2017 QuantEcon Workshops

Economic Modeling with Python and Julia

Introduction

September 2017

Team

- Chase Coleman – NYU
- Victoria Gregory – NYU
- Spencer Lyon – NYU
- Matthew McKay – QuantEcon
- John Stachurski – ANU
- Natasha Watkins – QuantEcon

Thanks to the Alfred P. Sloan Foundation ;-)

Workshop Timeline

1. **9:30 am – 1 pm**: Python
2. **1 pm – 2 pm**: Lunch
3. **2 pm – 4:30 pm**: Julia

Morning Timeline

1. **9:30-10:30** Introduction and First Steps
   - John Stachurski
2. 10:30-10:45 break
3. **10:45-11:45** Data Analysis with Python
   - Natasha Watkins
4. 11:45-12:00 break
5. **12:00-13:00** Advanced Data Analysis with Python
   - Matt McKay

Afternoon Timeline

1. **14:00-15:00** Introduction to Julia with Applications
   - Chase Coleman
2. 15:00-15:30 break
3. **15:30-14:30** Dynamic Programming with Julia
   - Victoria Gregory

Aims / Outcomes / Expectations

Aims != learn Python + Julia + libraries + etc.

Aims =

- Give an overview
- Show some examples
- Discuss / argue
- Resources for further study

Aims / Outcomes / Expectations

Aims != learn Python + Julia + libraries + etc.

Aims =

- Give an overview
- Show some examples
- Discuss / argue
- Resources for further study

Workshop Resources

Cheatsheets, downloads, etc. — see

`https://quantecon.org/2017-phd-workshops-on-computational-methods`

Look for Resources

Download workshop files from the GitHub repo

- via `git` or the Download button

Downloads / Installation / Troubleshooting

Install Python + Scientific Libs

- Install Anaconda from `https://www.anaconda.com/downloads`
- Not plain vanilla Python

Jupyter notebooks

A browser based interface to Python / Julia / R / etc.

Step 1: Open a terminal

- on Windows, use `Anaconda Command Prompt`

Step 2: type `jupyter notebook`

Cloud-based server

`workshop.quantecon.org:8000`

Password = `economics`

Topics:

- opening a notebook
- executing code
- edit / command mode
- installing `quantecon`
- getting help
- introspection
- math and rich text

Background — Open Source

Proprietary

- MATLAB
- STATA, etc.

Open Source

- Python
- Julia
- R

closed and stable vs open and fast moving

Background — Open Source

Proprietary

- MATLAB
- STATA, etc.

Open Source

- Python
- Julia
- R

closed and stable vs open and fast moving

Background — Open Source

Proprietary

- MATLAB
- STATA, etc.

