2017 QuantEcon Workshops

Economic Modeling with Python and Julia

Introduction

August-September 2017

Personnel

- Matthew McKay (QuantEcon / ANU)
- Natasha Watkins (QuantEcon / ANU)
- John Stachurski (ARC / ANU) $\leftarrow$ **me**

Thanks to the Alfred P. Sloan Foundation

Workshop Timeline

Session 1: Python

- Tue 2pm–5pm

Session 2: Julia

- Wed 2pm–4pm

Session 1 Timeline

1. Introduction and First Steps
   - John Stachurski
2. Data Analysis with Python
   - Natasha Watkins
3. Advanced Data Analysis with Python
   - Matt McKay

Aims / Outcomes / Expectations

Please don't expect to "learn" Python + libraries + etc.

What we can do:

- Give an overview
- Show some examples
- Discuss
- Set you up to learn if you then wish to

Aims / Outcomes / Expectations

Please don't expect to "learn" Python + libraries + etc.

What we can do:

- Give an overview
- Show some examples
- Discuss
- Set you up to learn if you then wish to

Workshop Resources

Cheatsheets, downloads, code examples, etc.

All available from

https://quantecon.org/WAMS-2017

Look for **Resources**

Get this PDF from the GitHub repo

- via `git` or the Download button

Downloads / Installation / Troubleshooting

⋆ **Python**

Step 1: Installation

- Install Anaconda from `https://www.continuum.io/downloads`
- Not plain vanilla Python

Step 2: Testing and next steps

Try following the instructions at `lectures.quantecon.org`

- Python → Setting up your Python Environment
- `https://lectures.quantecon.org/py/getting_started.html`

Interacting with Python

There are many options, such as **spyder**

But today we're going to use **jupyter notebook**

Launch from

- `anaconda prompt` (Windows)
- a terminal (Mac, Linux, etc.)

