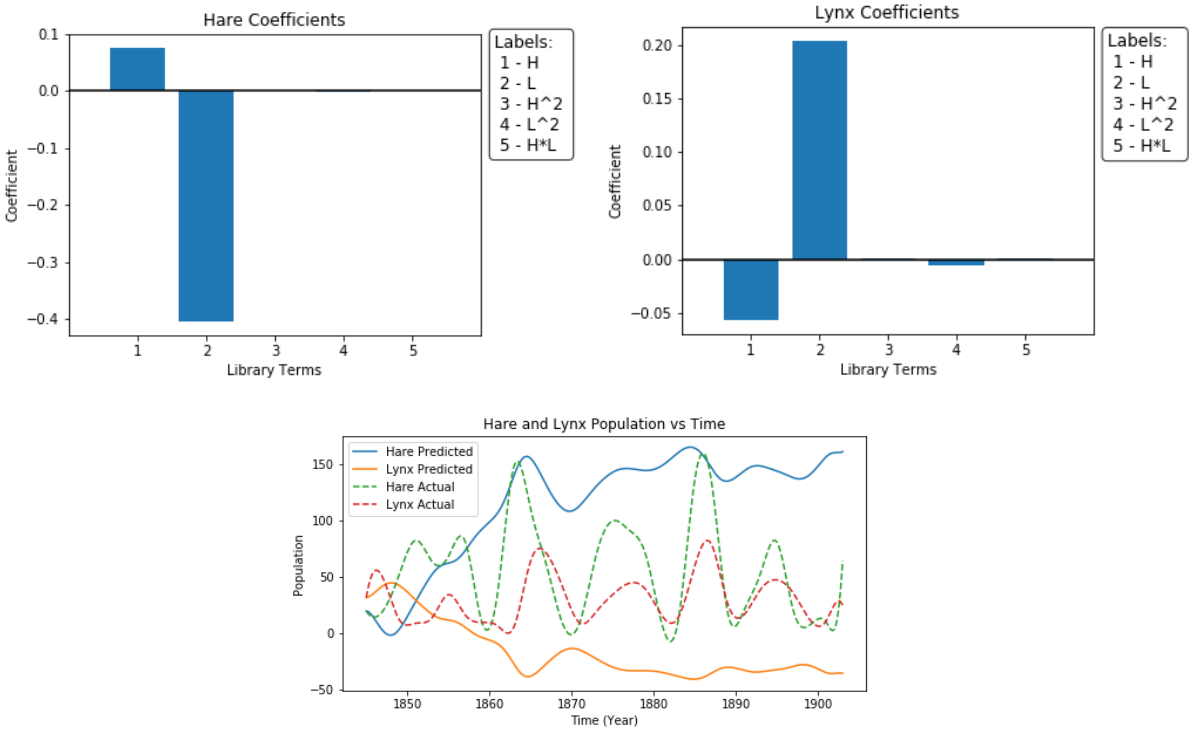

Figure 3: Model 2 for Hare and Lynx pelts

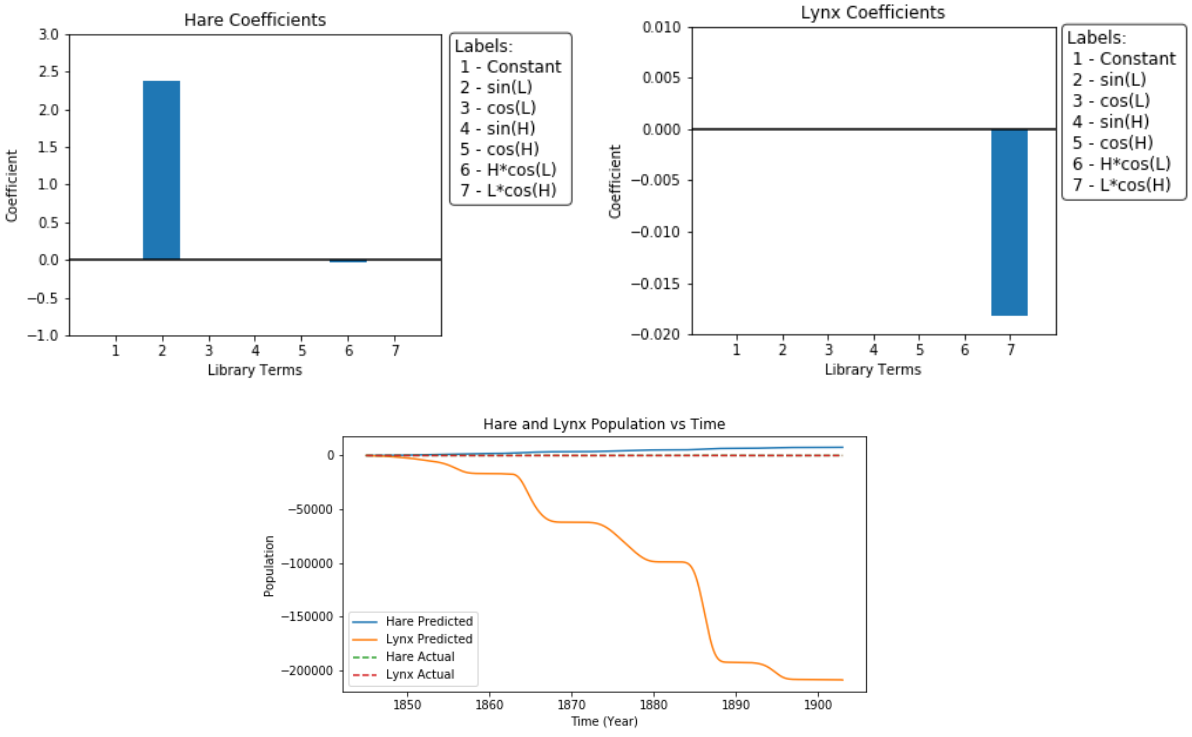

Figure 4: Model 3 for Hare and Lynx pelts

Within our three models, it is fairly common for our simulated results to blow up towards positive or negative infinity. However, re-regressing on previous models did help with preventing the blow up, as seen in the cycle from Model 1 to Model 2. In the case of Model 2, we were able to achieve an oscillatory nature to the solution but it does not follow the same pattern as the original hare and lynx data. As for the information criteria scores:

	Model 1	**Model 2**	**Model 3**
KL Divergence - Hare	∞	∞	∞
