Modern Methods in Computational Economics

University of Economics Ho Chi Minh City

John Stachurski

November 2016

Topics

- Modern computational methods with Python and Julia

Assumptions

- Some programming but unfamiliar with Python / Julia

Aims

- Background, overview and comparisons
- Lower fixed costs to getting started
- Provide resources for further study

Topics

- Modern computational methods with Python and Julia

Assumptions

- Some programming but unfamiliar with Python / Julia

Aims

- Background, overview and comparisons
- Lower fixed costs to getting started
- Provide resources for further study

Topics

- Modern computational methods with Python and Julia

Assumptions

- Some programming but unfamiliar with Python / Julia

Aims

- Background, overview and comparisons
- Lower fixed costs to getting started
- Provide resources for further study

Thanks

1. **UEH**
2. Sponsors supporting quantecon.org

See

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

Resources

Workshop homepage:

- https://github.com/QuantEcon/HCMC_workshop_2016

Other

- Cheat sheets: http://cheatsheets.quantecon.org/
- Lectures: http://lectures.quantecon.org/
- Discourse forum: http://discourse.quantecon.org/

Resources

Workshop homepage:

- https://github.com/QuantEcon/HCMC_workshop_2016

Other

- Cheat sheets: http://cheatsheets.quantecon.org/
- Lectures: http://lectures.quantecon.org/
- Discourse forum: http://discourse.quantecon.org/

Software options

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

Notes:

- See workshop home page for more instructions
- Remember that `http://workshop.quantecon.org:8000/` is temporary!

Software options

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

Notes:

- See workshop home page for more instructions
- Remember that http://workshop.quantecon.org:8000/ is temporary!

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 of Scientific Computing

Tasks

- Solve numerical problems
- Produce figures and graphs
- Manipulate data
- Explore (simulate, plot, visualize, etc.)

And sometimes we need **speed**

Overview of Scientific Computing

Tasks

- Solve numerical problems
- Produce figures and graphs
- Manipulate data
- Explore (simulate, plot, visualize, etc.)

And sometimes we need **speed**

The Need for Speed

Maximum speed:

- Optimal use of hardware
- High level of control over calculations / logic

First best = **assembly** / machine code

- Individual instructions at the CPU level

Example here sums $1 + 2$

The Need for Speed

Maximum speed:

- Optimal use of hardware
- High level of control over calculations / logic

First best = **assembly** / machine code

- Individual instructions at the CPU level

Example here sums $1 + 2$

The Need for Speed

Maximum speed:

- Optimal use of hardware
- High level of control over calculations / logic

First best = **assembly** / machine code

- Individual instructions at the CPU level

Example here sums $1 + 2$

Now imagine a large general equilibrium model

And of course you need to optimize for specific hardware

- pipelining
- cache hierarchies
- branch prediction
- coprocessors
- etc.

And then Intel brings out a new processor...

Now imagine a large general equilibrium model

And of course you need to optimize for specific hardware

- pipelining
- cache hierarchies
- branch prediction
- coprocessors
- etc.

And then Intel brings out a new processor...

Now imagine a large general equilibrium model

And of course you need to optimize for specific hardware

- pipelining
- cache hierarchies
- branch prediction
- coprocessors
- etc.

And then Intel brings out a new processor...

Lesson: There's a **trade off** between

- machine speed
- coding efficiency

Low level languages give us **fine grained control**

High level languages give us

- **abstraction**
- **automation** of some tasks
- **natural language** representations

Lesson: There's a **trade off** between

- machine speed
- coding efficiency

Low level languages give us **fine grained control**

High level languages give us

- **abstraction**
- **automation** of some tasks
- **natural language** representations

