Modern Tools for Computational Economics

John Stachurski

October 2016

Aims

1. An overview and comparisons
2. Help with installation
3. Lower fixed costs to getting started
4. Provide resources for further study

Thanks

1. **Keio University** (Ippei Fujiwara, Sagiri Kitao)
2. Sponsors supporting `quantecon.org`

See

- `http://quantecon.org/`
- `http://www.numfocus.org/`

Japan is great

Team / Resources

Will be assisted by Matt McKay

- Pythonista
- Computational development economist

Workshop homepage:

- https://github.com/QuantEcon/Keio_workshop

Team / Resources

Will be assisted by Matt McKay

- Pythonista
- Computational development economist

Workshop homepage:

- https://github.com/QuantEcon/Keio_workshop

Software options for this workshop

1. Install Anaconda Python (Python only)
2. Install Julia + Anaconda Python
3. Use our server http://workshop.quantecon.org:8000/

Remark: We'll use Julia through Jupyter, from Anaconda Python

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

Julia Set Up

Get from http://julialang.org/downloads/

Extra instructions

- http://julialang.org/downloads/platform.html

Testing, testing

Try starting

- Julia REPL (REPL = Read Eval Print Loop)
- Python and IPython REPLs

Use menus or terminal

- `terminal` in UNIX/macOS and `cmd` in Windows

Overview

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

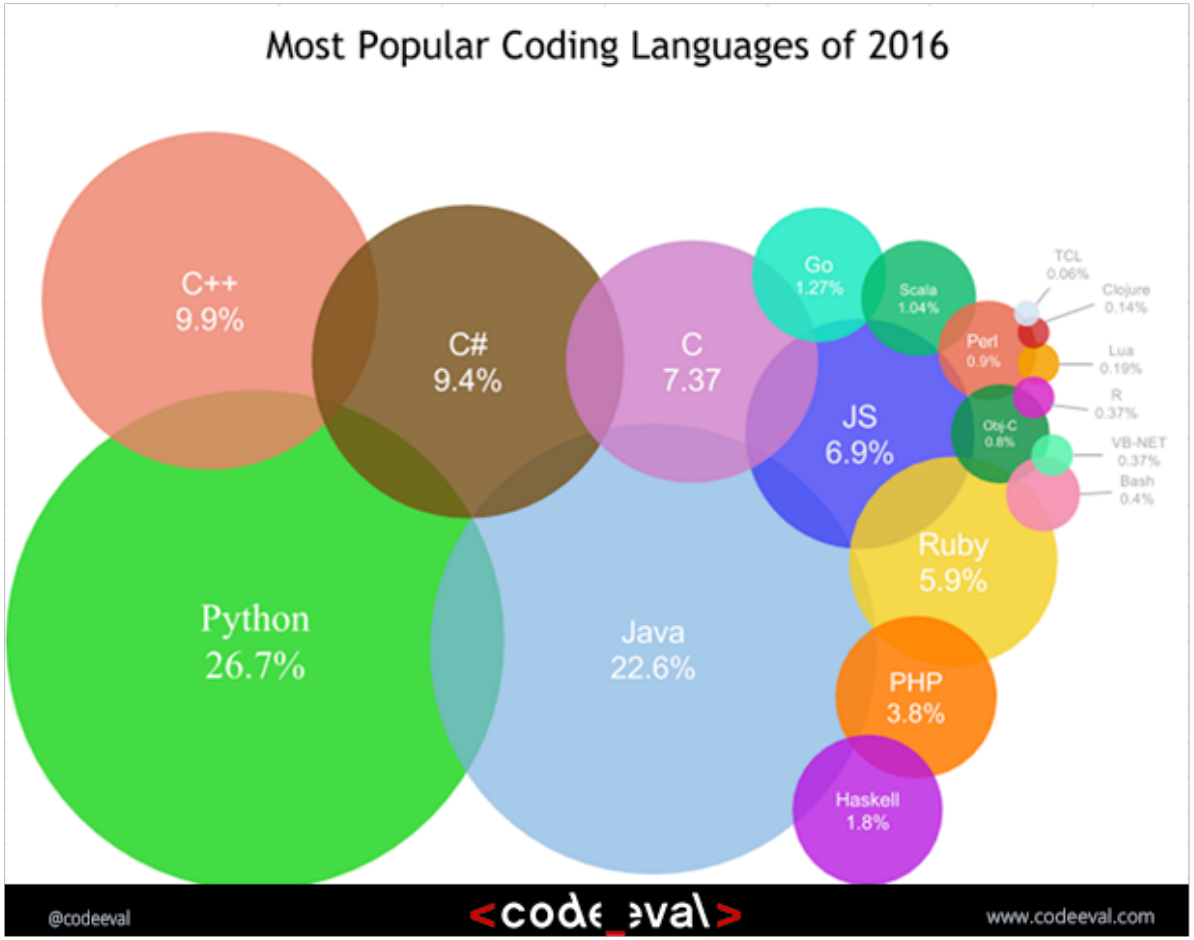
What's Python?

