Data Analysis and Machine Learning: Introduction and Representing data

Morten Hjorth-Jensen[1,2]

Department of Physics, University of Oslo[1]

Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University[2]

May 9, 2018

Introduction

Statistics, data science and machine learning form important fields of research in modern science. They describe how to learn and make predictions from data, as well allowing us to extract important correlations about physical process and the underlying laws of motion in large data sets. The latter, big data sets, appear frequently in essentially all disciplines, from the traditional Science, Technology, Mathematics and Engineering fields to Life Science, Law, education research, the Humanities and the Social Sciences. It has become more and more common to see research projects on big data in for example the Social Sciences where extracting patterns from complicated survey data is one of many research directions. Having a solid grasp of data analysis and machine learning is thus becoming central to scientific computing in many fields, and competences and skills within the fields of machine learning and scientific computing are nowadays strongly requested by many potential employers. The latter cannot be overstated, familiarity with machine learning has almost become a prerequisite for many of the most exciting employment opportunities, whether they are in bioinformatics, life science, physics or finance, in the private or the

Learning outcomes

These lectures aim at giving you an overview of central aspects of statistical data analysis as well as some of the central algorithms used in machine learning. We will introduce a variety of central algorithms and methods essential for studies of data analysis and machine learning.

Hands-on projects and experimenting with data and algorithms plays a central role in these lectures, and our hope is, through the various projects and exercies, to expose you to fundamental research problems in these fields, with the aim to reproduce state of the art scientific results. You will learn to develop and structure large codes for studying these systems, get acquainted with computing facilities and learn to handle large scientific projects. A good scientific and ethical conduct is emphasized throughout the course. More specifically, you will

1. learn about basic data analysis, Bayesian statistics, Monte Carlo methods, data optimization and machine learning;
2. be capable of extending the acquired knowledge to other systems and cases;

Types of Machine Learning

The approaches to machine learning are many, but are often split into two main categories. In *supervised learning* we know the answer to a problem, and let the computer deduce the logic behind it. On the other hand, *unsupervised learning* is a method for finding patterns and relationship in data sets without any prior knowledge of the system. Some authours also operate with a third category, namely *reinforcement learning*. This is a paradigm of learning inspired by behavioral psychology, where learning is achieved by trial-and-error, solely from rewards and punishment.

Another way to categorize machine learning tasks is to consider the desired output of a system. Some of the most common tasks are:

- Classification: Outputs are divided into two or more classes. The goal is to produce a model that assigns inputs into one of these classes. An example is to identify digits based on pictures of hand-written ones. Classification is typically supervised learning.
- Regression: Finding a functional relationship between an input data set and a reference data set. The goal is to construct a

Software and needed installations

We will make intensive use of python as programming language and the myriad of available libraries. Furthermore, you will find IPython/Jupyter notebooks invaluable in your work. You can run **R** codes in the Jupyter/IPython notebooks, with the immediate benefit of visualizing your data.

If you have Python installed (we recommend Python3) and you feel pretty familiar with installing different packages, we recommend that you install the following Python packages via **pip** as

1. pip install numpy scipy matplotlib ipython scikit-learn mglearn sympy pandas pillow

For Python3, replace **pip** with **pip3**.

For OSX users we recommend also, after having installed Xcode, to install **brew**. Brew allows for a seamless installation of additional software via for example

1. brew install python3

For Linux users, with its variety of distributions like for example the widely popular Ubuntu distribution you can use **pip** as well and simply install Python as

Python installers

If you don't want to perform these operations separately, we recommend two widely used distrubutions which set up all relevant dependencies for Python, namely

1. Anaconda Anaconda is an open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system **conda**
2. Enthought canopy is a Python distribution for scientific and analytic computing distribution and analysis environment, available for free and under a commercial license.

Installing R, C++, cython or Julia

You will also find it convenient to utilize R. Jupyter/Ipython notebook allows you run **R** code interactively in your browser. The software library **R** is tuned to statistically analysis and allows for an easy usage of the tools we will discuss in these texts.
To install **R** with Jupyter notebook following the link here

Installing R, C++, cython or Julia

For the C++ aficionados, Jupyter/IPython notebook allows you also to install C++ and run codes written in this language interactively in the browser. Since we will emphasize writing many of the algorithms yourself, you can thus opt for either Python or C++ as programming languages.