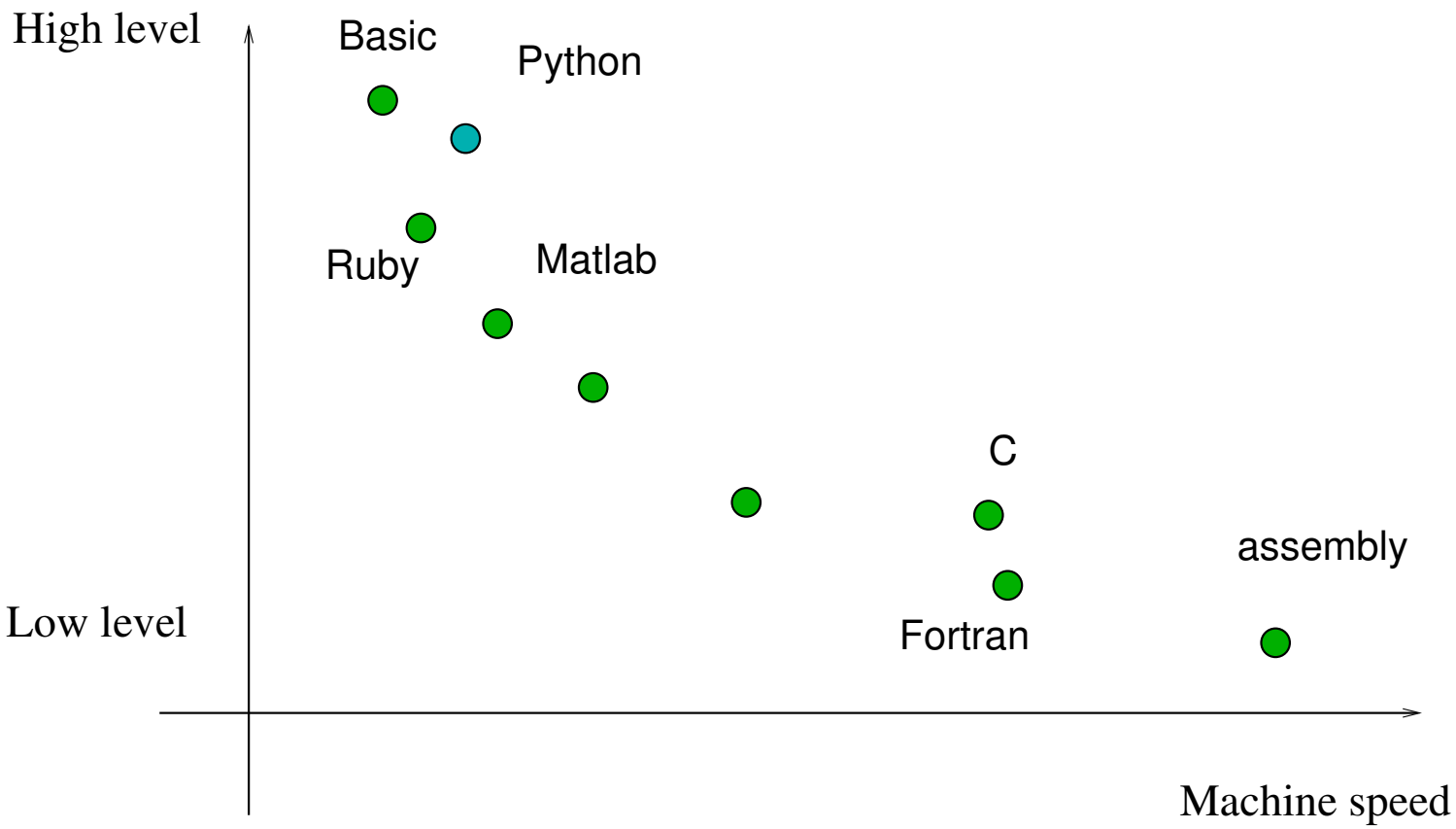
Lesson: There's a **trade off** between

- machine speed
