

Econometric Society Workshop on Scientific Computing with Python and Julia

John Stachurski

June 2016

Agenda

1. **09:00am – 10:15am** Overview & Introduction to Python
2. **10:15am – 10:45am** Coffee break
3. **10:45am – 12:00pm** Introduction to Julia

Personnel

NYU:

- Felipe Alvez
- Chase Coleman
- Pierre Mabilie
- Matthew McKay
- John Stachurski

FRBNY:

- Erica Moszkowski
- Pearl Li

Aims

1. An overview and comparisons
2. Help with installation
3. Lower fixed costs to getting started

Sponsors



See

- <http://quantecon.org/>
- <http://www.numfocus.org/>

Resources

- http://quantecon.org/econometric_society_workshop.html
- entry point at bottom of <http://quantecon.org>

Python Set Up

Anaconda

- Python + the main scientific libraries
- Free from <http://continuum.io/downloads>
- Choose the Python 3.5 version
- Make it your default Python distribution

Explore:

- Read the docs at <https://docs.continuum.io/anaconda>
- Try starting `anaconda-navigator` (in a terminal or search apps)

Overview

- What's Python?
- What's Julia?
- Pros and cons
- Which to choose?

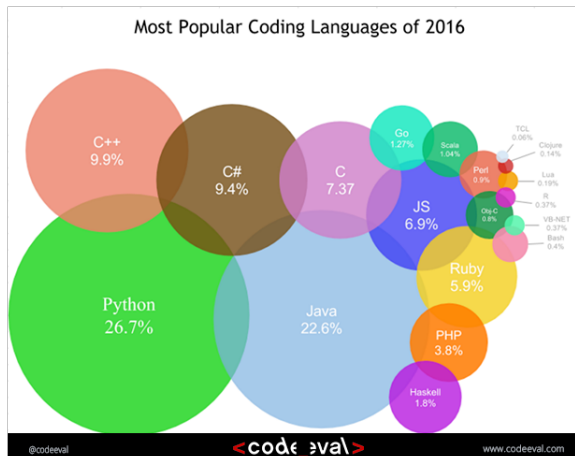
What's Python?

Modern, high level, open source, general purpose programming language

Used extensively by

- Tech firms (YouTube, Dropbox, Reddit, etc., etc.)
- Hedge funds and finance industry
- Scientists (academia, NASA, CERN, etc.)
- etc., etc.

Very popular in “data science” / machine learning



Strengths

- Modern design
- Clean syntax, readability
- Great libraries
- Friendly community
- High productivity

What's Julia?

Modern, high level, open source, scientific programming language

Strengths:

- Nice design
- High productivity...
- and high performance!

Julia vs Python

Who will benefit more from Julia?

- Focused on scientific programming
- Write your own algorithms
- Need optimization / high performance

Negatives

- Some instability
- Some libraries still under development

Who will benefit more from Python?

- Care about stability and high productivity
- Diverse coding needs
- Use a lot of data / empirics

Negatives:

- Optimization is a bit more work than Julia

Python/Julia vs C/Fortran

But isn't C/Fortran faster?

Not really...

And what matters is productivity

Python and Julia help us maximize productivity, so

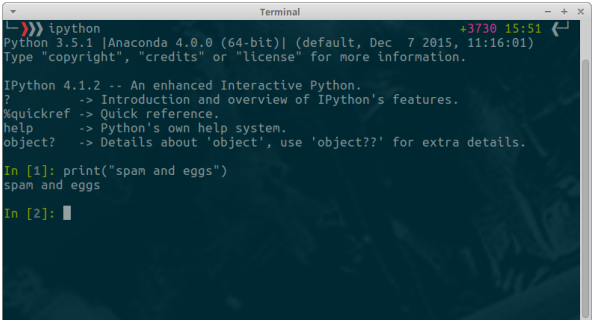
- Write your programs in Python or Julia
- Optimize your hot loops

Interacting with Python

Options

- The IPython REPL
- IDEs like Spyder
- Text editor (e.g., [Atom](#) or [Sublime](#)) plus the REPL
- Jupyter notebooks

IPython REPL (Read Eval Print Loop)

A terminal window titled "Terminal" with standard window controls. The prompt is a red prompt character followed by three green chevrons. The text in the terminal is as follows:

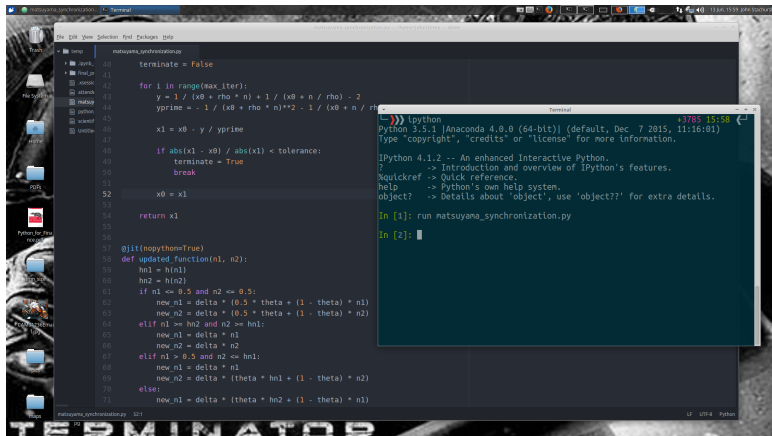
```
>>> ipython
Python 3.5.1 |Anaconda 4.0.0 (64-bit)| (default, Dec  7 2015, 11:16:01)
Type "copyright", "credits" or "license" for more information.

IPython 4.1.2 -- 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]: print("spam and eggs")
spam and eggs

In [2]:
```

Atom text editor + IPython REPL

A screenshot showing the Atom text editor and the IPython REPL. The Atom editor window displays a Python script named 'matsuyama_synchronization.py'. The script contains a function 'updated_function' and a loop that iterates over a range of values. The IPython REPL window is open in the foreground, showing the prompt 'In [1]:' and the command 'run matsuyama_synchronization.py'. The output of the script is displayed in the REPL, showing the value of 'x1' as 0.0 and the value of 'yprime' as 1.0. The background of the Atom editor window shows a 'TERMINATOR' wallpaper.

```
1 terminate = False
2
3 for i in range(max_iter):
4     y = 1 / (x0 + rho * n) + 1 / (x0 + n / rho) - 2
5     yprime = - 1 / (x0 + rho * n)**2 - 1 / (x0 + n / rho)
6
7     x1 = x0 - y / yprime
8
9     if abs(x1 - x0) / abs(x1) < tolerance:
10         terminate = True
11         break
12
13 x0 = x1
14
15 return x1
16
17 @jit(nopython=True)
18 def updated_function(n1, n2):
19     hn1 = h(n1)
20     hn2 = h(n2)
21
22     if n1 <= 0.5 and n2 <= 0.5:
23         new_n1 = delta * (0.5 * theta + (1 - theta) * n1)
24         new_n2 = delta * (0.5 * theta + (1 - theta) * n2)
25     elif n1 >= hn2 and n2 >= hn1:
26         new_n1 = delta * n1
27         new_n2 = delta * n2
28     elif n1 > 0.5 and n2 <= hn1:
29         new_n1 = delta * n1
30         new_n2 = delta * (theta * hn1 + (1 - theta) * n2)
31     else:
32         new_n1 = delta * (theta * hn2 + (1 - theta) * n1)
```

```
In [1]: run matsuyama_synchronization.py
In [2]:
```

Spyder

Spyder (Python 3.5)