Open Source

- Python
- Julia
- R

closed and stable vs open and fast moving

Background — Language Types

Low level languages give us fine grained control

Example. $1 + 1$ in assembly

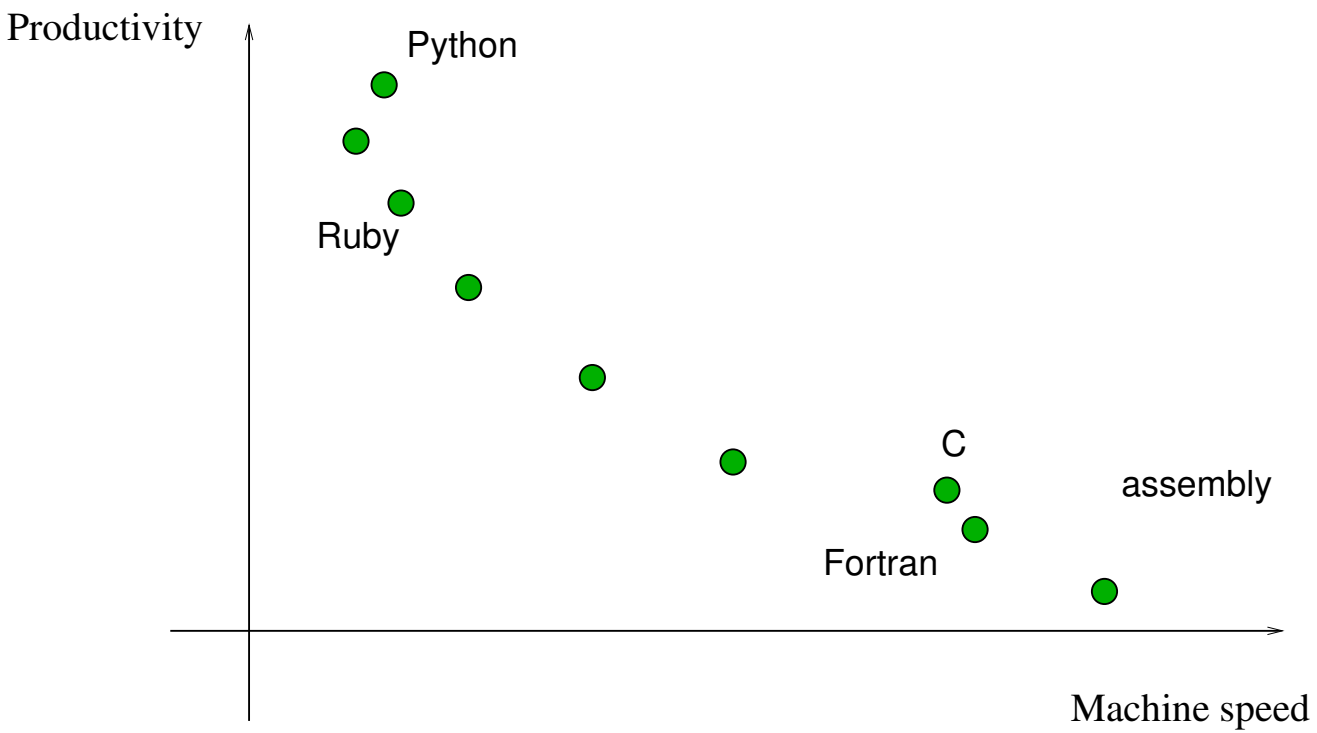
```
pushq   %rbp
movq    %rsp, %rbp
movl    $1, -12(%rbp)
movl    $1, -8(%rbp)
movl    -12(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
popq    %rbp
```

High level languages give us abstraction, automation, etc.

Example. Reading from a file in Python

```
data_file = open("data.txt")
for line in data_file:
    print(line.capitalize())
data_file.close()
```