- coding efficiency

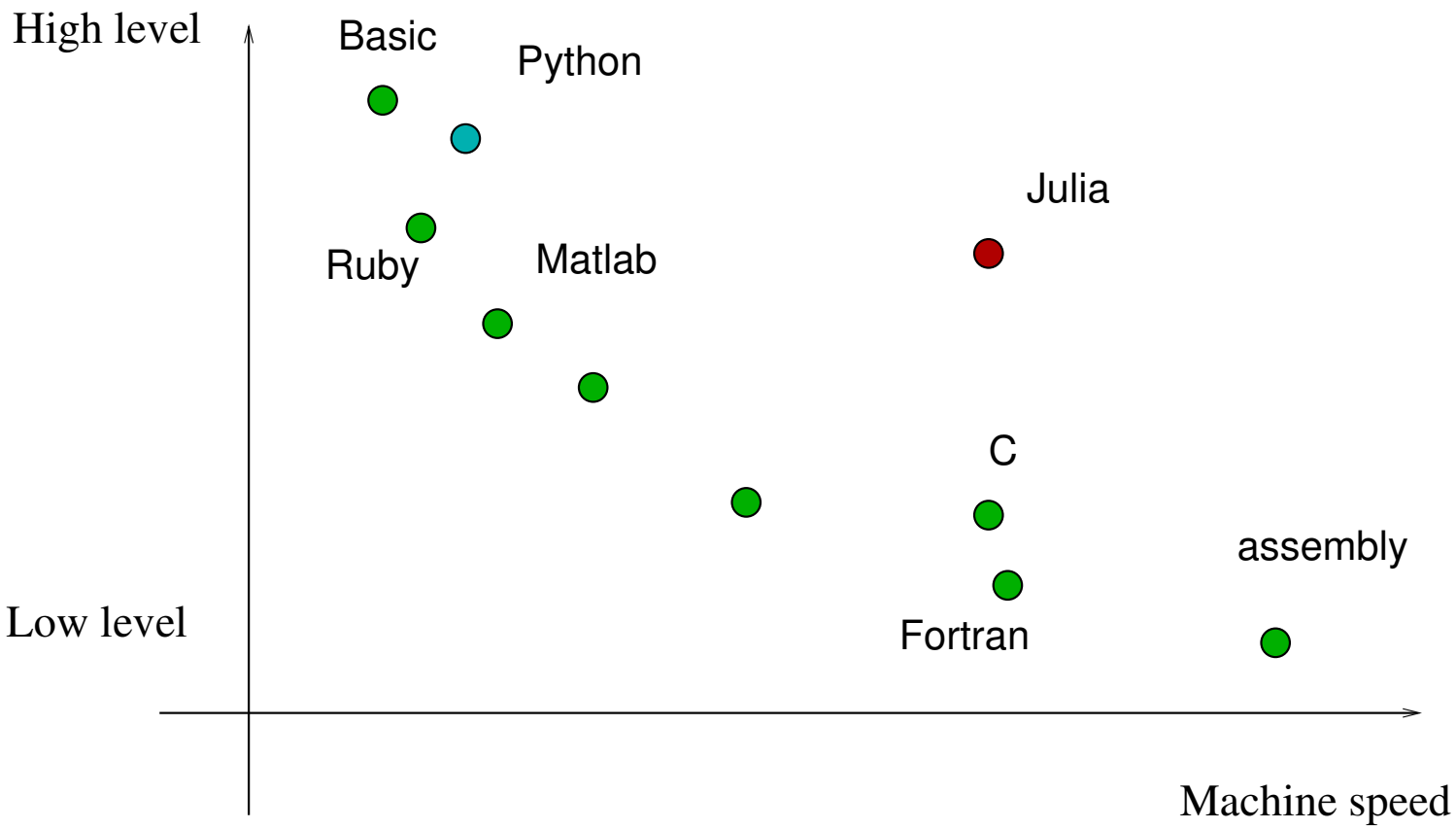
Low level languages give us **fine grained control**

