Python Tutorial

Jannicke Pearkes, Physics 67, 2018

Introduction

This tutorial will go over the essentials that you will need to do your data analysis in python. We will go over the basics of programming in python, how to read in a data file, and fit a curve to data.

Work through this tutorial at your own pace. Ideally, by the end of class you will have gotten to the challenge exercise, but everyone will be working at a different speed. That is okay! If you are new to programming, don't worry if this seems difficult at first, with a bit of effort you will get the hang of it.

As always the TAs and course instructors are there to help you out. Ask questions during class and please come to office hours for help if you are having difficulties. You are also encouraged to ask your peers for help if you are stuck.

Installing Anaconda

If you have not already done so, follow the following start up instructions to install Anaconda:

https://docs.google.com/a/stanford.edu/document/d/1TSimC3ZxhPAhzQJDKWUibM6KgEMk8QIih5CwdG0W
usp=sharing
(https://docs.google.com/a/stanford.edu/document/d/1TSimC3ZxhPAhzQJDKWUibM6KgEMk8QIih5CwdG0W
usp=sharing)

Make sure that you install the python 2 version, not python 3.

Test

The instructions for installing Anaconda explain how to open a notebook. Once you have opened your first notebook, type in the following command. Hit shift+enter, or press the play button to run the cell.

In [ ]:

```
print "hello world!"
```

If it outputs hello world!, you are good to move onto the next section. If you are stuck here, ask for help either from a neighbour or the instructor.

Basic Programming

If you are not familiar with python, try typing in and running the following commands to get familiar with the basics. Otherwise skip ahead to the next section 'Writing our own functions'. After entering each command press shift+enter to see the output.

In [ ]:

```
# Addition
1+1
```

This should output 2

In [ ]:

```
# Multiplication
1*2
```

In [ ]:

```
# Integer division (usually not recommended)
1/2
```

In [ ]:

```
# Floating point division (recommended way to do division)
# The decimal point tells python to use a float
1./2
```

In [ ]:

```
# Power
2**3
```

In [ ]:

```
# Assigning values to variables
# Try playing around with different assignments
a = 5.5
b = 4
c = a-b
print c
```

In [ ]:

```
# Making a list
my_list = [1,2,3,4,5,6]

# Reading the first value (lists are 0-indexed)
my_list[0] # the number in the bracket tells what position to output
```

In [ ]:

```
# Reading first 3 values, this is called slicing
my_list[0:3]
```

Importing a library

Nice! Now that we are familiar with the basics it is time to get a little more sophisticated. Remember that if there is anything you don't know how to do, you can always ask Google. For example try googling "python squareroot" to find out how to take a square root of a number.

One of the first results is: https://www.tutorialspoint.com/python/number_sqrt.htm (https://www.tutorialspoint.com/python/number_sqrt.htm)

It tells us to try:

In [ ]:

```
import math
x = 4
math.sqrt(x)
```

The import statement allows us to import a library. This is convient because someone else has gone to the trouble of writing a math library which has all types of functions that you might want to use during this course.

The documentation for math is here, take a look at the link to find out what other things 'math' can do: https://docs.python.org/2/library/math.html (https://docs.python.org/2/library/math.html)

Other libraries we will use in this class are numpy (numeric python, good for dealing with arrays), matplotlib (a plotting tool), and scipy (scientific python). These all come with your standard anaconda installation and we will learn about them later in this tutorial.

Writing our own functions

In this section we will write one function to compute the mean of a set of data and another to compute the standard deviation.

One thing to remeber is that white-space matters in python. Remember to indent consistently and carefully, and make sure to start a new line at the right place.

In [ ]:

```
def mean(data):
    # this is a function to find the mean
    L = len(data) # len is an inbuilt python function
    m = 0
    for i in range(0,L): # find the sum of all data points
        m = m + data[i]
    m = m/L
    # a short way is: m /= L
    # return the result for later use
    return m

def sample_std_dev(data):
    # this is a function to find the standard deviation
    L = len(data)
    m = mean(data)
    # initialise error from mean of each # data point
    err = 0
    for i in range(0, L):
        err = err + (data[i] - m)**2
    err = math.sqrt(err / (L-1))
    return err

data = [2.2, 2.4, 2.5 ,2.6 , 2.8] # sample population

mean_val = mean(data)
std_dev_val = sample_std_dev(data)

print 'The mean value is: ', mean_val
print 'The sample standard deviation is: ', std_dev_val
```

Okay, great, so now we know how to define our own functions.

Using Numpy

We can also do this more easily using numpy. Numpy is a pre-existing library that can do a lot of things for you. Try the following code and see if it agrees with what we calculated ourselves.

In [ ]:

```
import numpy as np

np_mean_val = np.mean(data)
np_std_dev_val = np.std(data)

print 'The mean value is: ', np_mean_val
print 'The standard deviation is: ', np_std_dev_val
```

Chances are that the standard deviation calculated by numpy and the one we calculated are slightly different. Why do you think this is?

