Short Workshop on Computational Methods

Computational Economics with Python

Introduction

July 12th 2019

Introduction

Personnel

- Matt McKay
- John Stachurski

Duration: $\approx$ 2 hours, with 10 minutes break

Prereqs / Aims / Outcomes

Aims =

- Super-quick intro to Python
- Overview of scientific computing
- Some examples, discuss
- Resources for further study

Assumptions =

- computer/maths/stats literate
- not familiar with Python

Workshop Resources

Downloads, slides, etc. — see

`https://github.com/QuantEcon/short_python_workshop`

Download via `git` or the Download button

Downloads / Installation / Troubleshooting

Install Python + Scientific Libs

- Install Anaconda from `https://www.anaconda.com/distribution/`
  - Select latest Python version (3.7)
  - For your OS!
- Not plain vanilla Python

Remote options

- `https://colab.research.google.com`
- `https://notebooks.azure.com/`

Jupyter Notebooks

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

Option 1: Open a terminal and type `jupyter notebook`

- on Windows, use `Anaconda Command Prompt`

Option 2: Find Jupyter Notebooks in your application menu

Option 3: Open “Anaconda Navigator”, click on “Jupyter Notebooks”

- opening a notebook
- writing code
- executing code blocks
- edit / command mode
- getting help
- math and rich text
- unicode
- Jupyter Lab

Overview: Why Python?

Why are we focusing on Python?

Popular alternatives for scientific computing:

- C/C++
- Fortran
- MATLAB
- Julia
- R
- Excel, STATA, Eviews, etc, etc.

Let's do an overview of scientific computing

Background — Language Types

Proprietary

- Excel
- MATLAB
- STATA, etc.

Open Source

- Python
- Julia
- R

closed and stable vs open and fast moving

Background — Language Types

Low level

- C/C++
- Fortran
- Java

High level

- Python
- Ruby
- Javascript

Low level languages give us fine grained control

Example. $1 + 1$ in assembly

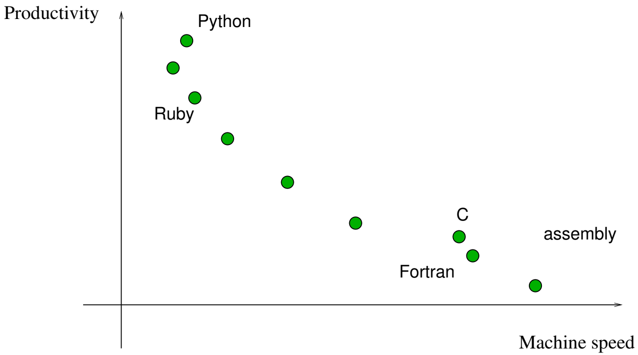
```
pushq   %rbp
movq    %rsp, %rbp
movl    $1, -12(%rbp)
movl    $1, -8(%rbp)
movl    -12(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
popq    %rbp
```

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

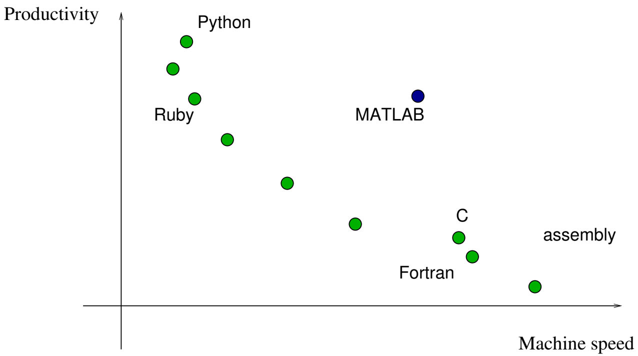

Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

But what about scientific computing?