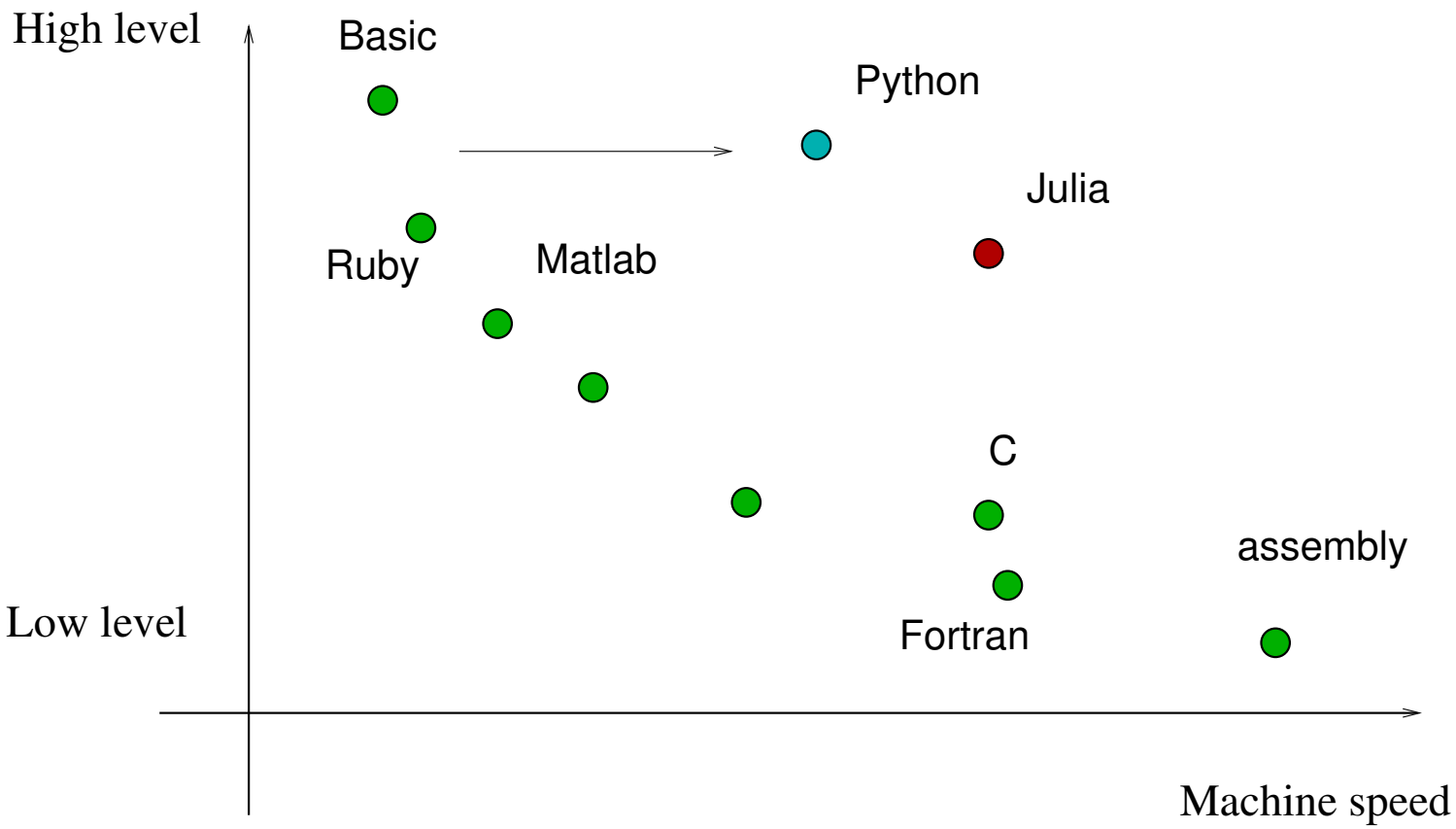
High level languages give us

- **abstraction**
- **automation** of some tasks
- **natural language** representations


High level
Low level
Basic
Python
Ruby
Matlab
C
Fortran
assembly
Machine speed

Although the curve is starting to shift...


High level
Low level
Basic
Python
Ruby
Matlab
Julia
C
Fortran
assembly
Machine speed


High level
Low level
Basic
Python
Julia
Ruby
Matlab
C
assembly
Fortran
Machine speed

A horse race

- See fast loops

Overview of Python and Julia

- History
- Pros and cons
- Which to choose?