Trade-Offs

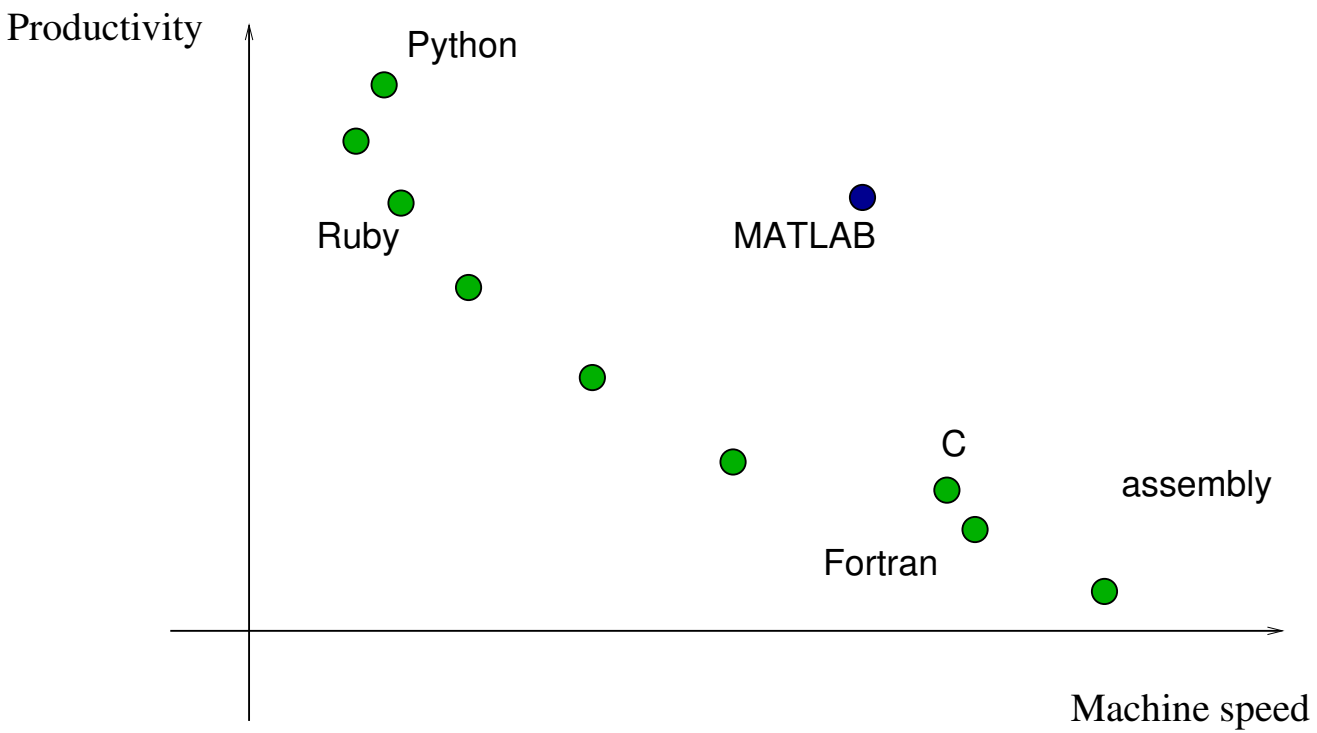

Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

But what about scientific computing?

