HW2: SINDy and You

Alexander Van Roijen

May 4, 2019

I Introduction and Overview

In attempts to understand the inner workings of real life systems, we gather data to visualize the process. However, this wont tell us the governing equations of said system. I explored two datasets and how we can use Sparse Identification of Nonlinear Dynamics (SINDy), leveraging particular python library packages [1],[2], to determine their form. First, we will attempt to find governing equations to a small predator prey dataset on snowshoe hares and lynx. We will then explore a data rich source of image files we will refer to as BZ. In both cases, forecasting will prove difficult, but we will identify significant predictors in both systems.

II Theoretical Background

We are attempting to solve a problem of the form $Ax = b$, where our x and A will approximate the equations of our system. In particular, we are looking to find the driving equations for a 2 dimensional ODE problem for the predator prey dataset, and the n dimensional PDE problem for the video source.

In this problem set, we only have the measurements from our system to work with. In order to determine equation predictors, we seek to find their **coefficients** and **transformations**. So our A will represent the collection transformations of the measurements.Consider measurements z and y, we create an A with z^2, zy, $\frac{1}{z}$, and so on. Our b will be represented in one of two ways. Either $b = \langle z_2, z_3 ... z_n \rangle$ or $b = \dot{z}$. This means we either want to find equations such that $f(z_n) = z_{n+1}$ or $f(z_n) = \dot{z}_n$. Finally, our x represents the coefficients of our system $\langle 0, 1.2, -1, ... \rangle$ such that we minimize some loss between our transformed data and $\dot{z}$.

Now since we can see this is an $Ax = b$ problem, and want to find some x, we can use convex methods including Lasso and ridge regression. However, we will promote sparsity

as most models in reality follow some finite set of variables. Consequently, we will be using a combination of lasso regression and thresholds to create better understood models.

II.I Evaluating our predatory prey models

We can compute many different scores to determine the accuracy of our system. However, we will be using **AIC**, **BIC**, and **KL Divergence** [3]. Below are the listed formulas.

- $AIC = 2K - 2\log(\mathcal{L}(\hat{\mu}|x))$
- $BIC = \log(n)K - 2\log(\mathcal{L}(\hat{\mu}|x))$
- Kullback-Leibler(KL) divergence
  $\int f(X,\beta)\log(\frac{f(X,\beta)}{g(X,\mu)})$
  or
  $\sum_{i=1}^{n} f(x_i,\beta)\log(\frac{f(x_i,\beta)}{g(x_i,\mu)})$

In these equations, our K is the number of predictors in our model β. For KL divergence, it is assumed we have the true distribution of our data, and can compute the score exactly. In practice we cannot, so we approximate this as best we can with our data. Meanwhile, AIC and BIC do not assume this and lets us approximate the value using maximum likelihood estimation given our data and β. The relative lower scores are better, which makes it useful only when comparing different models rather than some absolute measure such as classification error. Finally, we are assuming so far that our data contains all the information necessary to accurately model the system. Consequently we will look to **Singular Value Decomposition(SVD)** on a **Hankel Matrix** to determine latent variables we may be missing

II.II SVD and latent variables

We understand that SVD expresses the ability of a matrix to stretch and rotate a set of vectors. In general it is of form $A = U\Sigma V^T$ where U and V are unitary transformations and Σ represents the stretching. When studying our hankel Matrix, we want to do the same decomposition.
If we preform SVD on this matrix, we are asking how we can explain the variation of our data over time through this decomposition of stretching and rotation. The diagonals of our Σ tell us the amount of variation achieved per variable, thus giving us a method to determine how many variables we should be looking for in our ultimate equation $f(z)$

Table 1: 2 rows of example Hankel Matrix

SS_t	$Lynx_t$	SS_{t+1}	$Lynx_{t+1}$	...
20	32	17.59	41.77	...
17.59	41.77	15.12	42.3	...

III Algorithm Implementation and Development

For our predator prey problem, we are looking for $f(SS_n, w_n) = SS_{n+1}$ and $g(L_n, w_n) = L_{n+1}$ where SS and L are our measurements of hares and lynx, and w is our transformation vector. Our original dataset is only of size $n = 30$, so we will use cubic spline interpolation from scipy[4] to smooth out our data to $n' = 720$ and make our calculations more feasible.

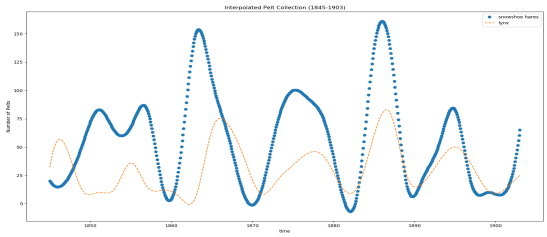


Figure 1: cubic interpolated data on a monthly level

A mixture of model selection schemes were used to generate three separate models. The models fit to this data used sklearn's lasso [1] with various penalties and my own thresholding limit set to ≥ 0.0002. The motivation for promoting a threshold and sparsity is to prevent our models predictions from exploding to either ∞ when forecasting and gain a better understanding of their predictors. These models then have their KL, AIC, and BIC scores calculated and compared. In better detail, the scores were computed as follows

- $AIC = n \log \sigma + 2K \quad , \quad BIC = n \log \sigma + K \log(n)$ with $\sigma = \frac{RSS}{n}$ with $RSS = ||X^T \beta - y||_2$
- KL divergence= $\sum_{i=1}^{n} f(x_i, \beta) \log(\frac{f(x_i,\beta)}{g(x_i,\mu)})$

It is important to note that the AIC and BIC scores are now calculated assuming our data follows a normal distribution. This is not always the case, but due to the law of large numbers and lack of insight on the data, I will assume this is the case for evaluation purposes. Second, KL divergence is calculated knowing the true probability of our distributions f and g. However, since we do not know these, we will approximate them using a simple histogram approach highlighted in the algorithm below.

Algorithm 1: Calculate means of true data SS : μ_{SS} and $Lynx$: μ_{Lynx} and put the datapoints in bins as follows
$(SS_n, Lynx_n) \in$ bin 1 if $SS_n < \mu_{SS}, Lynx_n < \mu_{Lynx}$
$(SS_n, Lynx_n) \in$ bin 2 if $SS_n < \mu_{SS}, Lynx_n \geq \mu_{Lynx}$
$(SS_n, Lynx_n) \in$ bin 3 if $SS_n \geq \mu_{SS}, Lynx_n < \mu_{Lynx}$
$(SS_n, Lynx_n) \in$ bin 4 if $SS_n \geq \mu_{SS}, Lynx_n \geq \mu_{Lynx}$
We then divide by n to determine their relative probabilities.

Thus to calculate the KL divergence for our models we use the same partitioning scheme

with the same means to create these probabilities and divide by those of our true model. Lastly, the SVD on the Hankel Matrix described earlier will be accomplished using numpy's svd method[5]

Finally,I will be exploring the potential PDE space driving the change in pixel values of the BZ video data. However,due to computational complexity (1200 shots of 451x351 images), I only look at the first column of data and how it changes over time. To evaluate the performance of my models, I use a combination of **Mean Squared Error (MSE)** and baseline prediction heuristics including one step delay evaluation. To calculate derivatives for both our library and $\hat{z}$, I use numpy's gradient library[2], which implements a finite difference method, with only one direction difference on the end points.