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


Most Popular Coding Languages of 2016
C++
9.9%
C#
9.4%
C
7.37
Go
1.27%
Scala
1.04%
TCL
0.06%
Clojure
0.14%
Perl
0.9%
Lua
0.19%
R
0.37%
JS
6.9%
Obj-C
0.8%
VB-NET
0.37%
Bash
0.4%
Ruby
5.9%
Python
26.7%
Java
22.6%
PHP
3.8%
Haskell
1.8%
@codeeval
<code_eval>
www.codeeval.com

Strength 1: Intelligent, modern design

- Often used to teach first courses in comp sci
  - MIT
  - Stanford
  - Chicago, etc.

Strength 2: Simple, readable syntax

```
for name in name_list:
    if name not in list_of_users:
        print("Your name is not found")
```

Strength 1: Intelligent, modern design

- Often used to teach first courses in comp sci
  - MIT
  - Stanford
  - Chicago, etc.

Strength 2: Simple, readable syntax

```
for name in name_list:
    if name not in list_of_users:
        print("Your name is not found")
```

Other strengths

- 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

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

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?

Sometimes, but

"**Premature optimization is the root of all evil**"
– Donald Knuth

For $\approx$ 97% of your code, high productivity and clarity are what matter

Example: Astropy

Supports:

- data collection, data analysis, modeling, estimation

Essential to

- all NASA mission pipelines
- national laboritories, academia, industry, etc.

Code base:

- 325,000 lines of Python code
- 14,000 lines of C code

Example: Astropy

Supports:

- data collection, data analysis, modeling, estimation

Essential to

- all NASA mission pipelines
- national laboritories, academia, industry, etc.

Code base:

- 325,000 lines of Python code
- 14,000 lines of C code

Best scientific programming paradigm is

1. write your program using a high level language
2. test and profile
3. optimize **hot loops** only

Options for step 3

- call out to Fortran / C
- fast loops **within** Python and Julia!
  - unlike MATLAB / Octave / GAUSS / etc

Best scientific programming paradigm is

1. write your program using a high level language
2. test and profile
3. optimize **hot loops** only

Options for step 3

- call out to Fortran / C
- fast loops **within** Python and Julia!
  - unlike MATLAB / Octave / GAUSS / etc

Best scientific programming paradigm is

1. write your program using a high level language
2. test and profile
3. optimize **hot loops** only