To add more entropy, **cython** can also be used when running your notebooks. It means that Python with the Jupyter/IPython notebook setup allows you to integrate widely popular softwares and tools for scientific computing. With its versatility, including symbolic operations, Python offers a unique computational environment. Your Jupyter/IPython notebook can easily be converted into a nicely rendered **PDF** file or a Latex file for further processing. For example, convert to latex as

```
jupyter nbconvert filename.ipynb --to latex
```

If you use the light mark-up language **doconce** you can convert a standard ascii text file into various HTML formats, ipython notebooks, latex files, pdf files etc.

Introduction to Jupyter notebook and available tools

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import sparse
import pandas as pd
from IPython.display import display
eye = np.eye(4)
print(eye)
sparse_mtx = sparse.csr_matrix(eye)
print(sparse_mtx)
x = np.linspace(-10,10,100)
y = np.sin(x)
plt.plot(x,y,marker='x')
plt.show()
data = {'Name': ["John", "Anna", "Peter", "Linda"], 'Location': ["Nair
data_pandas = pd.DataFrame(data)
display(data_pandas)
```

Representing data, more examples

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import sparse
import pandas as pd
from IPython.display import display
import mglearn
import sklearn
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
x, y = mglearn.datasets.make_wave(n_samples=100)
line = np.linspace(-3,3,1000,endpoint=False).reshape(-1,1)
reg = DecisionTreeRegressor(min_samples_split=3).fit(x,y)
plt.plot(line, reg.predict(line), label="decision tree")
regline = LinearRegression().fit(x,y)
plt.plot(line, regline.predict(line), label= "Linear Regression")
plt.show()
```

Simple regression model

Add info about the equations

```
# Importing various packages
from random import random, seed
import numpy as np
import matplotlib.pyplot as plt

x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.randn(100,1)

xb = np.c_[np.ones((100,1)), x]
theta = np.linalg.inv(xb.T.dot(xb)).dot(xb.T).dot(y)
xnew = np.array([[0],[2]])
xbnew = np.c_[np.ones((2,1)), xnew]
ypredict = xbnew.dot(theta)

plt.plot(xnew, ypredict, "r-")
plt.plot(x, y ,'ro')
plt.axis([0,2.0,0, 15.0])
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r'Linear Regression')
plt.show()
```

Simple regression model, now using **scikit-learn**

Add info about the equations

```
# Importing various packages
from random import random, seed
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.randn(100,1)
linreg = LinearRegression()
linreg.fit(x,y)
xnew = np.array([[0],[2]])
ypredict = linreg.predict(xnew)

plt.plot(xnew, ypredict, "r-")
plt.plot(x, y ,'ro')
plt.axis([0,2.0,0, 15.0])
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r'Random numbers ')
plt.show()
```

Simple regression model with gradient descent

Add info about the equations, play around with different learning rates

```
# Importing various packages
from math import exp, sqrt
from random import random, seed
import numpy as np
import matplotlib.pyplot as plt

x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.randn(100,1)

xb = np.c_[np.ones((100,1)), x]
theta_linreg = np.linalg.inv(xb.T.dot(xb)).dot(xb.T).dot(y)
print(theta_linreg)
theta = np.random.randn(2,1)

eta = 0.1
Niterations = 1000
m = 100

for iter in range(Niterations):
    gradients = 2.0/m*xb.T.dot(xb.dot(theta)-y)
    theta -= eta*gradients

print(theta)
xnew = np.array([[0],[2]])
xbnew = np.c_[np.ones((2,1)), xnew]
```

Simple regression model with stochastic gradient descent

Add info about the equations, play around with different learning rates

```
# Importing various packages
from math import exp, sqrt
from random import random, seed
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor

x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.randn(100,1)

xb = np.c_[np.ones((100,1)), x]
theta_linreg = np.linalg.inv(xb.T.dot(xb)).dot(xb.T).dot(y)
print(theta_linreg)
sgdreg = SGDRegressor(n_iter = 50, penalty=None, eta0=0.1)
sgdreg.fit(x,y.ravel())
print(sgdreg.intercept_, sgdreg.coef_)
```