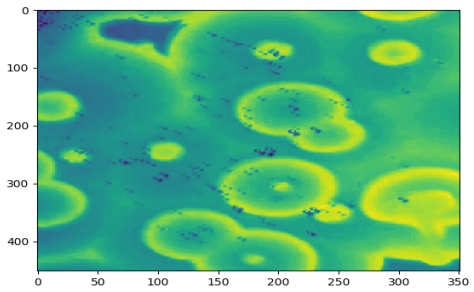


Figure 2: plot of first (451x351) frame

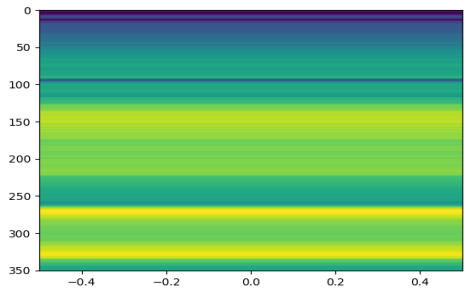


Figure 3: imshow plot of a single 351x1 slice

IV Computational Results

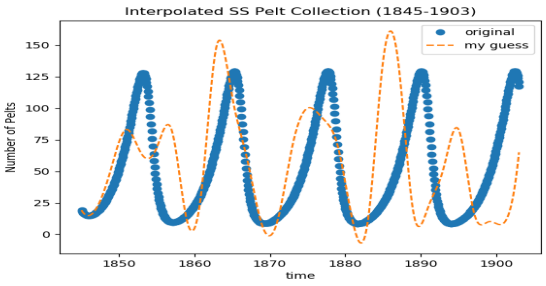
IV.I Predator Prey

Threshold	Penalty	Model	KL	AIC	BIC	SS #	Lynx#	MN
False	1	Lasso	2.7251	6736.586	6772.9492	8	6	1
False	0.0001	Lasso	2.6527	6175.5940	6302.8637	28	28	2
True	0.1	Lasso	0.1571	5677.0050	5695.1863	4	3	3

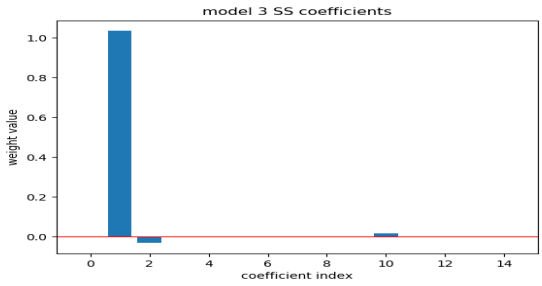
Table 2: Table includes pertinent parameters of model performance, Model Number (MN), and number of coefficients used(SS#,Lynx#). The boolean Threshold indicates whether the thresholding level of 0.0002 was used.

Model performance can be found in table 2 above. The most striking result is that the third model achieves the best performance despite it containing the least parameters. However, the models were computed with different libraries and penalties, thus changing their coefficients and interactions. We can find in the figures below the forecasts of model

3 and its coefficients. Find a table in the appendix highlighting parameters used in each models respective library.



Model 3 SS predictions vs actual



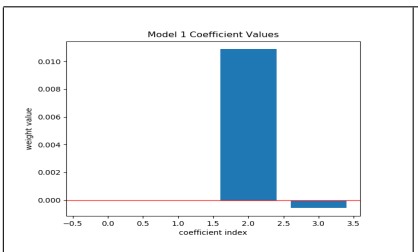
Model 3 SS predictor coefficients

IV.II BZ video data

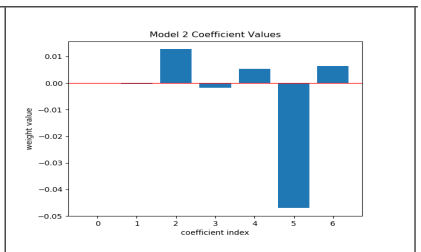
Base Model	Penalty	Model	MSE_P	MSE_D	Coef #	Lib Size	Model Number
True	N/A	N/A	1.6562	0.718	N/A	N/A	N/A
False	0.00001	Lasso	299.301	0.841	3	4	1
False	0.00001	Lasso	508.232	0.843	6	7	2

Table 3: Comparative results of a small and large model on our data. *MSEP* is the Mean Squared Error of Predicitions, while *MSED* is the Mean Squared Error of Derivatives. The boolean *basemodel* indicates if we simply guessed the previous derivative or value as the next iterations value

From table 3 above, we can tell that our attempted model fits did not succeed greatly. However, this is to be expected, as we often want to predict more than one time point ahead at a time which can not be done with a base previous value guess model. Further, when looking at the figures below, we can see that the smaller model has small coefficient values that cause our model to change pixel values less aggressively as our larger model.



Model 1 coefficient values.



Model 2 coefficient values.

Furthermore we can explain the difference in model performance by looking at time slices of the column. Our smaller model has little to no impact between the beginning and end frame. This also explains the higher MSE score of our larger model as it is likely trending in a particular direction that the smaller model does not explore.

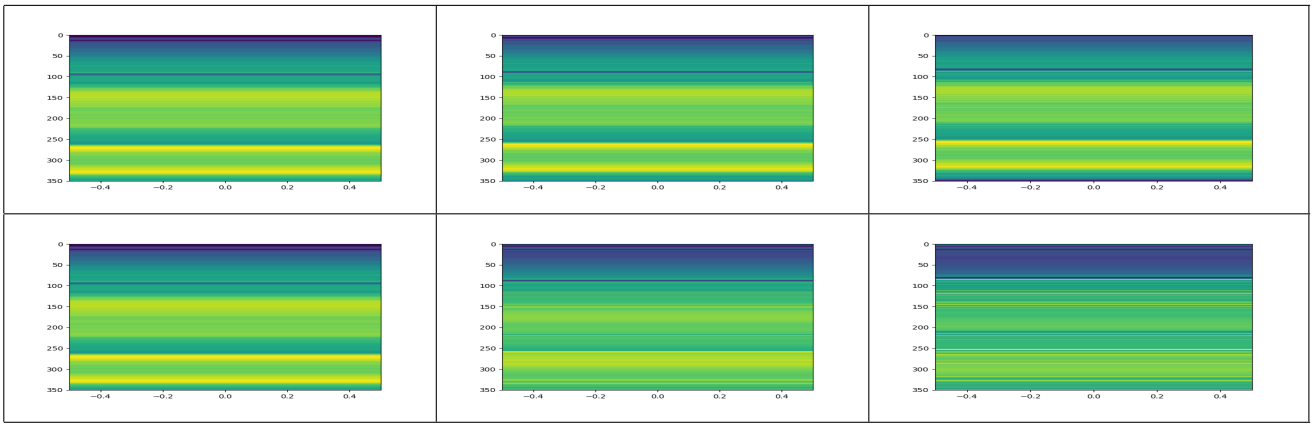


Table 4: The top row shows us the progression of our small models predictions at time slice 0, 600, and 1200. The bottom show the progression of our larger model

V Discussions and Summary

Looking at both our attempts to solve the driving equations of a PDE and ODE, we find that our models arent great future state predictors. This is partially due to us extrapolating data and lacking expertise in the data domains. Fortunately, we have learned what transformations on our data appear to have some predictive prowess. We can reference the appendix to see that for the predator prey problem, both SS_n and $Lynx_n$ populations are both strongly determined by SS_{n-1},$Lynx_{n-1}$, usually with opposite signs.Intuitively, we can understand how these populations would fluctuate around each other. Looking at the results of our SVD below, we can see an elbow occurs around two predictors, again agreeing with the two parameters all our models deem important.