KL Divergence - Lynx	0.510	∞	∞
AIC - Hare	-18.421	-11.733	-27.861
AIC - Lynx	-33.354	-22.104	-52.554
BIC - Hare	-12190.168	-8845.528	-16910.054
BIC - Lynx	-13741.968	-8116.52	-23341.646

We see that among Model 2, it has the lowest AIC and BIC scores in terms of magnitude, which indicates the least amount of data lost compared to the other two. Then finally for our time delay embedding:

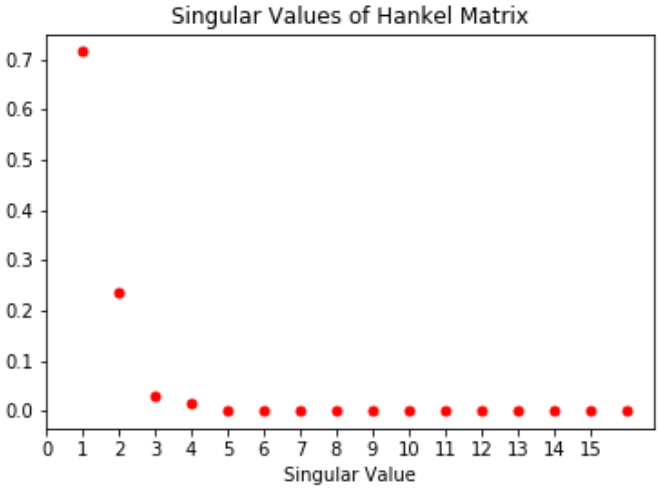

Figure 5: Singular values of our Hankel Matrix

We see that other than the first two singular points, we see that the third and fourth singular values are also non-zero, which indicates that there may be one or two latent variables within our system.