Polynomial regression

Predator-Prey model from ecology

The population dynamics of a simple predator-prey system is a classical example shown in many biology textbooks when ecological systems are discussed. The system contains all elements of the scientific method:

- The set up of a specific hypothesis combined with
- the experimental methods needed (one can study existing data or perform experiments)
- analyzing and interpreting the data and performing further experiments if needed
- trying to extract general behaviors and extract eventual laws or patterns
- develop mathematical relations for the uncovered regularities/laws and test these by per forming new experiments

Case study from Hudson bay

Lots of data about populations of hares and lynx collected from furs in Hudson Bay, Canada, are available. It is known that the populations oscillate. Why? Here we start by

1. plotting the data
2. derive a simple model for the population dynamics
3. (fitting parameters in the model to the data)
4. using the model predict the evolution other predator-pray systems

Hudson bay data

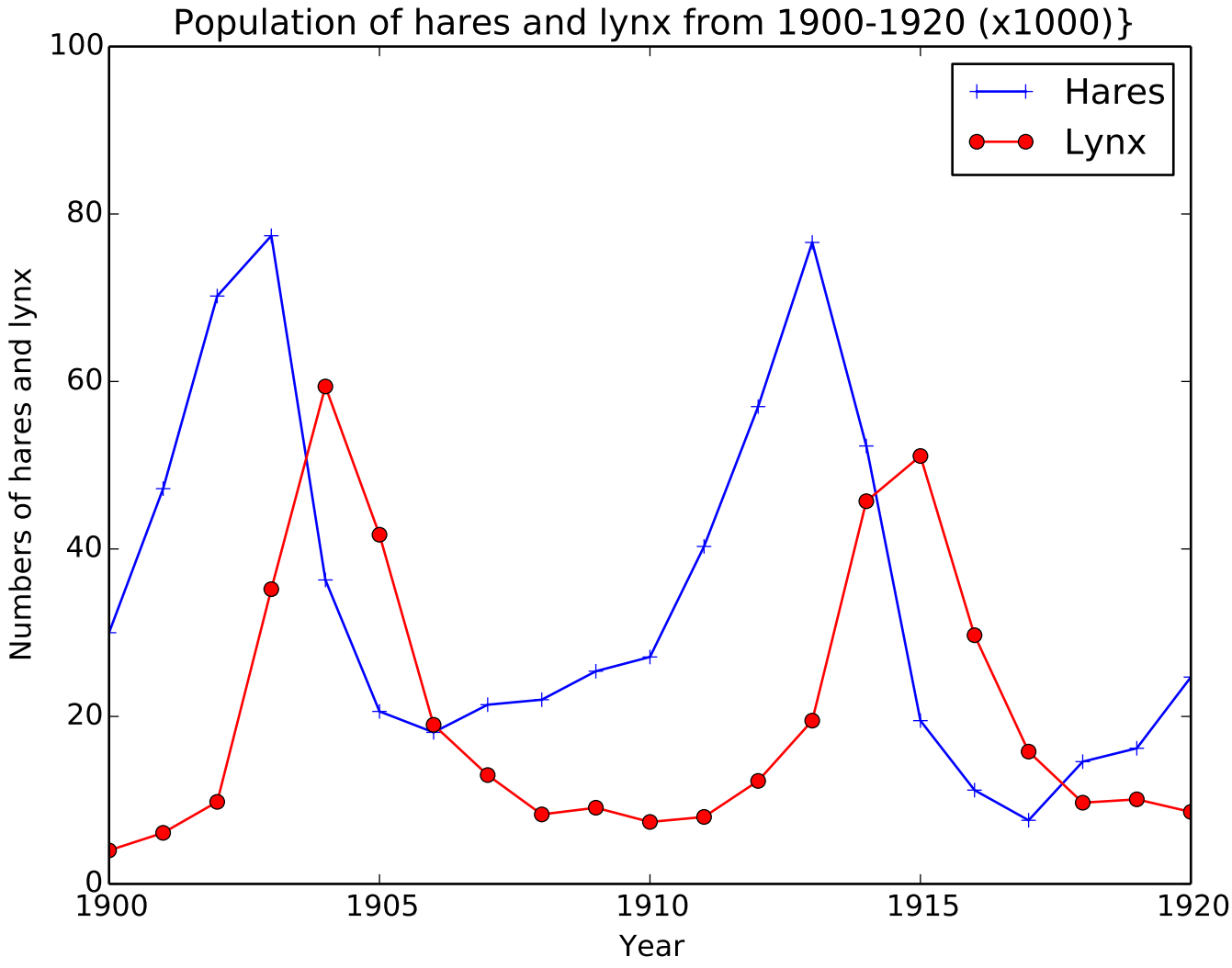
Most mammalian predators rely on a variety of prey, which complicates mathematical modeling; however, a few predators have become highly specialized and seek almost exclusively a single prey species. An example of this simplified predator-prey interaction is seen in Canadian northern forests, where the populations of the lynx and the snowshoe hare are intertwined in a life and death struggle. One reason that this particular system has been so extensively studied is that the Hudson Bay company kept careful records of all furs from the early 1800s into the 1900s. The records for the furs collected by the Hudson Bay company showed distinct oscillations (approximately 12 year periods), suggesting that these species caused almost periodic fluctuations of each other's populations. The table here shows data from 1900 to 1920.