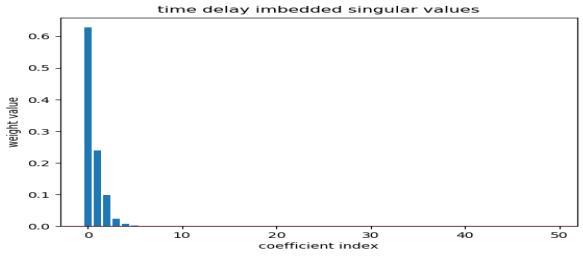


Figure 4: diagonal components of SVD decomposition of a 50 time step delay embedded Hankel matrix

Meanwhile, for our video data, $\frac{dv}{dt}$ is some combination of $\frac{dv}{ds}$ and $\sin(\frac{dv}{ds})$ where $\frac{dv}{dt}$ is the gradient of our pixel values over time and $\frac{dv}{ds}$ is the gradient of our pixel values in space. Overall, we can see that for noisy and potentially nonlinear systems, we can use SINDy to gauge important factors in our system of equations despite difficulties in developing accurate forecasters.

VI Future Work

In the future, I hope to be able ot use this with less brute force trial and error, and implement smarter schemes to find driving equations faster. I can see this being very useful for many different applications, but I found that the penaltys an choice of libraries are quite important in determining good forecasters.

VII References

- 1: Scikit-learn. Lasso. May 1, 2019. Electronic. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
- 2: Numpy. Gradient. May 2, 2019. Electronic. https://docs.scipy.org/doc/numpy/reference/generated/numpy.gradient.html#numpy-gradient
- 3: MathWorks. AIC and BIC calculation May 1, 2019. Electronic. https://www.mathworks.com/matlabcentral/answers/195674-neu ral-network-aic-and-bic-calculation-number-of-parameters
- 4: Scipy. interp1d. April 28, 2019. Electronic. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html
- 5: Numpy. SVD. April 30, 2019. Electronic. https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.svd.html

VIII Results Appendix

More code,images, and results can be found on my github.

Model Number	Dataset	Library
1	Predator vs Prey	see code appendix **simpleLibrary**
2	Predator vs Prey	see code appendix **simpleLibrary**
3	Predator vs Prey	see code appendix **getLibrary**
1	BZ Video Data	$c, v, \frac{dv}{ds}, \frac{d^2v}{ds^2}$
2	BZ Video Data	$c, v, \frac{dv}{ds}, \frac{d^2v}{ds^2}, \sin(v), \sin(\frac{dv}{ds}), \sin(\frac{d^2v}{ds^2})$

Table 5: Highlights the dataset and library used to determine predictors of our models. c is a constant and v represents the pixel value between 0 and 255.

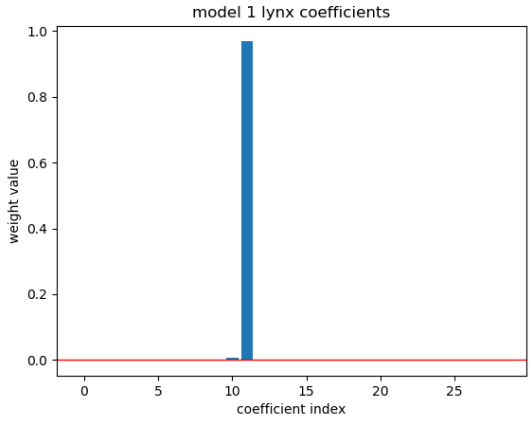


Figure 5: first model coefficient values. Note that index 11 is the previous lynx value.

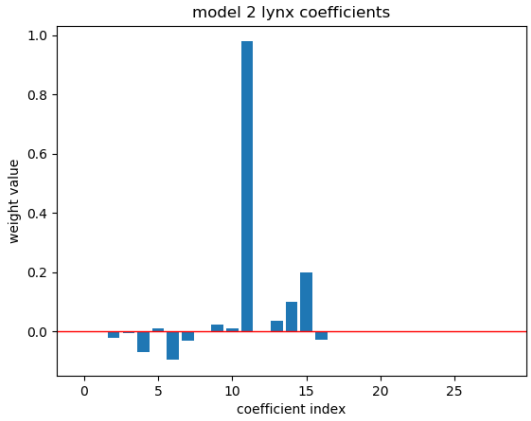


Figure 6: second model coefficient values. Note that index 11 is the previous lynx value. Further, note the many non zero terms due to the low penalty

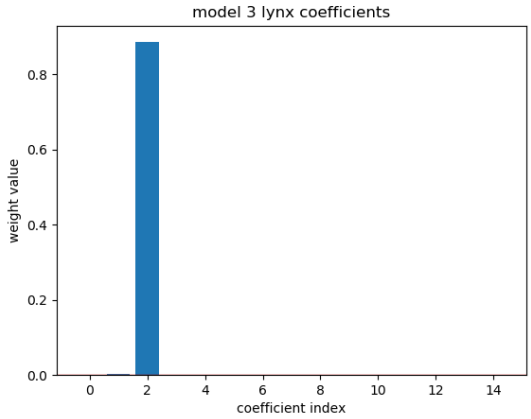


Figure 7: third model coefficient values. Note that index 2 is the previous lynx value.