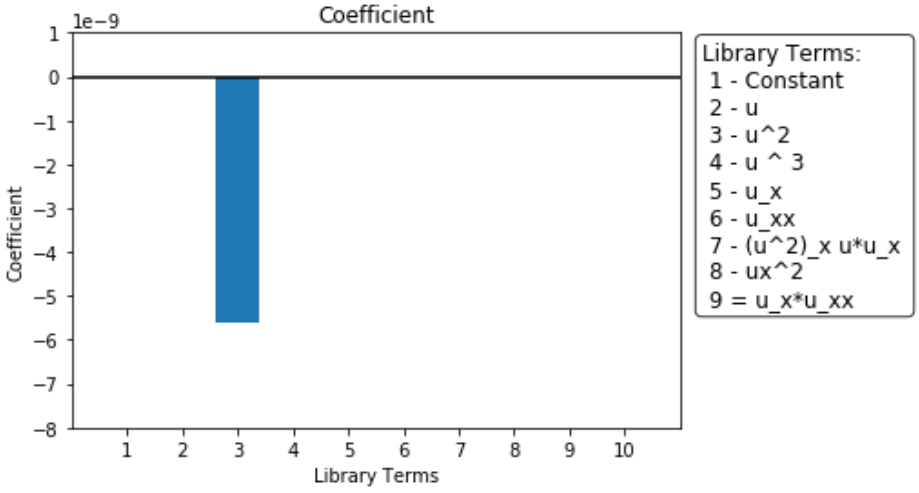

Figure 6: Coefficients of our Belousov-Zhabotinsky chemical oscillator movie data

We see here that all the coefficients are fairly close to zero. This might be due to the frame slice chosen not containing enough data, in which case we can experiment with more frames in a future experiment.

6 Summary and Conclusions

In model discovery, a few shortcomings are important to keep in mind. One being that noise can really throw off a solution. The second being that if the library used does not contain a function that correctly defines the system then is very hard to obtain accurate the results. We can see that despite trying a number of different libraries, it is still a little difficult to come with the true solution of the system. In the future, we can look into creating more robust and diverse libraries.