Year	Hares (x1000)	Lynx (x1000)
1900	30.0	4.0
1901	47.2	6.1
1902	70.2	9.8
1903	77.4	35.2
1904	36.3	59.4

Plotting the data

```
import numpy as np
from  matplotlib import pyplot as plt

# Load in data file
data = np.loadtxt('src/Hudson_Bay.csv', delimiter=',', skiprows=1)
# Make arrays containing x-axis and hares and lynx populations
year = data[:,0]
hares = data[:,1]
lynx = data[:,2]

plt.plot(year, hares ,'b-+', year, lynx, 'r-o')
plt.axis([1900,1920,0, 100.0])
plt.xlabel(r'Year')
plt.ylabel(r'Numbers of hares and lynx ')
plt.legend(('Hares','Lynx'), loc='upper right')
plt.title(r'Population of hares and lynx from 1900-1920 (x1000)}')
plt.savefig('Hudson_Bay_data.pdf')
plt.savefig('Hudson_Bay_data.png')
plt.show()
```

Hares and lynx in Hudson bay from 1900 to 1920



Why now create a computer model for the hare and lynx populations?

We see from the plot that there are indeed fluctuations. We would like to create a mathematical model that explains these population fluctuations. Ecologists have predicted that in a simple predator-prey system that a rise in prey population is followed (with a lag) by a rise in the predator population. When the predator population is sufficiently high, then the prey population begins dropping. After the prey population falls, then the predator population falls, which allows the prey population to recover and complete one cycle of this interaction. Thus, we see that qualitatively oscillations occur. Can a mathematical model predict this? What causes cycles to slow or speed up? What affects the amplitude of the oscillation or do you expect to see the oscillations damp to a stable equilibrium? The models tend to ignore factors like climate and other complicating factors. How significant are these?

- We see oscillations in the data

The traditional (top-down) approach

The classical way (in all books) is to present the Lotka-Volterra equations:

$$\frac{dH}{dt} = H(a - bL)$$
$$\frac{dL}{dt} = -L(d - cH)$$

Here,

- H is the number of preys
- L the number of predators
- a, b, d, c are parameters

Most books quickly establish the model and then use considerable space on discussing the qualitative properties of this *nonlinear system of ODEs* (which cannot be solved)

Basic mathematics notation

- Time points: $t_0, t_1, \ldots, t_m$
- Uniform distribution of time points: $t_n = n\Delta t$
- H^n: population of hares at time t_n
- L^n: population of lynx at time t_n
- We want to model the changes in populations, $\Delta H = H^{n+1} - H^n$ and $\Delta L = L^{n+1} - L^n$ during a general time interval $[t_{n+1}, t_n]$ of length $\Delta t = t_{n+1} - t_n$

