

Python for Scientific Computing in Economics

John Stachurski

January 2016

Before We Start

Installed Anaconda?

- Free from <http://continuum.io/downloads>
- Make it your default Python distribution

Installed Anaconda a while ago?

- In a terminal type `conda update anaconda`

Need a temporary solution?

- <https://try.jupyter.org/>

Topics

- Getting started
 - Scientific computing background
 - Intro to Python
 - How to run Python code
- Scientific programming
 - The scientific libraries
 - Graphics
- Learning Python
 - Basic syntax, data types
- Exercise

Scientific Programming Environments

Low level languages:

- C, C++
- Fortran

High level languages

- Matlab
- Julia
- Python
- R, etc.

Low level languages require more details from the user

```
/* Creates a linear array. */
void linspace(double *ls, double a, double b, int n)
{
    double step = (b - a) / (n - 1);
    int i;
    for (i = 0; i < n; i++)
    {
        ls[i] = a;
        a += step;
    }
}
```

High level languages require less information

```
function linspace!(ls, a, b)
    n = length(ls)
    step = (b - a) / (n - 1)
    for i in 1:n
        ls[i] = a
        a += step
    end
    return ls
end
```

High level languages tend to be convenient for the user

```
>>> a, b = 1, 4
>>> a + b
5
>>> a, b = 'foo', 'bar'
>>> a + b
'foobar'
```

But harder to convert into optimized machine code

Although **big** steps forward in recent years

- Julia, Python + Numba

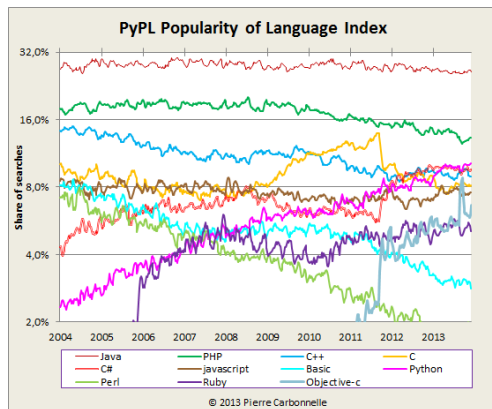
What's Python?

A high level, general purpose programming language

Used extensively by

- Tech firms (YouTube, Dropbox, Reddit, etc., etc.)
- Hedge funds and finance industry
- Gov't agencies (NASA, CERN, etc.)
- Academia

Free and open source



Python is noted for

- Elegant, modern design
- Clean syntax, readability
- High productivity

Often used to teach first courses in comp sci and programming

- MIT, Udacity, edX, etc.

Scientific Programming

Rapid adoption by the scientific community

- Artificial intelligence
- engineering
- computational biology
- chemistry
- physics, etc., etc.

Major Scientific Libraries

NumPy

- basic data types
- simple array processing operations

SciPy

- built on top of NumPy
- provides additional functionality

Matplotlib

- 2D and 3D figures

NumPy Example: Mean and standard dev of an array

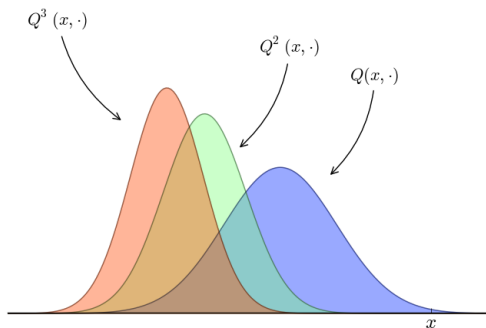
```
>>> import numpy as np
>>> a = np.random.randn(10000)
>>> a.mean()
0.0020109779347995344
>>> a.std()
1.0095758844793006
```

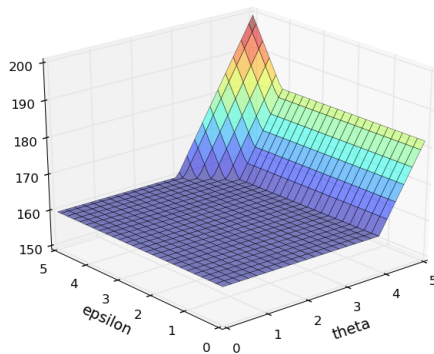
SciPy Example: Calculate

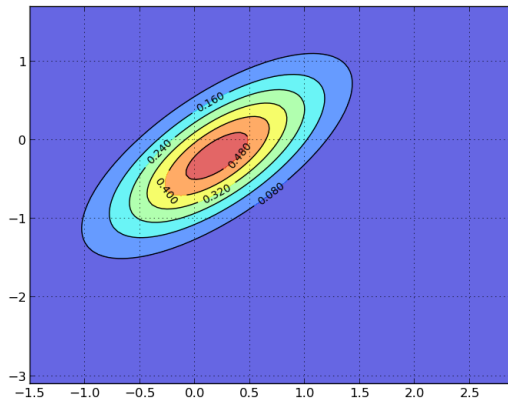
$$\int_{-2}^2 \phi(z) dz \quad \text{where} \quad \phi \sim N(0,1)$$