A Python Code - Part 1

```
# %%
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model as lm
import numpy.linalg as la

dat = loadmat('population.mat')
year = np.squeeze(dat['year'])
year = np.array([float(x) for x in year])
hare = np.squeeze(dat['hare'])
hare = np.array([float(x) for x in hare])
lynx = np.squeeze(dat['lynx'])
lynx = np.array([float(x) for x in lynx])
leng = len(hare)

#%% Plotting Data
from scipy.interpolate import UnivariateSpline

hs = UnivariateSpline(year, hare)
years = np.linspace(1845, 1903, 1000)
hares = hs(years)
ls = UnivariateSpline(year, lynx)
lynxs = ls(years)
leng = len(lynxs)

plt.figure(figsize=(10,6))
plt.plot(years, hares, 'c', linewidth=3)
plt.plot(years, lynxs, 'r', linewidth=3)
plt.plot(year, hare, 'g.', markersize=15)
plt.plot(year, lynx, 'm.', markersize=15)
plt.legend(['Hare Interpolation', 'Lynx Interpolation', 'Original Hare Data', 'Original
plt.xlabel('Time (Year)')
plt.ylabel('Population')
plt.title('Hare and Lynx Population vs Time')
plt.savefig('pop_fig.png')
plt.show()

#%% Using built in spline method
haredot = hs.derivative()(years)
lynxdot = ls.derivative()(years)
```

```
# %% Find derivative - Using Central Difference Schema
leng = len(hares)
haredot = np.zeros(leng)
lynxdot = np.zeros(leng)
dt = years[1] - years[0]
lynxdot[0] = (-3*lynxs[0] + 4*lynxs[1] - lynxs[2]) / (2*dt)
haredot[0] = (-3*hares[0] + 4*hares[1] - hares[2]) / (2*dt)
for k in (np.arange(leng)[1:-1]):
    haredot[k] = (hares[k+1] - hares[k-1]) / (2*dt)
    lynxdot[k] = (lynxs[k+1] - lynxs[k-1]) / (2*dt)

lynxdot[-1] = (-3*lynxs[-1] + 4*lynxs[-2] - lynxs[-3]) / (2*dt)
haredot[-1] = (-3*hares[-1] + 4*hares[-2] - hares[-3]) / (2*dt)

# %% Create Library for Model 1
##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################
A = np.transpose(np.array([hares/hares, hares, lynxs, hares ** 2, lynxs ** 2, hares * l
     (hares ** 2)* lynxs, (lynxs ** 2)*hares, lynxs** 3, hares ** 3, lynxs ** 4, hares
     np.sin(lynxs), np.cos(lynxs), np.sin(hares), np.cos(hares)]))

# %%  Solve
clf_h = lm.Lasso(alpha=1.5, max_iter=10000)
clf_l = lm.Lasso(alpha=1.5, max_iter=10000)
clf_h.fit(A, haredot)
clf_l.fit(A, lynxdot)

xhare = clf_h.coef_
xlynx = clf_l.coef_
bhare = clf_h.intercept_
blynx = clf_l.intercept_


# %% Plot Coefficients of Hares
labels = np.arange(len(xhare)) + 1
text = 'Labels: \n 1 - Constant \n 2 - H \n 3 - L \n 4 - H^2 \n 5 - L^2 \n 6 - H*L \n 7
props = props = dict(boxstyle='round', facecolor='white', alpha=0.8)

terms_num = len(labels)
plt.figure(figsize=(6,4))
plt.bar(labels, xhare)
```

```
plt.xticks(labels)
plt.title('Hare Coefficients')
plt.xlabel('Library Terms')
plt.ylabel('Coefficient')
plt.hlines(0, -1, 20)
plt.xlim(0,terms_num + 1)
plt.text(19.5, -2.80,text, fontsize=12, horizontalalignment='left', bbox = props)
#plt.tight_layout()
plt.savefig('hare_coeff.png', bbox_inches="tight")
plt.show()

# %% Plot Coefficients of Lynx
plt.figure(figsize=(6,4))
plt.bar(labels, xlynx)
plt.xticks(labels)
plt.title('Lynx Coefficients')
plt.xlabel('Library Terms')
plt.ylabel('Coefficient')
plt.hlines(0, -1, 20)
plt.xlim(0, 14)
plt.text(14.5, -.045,text, fontsize=12, horizontalalignment='left', bbox = props)
#plt.tight_layout()
plt.savefig('lynx_coeff.png', bbox_inches="tight")
plt.show()


# %% Recreate Data
dt = years[1] - years[0]
hare_der = xhare[2]*lynxs + xhare[1]*hares + bhare
lynx_der = xlynx[2]*lynxs + xlynx[1]*hares + xlynx[5]*hares*lynxs + blynx

hare_r = np.zeros(leng)
hare_r[0] = hares[0]

lynx_r = np.zeros(leng)
lynx_r[0] = lynxs[0]

for k in np.arange(leng-1):
    hare_r[k+1] = dt*hare_der[k] + hare_r[k]
    lynx_r[k+1] = dt*lynx_der[k] + lynx_r[k]

# %% Compare the Recreated data and original data
plt.figure(figsize=(8,4))
