Scientific Computing in Python and Julia

Part 1: Introduction to Python

John Stachurski Pablo Winant Sébastien Villemot

June 2014

Before We Start

Have you installed Anaconda?

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

Installed Anaconda a while ago (more than one month)?

- In a terminal type conda update anaconda

Problems?

- Go to http://www.wakari.io
- Sign up for a free plan

Today

- Morning session: Intro to Python with John Stachurski
- Mid afternoon session: HPC with Python by Pablo Winant
- Late afternoon session: Julia with Sébastien Villemot

Topics for the Morning Session

- Getting started
  - How to run Python programs
- Learning Python
  - Basic syntax
  - programming techniques
- Scientific programming
  - The scientific libraries
  - Graphics
- Problems
  - Exercise on Markov chains

Resources

See http://quant-econ.net/resources.html for

- Basic instructions
- Lecture PDFs
- etc.

What’s Python?

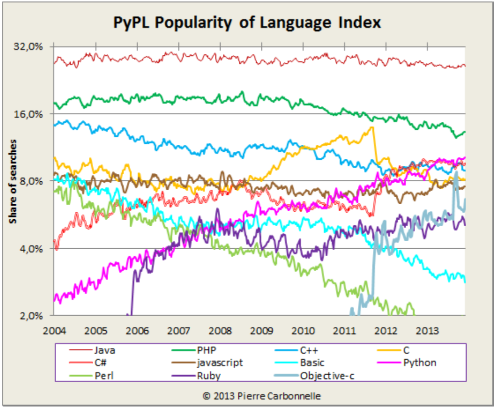
```
>>> print "Hello world"
Hello world
```

A general purpose programming language

Free and open source

Used extensively by

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


PyPL Popularity of Language Index
32,0%
16,0%
8,0%
4,0%
2,0%
Share of searches
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
Java
PHP
C++
C
C#
javascript
Basic
Python
Perl
Ruby
Objective-c
© 2013 Pierre Carbonnelle

Python is noted for

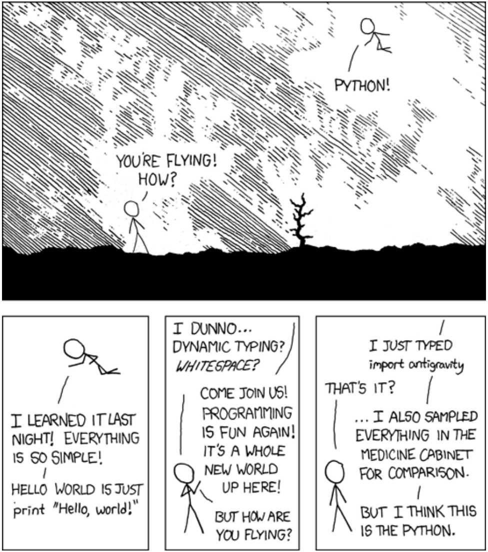
- Elegant, modern design
- Clean syntax, readability
- High productivity

Often used to teach first courses in comp sci and programming

- MIT, Udacity, edX, etc.

Example of readability

```
»> 1 < 2 and 'f' in 'foo'
True
»> 1 < 2 or 'g' in 'foo'
True
»> 'g' not in 'foo'
True
```


YOU'RE FLYING! HOW?
PYTHON!
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Scientific Programming

Rapid adoption by the scientific community

- Artifical 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

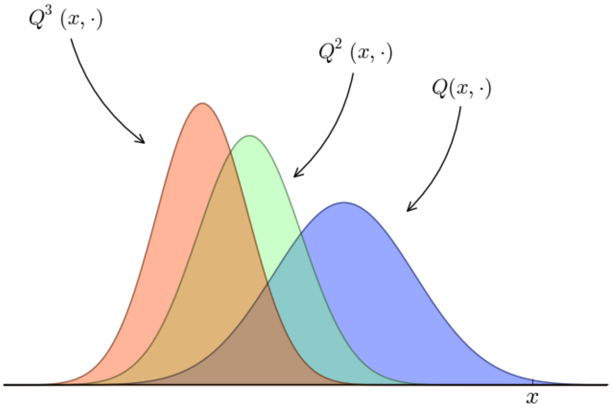
```
>>> import numpy as np
>>> a = np.random.randn(10000)
>>> a.mean()
0.0020109779347995344
>>> a.std()
1.0095758844793006
```