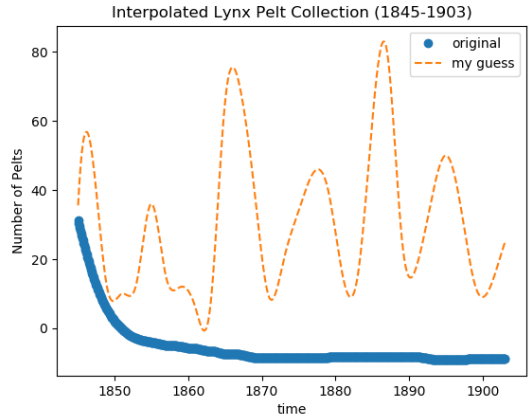


Figure 8: first model forecasting lynx populations over time

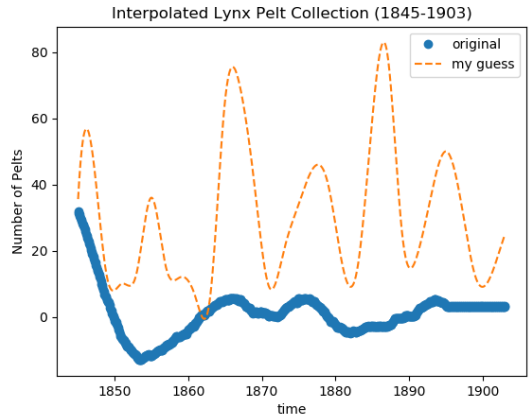


Figure 9: Second model forecasting lynx populations over time

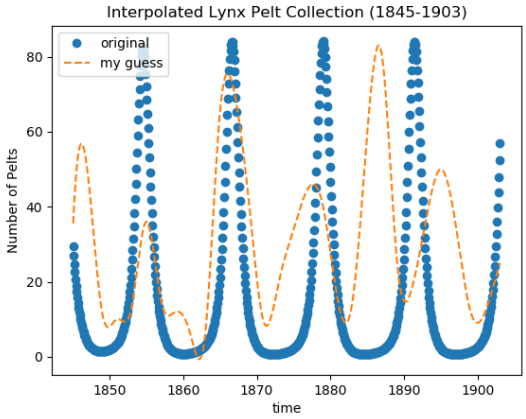


Figure 10: third model forecasting lynx populations over time

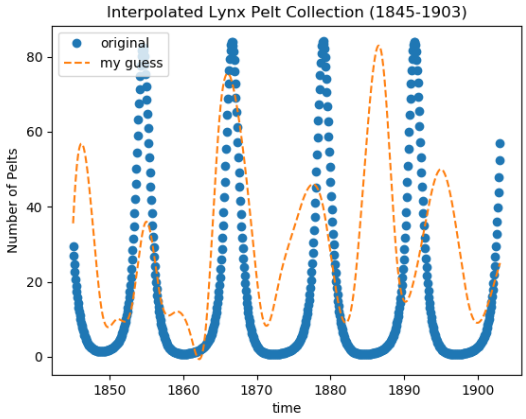


Figure 11: third model forecasting lynx populations over time

IX Code Appendix

All library functions used to achieve results listed have been brought to the top of the verbatim code below to prevent digging through the code.

IX.I predatorVPrey.py

```
import numpy as np
import struct
import cvxpy as cp
import os
```

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.dates as mdates
import matplotlib.lines as lines
import matplotlib.cbook as cbook
import cvxopt as cx
from cvxopt.modeling import op, dot, variable
from sklearn.linear_model import Ridge
from scipy.sparse.linalg import lsmr
from sklearn import linear_model
from decimal import Decimal
from scipy.interpolate import interp1d
import h5py
from scipy import io

'''
useful links
https://www.youtube.com/watch?v=gSCa78TIldg the meat of this assignment
https://www.pnas.org/content/113/15/3932
https://mathinsight.org/ordinary_differential_equation_introduction for refreshing ODE
https://imedea.uib-csic.es/master/cambioglobal/Modulo_V_cod101615/Theory/TSA_theory_pa
'''
YI = 0
SHPI = 1
CLPI = 2

filePath = '/home/bdvr/Documents/Github/AMATH563/hw2/'
interps = [(1,1),(2,3),(3,7)] # represent every year, every 6 months, and every 3 mont

def simpleLibrary(y1,y2,single=False):
'''
creates the library for models 1 and 2
'''
types = ['const','y1','y2',]
if(single):
numRows=1
else:
numRows = len(y1)

numCols = 29
lib = np.ones([numRows,numCols])
lib[:,1]= np.sin(y1)
lib[:,2]= np.sin(y2)
lib[:,3] = np.sin(y1)*np.cos(y2)
```

```
lib[:,4] = np.cos(y1)
lib[:,5] = np.cos(y2)
lib[:,6] = np.sin(y2**2)*np.cos(y1)
lib[:,7] = np.sin(y1**2)*np.cos(y2)
lib[:,8] = np.sin(y2)*np.cos(y1**2)
lib[:,9] = np.sin(y1)*np.cos(y2**2)
lib[:,10] = y1
lib[:,11] = y2
lib[:,12] = y1*y2
lib[:,13] = np.sin(y1*y2)
lib[:,14] = np.sin(y1+y2)
lib[:,15] = np.cos(y1*y2)
lib[:,16] = np.cos(y1+y2)
lib[:,17] = y1*np.sin(y1)
lib[:,18] = y1*np.cos(y1)
lib[:,19] = y2*np.sin(y1)
lib[:,20] = y2*np.cos(y1)
lib[:,21] = y2*np.sin(y2)
lib[:,22] = y2*np.cos(y2)
lib[:,23] = y1*np.sin(y2)
lib[:,24] = y1*np.cos(y2)
lib[:,25] = (y1+y2)*np.sin(y1)
lib[:,26] = (y1+y2)*np.cos(y1)
lib[:,27] = (y1-y2)*np.sin(y1)
lib[:,28] = (y1-y2)*np.cos(y1)

return lib

def getLibrary(y1,y2,single=False):
'''
creates the library for model 2
'''
types = ['const','y1','y2',]
if(single):
numRows=1
else:
numRows = len(y1)
numCols = 15
lib = np.ones([numRows,numCols])
lib[:,0]#nothing as its our constant column
lib[:,1] = y1
lib[:,2] = y2
lib[:,3]= y1*y1
lib[:,4]= y2*y2
```

```
lib[:,5] = y1*y2
lib[:,6] = y1*y1*y2
lib[:,7] = y2*y2*y1
lib[:,8] = np.sin(y1)
lib[:,9] = np.cos(y1)
lib[:,10] = np.sin(y2)
lib[:,11] = np.cos(y2)
lib[:,12] = np.cos(y2)*np.sin(y1)
lib[:,13] = np.sin(y2)*np.cos(y1)
lib[:,14] = y1*y1*y1

return lib


def splineInterpolation(xs,level=0,plotSplines = False,title='peltCounts',clearFigure='
'''
does cubic spline interpolation for our problem with various levels of granularity
'''
size = len(xs)
y = xs
#f = interp1d(x, y)
allfactors = [2,4,8,24,730]
factor=allfactors[level]
#print(size)
if(test is False):
x = np.linspace(1845, 1903, num=size, endpoint=True)
xnew = np.linspace(1845, 1903, num=factor*size-(factor-1), endpoint=True)
else:
x = np.linspace(-5, 5, num=size, endpoint=True)
xnew = np.linspace(-5, 5, num=factor*size-(factor-1), endpoint=True)
f2 = interp1d(x, y, kind='cubic')

if(clearFigure):
plt.clf()
plt.plot(x, y, 'o', xnew, f2(xnew), '--')
plt.title('Interpolated Pelt Collection (1845-1903)')
plt.legend(['original', 'interpolated (cubic)'], loc='best')
if(plotSplines is True):
plt.show()
else:
plt.savefig(saveLoc)
return xnew,f2(xnew)

def createPointPairs(data1,data2,mean1,mean2):
```

```
'''
implements algorithm 1 highlighted in paper
'''
size = len(data1)
counter = 0
bins=[1,1,1,1]
while(counter<size):
p1 = data1[counter]
p2 = data2[counter]
if(p1>mean1 and p2>mean2):
bins[0]+=1
elif(p1<=mean1 and p2>mean2):
bins[1]+=1
elif(p1>mean1 and p2<=mean2):
bins[2]+=1
elif(p1<=mean1 and p2<=mean2):
bins[3]+=1
counter += 1
bins = np.array(bins)
return (bins/size)

def linearInterpolation(thisPoint,nextPoint):
curYear = thisPoint[YI]
nextYear = nextPoint[YI]-1
startSHP = thisPoint[SHPI]
startCLP = thisPoint[CLPI]
yearDiff = nextPoint[YI] - thisPoint[YI]
shpDelta = ((nextPoint[SHPI] - thisPoint[SHPI])/yearDiff)
clpDelta = ((nextPoint[CLPI] - thisPoint[CLPI])/yearDiff)
newPoints=[]
counter=1
while(curYear<nextYear):
curYear+=1
newSHP = startSHP + shpDelta*counter
newCLP = startCLP + clpDelta*counter
newPoints.append([curYear,newSHP,newCLP])
counter+=1
return np.array(newPoints)

testData = np.array([[2000,1,2],[2003,7,11],[2006,11,2],[2009,5,11]])
actualData = np.genfromtxt('peltData.csv',delimiter=',')
def interpolateData(data):
numEntries = data.shape[0]
counter = 0
```