plt.plot(years, hare_r)
```

```
plt.plot(years, lynx_r)
plt.plot(years, hares, '--')
plt.plot(years, lynxs, '--')
plt.legend(['Hare Predicted', 'Lynx Predicted', 'Hare Actual', 'Lynx Actual' ])
plt.xlabel('Time (Year)')
plt.ylabel('Population')
plt.title('Hare and Lynx Population vs Time')
plt.savefig('pop_fig_recreate.png')
plt.show()

# %% Calculate KL Divergence and AIC/BIC
from scipy.stats import entropy as kl
kls_h = []
kls_l = []
AICs_h = []
AICs_l = []
BICs_h = []
BICs_l = []


klhare = kl(hares, hare_r) # = Inf
kllynx = kl(lynxs, lynx_r) # = Inf


kh = 3
sseh = np.sum((hares - hare_r) ** 2)
AIC_h = 2*kh - 2*np.log(sseh/leng)
BIC_h = kh*np.log(leng) - leng*np.log(sseh/leng)

kly = 4
ssel = np.sum((lynxs - lynx_r) ** 2)
AIC_l = 2*kly - 2*np.log(ssel)
BIC_l = kly*np.log(leng) - leng*np.log(ssel/leng)

kls_h.append(klhare)
kls_l.append(kllynx)
AICs_h.append(AIC_h)
AICs_l.append(AIC_l)
BICs_h.append(BIC_h)
BICs_l.append(BIC_l)


# %% Next Model %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
##########################################################################
```

```
#########################################################################
#########################################################################################
#########################################################################################
#########################################################################################

A2 = np.transpose(np.array([hares, lynxs, hares ** 2, lynxs ** 2, hares * lynxs]))

clf_h = lm.Lasso(alpha=1.5, max_iter=1000)
clf_l = lm.Lasso(alpha=1.5, max_iter=1000)
clf_h.fit(A2, haredot)
clf_l.fit(A2, lynxdot)

xhare2 = clf_h.coef_
xlynx2 = clf_l.coef_
bhare = clf_h.intercept_
blynx = clf_l.intercept_

# %% Plot Coefficients - Hare
labels = np.arange(len(xhare2)) + 1
text = 'Labels: \n 1 - H \n 2 - L \n 3 - H^2 \n 4 - L^2 \n 5 - H*L'
props = props = dict(boxstyle='round', facecolor='white', alpha=0.8)

plt.figure(figsize=(6,4))
plt.bar(labels, xhare2)
plt.xticks(labels)
plt.title('Hare Coefficients')
plt.xlabel('Library Terms')
plt.ylabel('Coefficient')
plt.hlines(0, -1, 20)
plt.xlim(0, 6)
plt.text(6.2, -0.105,text, fontsize=12, horizontalalignment='left', bbox = props)
plt.tight_layout()
plt.savefig('2hare_coeff.png')
plt.show()

# %% Plot Coefficients - Lynx
plt.figure(figsize=(6,4))
plt.bar(labels, xlynx2)
plt.xticks(labels)
plt.title('Lynx Coefficients')
plt.xlabel('Library Terms')
plt.ylabel('Coefficient')
plt.hlines(0, -1, 20)
plt.xlim(0, 6)
```

```
plt.text(6.2, .1,text, fontsize=12, horizontalalignment='left', bbox = props)
plt.tight_layout()
plt.savefig('2lynx_coeff.png')
plt.show()

# %% Recreate Data
dt = years[1] - years[0]
hare_der2 = xhare2[1]*lynxs + xhare2[0]*hares + bhare
lynx_der2 = xlynx2[1]*lynxs + xlynx2[0]*hares + blynx

hare_r2 = np.zeros(leng)
hare_r2[0] = hares[0]

lynx_r2 = np.zeros(leng)
lynx_r2[0] = lynxs[0]

for k in np.arange(leng-1):
    hare_r2[k+1] = dt*hare_der2[k] + hare_r2[k]
    lynx_r2[k+1] = dt*lynx_der2[k] + lynx_r2[k]

# %% Compare the Recreated data and original data
plt.figure(figsize=(8,4))
plt.plot(years, hare_r2)
plt.plot(years, lynx_r2)
plt.plot(years, hares, '--')
plt.plot(years, lynxs, '--')
plt.legend(['Hare Predicted', 'Lynx Predicted', 'Hare Actual', 'Lynx Actual' ])
plt.xlabel('Time (Year)')
plt.ylabel('Population')
plt.title('Hare and Lynx Population vs Time')
plt.savefig('2pop_fig_recreate.png')
plt.show()

# %%
from scipy.stats import entropy as kl

klhare = kl(hares, hare_r2)
kllynx = kl(lynxs, lynx_r2)

kh = 3
sseh = np.sum((hares - hare_r2) ** 2)
AIC_h = 2*kh - 2*np.log(sseh/leng)
BIC_h = kh*np.log(leng) - leng*np.log(sseh/leng)
```

```
kly = 4
ssel = np.sum((lynxs - lynx_r2) ** 2)
AIC_l = 2*kly - 2*np.log(ssel)