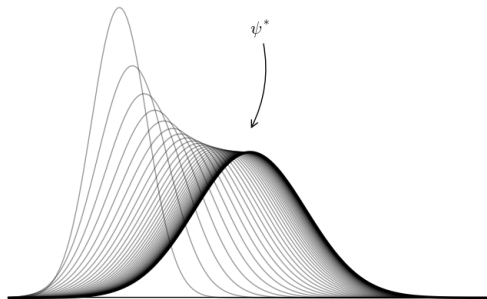
```
>>> from scipy.stats import norm
>>> from scipy.integrate import quad
>>> phi = norm()
>>> value, error = quad(phi.pdf, -2, 2)
>>> value
0.9544997361036417
```

Matplotlib examples









Other Scientific Libraries

Pandas

- statistics and data analysis

SymPy

- symbolic manipulations à la Mathematica

Still more:

- **statsmodels** — statistics / econometrics
- **scikit-learn** — machine learning in Python

The QuantEcon Libraries

Code for

- Markov chains
- Dynamic programming
- LQ control
- etc

Both Python and Julia versions

See <http://quantecon.org/>

Other Scientific Tools

Also tools for

- working with graphs (as in networks)
- parallel processing, GPUs
- manipulating large data sets
- interfacing C / C++ / Fortran
- cloud computing
- database interaction
- bindings to other languages, like R and Julia
- etc.

Why I Prefer Python to Matlab

1. Open source
 - no license hassles
 - can read / edit source code
2. General purpose
 - meets all my coding needs
3. Very well designed language
4. Other awesomeness
 - Fast loops through Numba
 - Jupyter, etc

How About Julia?

- Specialized to scientific computing
- Open source
- Fast

Why I personally use Python more than Julia

1. General purpose
2. Python has copied some goodies from Julia (Numba)
3. Language design fits my brain
4. Julia is still a bit unstable

Interacting with Python

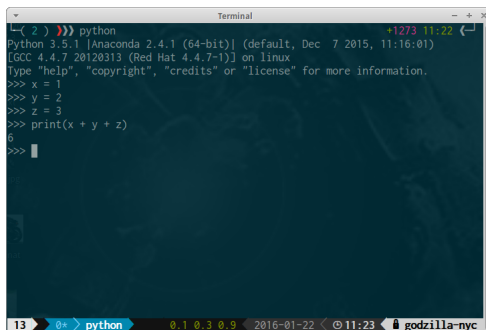
Here we'll interact with Python using **Jupyter notebooks**

- A browser based front end to Python, Julia, R, etc.
- Stores output as well as input
- Allows for rich text, graphics, etc.
- Easy to run remotely on servers / in cloud

But before then let's quickly review other options

One (not very good) option is the plain **Python REPL**

- REPL = read, eval, print loop

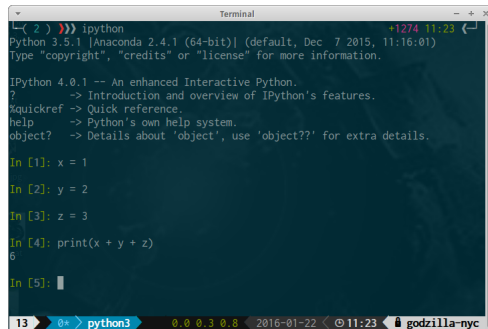
A terminal window titled "Terminal" with standard window controls (-, +, x). The terminal shows a prompt "(2) >>>" followed by the command "python". The output displays the Python version (3.5.1), the environment (Anaconda 2.4.1), and the system (Linux). It then shows a series of assignments: x = 1, y = 2, z = 3, followed by a print statement "print(x + y + z)" which outputs "6". The prompt ">>>" is shown again. At the bottom of the terminal, there is a status bar with "13", a "python" icon, version numbers "0.1 0.3 0.9", a date "2016-01-22", a time "11:23", and a username "godzilla-nyc".

```
(2 ) >>> python
Python 3.5.1 [Anaconda 2.4.1 (64-bit)] (default, Dec 7 2015, 11:16:01)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1
>>> y = 2
>>> z = 3
>>> print(x + y + z)
6
>>>
```