```
Times=[]
SHP=[]
CLP=[]
while(counter < numEntries-1):
thisDataPoint = data[counter]
nextDataPoint= data[counter+1]
allDelta = linearInterpolation(thisDataPoint,nextDataPoint)
counter+=1

def simpleLibrary2(y1,y2,single=False):
types = ['const','y1','y2',]
if(single):
numRows=1
else:
numRows = len(y1)

numCols = 22
lib = np.ones([numRows,numCols])
lib[:,1]= np.sin(y1)
lib[:,2]= np.sin(y2)
lib[:,3] = np.sin(y1)*np.cos(y2)
lib[:,4] = np.cos(y1)
lib[:,5] = np.cos(y2)
lib[:,6] = np.sin(y2**2)*np.cos(y1)
lib[:,7] = np.sin(y1**2)*np.cos(y2)
lib[:,8] = np.sin(y2)*np.cos(y1**2)
lib[:,9] = np.sin(y1)*np.cos(y2**2)
lib[:,10] = y1
lib[:,11] = y2
lib[:,12] = y1*y2
lib[:,13] = (y1**2)*y2
lib[:,14] = y2*y2
lib[:,15] = y1*y1
lib[:,16] = y1**3
lib[:,17] = y2**3
lib[:,18] = y1**4
lib[:,19] = y2**4
lib[:,20] = y1**5
lib[:,21] = y2**5

return lib


'''
```

```
try init sparse of 0.2, and then reregres son 0.5
'''
def getLibraryTest(y1,single=False):
types = ['const','y1','y2',]
if(single):
numRows=1
else:
numRows = len(y1)
lib = np.ones([numRows,4])
lib[:,0] = np.sqrt(y1)#nothing as its our constant column
lib[:,1] = y1
lib[:,2] = y1**2
lib[:,3]= y1**3
return lib

def getFormulaTest(derivs,data,penalty,iters):
A = getLibraryTest(data)
b=derivs
if(iters>0):
clf = linear_model.Lasso(alpha=penalty,max_iter=iters)
else:
clf = linear_model.Lasso(alpha=penalty)
clf.fit(A,np.reshape(derivs,(-1,1)))
return (clf.coef_),A

def smallTest():
#true func is x^2 with deriv 2x.
xs=np.arange(-5,6)
ys = xs**2
interpxs,interpolatedDatay = splineInterpolation(ys,3,True,test=True)
derivs = calcDiff(interpolatedDatay,np.diff(interpxs))
penalty = 0.001
print(np.diff(interpxs)[0])
coefs,SA = getFormulaTest(derivs,interpolatedDatay[1:-1],penalty,100000)
print(coefs)

def getFormula2(derivs,incomys,otherys,penalty=0.1,iters=0):
A = getLibrary(incomys,otherys)
#print(A)
#b=np.reshape(derivs,(derivs.shape[0],1))
b=derivs
reg = linear_model.LinearRegression()
reg.fit(A,b)
return (reg.coef_),A
```

```
def getFormula(derivs,incomys,otherys,penalty=0.1,iters=0):
A = getLibrary(incomys,otherys)
#print(A)
b=derivs
if(iters>0):
clf = linear_model.Lasso(alpha=penalty,max_iter=iters)
else:
clf = linear_model.Lasso(alpha=penalty)
clf.fit(A,b)
return (clf.coef_),A

def calcVals(SS,Lynx,time,coefSS,coefLynx,derivSS,derivLynx):
counter = 0
duration=len(time)
repeatSize = len(coefSS)
deltaSS=0
deltaLynx=0
mySS=SS[1]
myLynx = Lynx[1]
currentSS = SS[1]
currentLynx = Lynx[1]
myTimes = np.zeros([duration,5])
myDerivs =  np.zeros([duration,5])
stepSize = time[1]-time[0]
while(counter<duration):

toCoefSS = getLibrary(mySS,myLynx,True)
toCoefLynx = getLibrary(myLynx,mySS,True)

deltaSS = toCoefSS.dot(coefSS)
actualSS = derivSS[counter]
deltaLynx = toCoefLynx.dot(coefLynx)
actualLynx = derivLynx[counter]

mySS = mySS+stepSize*deltaSS
myLynx = myLynx+stepSize*deltaLynx

currentSS = SS[counter+1]
currentLynx = Lynx[counter+1]
myTimes[counter] = [time[counter],mySS,myLynx,currentSS,currentLynx]
myDerivs[counter] = [time[counter],deltaSS,deltaLynx,actualSS,actualLynx]

counter+=1
```

```
return myTimes,myDerivs

def marchForward(SS,Lynx,time,coefSS,coefLynx,initSS,initLynx,filter=None):
counter = 0
duration=len(time)
repeatSize = len(coefSS)
deltaSS=0
deltaLynx=0
mySS=initSS
myLynx = initLynx
currentSS = SS[0]
currentLynx = Lynx[0]
myGuesses = np.zeros([duration,5])
while(counter<duration):


toCoefSS = getLibrary(mySS,myLynx,True)
toCoefLynx = getLibrary(mySS,myLynx,True)
if(filter is not None):
tfilter = np.array(filter,dtype=bool)
toCoefSS = (toCoefSS.T[tfilter]).T
toCoefLynx = (toCoefLynx.T[tfilter]).T

mySS = toCoefSS.dot(coefSS)
myLynx = toCoefLynx.dot(coefLynx)

currentSS = SS[counter]
currentLynx = Lynx[counter]
myGuesses[counter] = [time[counter],mySS,myLynx,currentSS,currentLynx]
counter+=1
return myGuesses

def smallCalcs(startPointSS,startPointLynx,time,coefSS,coefLynx,diffSS,diffLynx):
counter = 0
duration=len(time)
repeatSize = len(coefSS)

myTimes = np.zeros([duration,3])
currentSS = startPointSS
currentLynx = startPointLynx

while(counter<10):
print('Step: '+str(counter))
myTimes[counter] = [time[counter],currentSS,currentLynx]
```

```
toCoefSS = getLibrary(currentSS,currentLynx,True)
#print(currentSS)
#print(toCoefSS)
#print(coefSS)
#print(toCoefSS*coefSS)
stepSize = time[counter+1]-time[counter]
toCoefLynx = getLibrary(currentLynx,currentSS,True)
deltaSS = np.sum(toCoefSS*coefSS)
print('Truth SS')
print(diffSS[counter]*stepSize)
print('mycalc SS')
print(deltaSS)
deltaLynx = toCoefLynx.dot(coefLynx)
print('Truth Lynx')
print(diffLynx[counter]*stepSize)
print('Mycalc lynx')
print(deltaLynx)
currentSS+= deltaSS
currentLynx += deltaLynx
counter+=1

def calcDiff(yvals, xdiffs):
numPoints = len(yvals)-1
allDifs = np.zeros(numPoints-1)
counter = 1
constDist = xdiffs[0]
while(counter < numPoints):
avgSlope = (yvals[counter+1]-yvals[counter-1])/(xdiffs[counter-1]+xdiffs[counter])
allDifs[counter-1] = avgSlope
counter+=1
return allDifs

def plotMyAttempt(ogv,myv,time,title = 'Interpolated Pelt Collection (1845-1903)',ylab
plt.clf()
plt.plot(time, ogv, 'o', time, myv, '--')
plt.title(title)
plt.ylabel(ylab)
plt.xlabel('time')
plt.legend(legendData, loc='best')
plt.show()

def timeDelayImbedData(data1,data2,numOffset):
size = len(data1)*2
hankellMatrix = np.zeros([numOffset,size])
```