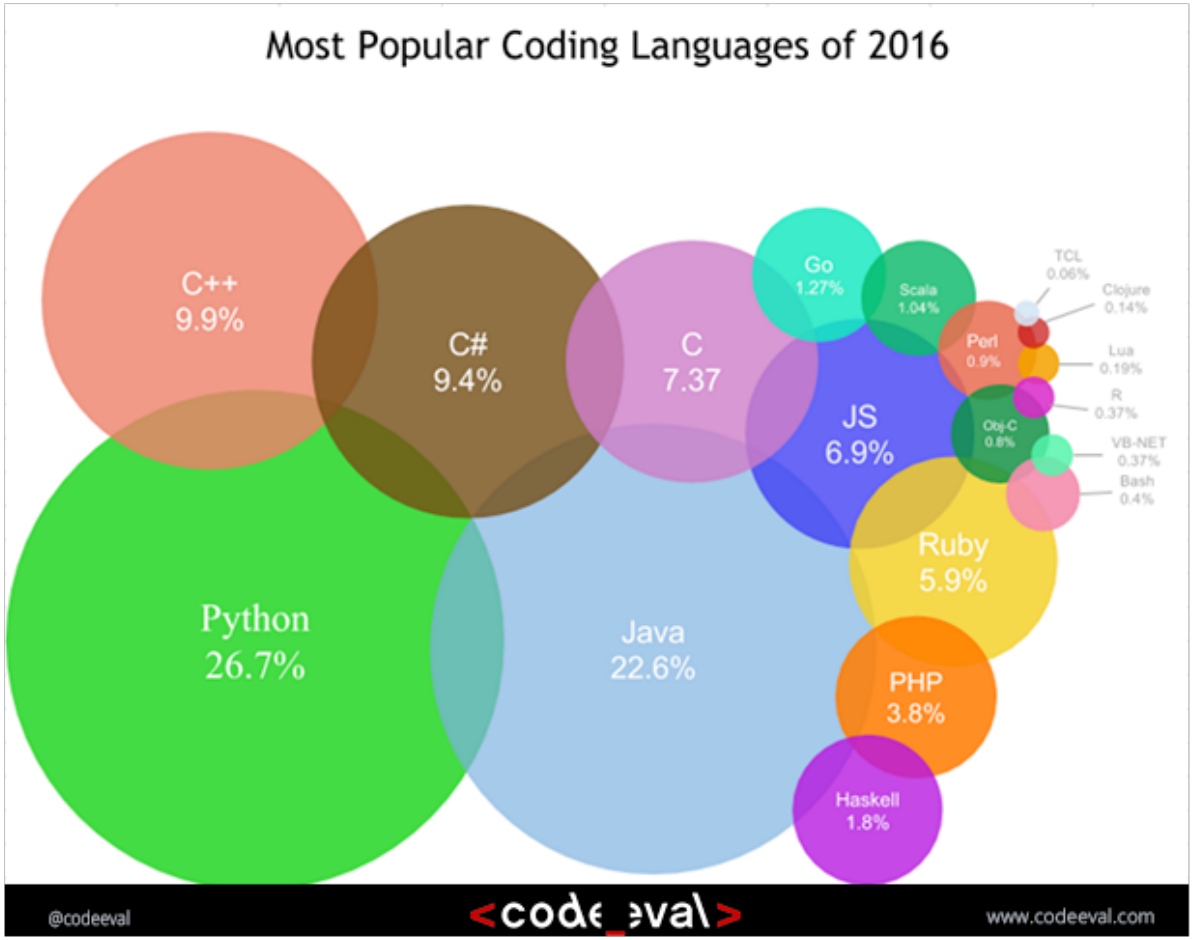
What's Python?

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
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 pros

- Great libraries
- Friendly community
- High productivity

Some negatives for scientific work

- Need scientific libraries
- Need some tricks to get speed

What's Julia?

Modern, high level, open source, scientific programming language

Strengths:

- High productivity...
- and high performance!

Negatives

- Still under development
- The "rabbit hole" of advanced features (plus or minus?)

What's Julia?

Modern, high level, open source, scientific programming language

Strengths:

- High productivity...
- and high performance!

