Introduction to Python Programming

Adrian Price-Whelan - adrn@nyu.edu

This document is designed to provide an introduction to Python and various packages (SciPy, NumPy, Matplotlib, etc.) that are important for data analysis. The problems assume a basic knowledge of computer programming, but should be at a level that even complete beginners can work through it. In either case, if you are new to Python I recommend taking a look at the Python tutorial, available at http://docs.python.org/tutorial/. If you have questions, or need hints on problems, don't hesitate to email me.

Problem 0: If you haven't already, download and install the Enthought distribution (http://www.enthought.com/). This will install Python, SciPy, NumPy, Matplotlib and a few other packages for you, but you are welcome to try to build from source instead. After the installation is complete, choose an editor (I don't recommend using the default IDLE editor included with the Python install). Some suggestions: emacs (all platforms), Xcode (Mac), TextWrangler (Mac), Eclipse (all platforms).

Open up the file 'problem-0.py,' included in the zip file, and insert your name where it says 'namehere,' and then run the file. Send the PNG file that is generated to adrn@nyu.edu.

Problem 1:

a) Write a script that collects user input, determines whether the input is a String (of only alphabetical characters), Integer, or Neither, and then prints the data type, length, and time it was entered.

b) Add to this script so that if the input is an Integer it prints the number squared, the square root of the number, and the number factorial.

c) Add again to the script so that if the input is a string, it outputs each word on a separate line, and prints the second to last character in the string 100 times.

d) Modify the code once more so that if the input is 'Neither' - it separates the letters from the numbers and prints them separately as strings.

Problem 2:

a) Write a script that asks for a number as input, and determines whether it is a prime number or not.

b) Modify the function so that if the input number is not a prime number, it will print the closest 3 prime numbers (limit this to the cases $N < 100$ and $N = int(N)$).

c) Add to this same .py file a means to output the Fibonacci series up to the boundary number N. For example, if the input number is 10, the module (when run) also outputs: 1, 1, 2, 3, 5, 8.

Problem 3:

For this problem, you will modify the file 'problem-3.py.'

a) Open and look at the code in the file 'problem-3.py.' Currently the code simply generates a one-dimensional array of 'data,' and plots a scatterplot of the values. Use matplotlib to create a histogram of this fake dataset.

b) The function random.uniform() samples from a uniform distribution, and so the histogram should be uninteresting. Instead of drawing our data from a uniform distribution, get it from a Gaussian distribution of Standard Deviation $\sigma = 3.0$ and Mean $\mu = 5.0$, again getting 10^5 samples. Make another histogram, this time using the Gaussian data, and think about what to set the bin size for this histogram.

c) Fit a line of best fit to the histogram.

**Problem 4:

Numerical integration is very important for solving Ordinary Differential Equations which cannot be solved analytically. In such cases, an approximation will have to do, and there are many different algorithms that achieve different levels of accuracy.

a) One of the simplest methods to understand and implement is Euler's method, which is essentially a 1st order approximation. One step using Euler's method from t_n to t_{n+1} is given by the expression:

$$\vec{y}_{n+1} = \vec{y}_n + hf(t_n, \vec{y}_n) \tag{1}$$

Or, for this case:

$$\begin{pmatrix} y \\ \dot{y} \end{pmatrix}_{n+1} = \begin{pmatrix} y \\ \dot{y} \end{pmatrix}_n + h \begin{pmatrix} \dot{y} \\ \ddot{y} \end{pmatrix}_n \tag{2}$$

$$\begin{pmatrix} y \\ \theta \end{pmatrix}_{n+1} = \begin{pmatrix} y \\ \theta \end{pmatrix}_n + h \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}_n \tag{3}$$

Where h is the step size.

Implement Euler's method to numerically integrate the following Differential Equation from $t_i = 0s$ to $t_f = 50s$, with initial condition $\theta(0) = 1.0$, and plot $\theta(t)$ over this interval for the three cases $h = 1.0$, $h = 0.1$, and $h = 0.01$. This equation is the equation of motion for a simple pendulum - and so you should expect to get some kind of oscillatory function for θ.

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0 \tag{4}$$

$$\text{Define a new variable } \tau = t\sqrt{\frac{g}{l}} \tag{5}$$

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0 \tag{6}$$

We can break this up into two coupled 1st order ODE's:

$$\frac{d\theta}{d\tau} = y \tag{7}$$

$$\frac{dy}{d\tau} = -\sin\theta \tag{8}$$

A good set of initial conditions would be something like $h = 0.01, \theta_0 = 1.0, y_0 = 0.0, t_0 = 0.0, t_{max} = 50.0$. Does this look like you expected? Play with the initial conditions. Make the initial angle large, or the step size large - what happens?

b) The Runge-Kutta methods for numerical integration is more complex iterative method for approximating the solution to ODEs. The most commonly used method is the 4th order, or RK4 method - which differs from the Euler method in the following way:

$$\vec{y}_{n+1} = \vec{y}_n + \frac{1}{6}h(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \tag{9}$$

Where:

$$\vec{k}_1 = f(t_n, \vec{y}_n) \tag{10}$$

$$\vec{k}_2 = f(t_n + \frac{1}{2}h, \vec{y}_n + \frac{1}{2}h\vec{k}_1) \tag{11}$$

$$\vec{k}_3 = f(t_n + \frac{1}{2}h, \vec{y}_n + \frac{1}{2}h\vec{k}_2) \tag{12}$$

$$\vec{k}_3 = f(t_n + h, \vec{y}_n + h\vec{k}_3) \tag{13}$$

Solve the same differential equation as in part a), this time using the RK4 method, and plot the result of part a) and part b) on the same figure for $h = 1.0$ and $h = 0.01$, keeping $\theta_0 = 1.0, y_0 = 0.0, t_0 = 0.0, t_{max} = 10.0$.