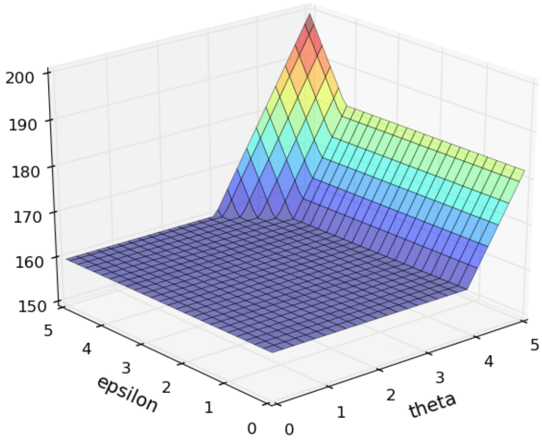
SciPy Example: Calculate

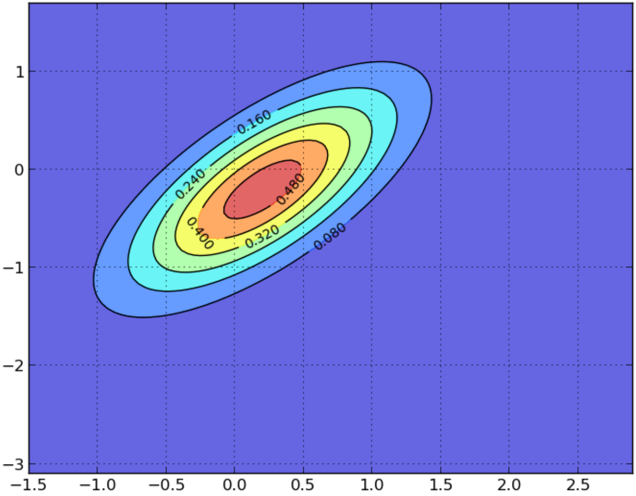
$$\int_{-2}^{2} \phi(z)dz \quad \text{where} \quad \phi \sim N(0,1)$$

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

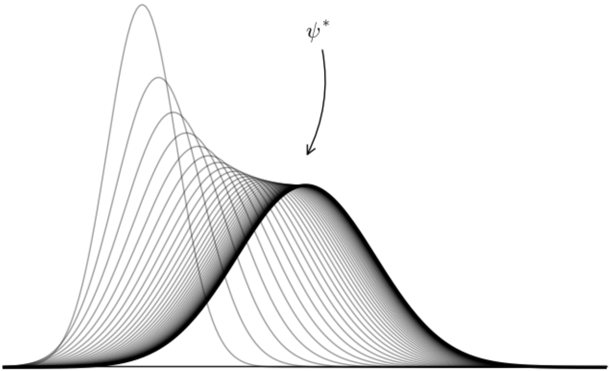
Matplotlib examples


$Q^3\ (x, \cdot)$
$Q^2\ (x, \cdot)$
$Q(x, \cdot)$
x


200
190
180
170
160
150
5
4
3
2
1
0
epsilon
0
1
2
3
4
5
theta

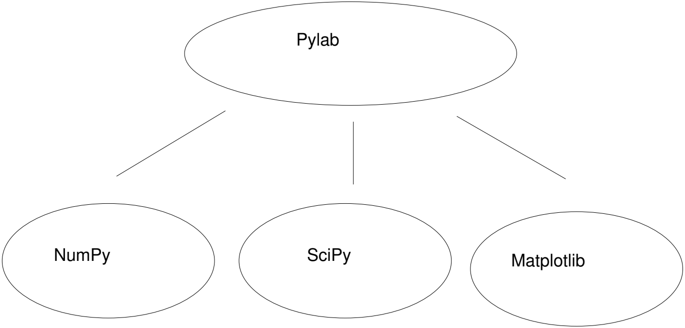


0.160
0.240
0.480
0.400
0.320
0.080
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


ψ^*

Pylab

Pylab combines core functionality of the big three



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

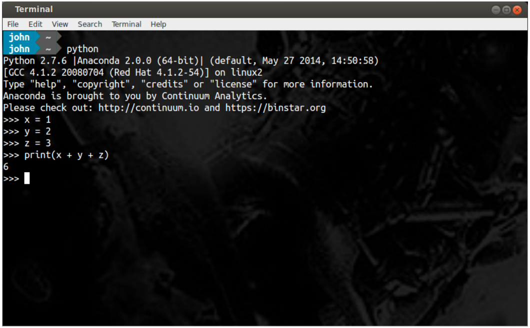
Other Scientific Tools

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.