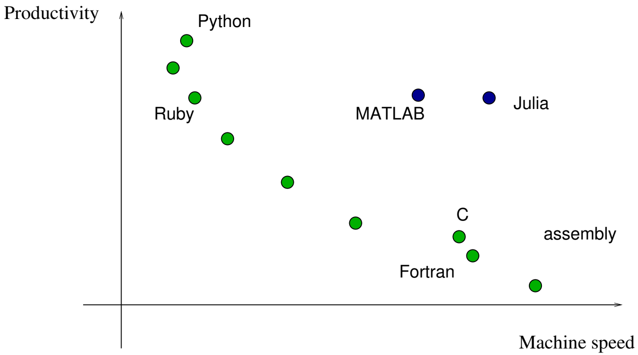
Requirements

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

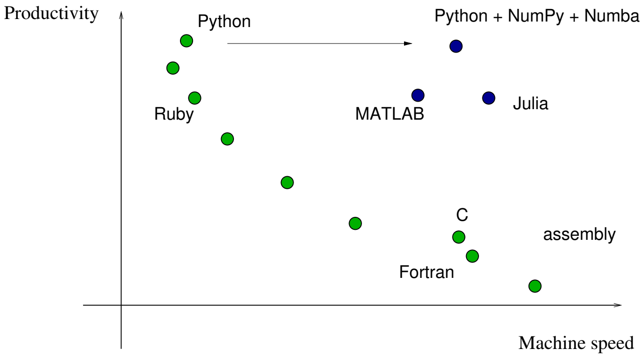
Trade-Offs


Productivity
Python
Ruby
MATLAB
C
assembly
Fortran
Machine speed

Trade-Offs

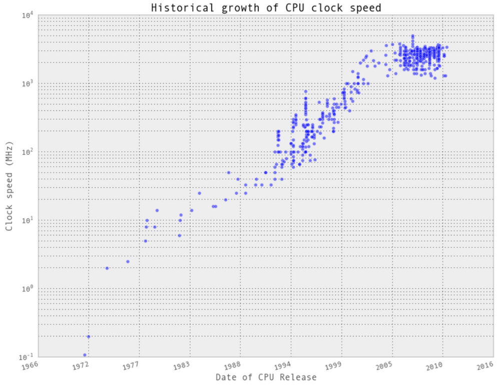

Productivity
Python
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

Trade-Offs

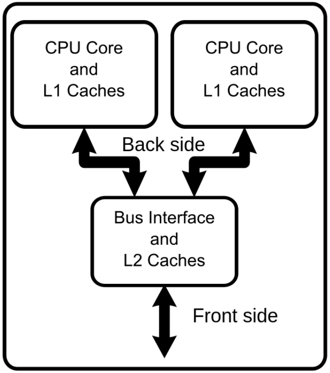

Productivity
Python
Python + NumPy + Numba
Ruby
MATLAB
Julia
C
assembly
Fortran
Machine speed

Programming Background — Hardware

CPU frequency (clock speed) growth is slowing

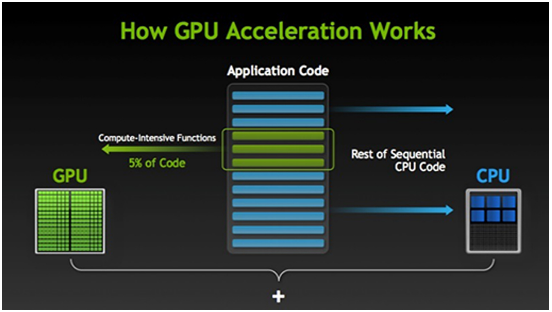


Chip makers have responded by developing multi-core processors



Source: Wikipedia

GPUs / ASICs are also becoming increasingly important


How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
GPU
Rest of Sequential CPU Code
CPU
+

Applications: machine learning, deep learning, etc.

Distributed/Cloud Computing

Advantages:

- run code on big machines we don't have to buy
- customized execution environments
- circumvent internal IT rules

Options:

- University machines
- AWS
- Google Colab, etc.

Further Advantages of Python

We have seen that Python + scentific libraries give us

high productivity **and** high performance / execution speed

Any other advantages?

Example. How does it relate to new trends like

- multiple cores / GPUs / parallelization
- cloud computing?

Exploiting multiple cores / threads is nontrivial

But Python has very strong tools in this area:

- Numpy (implicit multithreading)
- Numba + `@vectorize` + `@jit + prange`
- Dask
- Keras / tensorflow
- PyTorch
- JAX, etc.

Python is also very popular for / well adapted to cloud computing

- Jupyter notebooks
- JupyterHub
- Google colab
- AWS
- etc.

In short, Python is well adapted to **new trends** in scientific computing

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*

But how?

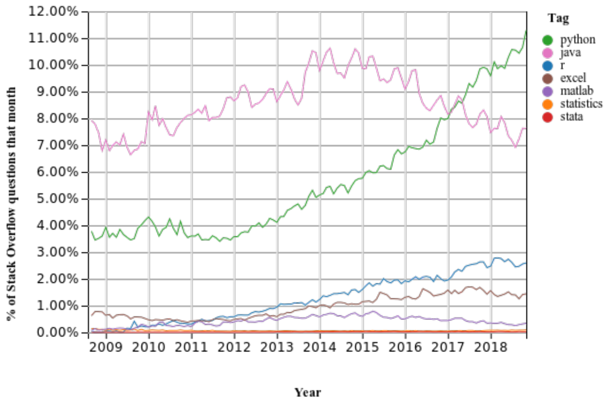
- core language is small
- **lots** of third party libraries

Python for Data Science

Python has gotten sufficiently weapons grade that we don't descend into R anymore. Sorry, R people. I used to be one of you but we no longer descend into R.

– Chris Wiggins, Chief Data Scientist at The New York Times

As a result of these advantages:



Conclusion

Python combines

- stability and maturity
- high productivity
- high popularity
- high performance through scientific libraries

Current best tool for **general purpose** scientific computing in econ