Basic dynamics of the population of hares

The population of hares evolves due to births and deaths exactly as a bacteria population:

$$\Delta H = a\Delta t H^n$$

However, hares have an additional loss in the population because they are eaten by lynx. All the hares and lynx can form $H \cdot L$ pairs in total. When such pairs meet during a time interval Δt, there is some small probablity that the lynx will eat the hare. So in fraction $b\Delta t HL$, the lynx eat hares. This loss of hares must be accounted for. Subtracted in the equation for hares:

$$\Delta H = a\Delta t H^n - b\Delta t H^n L^n$$

Basic dynamics of the population of lynx

We assume that the primary growth for the lynx population depends on sufficient food for raising lynx kittens, which implies an adequate source of nutrients from predation on hares. Thus, the growth of the lynx population does not only depend of how many lynx there are, but on how many hares they can eat. In a time interval ΔtHL hares and lynx can meet, and in a fraction $b\Delta tHL$ the lynx eats the hare. All of this does not contribute to the growth of lynx, again just a fraction of $b\Delta tHL$ that we write as $d\Delta tHL$. In addition, lynx die just as in the population dynamics with one isolated animal population, leading to a loss $-c\Delta tL$.

The accounting of lynx then looks like

$$\Delta L = d\Delta t H^n L^n - c\Delta t L^n$$

Evolution equations

By writing up the definition of ΔH and ΔL, and putting all assumed known terms H^n and L^n on the right-hand side, we have

$$H^{n+1} = H^n + a\Delta t H^n - b\Delta t H^n L^n$$

$$L^{n+1} = L^n + d\Delta t H^n L^n - c\Delta t L^n$$

Note:

- These equations are ready to be implemented!
- But to start, we need H^0 and L^0 (which we can get from the data)
- We also need values for a, b, d, c