Negatives

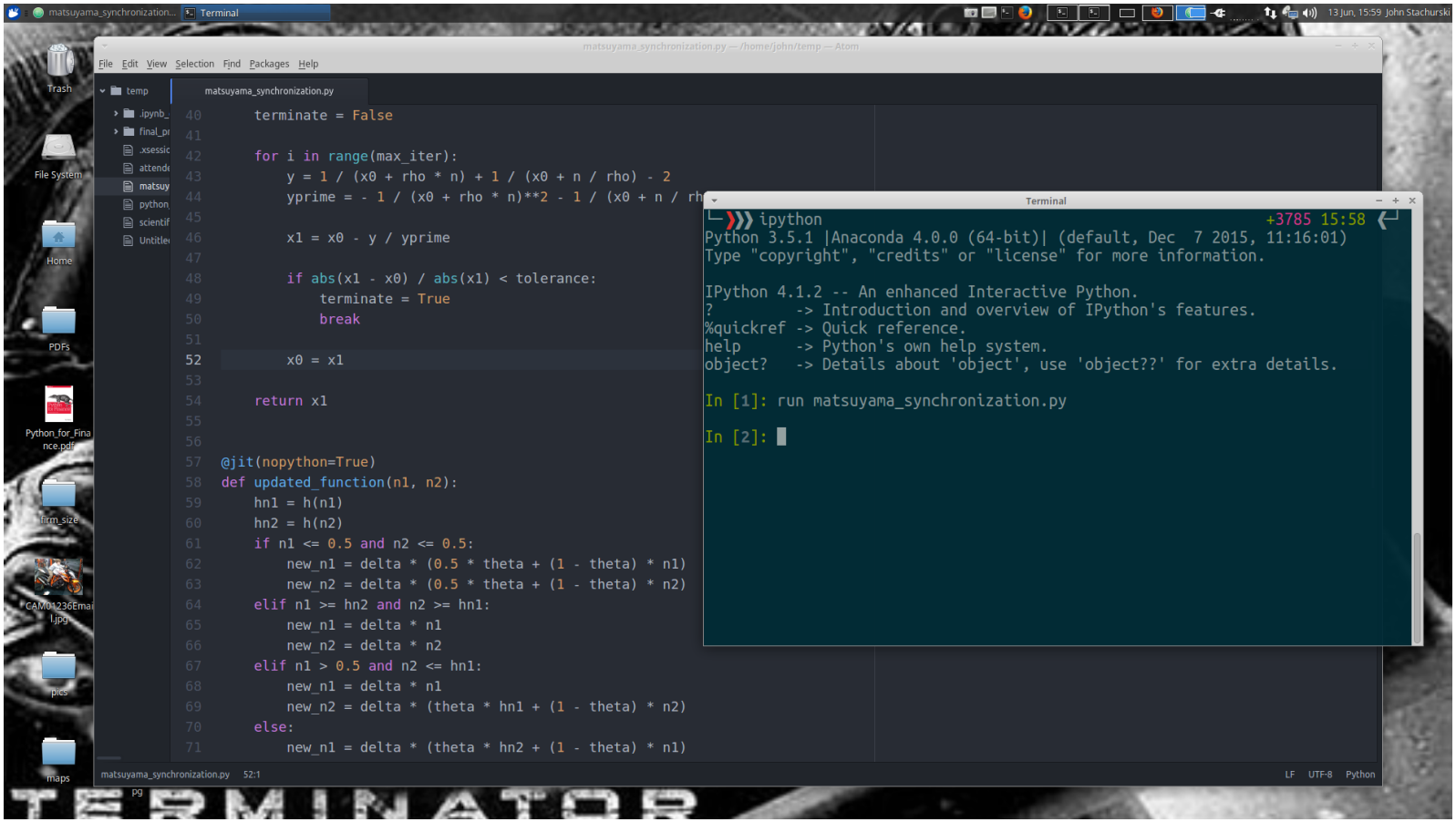
- Still under development
- The "rabbit hole" of advanced features (plus or minus?)