Language Types

Low level languages give us

- fine grained control (hardware specific)
- high execution speed

Language Types

Low level languages give us

- fine grained control (hardware specific)
- high execution speed

Example. Computing $1 + 1$ 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
```

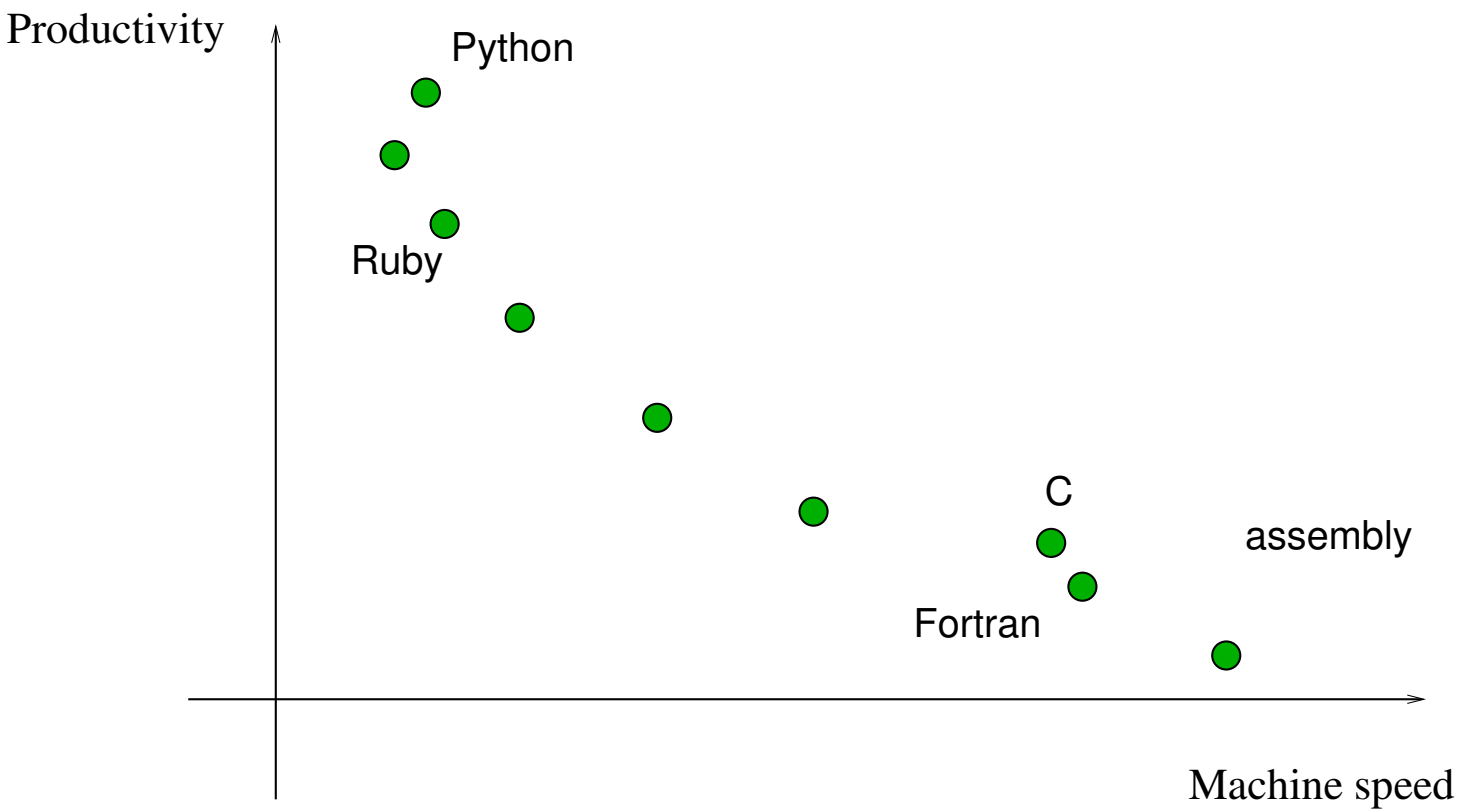
High level languages give us

- abstraction
- automation
- natural language representation

$1 + 1$ in Python:

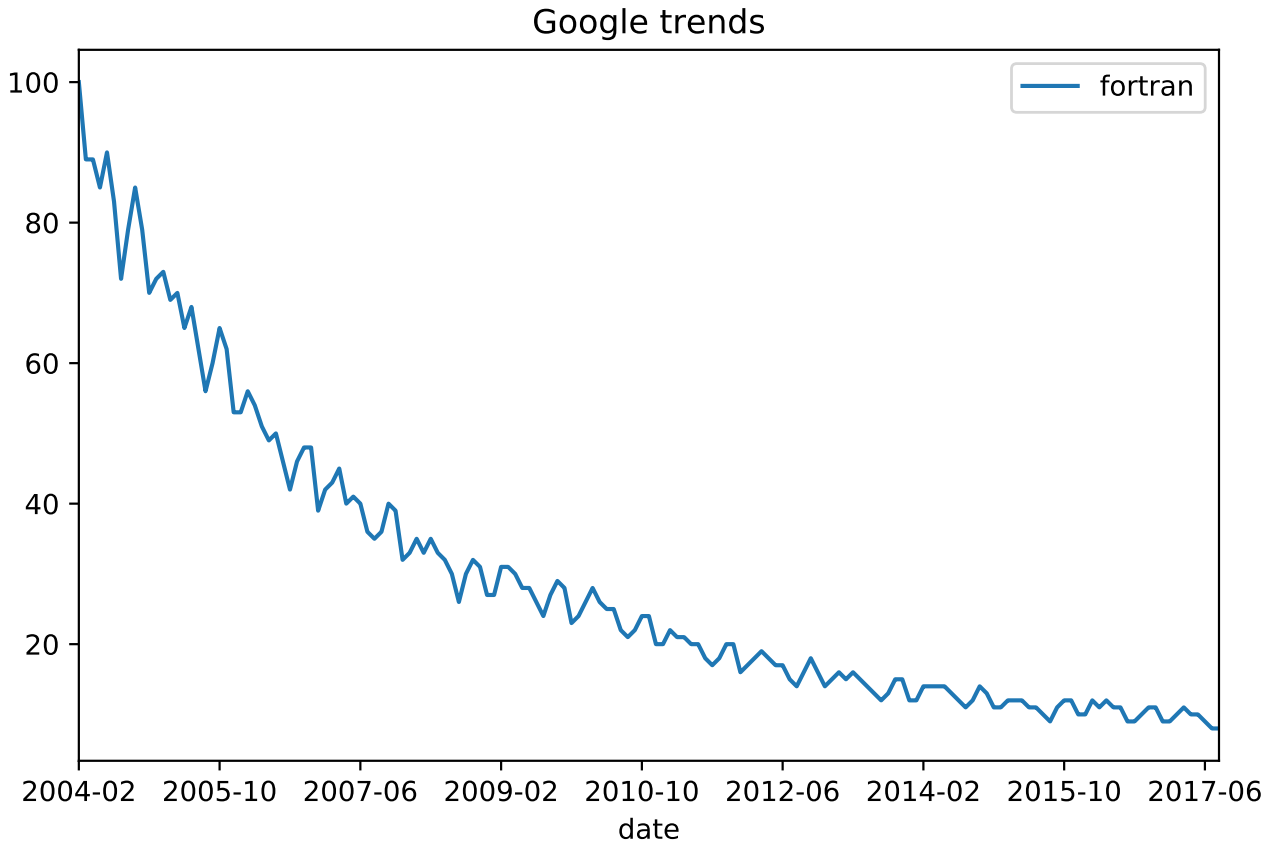
```
1 + 1
```

Trade-Offs


Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

Trends

- Low level languages are down


Google trends
fortran
100
80
60
40
20
2004-02
2005-10
2007-06
2009-02
2010-10
2012-06
2014-02
2015-10
2017-06
date

Why?

- CPU cycles are getting cheaper
- ...and complexity is increasing

Jane Street on readability:

> There is no faster way for a trading firm to destroy itself than to deploy a piece of trading software that makes a bad decision over and over in a tight loop.
>
> Part of Jane Street's reaction to these technological risks was to put a very strong focus on building software that was easily understood—software that was readable.
>
> – Yaron Minsky, Jane Street

Conversely, high level languages — such as Python — are ↑

Example. Instagram

- 600 million users, 400 million of whom are active daily
- 95 million images / videos per day

All built on Python...

Example. Data science

> Honestly, I would have to think at this point virtually all tech companies use Python to some degree. It's ubiquitous, at least, for data science teams to use Python. And it's hard to find a company in the Bay Area that doesn't have a data science team these days.
>
> Heck, I've been to SF Python meetups (in the old AT&T building, now Yelp) where there are lines around the block to get in. It's crazy.
>
> – Random Redditor

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

But what about scientific computing?

Requirements

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

In short, we want **both high speed and high productivity**

But there's a trade off, right?

Or can we make Python fast?

In short, we want **both high speed and high productivity**

But there's a trade off, right?

Or can we make Python fast?

Step 1: Steal MATLAB's best ideas

Python's **NumPy** library

- Array processing
- Send operations in batches to optimized precompiled C / Fortran

$\Longrightarrow$ High productivity and high speed combined

Example. Draw $X_1, \ldots X_n \sim N(0,1)$, compute $\max_i |X_i|$