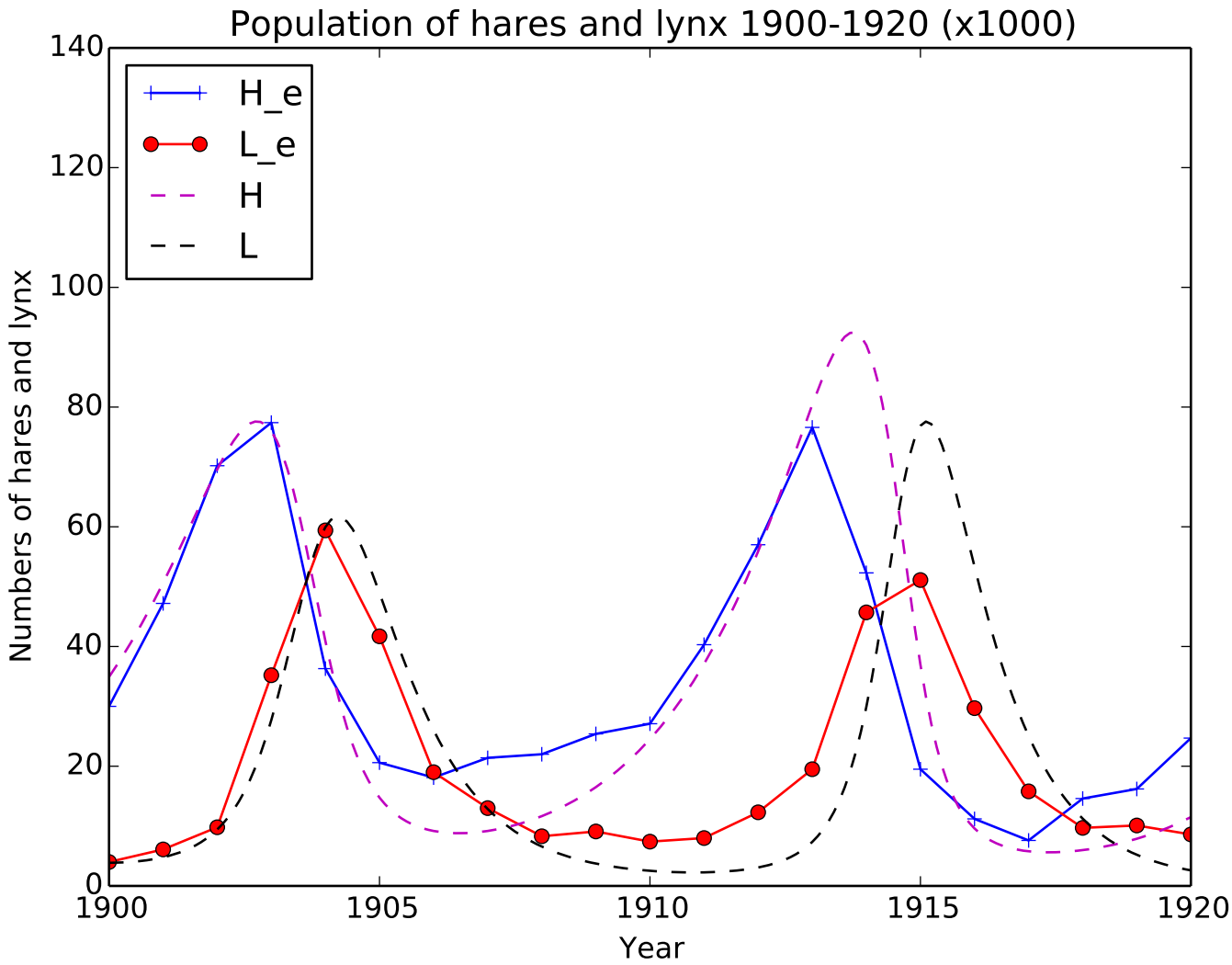
Adapt the model to the Hudson Bay case

- As always, models tend to be general - as here, applicable to "all" predator-pray systems
- The critical issue is whether the *interaction* between hares and lynx is sufficiently well modeled by constHL
- The parameters a, b, d, and c must be estimated from data
- Measure time in years
- $t_0 = 1900$, $t_m = 1920$

The program

```
import numpy as np
import matplotlib.pyplot as plt

def solver(m, H0, L0, dt, a, b, c, d, t0):
    """Solve the difference equations for H and L over m years
    with time step dt (measured in years."""

    num_intervals = int(m/float(dt))
    t = np.linspace(t0, t0 + m, num_intervals+1)
    H = np.zeros(t.size)
    L = np.zeros(t.size)

    print('Init:', H0, L0, dt)
    H[0] = H0
    L[0] = L0

    for n in range(0, len(t)-1):
        H[n+1] = H[n] + a*dt*H[n] - b*dt*H[n]*L[n]
        L[n+1] = L[n] + d*dt*H[n]*L[n] - c*dt*L[n]
    return H, L, t

# Load in data file
data = np.loadtxt('src/Hudson_Bay.csv', delimiter=',', skiprows=1)
# Make arrays containing x-axis and hares and lynx populations
t_e = data[:,0]
H_e = data[:,1]
```

The plot


Population of hares and lynx 1900-1920 (x1000)
H_e
L_e
H
L
Numbers of hares and lynx
Year
0
20
40
60
80
100
120
140
1900
1905
1910
1915
1920

Linear regression in Python

```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
import sklearn
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor


data = np.loadtxt('src/Hudson_Bay.csv', delimiter=',', skiprows=1)
x = data[:,0]
y = data[:,1]
line = np.linspace(1900,1920,1000,endpoint=False).reshape(-1,1)
reg = DecisionTreeRegressor(min_samples_split=3).fit(x.reshape(-1,1),y
plt.plot(line, reg.predict(line), label="decision tree")
regline = LinearRegression().fit(x.reshape(-1,1),y.reshape(-1,1))
plt.plot(line, regline.predict(line), label= "Linear Regression")
plt.plot(x, y, label= "Linear Regression")
plt.show()
```

Linear Least squares in R

```
HudsonBay = read.csv("src/Hudson_Bay.csv",header=T)
fix(HudsonBay)
dim(HudsonBay)
names(HudsonBay)
plot(HudsonBay$Year, HudsonBay$Hares..x1000.)
attach(HudsonBay)
plot(Year, Hares..x1000.)
plot(Year, Hares..x1000., col="red", varwidth=T, xlab="Years", ylab="H
summary(HudsonBay)
summary(Hares..x1000.)
library(MASS)
library(ISLR)
scatter.smooth(x=Year, y = Hares..x1000.)
linearMod = lm(Hares..x1000. ~ Year)
print(linearMod)
summary(linearMod)
plot(linearMod)
confint(linearMod)
predict(linearMod,data.frame(Year=c(1910,1914,1920)),interval="confide
```