Interacting with Python / Julia

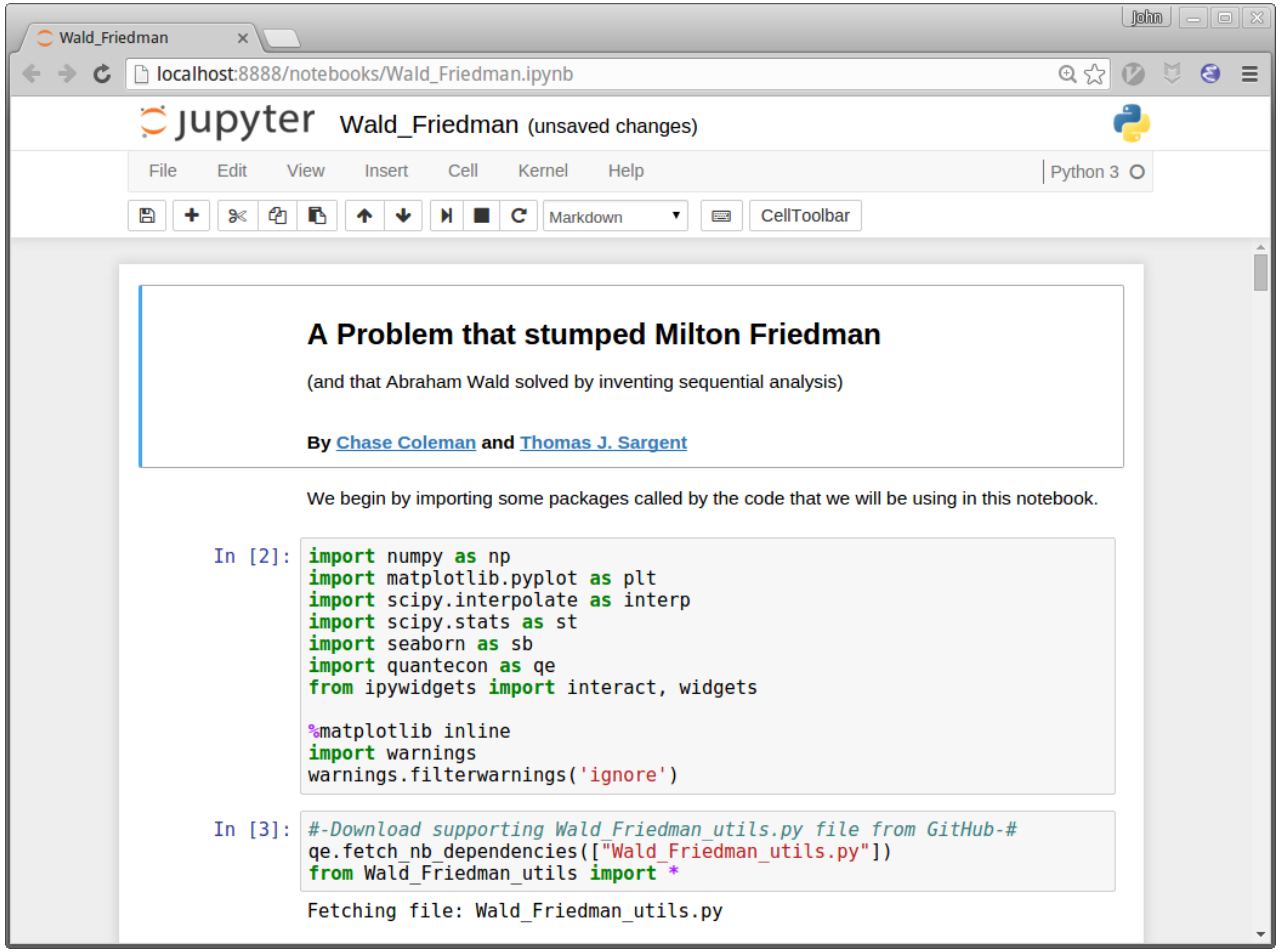
Options

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

- Connect via browser
- Dashboard and notebooks (multilingual)
- Modal interface
- Allows for rich text, equations, etc.
- Remote or local

Examples:

- http://notebooks.quantecon.org/

Jupyter Notebooks

Let's focus on Jupyter notebooks

- Connect via browser
- Dashboard and notebooks (multilingual)
- Modal interface
- Allows for rich text, equations, etc.
- Remote or local

Examples:

- http://notebooks.quantecon.org/