Interacting with Python

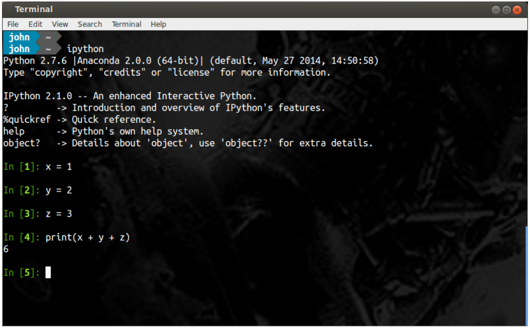
Python commands are read in through the **Python interpreter**



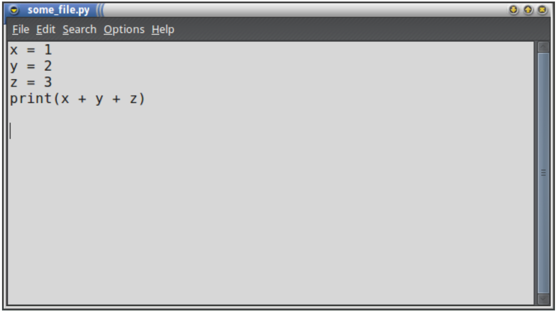
Open up a terminal (cmd for Windows) and type python

IPython Shell

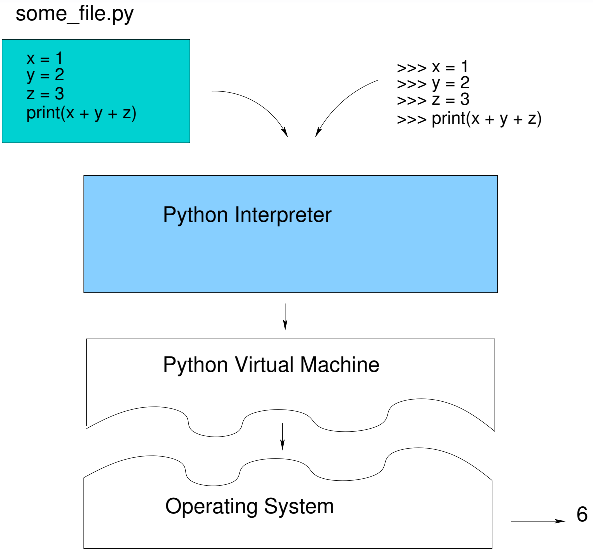
A nicer Python shell with support for file operations, timing code, etc.

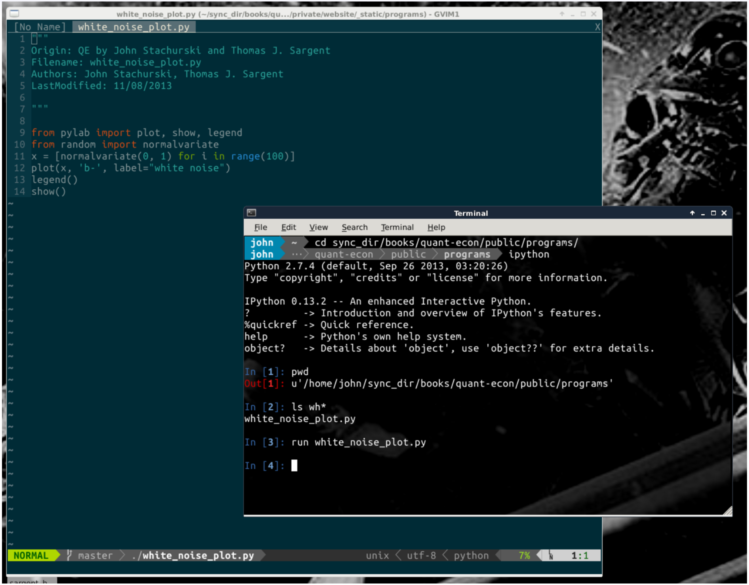


Sometimes it's better to write all the commands in a **text file**



...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


white_noise_plot.py (~/sync_dir/books/qu.../private/website/_static/programs) - GVIM1
[No Name] white_noise_plot.py
"""
Origin: QE by John Stachurski and Thomas J. Sargent
Filename: white_noise_plot.py
Authors: John Stachurski, Thomas J. Sargent
LastModified: 11/08/2013
"""
from pylab import plot, show, legend
from random import normalvariate
x = [normalvariate(0, 1) for i in range(100)]
plot(x, 'b-', label="white noise")
legend()
show()
Terminal
File Edit View Search Terminal Help
john ~ cd sync_dir/books/quant-econ/public/programs/
john ... quant-econ public programs ipython
Python 2.7.4 (default, Sep 26 2013, 03:20:26)
Type "copyright", "credits" or "license" for more information.
IPython 0.13.2 -- 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]: pwd
Out[1]: u'/home/john/sync_dir/books/quant-econ/public/programs'
In [2]: ls wh*
white_noise_plot.py
In [3]: run white_noise_plot.py
In [4]:
NORMAL master ./white_noise_plot.py unix utf-8 python 7% 1:1

Programming Setups

Options for interacting with Python

- (I)Python shell plus text editor
- Spyder or other IDEs
- IPython Notebook

We will focus on the latter

The IPython Notebook

- Starting the notebook
- Shift+Enter and multimodal editing
- Running short programs
- Tabs, on-line help
- Sharing is caring