```
counter =0
arrayIndex = 0
while(counter<numOffset):
j=0
arrayIndex=counter
while(j<size-2*counter):
hankellMatrix[counter,j] = data1[arrayIndex]
hankellMatrix[counter,j+1] = data2[arrayIndex]
arrayIndex+=1
j+=2
counter+=1
return hankellMatrix[:,0:(size-(numOffset-1)*2)]

def binData(data,numBins=20):
size = len(data)
hist, bin_edges = np.histogram(data, numBins)
probs = hist/size
return probs,bin_edges

def getProb(val,binEdges,probs):
counter = 0
while(counter<len(binEdges)):
if(val<=binEdges[counter+1]):
return probs[counter]
counter+=1
return 0

def createPointPairs(data1,data2,mean1,mean2):
size = len(data1)
counter = 0
bins=[1,1,1,1]
while(counter<size):
p1 = data1[counter]
p2 = data2[counter]
if(p1>mean1 and p2>mean2):
bins[0]+=1
elif(p1<=mean1 and p2>mean2):
bins[1]+=1
elif(p1>mean1 and p2<=mean2):
bins[2]+=1
elif(p1<=mean1 and p2<=mean2):
bins[3]+=1
counter += 1
bins = np.array(bins)
```

```
return (bins/size)

def lassoRegress(derivs,incomys,otherys,filter,penalty=0.1,iters=0):
holdA = getLibrary(incomys,otherys)
tfilter = np.array(filter,dtype=bool)
finalA = holdA.T[tfilter]
finalA = finalA.T
#print(A)
b=derivs
if(iters>0):
clf = linear_model.Lasso(alpha=penalty,max_iter=iters)
else:
clf = linear_model.Lasso(alpha=penalty)
clf = linear_model.LinearRegression()
clf.fit(finalA,b)
return (clf.coef_),finalA

def createBar(data,title):
'''
plots the magnitude of weights.
'''
plt.clf()
xs = np.arange(len(data))
newTitle = title
plt.title(newTitle)
plt.bar(x=xs,height=data)
plt.ylabel('weight value')
plt.xlabel('coefficient index')
plt.axhline(0, color='red', lw=1)
plt.show()

def calcKLDivergence(trueValsSS,myValsSS,trueValsLynx,myValsLynx):
meanss = np.mean(trueValsSS)
meanlynx = np.mean(trueValsLynx)
probActual = createPointPairs(trueValsSS,trueValsLynx,meanss,meanlynx)
probMine = createPointPairs(myValsSS,myValsLynx,meanss,meanlynx)
counter = 0
size = len(myValsSS)
score = 0
while(counter<size):
mp = probMine[counter]
ap = probActual[counter]
temp = -ap*np.log(mp/ap)
score+= temp
```

```
counter+=1
return score
'''
AIC/BIC
log(L(x|mu _hat)) ~~ -n/2 log(2pi) -n/2log(sigma^2) - 1/(2sigma^2)RSS
where sigma^2 = RSS/n (n is sample size, RSS = residual sum of squares = sum((y_i-f(x_
This assumes our error is gaussian (why? explain this in the paper)
Log likelihood for AIC and BIC. Likelihood of seeing the data given that model. Obviou

BOTH AIC AND BIC AND KL DIVERGENCE are both for one model in two dimensions, SO, you c
conseuqeuntly sigma squared is by summing hte squared distance between actual and my c

KL Divergence.
We are looking at the ratio of the distributions
equation is f(x,Beta) log((f(X,Beta)/g(X,Beta)))

where f is the amount of the animal

exploding to infinity is normal for this assignment when left on its own, so i got it

Now another approach is to do this

lookup ode45 equaivalent in python

difference equation is x_{n+1} = f(x_n,x_n-1...)

solved by look at x_n+1 rather than x', the derivative.

This difference equation and the x' are similar/ exact in certain situations


my x squared example is right .

for part 2, make it a small slice u(x,t), like a row of pixels.

SVD Can think of XV=UE (e is sum sigma) where sigma represents rank of matrix

want to throw both animals into this 2d matrix for the hankell matrix , like lynx ss l

Part 2
come up with a pde very similar to how we did part one but for part two

plot trajectories against each other. (lynx versus snowshoe..)
```

```
for KL Divergence dont bother for part 2.
'''
def main(data,penalty,level=2):
snowshoex , snowshoey = splineInterpolation(data[:,1],level,False,clearFigure=True,sav
lynxx , lynxy = splineInterpolation(data[:,2],level,False,clearFigure=True,saveLoc='ly
print(timeDelayImbedData(snowshoey,lynxy,10))
snowShoeDerivs = calcDiff(snowshoey,np.diff(snowshoex))
lynxDerivs = calcDiff(lynxy,np.diff(lynxx))
SSMarch = snowshoey[1:]
LynxMarch = lynxy[1:]
coefsSnowShoe,SA = getFormula(snowShoeDerivs,snowshoey[1:-1],lynxy[1:-1],penalty,10000
coefsLynx,LA = getFormula(lynxDerivs,lynxy[1:-1],snowshoey[1:-1],penalty,100000)

altss,idk = getFormula2(SSMarch,snowshoey[0:-1],lynxy[0:-1],penalty,100000)
altlynx,dik2 = getFormula2(LynxMarch,snowshoey[0:-1],lynxy[0:-1],penalty,100000)
filter = [1,1,1,0,0,1,0,0,1,1,1,1,1,1,0]
altss = altss*filter
altlynx = altlynx*filter
altss2,a2 = lassoRegress(SSMarch,snowshoey[0:-1],lynxy[0:-1],filter)
altlynx2,a3 = lassoRegress(LynxMarch,snowshoey[0:-1],lynxy[0:-1],filter)
#altss*[0,1,0,0]
print('snowshoe hare')
print(coefsSnowShoe)
print('lynx')
print(coefsLynx)
print(snowshoex.shape)
#vals,derivs= calcVals(snowshoey,lynxy,snowshoex[1:-1],coefsSnowShoe,coefsLynx,snowSho
#vals = marchForward(SSMarch,LynxMarch,snowshoex[1:],altss2,altlynx2,snowshoey[0],lynx
vals = marchForward(SSMarch,LynxMarch,snowshoex[1:],altss,altlynx,snowshoey[0],lynxy[0

print('total dist')
#totaldist = np.sum(np.abs(derivs[:,1]-derivs[:,3]))
#print(totaldist)
#print(np.sum(np.abs(derivs[:,3])))

MSSV = vals[:,1]
RSSV = vals[:,3]

MLV = vals[:,2]
RLV = vals[:,4]