Options for step 3

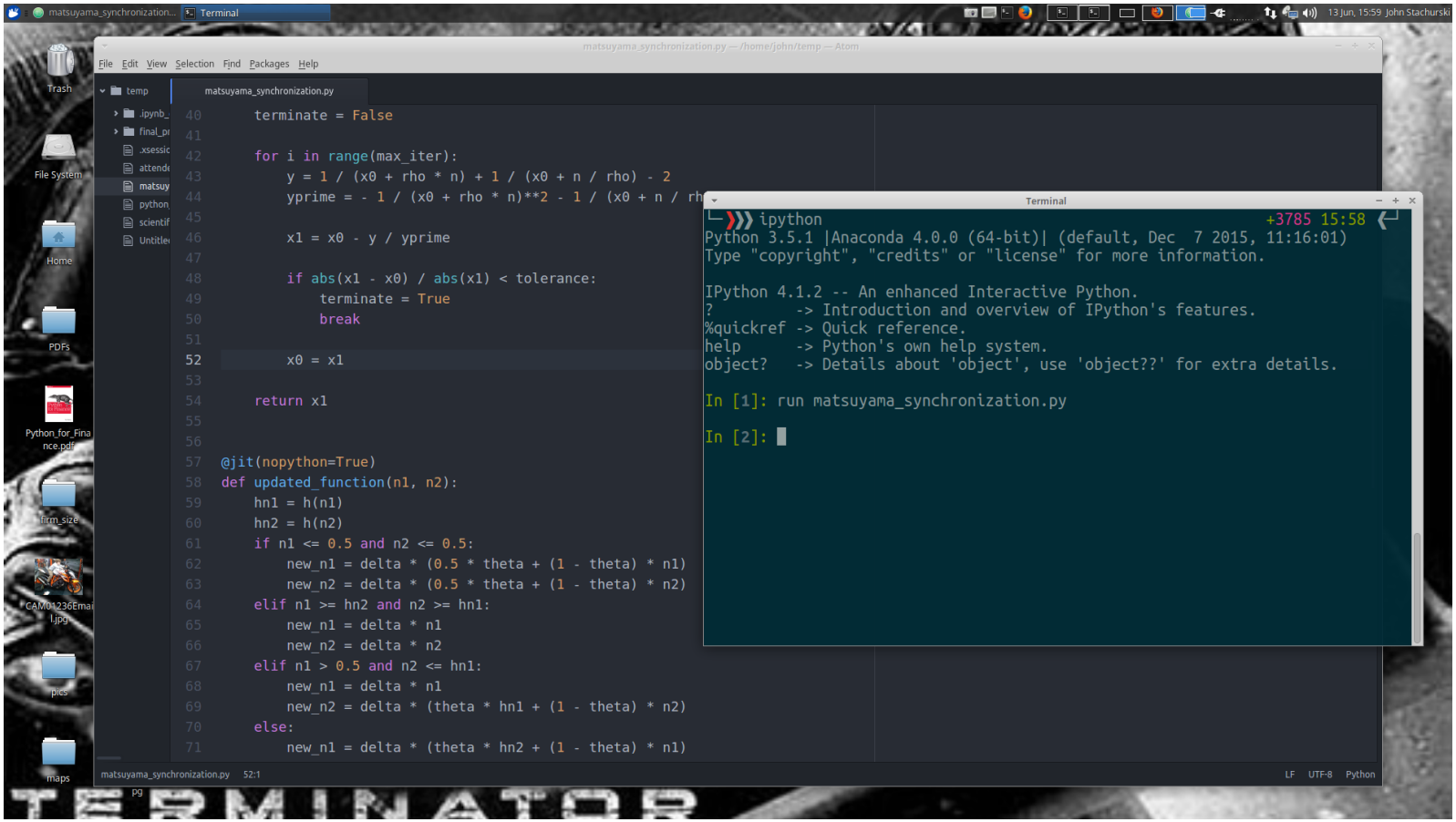
- call out to Fortran / C
- fast loops **within** Python and Julia!
  - unlike MATLAB / Octave / GAUSS / etc