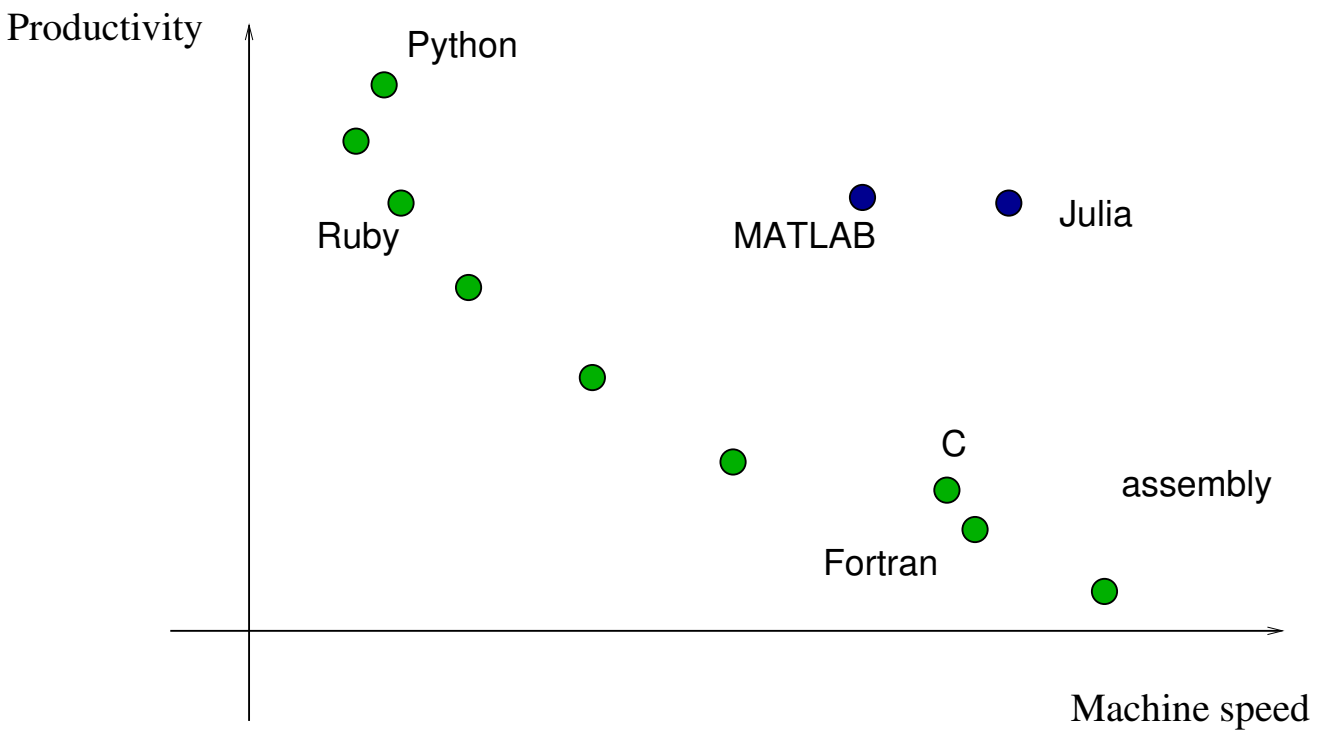
Requirements

- Productive — easy to read, write, debug, explore
- Fast computations

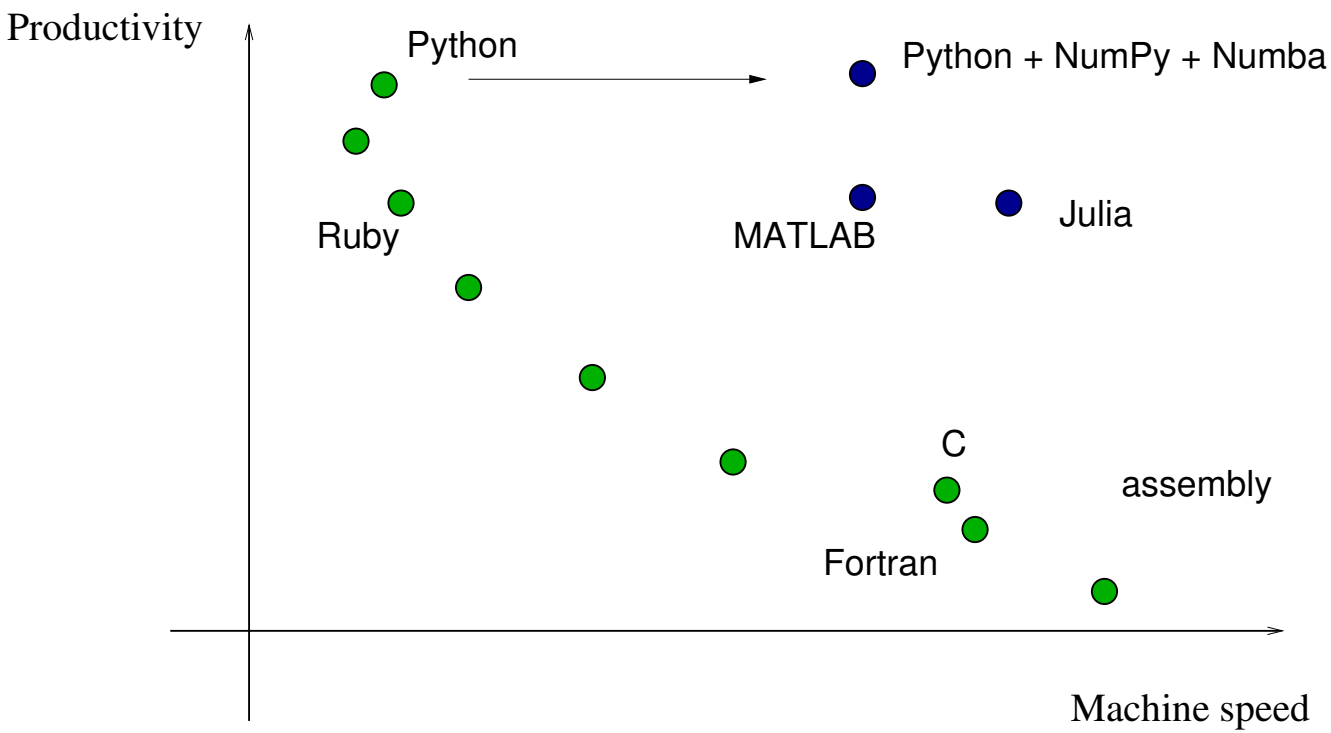
Trade-Offs


Productivity
Python
Ruby
MATLAB
C
assembly
Fortran
Machine speed

Trade-Offs


Productivity
Python
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

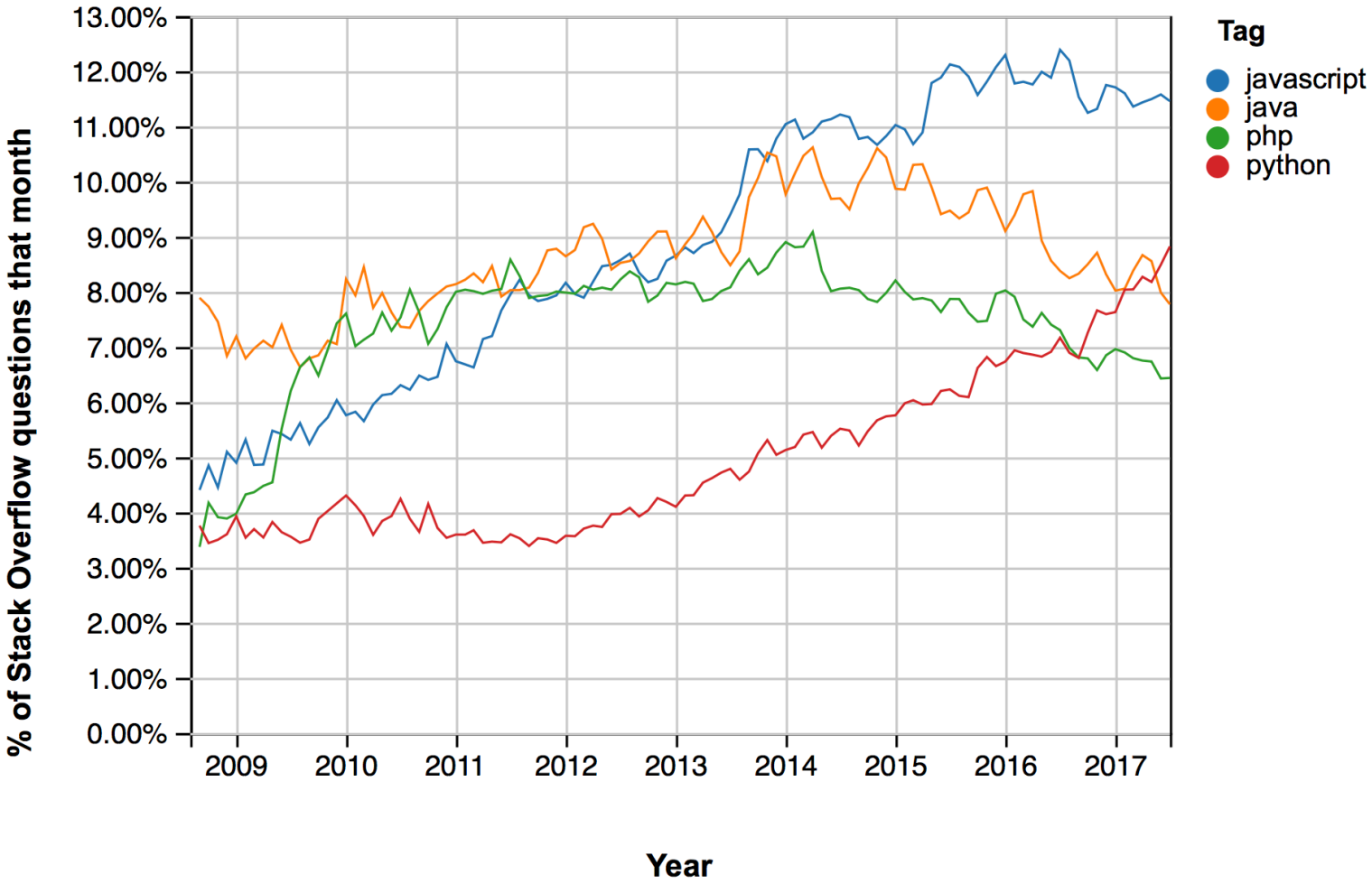
Trade-Offs


Productivity
Python
Python + NumPy + Numba
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

Python for Productivity

From local infrastructures to cloud-based systems to building websites to interfacing with SQL databases, Python has nearly limitless applications. Despite its wide-ranging impact, it remains gloriously clean and easy to learn.

– *mashable.com*


% of Stack Overflow questions that month
13.00%
12.00%
11.00%
10.00%
9.00%
8.00%
7.00%
6.00%
5.00%
4.00%
3.00%
2.00%
1.00%
0.00%
2009
2010
2011
2012
2013
2014
2015
2016
2017
Year
Tag
javascript
java
php
python

Python for High Performance Computing

- See `John/numba.ipynb`

Exercises

- See `John/plots.ipynb`