Open up a terminal (cmd for Windows) and type `python`

IPython

A nicer Python REPL with support for shell commands, etc.



```
Terminal
Python 3.5.1 [Anaconda 2.4.1 (64-bit)] (default, Dec 7 2015, 11:16:01)
Type "copyright", "credits" or "license()" for more information.

IPython 4.0.1 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]: x = 1
In [2]: y = 2
In [3]: z = 3
In [4]: print(x + y + z)
6
In [5]:
```

Open up a terminal (cmd for Windows) and type `ipython`

Sometimes it's better to write all the commands in a **text file**

2 x = 1
3 y = 2
4 z = 3
5 print(x + y + z)
6
7
8
9
10
11
12
13
14
15
16

NORMAL ~/temp/test.py unix < utf-8 < python 100% 16:1
"/temp/test.py" 16L, 47C written

...and then run it through the interpreter

some_file.py

```
x = 1  
y = 2  
z = 3  
print(x + y + z)
```

```
>>> x = 1  
>>> y = 2  
>>> z = 3  
>>> print(x + y + z)
```

Python Interpreter



Python Virtual Machine



Operating System

→ 6

Here's a text editor (vim) on one side and IPython on the other

```

51     x0 = x1
52
53     return x1
54
55 @jit(nopython=True)
56 def updated_function(n1, n2):
57     hn1 = h(n1)
58     hn2 = h(n2)
59     if n1 <= 0.5 and n2 <= 0.5:
60         new_n1 = delta * (0.5 * theta + (1 - theta) * n1)
61         new_n2 = delta * (0.5 * theta + (1 - theta) * n2)
62     elif n1 >= hn2 and n2 >= hn1:
63         new_n1 = delta * n1
64         new_n2 = delta * n2
65     elif n1 > 0.5 and n2 <= hn1:
66         new_n1 = delta * n1
67         new_n2 = delta * (theta * hn1 + (1 - theta) * n2)
68     else:
69         new_n1 = delta * (theta * hn2 + (1 - theta) * n1)
70         new_n2 = delta * n2
71     return new_n1, new_n2
72
73
74 def plot_trajectory(n1, n2, k):
75     """
76     Plot trajectory of length k from (n1, n2).
77     """
78     fig, ax = plt.subplots()
79     traj = np.empty(k)
80     for i in range(k):
81         new_n1, new_n2 = updated_function(n1, n2)
82         n1, n2 = new_n1, new_n2
83         traj[i] = abs(n1 - n2)
84     ax.plot(traj, 'b-')
NORMAL /matsuyama_synchronization.py < python 72% 84:5
12 > python3

```

```

Terminal
~/john/sync_dir/teaching/nyu/need_for_speed
~/s/t/n/need_for_speed ) @godzilla-nyc
( 2 ) >>> ipython
Python 3.5.1 [Anaconda 2.4.1 (64-bit)] (default, Dec 7 2015, 11:16:01)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.1 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]:

```

(Linux terminal, running through tmux)

Jupyter Notebooks

Wald_Friedman (unsaved changes)

File Edit View Insert Cell Kernel Help Python 3

A Problem that stumped Milton Friedman

(and that Abraham Wald solved by inventing sequential analysis)

By [Chase Coleman](#) and [Thomas J. Sargent](#)

We begin by importing some packages called by the code that we will be using in this notebook.

```
In [2]: import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate as interp
import scipy.stats as st
import seaborn as sb
import quantecon as qe
from ipywidgets import interact, widgets

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```
In [3]: #-Download supporting Wald_Friedman_utils.py file from GitHub-#
qe.fetch_nb_dependencies(["Wald_Friedman_utils.py"])
from Wald_Friedman_utils import *
```

Fetching file: Wald_Friedman_utils.py

To show

- Editing modes, execution
 - Markdown
 - Inline figures
 - Language agnostic
-
- Ref: quant-econ.net/py/getting_started.html

An Easy Python Program

Next step: write and pick apart small Python program

Notes

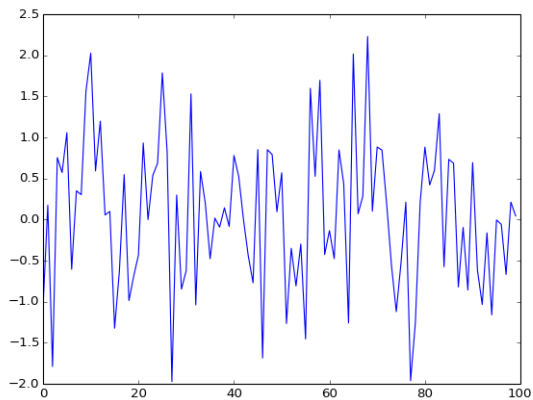
1. Source = quant-econ.net/python_by_example.html
2. Like all first programs, to some extent contrived
3. We focus as much as possible on pure Python

