Lab session 0: Introduction to Python

GP Summer School—Kampala, 6–9th of August 2013

The aim of this lab session is to get you familiar with coding in Python. We will illustrate how to execute some useful tasks in Python by solving a linear regression problem. Please follow the installation instructions from the outline document, in case your computer does not have Python 2.7 and GPy already installed.
We first open *ipython* and import the libraries we will need:

```python
import numpy as np
import pylab as pb

pb.ion()                        # Open one thread per plot
```

Remember to check what a command does, simply type:

```python
np.random.randn?
```

1 Linear regression: iterative solution

For this part we are going to load in some real data, we will use an example from the Olumpics: the pace of Marathon winners. To load their data (which is in comma separated values (csv) format) we need to download it from: `http://staffwww.dcs.shef.ac.uk/people/N.Lawrence/olympicMarathonTimes.csv`, and load it as follows:

```python
olympics = np.genfromtxt('olympicMarathonTimes.csv', delimiter=',')
```

This loads the data into a Python array
You can extract the Olympic years and the pace of the winner, respectively, as:

```
x = olympics[:, 0:1]
y = olympics[:, 1:2]
```

You can see what the values are by typing:

```
print(x)
print(y)
```

You can make a plot of y vs x with the following command:

```
pb.plot(x, y, "rx")
```

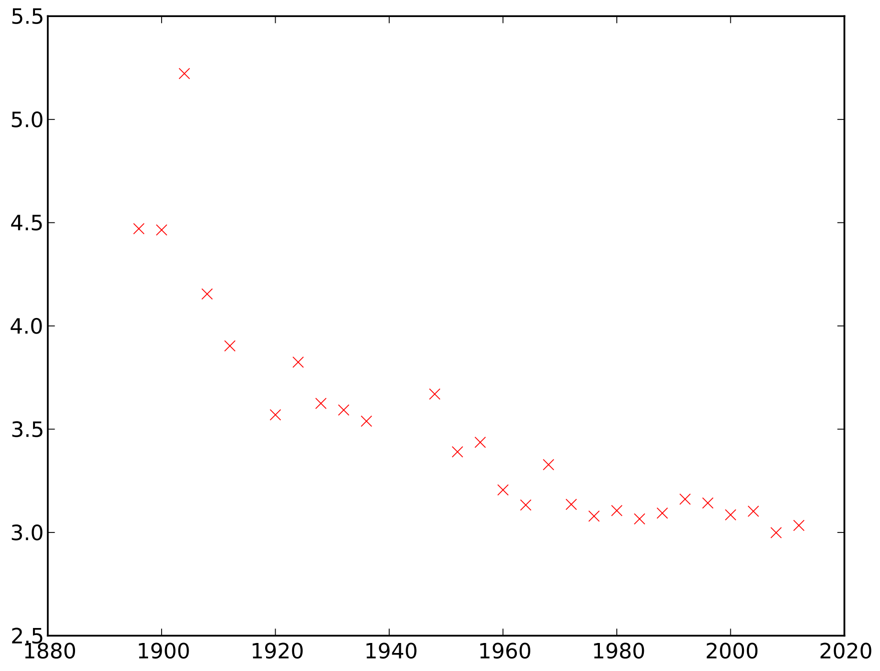

Figure 1: Olympic Marathon Winners' Times.

Now we are going to fit a line, $y_i = mx_i + c$, to the data you've plotted . We are trying to minimize this error:

$$E(m, c, \sigma^2) = \frac{n}{2} \log \sigma^2 + \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - mx_i - c)^2,$$

with respect to m, c and σ^2. We need to start with an initial guess for m, the actual value doesn't matter too much, try setting it to zero, then use this formula to set c:

$$c^* = \frac{\sum_{i=1}^{n} (y_i - m^* x_i)}{n},$$

followed by this formula to estimate m:

$$m^* = \frac{\sum_{i=1}^{n} x_i (y_i - c^*)}{\sum_{i=1}^{n} x_i^2}.$$

Iterate between the two formulae and compute the error after each iteration and see how much it changes. Finally when the error stops changing update the estimate of the noise variance:

$$\sigma^{2*} = \frac{\sum_{i=1}^{n} (y_i - m^* x_i - c^*)^2}{n}.$$

Print the final recorded values for the error, m, c and σ^2. Plot the error as a function of iterations. Ensure it goes down at each iteration.
You can add the line fitted to the figure you created earlier. To do this, we will create some 'test' data uniformly spaced along the x-axis. For example, you can create a set of values between two points with the command `np.linspace` :

```
xTest = np.linspace(1890,2010,100)
pb.plot(xTest,m*xTest + c,"b-")
```

Question 1a The error function used above assumes the following model $y_i = mx_i + c_i + \epsilon_i$, where ϵ_i corresponds to the observed noise. What is a sensible assumption about the noise's probability density?

Question 1b What are the mean and standard deviation of the observed noise?

Question 1c Can you write down the probability density for $\mathbf{y}$?

2 Basis functions

We don’t need to run the iterative algorithm. Since there was a gradient and an offset, we will use the ’trick’ of having a basis set containing the data and another basis set containing the ’constant function’ (i.e. set to 1).

```
Phi = np.hstack((np.ones(x.shape), x))
print(Phi)
```

We can use this basis set to learn m and c simultaneously. The maximum likelihood solution for $\mathbf{w}$ is:

$$\mathbf{w}^* = \left[\mathbf{\Phi}^\top\mathbf{\Phi}\right]^{-1}\mathbf{\Phi}^\top\mathbf{y},$$

but you should solve the matrix equation:

$$\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} = \mathbf{\Phi}^\top\mathbf{y}$$

directly to get the best solution (Hint: try `np.linalg.solve?`) and call the resulting variable `wStar`. Implement this update and show that w_1^* (the first element of $\mathbf{w}^*$) is equal to c and w_2^* is equal to m.
Create some ‘test’ data, as before, and create an associated feature matrix to add the bias term:

```
PhiTest = np.zeros((xTest.shape[0], 2))
PhiTest[:, 0] = np.ones(xTest.shape)
PhiTest[:, 1] = xTest
```

To obtain and plot the test outputs you can write:

```
yTest = np.dot(PhiTest, wStar)
pb.plot(xTest, yTest, "b-")
```

Now we will fit a non-linear basis function model. Start by creating a quadratic basis:

```
Phi = np.zeros((x.shape[0], 3))
for i in range(0, 3):
    Phi[:, i] = x.T**i
```

Question 2a Plot the fit and compute the final error using the non-linear basis function.

Question 2b Large order polynomial fits tend to do fairly extraordinary things outside an input range of 1896 to 2008. Have a think about why this is, in particular, what is the result of 2012^8?

Question 2c Do you think that is a suitable basis?