Interacting with Python / Julia

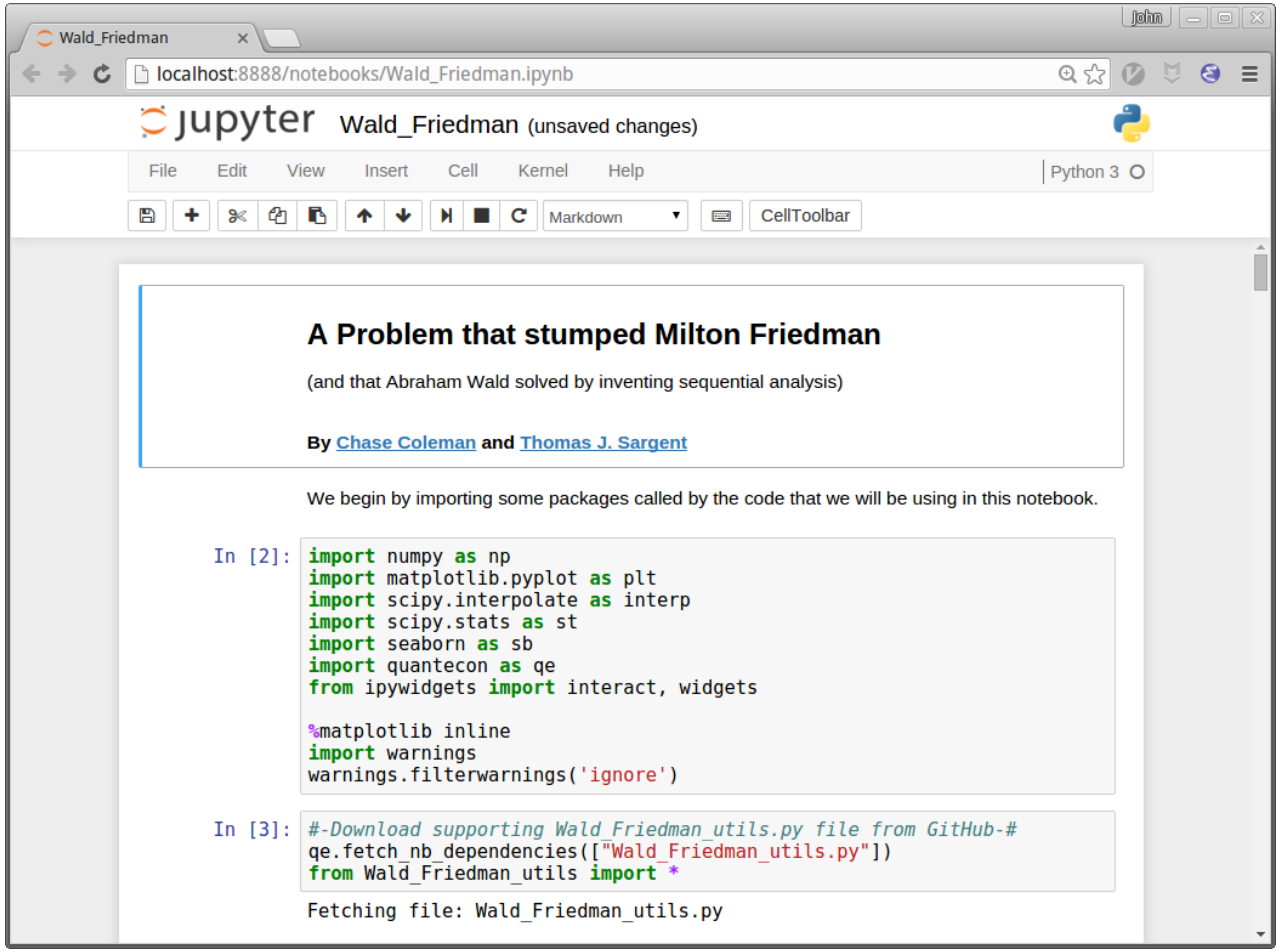
Options

1. The REPLs
2. Text editors (e.g., Atom or Sublime) plus the REPL
3. IDEs like Spyder and Juno
4. Jupyter notebooks

Atom + REPL


matsuyama_synchronization.py — /home/john/temp — Atom
File Edit View Selection Find Packages Help
temp
matsuyama_synchronization.py
terminate = False
for i in range(max_iter):
y = 1 / (x0 + rho * n) + 1 / (x0 + n / rho) - 2
yprime = - 1 / (x0 + rho * n)**2 - 1 / (x0 + n / rho
x1 = x0 - y / yprime
if abs(x1 - x0) / abs(x1) < tolerance:
terminate = True
break
x0 = x1
return x1
@jit(nopython=True)
def updated_function(n1, n2):
hn1 = h(n1)
hn2 = h(n2)
if n1 <= 0.5 and n2 <= 0.5:
new_n1 = delta * (0.5 * theta + (1 - theta) * n1)
new_n2 = delta * (0.5 * theta + (1 - theta) * n2)
elif n1 >= hn2 and n2 >= hn1:
new_n1 = delta * n1
new_n2 = delta * n2
elif n1 > 0.5 and n2 <= hn1:
new_n1 = delta * n1
new_n2 = delta * (theta * hn1 + (1 - theta) * n2)
else:
new_n1 = delta * (theta * hn2 + (1 - theta) * n1)
matsuyama_synchronization.py 52:1
LF UTF-8 Python
Terminal
ipython
+3785 15:58
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]: run matsuyama_synchronization.py
In [2]:

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/