Non-Linear Least squares in R

```
set.seed(1485)
len = 24
x = runif(len)
y = x^3+rnorm(len, 0,0.06)
ds = data.frame(x = x, y = y)
str(ds)
plot( y ~ x, main ="Known cubic with noise")
s  = seq(0,1,length =100)
lines(s, s^3, lty =2, col ="green")
m = nls(y ~ I(x^power), data = ds, start = list(power=1), trace = T)
class(m)
summary(m)
power = round(summary(m)$coefficients[1], 3)
power.se = round(summary(m)$coefficients[2], 3)
plot(y ~ x, main = "Fitted power model", sub = "Blue: fit; green: know
s = seq(0, 1, length = 100)
lines(s, s^3, lty = 2, col = "green")
lines(s, predict(m, list(x = s)), lty = 1, col = "blue")
text(0, 0.5, paste("y =x^ (", power, " +/- ", power.se, ")", sep = "")
```

Example: ecoli lab experiment

Typical pattern:

The population grows faster and faster. Why? Is there an underlying (general) mechanism?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

The program

Let us solve the difference equation in as simple way as possible, just to train some programming: $r = 1.5$, $N^0 = 1$, $\Delta t = 0.5$

```
import numpy as np

t = np.linspace(0, 10, 21)  # 20 intervals in [0, 10]
dt = t[1] - t[0]
N = np.zeros(t.size)

N[0] = 1
r = 0.5

for n in range(0, N.size-1, 1):
    N[n+1] = N[n] + r*dt*N[n]
    print 'N[%d]=%.1f' % (n+1, N[n+1])
```

The output

```
N[1]=1.2
N[2]=1.6
N[3]=2.0
N[4]=2.4
N[5]=3.1
N[6]=3.8
N[7]=4.8
N[8]=6.0
N[9]=7.5
N[10]=9.3
N[11]=11.6
N[12]=14.6
N[13]=18.2
N[14]=22.7
N[15]=28.4
N[16]=35.5
N[17]=44.4
N[18]=55.5
N[19]=69.4
N[20]=86.7
```

Parameter estimation

- We do not know r
- How can we estimate r from data?

We can use the difference equation with the experimental data

$$N^{n+1} = N^n + r\Delta t N^n$$

Say N^{n+1} and N^n are known from data, solve wrt r:

$$r = \frac{N^{n+1} - N^n}{N^n \Delta t}$$

Use experimental data in the fraction, say $t_1 = 600$, $t_2 = 1200$, $N^1 = 140$, $N^2 = 250$: $r = 0.0013$.

A program relevant for the biological problem

```
import numpy as np

# Estimate r
data = np.loadtxt('ecoli.csv', delimiter=',')
t_e = data[:,0]
N_e = data[:,1]
i = 2  # Data point (i,i+1) used to estimate r
r = (N_e[i+1] - N_e[i])/(N_e[i]*(t_e[i+1] - t_e[i]))
print 'Estimated r=%.5f' % r
# Can experiment with r values and see if the model can
# match the data better

T = 1200      # cell can divide after T sec
t_max = 5*T   # 5 generations in experiment
t = np.linspace(0, t_max, 1000)
dt = t[1] - t[0]
N = np.zeros(t.size)

N[0] = 100
for n in range(0, len(t)-1, 1):
    N[n+1] = N[n] + r*dt*N[n]

import matplotlib.pyplot as plt
plt.plot(t, N, 'r-', t_e, N_e, 'bo')
plt.xlabel('time [s]');  plt.ylabel('N')
plt.legend(['model', 'experiment'], loc='upper left')
plt.show()
```

Simulating financial transcations

The aim of this project is to simulate financial transactions among financial agents using Monte Carlo methods. The final goal is to extract a distribution of income as function of the income m. From Pareto's work (V. Pareto, 1897) it is known from empirical studies that the higher end of the distribution of money follows a distribution

$$w_m \propto m^{-1-\alpha},$$

with $\alpha \in [1, 2]$. We will here follow the analysis made by Patriarca and collaborators.

Here we will study numerically the relation between the micro-dynamic relations among financial agents and the resulting macroscopic money distribution.

We assume we have N agents that exchange money in pairs (i, j). We assume also that all agents start with the same amount of money $m_0 > 0$. At a given 'time step', we choose randomly a pair of agents (i, j) and let a transaction take place. This means that agent i's money m_i changes to m_i' and similarly we have $m_j \to m_j'$. Money is conserved during a transaction, meaning that