#MSSD = derivs[:,1]
#RSSD = derivs[:,3]
```

```
#MLD = derivs[:,2]
#RLD = derivs[:,4]
plotMyAttempt(vals[:,3],vals[:,1],snowshoex[0:-1])
plotMyAttempt(vals[:,4],vals[:,2],snowshoex[0:-1])

#plotMyAttempt(RSSV,MSSV,snowshoex[1:-1])
#plotMyAttempt(RLV,MLV,lynxx[1:-1])

#print(calcKLDivergence(RSSV,MSSV))
#print(calcKLDivergence(RLV,MLV))

def solveLasso(A,b,penalty,iters=0):
if(iters>0):
clf = linear_model.Lasso(alpha=penalty,max_iter=iters)
else:
clf = linear_model.Lasso(alpha=penalty)
clf.fit(A,b)
return (clf.coef_),A

def marchForward2(coefSS,coefLynx,initSS,initLynx,steps,filter=None):
counter = 0
myGuesses = np.zeros([steps,2])
mySS=initSS
myLynx = initLynx
while(counter<steps):
toCoefSS = simpleLibrary(mySS,myLynx,True)
toCoefLynx = simpleLibrary(mySS,myLynx,True)
if(filter is not None):
tfilter = np.array(filter,dtype=bool)
toCoefSS = (toCoefSS.T[tfilter]).T
toCoefLynx = (toCoefLynx.T[tfilter]).T

mySS = toCoefSS.dot(coefSS)
myLynx = toCoefLynx.dot(coefLynx)
myGuesses[counter] = [mySS,myLynx]
counter+=1
return myGuesses

def calcKLDivergence(trueBins,falseBins):
score = np.sum(trueBins * np.log(trueBins/falseBins))
return score

def calcAIC(trueData,falseData,numParam):
size = len(trueData)
```

```
counter=0
RSS=0
while(counter<size):
diff1 = (trueData[counter,0]-falseData[counter,0])**2
diff2 = (trueData[counter,1]-falseData[counter,1])**2
RSS+=(diff1+diff2)
counter+=1
#sigma2 = RSS/counter
#likelihood = (-1*counter/2)*np.log(2*np.pi) + (-1*counter/2)*np.log(sigma2) + (-1/(2*

#AIC = 2*numParam-likelihood
AIC = counter*(np.log(RSS/counter)) +2*numParam
return AIC

def calcBIC(trueData,falseData,numParam):
size = len(trueData)
counter=0
RSS=0
while(counter<size):
diff1 = (trueData[counter,0]-falseData[counter,0])**2
diff2 = (trueData[counter,1]-falseData[counter,1])**2
RSS+=(diff1+diff2)
counter+=1
#sigma2 = RSS/counter
#likelihood = (-1*counter/2)*np.log(2*np.pi) + (-1*counter/2)*np.log(sigma2) + (-1/(2*

#AIC = 2*numParam-likelihood
AIC = counter*(np.log(RSS/counter)) + numParam*np.log(counter)
return AIC

def main2(data,penalty,level=3):
snowshoex , snowshoey = splineInterpolation(data[:,1],level,False,clearFigure=True,sav
lynxx , lynxy = splineInterpolation(data[:,2],level,False,clearFigure=True,saveLoc='ly
hm =(timeDelayImbedData(snowshoey,lynxy,50))
plotMyAttempt(snowshoey,lynxy,snowshoex,title = 'Interpolated Pelt Collection (1845-19

lib = simpleLibrary(snowshoey[:-1],lynxy[:-1])
ssb = snowshoey[1:]
lynxb = lynxy[1:]

sscoef,idc1 = solveLasso(lib,ssb,penalty,10000)
lynxcoef,idc2 = solveLasso(lib,lynxb,penalty,10000)

sscoef0,idc3 = solveLasso(lib,ssb,1,10000)
```

```
lynxcoef0,idc4 = solveLasso(lib,lynxb,1,10000)

model0guesses = marchForward2(sscoef0,lynxcoef0,snowshoey[0],lynxy[0],len(ssb),filter=
guesses = marchForward2(sscoef,lynxcoef,snowshoey[0],lynxy[0],len(ssb),filter=None)
print('model 1 coefficeints SS:')
print(sscoef0)

plotMyAttempt(model0guesses[:,0],ssb,snowshoex[1:],'Interpolated SS Pelt Collection (1
print('model 1 coefficeints Lynx:')
print(lynxcoef0)

plotMyAttempt(model0guesses[:,1],lynxb,snowshoex[1:],'Interpolated Lynx Pelt Collectio

print('model 2 coefficeints SS:')
print(sscoef)

plotMyAttempt(guesses[:,0],ssb,snowshoex[1:],'Interpolated SS Pelt Collection (1845-19
print('model 2 coefficeints Lynx:')
print(lynxcoef)

plotMyAttempt(guesses[:,1],lynxb,snowshoex[1:],'Interpolated Lynx Pelt Collection (184

altss,AS = getFormula(ssb,snowshoey[0:-1],lynxy[0:-1],0.1,100000)
altlynx,AL = getFormula(lynxb,snowshoey[0:-1],lynxy[0:-1],0.1,100000)
print('model 3 coefficients SS: ')
print(altss)
print('model 3 coefficients Lynx: ')
print(altlynx)
filter = [1,1,1,0,0,1,0,0,1,1,1,1,1,1,0]

'''
For model 2, I threshold at ~0.0002

'''
altss = altss*filter
altlynx = altlynx*filter
vals = marchForward(ssb,lynxb,snowshoex[1:],altss,altlynx,snowshoey[0],lynxy[0])
plotMyAttempt(vals[:,1],ssb,snowshoex[1:],'Interpolated SS Pelt Collection (1845-1903)
plotMyAttempt(vals[:,2],lynxb,snowshoex[1:],'Interpolated Lynx Pelt Collection (1845-1
ssmean = np.mean(vals[:,3])
lynxmean = np.mean(vals[:,4])
truDistrib = createPointPairs(vals[:,3],vals[:,4],ssmean,lynxmean)
myDistrib0 = createPointPairs(model0guesses[:,0],model0guesses[:,1],ssmean,lynxmean)
myDistrib1 = createPointPairs(guesses[:,0],guesses[:,1],ssmean,lynxmean)
```

```
myDistrib2 = createPointPairs(vals[:,1],vals[:,2],ssmean,lynxmean)

truModelScore=calcKLDivergence(truDistrib,truDistrib)
myModel0Score = calcKLDivergence(truDistrib,myDistrib0)
myModel1Score = calcKLDivergence(truDistrib,myDistrib1)
myModel2Score = calcKLDivergence(truDistrib,myDistrib2)

model0coefss = np.count_nonzero(sscoef0)
model0coeflynx = np.count_nonzero(lynxcoef0)

model1coefss = np.count_nonzero(sscoef)
model1coeflynx = np.count_nonzero(lynxcoef)

model2coefss = np.count_nonzero(altss)
model2coeflynx = np.count_nonzero(altlynx)

myModel0AIC = calcAIC(vals[:,3:5],model0guesses,model0coefss)
myModel1AIC = calcAIC(vals[:,3:5],guesses,model1coefss)
myModel2AIC = calcAIC(vals[:,3:5],vals[:,1:3],model2coefss)

myModel0BIC = calcBIC(vals[:,3:5],model0guesses,model0coefss)
myModel1BIC = calcBIC(vals[:,3:5],guesses,model1coefss)
myModel2BIC = calcBIC(vals[:,3:5],vals[:,1:3],model2coefss)

print(truDistrib)
print(truModelScore)

print('Bin values in order ')
print(myDistrib0)
print(myDistrib1)
print(myDistrib2)
print('KL Divergence values in order ')
print(myModel0Score)
print(myModel1Score)
print(myModel2Score)
print('AIC scores in order')
print(myModel0AIC)
print(myModel1AIC)
print(myModel2AIC)
print('BIC scores in order')
print(myModel0BIC)
print(myModel1BIC)
print(myModel2BIC)
```