```
import numpy as np
n = 1_000_000

# Non-vectorized code
m = -np.inf
for i in range(n):
    x = abs(np.random.randn())
    if x > m:
        m = x

# Vectorized code
m = max(abs(np.random.randn(n)))
```

Example. Linear algebra

```
import numpy as np

A = np.random.randn(2, 2)
b = np.ones(2)
np.linalg.solve(A, b)
```

But we can still do better:

1. Vectorized array processing is actually kind of a hack
   - Example. The $\max_i |X_i|$ example
2. Many algorithms are hard to vectorize
   - Example. Consider simulating $x_{t+1} = \alpha x_t + \beta + \sigma \xi_{t+1}$

But we can still do better:

1. Vectorized array processing is actually kind of a hack
   - Example. The $\max_i |X_i|$ example
2. Many algorithms are hard to vectorize
   - Example. Consider simulating $x_{t+1} = \alpha x_t + \beta + \sigma \xi_{t+1}$

Step 2: JIT compilation

Python + NumPy + Numba provide

- interactive, high productivity environment
- vectorized operations
- and fast loops

(Note: Julia's JIT compilation is better)

See `vectorization-py.ipynb`

Step 2: JIT compilation

Python + NumPy + Numba provide

- interactive, high productivity environment
- vectorized operations
- and fast loops

(Note: Julia's JIT compilation is better)

See `vectorization-py.ipynb`

Step 2: JIT compilation

Python + NumPy + Numba provide

- interactive, high productivity environment
- vectorized operations
- and fast loops

(Note: Julia's JIT compilation is better)

See `vectorization-py.ipynb`

Exercise: Generate some plots

- See `plots/plots-py.ipynb`