File Edit Search Source Run Debug Consoles Tools View Help

Editor: /home/john/sync_dir/teaching/nyu/more_python/plot_example_5.py

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from scipy.stats import norm
4 from random import uniform
5 num_rows, num_cols = 2, 3
6 fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 8))
7 for i in range(num_rows):
8     for j in range(num_cols):
9         m, s = uniform(-1, 1), uniform(1, 2)
10        x = norm.rvs(loc=m, scale=s, size=100)
11        axes[i, j].hist(x, alpha=0.6, bins=20)
12        t = r'$\mu = {0:.1f}, \sigma = {1:.1f}'.format(m, s)
13        axes[i, j].set_title(t)
14        axes[i, j].set_xticks([-4, 0, 4])
15        axes[i, j].set_yticks([])
16 plt.show()
17
18

```

Variable explorer

Name	Type	Size	Value
axes	object	(2, 3)	array([[<matplotlib.axes._su...
i	int	1	1
j	int	1	2
m	float	1	0.2753487585185199
num_cols	int	1	3
num_rows	int	1	2
s	float	1	1.177823269914244
t	str	1	'\$\mu = 0.3, \sigma = 1.2\$'

Variable explorer File explorer

IPython console

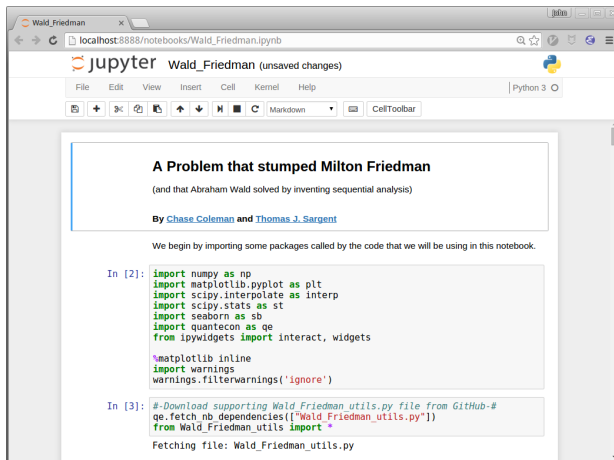
Console 1/A

In [2]:

Console History log IPython console

Permissions: RW End-of-lines: LF Encoding: UTF-8-GSS54B Line: 3 Column: 3 Memory: 23 %

Jupyter notebooks



Jupyter Notebooks

Let's focus on Jupyter notebooks

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

References and examples:

- Ref: http://quant-econ.net/py/getting_started.html
- Examples: <http://notebooks.quantecon.org/>

An Easy Python Program

Next step: write and pick apart small Python program

Aim: To simulate and plot

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

Notes

1. Like all first programs, to some extent contrived
2. We focus as much as possible on pure Python

First pass

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

ts_length = 100
epsilon_values = []
for i in range(ts_length):
    e = normalvariate(0, 1)
    epsilon_values.append(e)
plt.plot(epsilon_values, 'b-')
plt.show()
```

- http://quant-econ.net/py/python_by_example.html

Import Statements

Importing a module causes Python to

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

```
In [1]: import random
```

```
In [2]: random.normalvariate(0, 1)
```

```
Out[2]: 0.18415513098683509
```

```
In [3]: random.uniform(0, 1)
```

```
Out[3]: 0.11883707116624409
```


You can also just import the names directly:

```
In [4]: from random import normalvariate, uniform
```

```
In [5]: normalvariate(0, 1)
```

```
Out[5]: -0.7248742909651001
```

```
In [6]: uniform(0, 4)
```

```
Out[6]: 0.24696251658189228
```

Lists

Statement `epsilon_values = []` creates an empty list

Lists: a Python data structure used to group objects

```
In [7]: x = [10, 'foo']
```

```
In [8]: type(x)
```

```
Out[8]: list
```

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

Adding a value to a list:

```
In [10]: x
```

```
Out[10]: [10, 'foo']
```

```
In [11]: x.append(0.5)
```

```
In [12]: x
```

```
Out[12]: [10, 'foo', 0.5]
```

Here `append()` is an example of a **list method**

- a function “attached to” an object

Another example of a list method:

```
In [13]: x
```

```
Out[13]: [10, 'foo', 0.5]
```

```
In [14]: x.pop()
```

```
Out[14]: 0.5
```

```
In [15]: x
```

```
Out[15]: [10, 'foo']
```

Each data type has its own methods

```
In [1]: s = 'foobar'
```

```
In [2]: type(s)
```

```
Out[2]: str
```

```
In [3]: s.capitalize()
```

```
Out[3]: 'Foobar'
```

```
In [4]: s.upper()
```

```
Out[4]: 'FOOBAR'
```

To see all string methods, type `s.` and hit “Tab”

Loops

Consider again these lines from `test_program_1.py`

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

Lines 7–8 are the **code block** of the `for` loop

Reduced indentation signals end of code block

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

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

Solution

```
import matplotlib.pyplot as plt
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)
plt.plot(x_values, 'b-')
plt.show()
```