```
createBar(sscoef0, "model 1 SS coefficients")
createBar(lynxcoef0,"model 1 lynx coefficients")
createBar(sscoef, "model 2 SS coefficients")
createBar(lynxcoef,"model 2 lynx coefficients")
createBar(altss,"model 3 SS coefficients")
createBar(altlynx,"model 3 lynx coefficients")

print('SS num coef in order')
print(model0coefss)
print(model1coefss)
print(model2coefss)
print('lynx num coef in order')
print(model0coeflynx)
print(model1coeflynx)
print(model2coeflynx)


'''
a matrix A multiplied to a vector does a certain amount of stretching and rotation, tw
https://www.youtube.com/watch?v=EokL7E6o1AE
think of x many vectors defines a x dimensional hypersphere
our A will alter these x that define a sphere S into a ellipse of n dimensions with a
we can break up these new x into a sigma*mu where mu is a directional vector and sigma
so in the AV=UE, V is a rotation, U is a rotation (also called unitary transformation)
A=UEV^T is called the reduced singular value decomp. Says that to get correct rotation
'''
u,s,vh=np.linalg.svd(hm) # this is our results from our hankell matrix
createBar(s/np.sum(s),"time delay imbedded singular values")

#smallTest()

main2(actualData,0.0001)
'''
part 2
use np.gradient to create the library.
so gradient over space is the transformation library for our data at a time steps

'''
#main(actualData,1)
#main(actualData,10)
#main(actualData,100)
#main(actualData,1000) #this returns negative KL scores!
```

```
#for time delay imbedding https://stackoverflow.com/questions/48967169/time-delay-embe
```

IX.II BZVideo.py

```
import numpy as np
import struct
import cvxpy as cp
import os
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.dates as mdates
import matplotlib.lines as lines
import matplotlib.cbook as cbook
import cvxopt as cx
from cvxopt.modeling import op, dot, variable
from sklearn.linear_model import Ridge
from scipy.sparse.linalg import lsmr
from sklearn import linear_model
from decimal import Decimal
from scipy.interpolate import interp1d
import h5py

smallModel = True
grounded = False

def fitModel(derivs,A,penalty=0.1,iters=0):
b=derivs
if(iters>0):
clf = linear_model.Lasso(alpha=penalty,max_iter=iters)
else:
clf = linear_model.Lasso(alpha=penalty)
clf.fit(A,b)
return (clf.coef_)

def transforms(inputs,single=False):
numRows = inputs.shape[0]
if(smallModel):
numCols=4
lib = np.ones([numRows,numCols])
lib[:,1] = inputs
lib[:,2] = np.gradient(inputs)
lib[:,3] = np.gradient(np.gradient(inputs))
```

```
else:
numCols=7
lib = np.ones([numRows,numCols])
lib[:,1] = inputs
lib[:,2] = np.gradient(inputs)
lib[:,3] = np.gradient(np.gradient(inputs))
lib[:,4] = np.sin(inputs)
lib[:,5] = np.sin(np.gradient(inputs))
lib[:,6] = np.sin(np.gradient(np.gradient(inputs)))
return lib

def rePredict(initPoints,coefs,numsteps,actualPoints):
myPoints=np.zeros([initPoints.shape[0],1])
myPoints[:,0] = initPoints.flatten()
myGuesses = np.zeros([initPoints.shape[0],numsteps])
myADerivs = np.zeros([initPoints.shape[0],numsteps])

counter = 0
while(counter<numsteps):
myGuesses[:,counter]=myPoints.flatten()
if(grounded):
myDerivs = transforms(actualPoints[counter])
else:
myDerivs = transforms(myPoints.flatten())
myADerivs[:,counter] = myDerivs.dot(coefs).T.flatten()
myPoints = myPoints.flatten() + myDerivs.dot(coefs).T.flatten()
counter+=1
return myGuesses,myADerivs

def calcMSE(trueData,myData):
counter = 0
totalErr = 0
for y in trueData:
totalErr += np.sum((myData[:,counter] -y)**2)/351
counter+=1
return (totalErr/counter)

def showSlice(newData):
plt.clf()
plt.imshow(np.reshape(newData,(351,1)),aspect='auto')
plt.show()

def createBar(data,title):
'''
```

```
plots the magnitude of weights.
'''
plt.clf()
xs = np.arange(len(data))
newTitle = title
plt.title(newTitle)
plt.bar(x=xs,height=data)
plt.ylabel('weight value')
plt.xlabel('coefficient index')
plt.axhline(0, color='red', lw=1)
plt.show()

def main(penalty):
file = h5py.File('BZ.mat','r')
l = list(file.keys())
data = file['BZ_tensor']
counter=0
altData = data[:,0,:]
x = (np.gradient(altData))

inputValues = altData.flatten()
output = x[0].flatten()
inputMatrix = transforms(inputValues)
file.close()
coefs = fitModel(output,inputMatrix,penalty)
print('penalty: ' + str(penalty))
print('coefs')
print(coefs)
guesses,mypderivs = rePredict(altData[0],coefs,len(altData),altData)
print('time:'+str(0))
showSlice(altData[0])
showSlice(guesses[:,0])
print('time: 600')
showSlice(altData[600])
showSlice(guesses[:,600])
print('time: 1200')
showSlice(altData[1199])
showSlice(guesses[:,1199])
print('diff: ' + str(guesses[:,0]-guesses[:,600]))

print('MSE of this model:')
print(calcMSE(altData,guesses))
print('MSE of this model derivs:')
print(calcMSE(np.array(x)[0],mypderivs))
```

```
print('baseline')
print(calcMSE(altData[0:-1,:],altData[1:,:].T))
print('baseline')
print(calcMSE(np.array(x)[0][0:-1,:],np.array(x)[0][1:,:].T))
print('MSE of this model (first 5 steps):')
print(calcMSE(altData[0:5],guesses[:,0:5]))
print('MSE of this model derivs (first 5 steps):')
print(calcMSE(np.array(x)[0][0:5],mypderivs[:,0:5]))
print('baseline')
print(calcMSE(altData[0:5,:],altData[1:6,:].T))
print('baseline')
print(calcMSE(np.array(x)[0][0:5,:],np.array(x)[0][1:6,:].T))
print('MSE of this model (first step):')
print(calcMSE(altData[0:1],guesses[:,0:1]))
print('MSE of this model derivs (first step):')
print(calcMSE(np.array(x)[0][0:1],mypderivs[:,0:1]))
print('baseline')
print(calcMSE(altData[0:1,:],altData[1:2,:].T))
print('baseline')
print(calcMSE(np.array(x)[0][0:1,:],np.array(x)[0][1:2,:].T))

createBar(coefs,'Model 1 Coefficient Values')
main(0.00001)
```