BIC_l = kly*np.log(leng) - leng*np.log(ssel/leng)

kls_h.append(klhare)
kls_l.append(kllynx)
AICs_h.append(AIC_h)
AICs_l.append(AIC_l)
BICs_h.append(BIC_h)
BICs_l.append(BIC_l)

# %%  Model 3
##############################################################################################
##############################################################################################
##############################################################################################
##############################################################################################
##############################################################################################

A3 = np.transpose(np.array([hares/hares, np.sin(lynxs),
     np.cos(lynxs), np.sin(hares), np.cos(hares), hares*np.cos(lynxs), lynxs*np.cos(har

clf_h = lm.Lasso(alpha=1)
clf_l = lm.Lasso(alpha=1)
clf_h.fit(A3, haredot)
clf_l.fit(A3, lynxdot)

xhare3 = clf_h.coef_
xlynx3 = clf_l.coef_
bhare = clf_h.intercept_
blynx = clf_l.intercept_


# %%
labels3 = np.arange(len(xhare3)) + 1
text = 'Labels: \n 1 - Constant \n 2 - sin(L) \n 3 - cos(L) \n 4 - sin(H) \n 5 - cos(H)
props = props = dict(boxstyle='round', facecolor='white', alpha=0.8)

plt.figure(figsize=(6,4))
plt.bar(labels3, xhare3)
plt.xticks(labels3)
plt.title('Hare Coefficients')
plt.xlabel('Library Terms')
plt.ylabel('Coefficient')
```

```
plt.hlines(0, -1, 20)
plt.xlim(0, len(xhare3)+1)
plt.ylim(-1, 3)
plt.text(8.3, 0.85,text, fontsize=12, horizontalalignment='left', bbox = props)
plt.tight_layout()
plt.savefig('3hare_coeff.png')
plt.show()
# %%
plt.figure(figsize=(6,4))
plt.bar(labels3, xlynx3)
plt.xticks(labels3)
plt.title('Lynx Coefficients')
plt.xlabel('Library Terms')
plt.ylabel('Coefficient')
plt.hlines(0, -1, 20)
plt.xlim(0, len(xhare3) + 1)
plt.ylim(-0.02, 0.01)
plt.text(8.25, -0.006,text, fontsize=12, horizontalalignment='left', bbox = props)
plt.tight_layout()
plt.savefig('3lynx_coeff.png')
plt.show()

# %% Recreate Data
dt = years[1] - years[0]
hare_der3 = xhare3[1]*A[:,1] + bhare
lynx_der3 = xlynx3[6]*A[:,6] + xlynx3[5]*A[:,5] + blynx

hare_r3 = np.zeros(leng)
hare_r3[0] = hares[0]

lynx_r3 = np.zeros(leng)
lynx_r3[0] = lynxs[0]

for k in np.arange(leng-1):
    hare_r3[k+1] = dt*hare_der3[k] + hare_r3[k]
    lynx_r3[k+1] = dt*lynx_der3[k] + lynx_r3[k]

# %% Compare the Recreated data and original data
plt.figure(figsize=(8,4))
plt.plot(years, hare_r3)
plt.plot(years, lynx_r3)
plt.plot(years, hares, '--')
plt.plot(years, lynxs, '--')
plt.legend(['Hare Predicted', 'Lynx Predicted', 'Hare Actual', 'Lynx Actual' ])
```

```
plt.xlabel('Time (Year)')
plt.ylabel('Population')
plt.title('Hare and Lynx Population vs Time')
plt.savefig('3pop_fig_recreate.png')
plt.show()

# %%
from scipy.stats import entropy as kl

klhare = kl(hares, hare_r3)
kllynx = kl(lynxs, lynx_r3)

kh = 3
sseh = np.sum((hares - hare_r3) ** 2)
AIC_h = 2*kh - 2*np.log(sseh/leng)
BIC_h = kh*np.log(leng) - leng*np.log(sseh/leng)

kly = 4
ssel = np.sum((lynxs - lynx_r3) ** 2)
AIC_l = 2*kly - 2*np.log(ssel)
BIC_l = kly*np.log(leng) - leng*np.log(ssel/leng)

kls_h.append(klhare)
kls_l.append(kllynx)
AICs_h.append(AIC_h)
AICs_l.append(AIC_l)
BICs_h.append(BIC_h)
BICs_l.append(BIC_l)

#%% Time embedd
time_em = 8
H1 = np.zeros([time_em*2, 500])
k = 0
startt = 0
stopp = 500
for n in np.arange(time_em):
    H1[k, :] = hares[startt:stopp]
    H1[k+1, :] = lynxs[startt:stopp]
    k = k+2
    startt = startt+1
    stopp = stopp+1


u,s,vh = la.svd(H1, full_matrices=False)
```

```
#%%
plt.plot(np.arange(time_em*2) + 1, s/np.sum(s), 'r.', markersize=10)
plt.xticks(np.arange(time_em*2))
plt.title('Singular Values of Hankel Matrix')
plt.xlabel('Singular Value')
plt.savefig('hsv.png')
plt.show()
```