Example: Plotting a White Noise Process

Suppose we want to simulate and plot

$$\epsilon_0, \epsilon_1, \dots, \epsilon_T \quad \text{where} \quad \{\epsilon_t\} \stackrel{\text{iid}}{\sim} N(0, 1)$$

In other words, we want to generate figures like this



A first pass (from quant-econ.net/py/python_by_example.html)

```
1 from random import normalvariate
2 import matplotlib.pyplot as plt
3 ts_length = 100
4 epsilon_values = []    # An empty list
5 for i in range(ts_length):
6     e = normalvariate(0, 1)
7     epsilon_values.append(e)
8 plt.plot(epsilon_values, 'b-')
9 plt.show()
```

Import Statements

First, consider the lines

```
1 from random import normalvariate
2 import matplotlib.pyplot as plt
```

Here matplotlib and random are two separate **modules**

- module = file containing Python code

Importing a module causes Python to

- run the code in those files
- set up a matching "namespace" to store variables

```
>>> import random
>>> random.normalvariate(0, 1)
-0.12451500570438317
>>> random.uniform(-1, 1)
0.35121616197003336
```

You can also just import the names directly, like so

```
>>> from random import normalvariate, uniform
>>> normalvariate(0, 1)
-0.38430990243287594
>>> uniform(-1, 1)
0.5492316853602877
```

Lists

Statement `epsilon_values = []` creates an empty list

Lists: a Python data structure used to group objects

```
>>> x = [10, 'foo', False]
>>> type(x)
<type 'list'>
```

Note that different types of objects can be combined in a single list

Adding a value to a list: `list_name.append(some_value)`

```
>>> x
[10, 'foo', False]
>>> x.append(2.5)
>>> x
[10, 'foo', False, 2.5]
```

- `append()` is an example of a **method**
- method = a function "attached to" an object

Another example of a list method:

```
>>> x
[10, 'foo', False, 2.5]
>>> x.pop()
2.5
>>> x
[10, 'foo', False]
```

An example of a string method:

```
>>> s = 'foobar'
>>> s.upper()
'FOOBAR'
```

As in C, Java, etc., lists in Python are zero based

```
>>> x
[10, 'foo', False]
>>> x[0]
10
>>> x[1]
'foo'
```

The `range()` function creates a sequential list of integers

```
>>> range(4)
[0, 1, 2, 3]
>>> range(5)
[0, 1, 2, 3, 4]
```

Note: `range(n)` gives indices of list `x` when `len(x)` equals `n`

The for Loop

Consider again these lines from `test_program_1.py`

```
5 for i in range(ts_length):  
6     e = normalvariate(0, 1)  
7     epsilon_values.append(e)  
8 plt.plot(epsilon_values, 'b-')
```

Lines 6–7 are the **code block** of the `for` loop

Reduced indentation signals lower limit of the code block

Comments on Indentation

In Python **all** code blocks are delimited by indentation

This is a **good** thing (more consistency, less clutter)

But tricky at first, so please remember

- Line before start of code block always ends in a colon
- All lines in a code block must have same indentation
- The Python standard is 4 spaces—please use it
- Tabs and spaces are different

Using the Scientific Libraries

In fact the scientific libraries will do all this more efficiently

For example, try

```
>>> from numpy.random import randn
>>> epsilon_values = randn(3)
>>> epsilon_values
array([-0.15591709, -0.67383208, -0.45932047])
```

Exercise

Simulate and plot the correlated time series

$$x_{t+1} = \alpha x_t + \epsilon_{t+1} \quad \text{where} \quad x_0 = 0 \quad \text{and} \quad t = 0, \dots, T$$

Here $\{\epsilon_t\} \stackrel{\text{iid}}{\sim} N(0, 1)$

In your solution, restrict your import statements to

```
from random import normalvariate
import matplotlib.pyplot as plt
```

Set $T = 200$ and $\alpha = 0.9$

Solution

```
import matplotlib.pyplot as plt
from random import normalvariate

alpha = 0.9
ts_length = 200
x = 0

x_values = []
for i in range(ts_length):
    x_values.append(x)
    x = alpha * x + normalvariate(0, 1)
plt.plot(x_values, 'b-')
plt.show()
```

Further Resources

See <http://quant-econ.net> for

- Basic instructions
- Python lectures
- Julia lectures
- Code related to econ