- Ref: quant-econ.net/getting_started.html

An Easy Python Program

Next step: write and pick apart small Python program

Notes

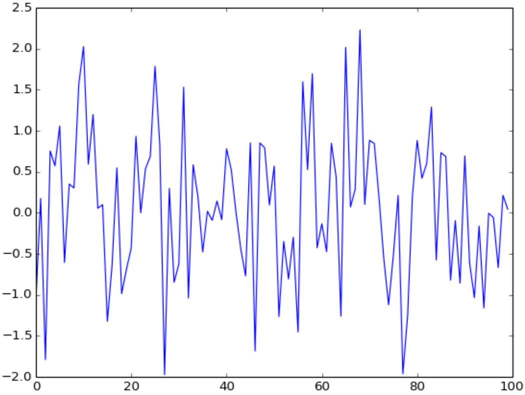
1. Source = quant-econ.net/python_by_example.html
2. Like all first programs, to some extent contrived
3. We focus as much as possible on pure Python

Example: Plotting a White Noise Process

Suppose we want to simulate and plot

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

In other words, we want to generate figures like this



Here's a first pass

You can cut and paste from quant-econ.net/python_by_example.html

```
import pylab
from random import normalvariate
ts_length = 100
epsilon_values = []   # An empty list
for i in range(ts_length):
    e = normalvariate(0, 1)
    epsilon_values.append(e)
pylab.plot(epsilon_values, 'b-')
pylab.show()
```

Import Statements

First, consider the lines

```
import pylab
from random import normalvariate
```

Here pylab and random are two separate ”modules”

- module = file or hierachy of linked files containing Python code
- Importing causes Python to run the code in those files

Option 1:

```
>> import random
>> random.normalvariate(0, 1)
-0.12451500570438317
>> random.uniform(-1, 1)
0.3512161619700336
```

Option 2:

```
>> 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 pylab.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

While Loops

Here's the same program with a while loop (test_program_2.py)

```
import pylab
from random import normalvariate
ts_length = 100
epsilon_values = []
i = 0
while i < ts_length:
    e = normalvariate(0, 1)
    epsilon_values.append(e)
    i = i + 1
pylab.plot(epsilon_values, 'b-')
pylab.show()
```

User-Defined Functions

Now let's go back to the for loop

—but restructure our program to make the logic clearer

To this end, we will break our program into two parts:

1. A *user-defined function* that generates a list of random variables
2. The main part of the program, which
    1. calls this function to get data
    2. plots the data

test_program_3.py

```
import pylab
from random import normalvariate

def generate_data(n):
    epsilon_values = []
    for i in range(n):
        e = normalvariate(0, 1)
        epsilon_values.append(e)
    return epsilon_values

data = generate_data(100)
pylab.plot(data, 'b-')
pylab.show()
```

Our function generate_data() is rather limited

Let’s make it more flexible by giving it the ability to return either

- standard normals, or
- uniform rvs on $(0, 1)$

This is done in test_program_4.py

```python
import pylab
from random import normalvariate, uniform

def generate_data(n, generator_type):
    epsilon_values = []
    for i in range(n):
        if generator_type == 'U':
            e = uniform(0, 1)
        else:
            e = normalvariate(0, 1)
        epsilon_values.append(e)
    return epsilon_values

data = generate_data(100, 'U')
pylab.plot(data, 'b-')
pylab.show()
```

In fact we can get rid of the conditionals all together

Method: pass the desired generator type *as a function*

To understand this, consider test_program_6.py

```python
import pylab
from random import normalvariate, uniform

def generate_data(n, generator_type):
    epsilon_values = []
    for i in range(n):
        e = generator_type(0, 1)
        epsilon_values.append(e)
    return epsilon_values

data = generate_data(100, uniform)
pylab.plot(data, 'b-')
pylab.show()
```

List Comprehensions

We can also simplify the for loop by using a **list comprehension**

```
>>> animals = ['dog', 'cat', 'bird']
>>> plurals = [animal + 's' for animal in animals]
>>> plurals
['dogs', 'cats', 'birds']
```

With the list comprehension syntax, we can simplify the lines

```
epsilon_values = []
for i in range(n):
    e = generator_type(0, 1)
    epsilon_values.append(e)
```

into

```
epsilon_values = [generator_type(0, 1) for i in range(n)]
```

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(4)
>>> epsilon_values
array([-0.15591709, -1.42157676, -0.67383208, -0.45932047])
```

Exercise

Simulate and plot the correlated time series

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

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

In your solution, restrict your import statements to

```
from pylab import plot, show
from random import normalvariate
```

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

Solution

```
from pylab import plot, show, legend
from random import normalvariate

alpha = 0.9
ts_length = 200
current_x = 0

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