B Python Code - Part 2

```
# %%
from scipy.io import loadmat
import numpy as np

dat = loadmat('BZ_reformat.mat')
BZ_original = np.array(dat['BZ_tensor'])

BZx = BZ_original[150:200, 190:240, :];

[x,y,time] = np.shape(BZx)
dt = 1;
dx = 1;
m = x*y
BZr = np.zeros([x*y, time])
for t in np.arange(time):
    BZr[:,t] = np.reshape(BZx[:,:,t], (x*y,))

# %%
BZdot = np.zeros(np.shape(BZx))
for i in np.arange(x):
    for j in np.arange(y):
        for k in np.arange(time)[1:-1]:
            BZdot[i,j,k] = (BZx[i,j,k+1] - BZx[i,j,k-1]) / (2*dt)



#%%
BZrdot = np.zeros(np.shape(BZr))
for t in np.arange(time)[1:-1]:
    for n in np.arange(x*y):
```

```
        BZrdot[n, t] = (BZr[n, t+1] - BZr[n, t-1]) / (2*dt)

BZrdot = BZrdot[:, 1:-1]

# %% Create Derivatives
m = x*y;
D = np.zeros([m,m])
D2 = np.zeros([m,m])

for j in np.arange(m-1):
  D[j,j+1]=1;
  D[j+1,j]=-1;

  D2[j,j+1]=1;
  D2[j+1,j]=1;
  D2[j,j]=-2;


D[m-1,1]=1
D[1,m-1] = -1

D2[m-1,m-1]=-2;
D2[m-1,1]=1;
D2[1,m-1]=1;
D2 = D2/( dx ** 2);

# %%
u = np.reshape(np.transpose(BZr[:,1:-1]),[(time-2)*m,1] );

#%%
ux = np.zeros(np.shape(np.transpose(BZr)))
uxx = np.zeros(np.shape(np.transpose(BZr)))
u2x = np.zeros(np.shape(np.transpose(BZr)))

for jj in np.arange(2, time-1):
   ux[jj-1,:] = D@BZr[:,jj]  # u_x
   uxx[jj-1,:] = D2@BZr[:,jj]  # u_xx
   u2x[jj-1,] = D@(BZr[:,jj] ** 2)   # (u^2)_x

# %%
ux1 = ux[1:-1, :]
uxx1 = uxx[1:-1, :]
u2x1 = u2x[1:-1, :]
```

```
# %%
Ux = np.reshape(ux1,[(time-2)*m,1])
Uxx = np.reshape(uxx1,[(time-2)*m,1])
U2x = np.reshape(u2x1,[(time-2)*m,1])

# %%
A= np.transpose(np.squeeze(np.array([u/u, u, u ** 2, u ** 3, Ux, Uxx, U2x, Ux*u, Ux*Ux,



# %%
Udot = np.reshape(BZrdot, [(time-2)*m, 1])


# %%
from sklearn import linear_model as lm
clf = lm.Lasso(alpha=0.8)
clf.fit(A, Udot)
coef = clf.coef_
b = clf.intercept_


# %%
import matplotlib.pyplot as plt

text = 'Library Terms: \n 1 - Constant \n 2 - u \n 3 - u^2 \n 4 - u ^ 3 \n 5 - u_x \n 6
props = props = dict(boxstyle='round', facecolor='white', alpha=0.8)

labels =np.arange(len(coef))
plt.bar(labels, coef)
plt.ylim(-8e-9,1e-9)
plt.hlines(0, -1, len(coef)+1)
plt.xlabel('Library Terms')
plt.ylabel('Coefficient')
plt.xticks(labels + 1)
plt.xlim(0,len(coef) + 1)
plt.title('Coefficient')
plt.text(11.4, -5.2e-9,text, fontsize=12, horizontalalignment='left', bbox = props)
plt.savefig('lt.png', bbox_inches="tight")
plt.show()
```