Hint: think about sample population vs parent population as discussed in class

How could we change the parameters given to np.std() to give us a result that agrees with the standard deviation we calculated ourselves?

Check out the Notes section in the numpy documentation on np.std() https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.std.html (https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.std.html) to figure this out.

To Download

Download falling_object.csv and challenge_data.csv from the Files/Python Tutorial folder in Canvas and make sure that it is in the same folder as your notebook is in. **It is very important that these items are in the same folder as your notebook.**

Reading a Data File

Next we are going to read in a data file with numpy. First open up the 'falling_object.csv' data file and take a look at it. You will notice values that are seperated by commas (the delimiter). This is basically like an not very fancy Excel spreadsheet, with the commas seperating the columns, and the new lines seperating the rows.

Here our first column denotes time and the second column is the position of a falling object.

In [ ]:

```
# load data that we stored in .csv format
data = np.loadtxt('falling_object.csv',delimiter=',')

# take a look at how many rows and columns we have in our data
print(data.shape)
```

Next we will want to split up our data array into time and position.

The syntax for "slicing" or splitting up your array goes as follows:

$ data[rows, columns]

Putting a colon into either rows or columns allows us to choose all rows or all columns:

$ data[:, 0] # choose all rows, column 0

$ data[0, :] # choose all columns, row 0

$ data[0:5, 0] # choose first 5 rows, column 0

Google, numpy slicing for more info if you are interested.

In [ ]:

```
# Split data into time and position
# We know that the first column is time and the second is position
# So we slice up our data array as follows

time = data[:,0]
position = data[:,1]

print "time"
print time
print "position"
print position
```

Plotting our data

Next we will make some plots using matplotlib!

In [ ]:

```
# import packages we need to use
%matplotlib inline

import matplotlib.pyplot as plt

# plot data as a scatter plot
plt.plot(time,position,'.')
# the '.' stands for the type of marker we are using
# 'x','o' are other valid types of markers
# Google "markers matplotlib" to find out what other types of marker you can use

# always label your axes
plt.title("Position of Falling Object")
plt.xlabel('time (second)')
plt.ylabel('position (meter)')
```

Fitting a curve

Here we are going to use scipy's curve fitting tool to extract gravity.

In [ ]:

```
from scipy.optimize import curve_fit

# we are going to use the curve_fit function from the scipy package
# to fit a polynomial through the data points.
# this will take a couple of steps

# first we need to define the functional form to which to fit.
# also known as defining a model.

# IMPORTANT: it's important that time, the independent variable here,
# be the first parameter in this function
def free_fall_func(time, initial_vel, g, initial_pos):
    position = time * initial_vel + g / 2.0 * time**2.0 + initial_pos
    return position
```

In [ ]:

```
# we use the curve_fit function, which we imported earlier
# putting a question mark after a function and running the code will
# load the docstring of the function, i.e. some documentation telling you
# how to use it. Extremely useful.
curve_fit?
```

In [ ]:

```
# Now actually tun curve_fit
# what do you see? Hopefully two arrays and a bunch of numbers
curve_fit(free_fall_func, time, position)
```

In [ ]:

```
# store your results in some variable
# we are going to need it later
params, _ = curve_fit(free_fall_func, time, position)
print(params)

# What do these parameters mean?
# What is the acceleration, the initial velocity and initial position?
# Does this make sense?
```

In [ ]:

```
# Visualize your fitted curve
# first, compute what the fitted curve is from all the time data points
fitted_curve = free_fall_func(time, *params)
```

In [ ]:

```
# plot data as a sactter plot
plt.plot(time,position,'.',label = 'Data')

# plot fitted curve as line
plt.plot(time, fitted_curve,'-', label = 'Fitted curve')

# label your axes and add a title
plt.xlabel('time (second)')
plt.ylabel('position (meter)')
plt.title("Position of Falling Object")
plt.legend()
```

Challenge Activity

Okay, we are going to try fitting a gaussian to the data in 'challenge_data.csv'.

A gaussian function is a function of the form:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

Here μ is the mean of the Gaussian, and σ is the standard deviation.

- Load data
- Plot data
- Estimate what the mean and standard deviation of this data set is
- Use the curve fitting tool to fit a gaussian to the data points
- What is the mean and standard deviation of the gaussian?
- Plot the data + fitted curve
- Annotate the graph with the mean and standard deviation you found with the curve fitting tool

Extra Documentation

Matplotlib: https://matplotlib.org/index.html# (https://matplotlib.org/index.html#)

Numpy Userguide: https://docs.scipy.org/doc/numpy-dev/user/index.html (https://docs.scipy.org/doc/numpy-dev/user/index.html)

Scipy Reference: http://scipy.github.io/devdocs/ (http://scipy.github.io/devdocs/)

Python 3 Changes

If you are using python 3, note that you will need to add a bunch of brackets into the print statements. Change all print statements from

print "thing to print"

to

print("thing to print")