

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

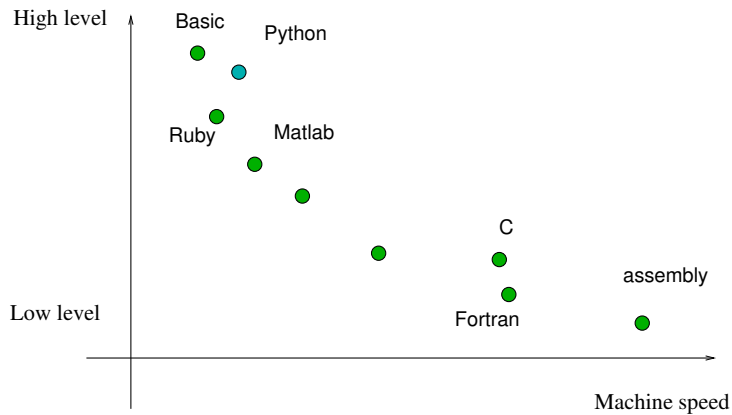
Lesson: There's a **trade off** between

- machine speed
- coding efficiency

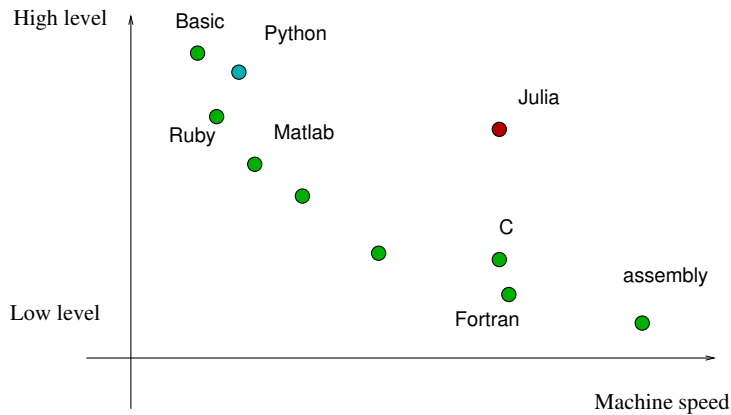
Low level languages give us **fine grained control**

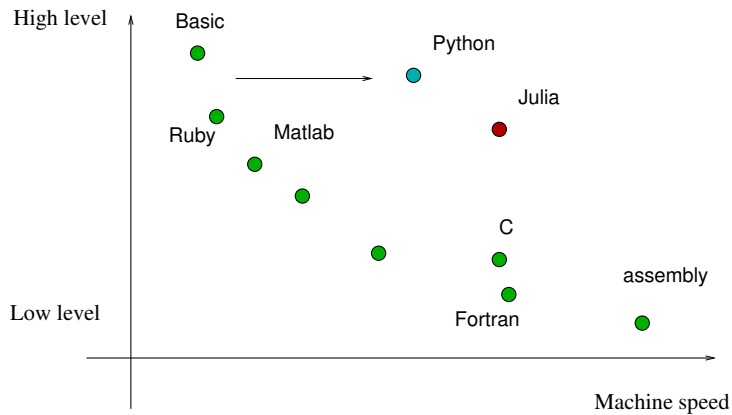
High level languages give us

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



Although the curve is starting to shift...





A Horse Race

Task is to **simulate** the time series

$$X_{t+1} = \alpha X_t + \beta + \sigma W_{t+1}$$

This gives us

$$X_1, X_2, X_3, \dots, X_N$$

Set $N = 10,000,000$ and then compute the sample mean

- See **fast loops**

Overview of Python and Julia

- History
- Pros and cons
- Which to choose?

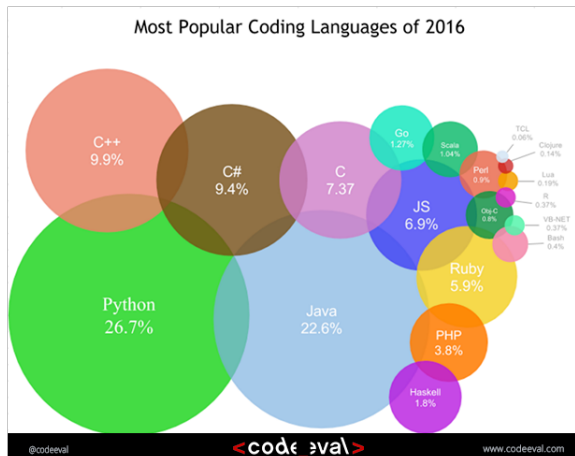
What's Python?

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



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

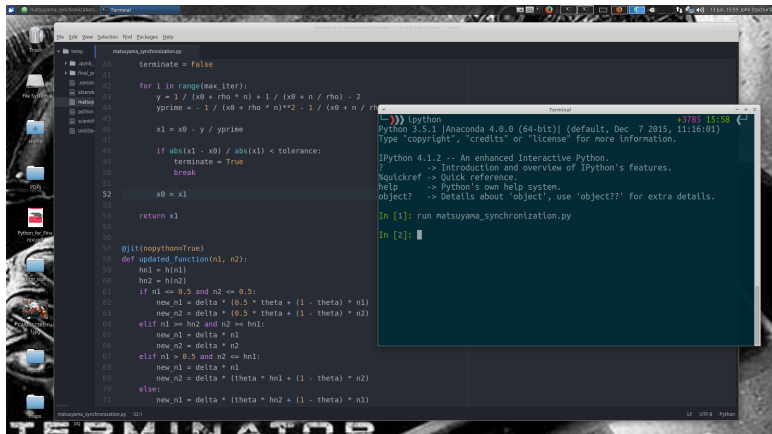
- Still under development
- The “rabbit hole” of advanced features (plus or minus?)

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



The screenshot shows the Atom text editor interface. The main editor window displays a Python file named `matsuyama_synchronization.py`. The code defines a function `updated_function` and a loop that iterates over a range of values, updating variables `x0` and `x1` based on a tolerance. The code is as follows:

```
40 terminate = False
41
42 for i in range(max_iter):
43     y = 1 / (x0 + rho * n) + 1 / (x0 + n / rho) - 2
44     yprime = - 1 / (x0 + rho * n)**2 - 1 / (x0 + n / rho)
45
46     x1 = x0 - y / yprime
47
48     if abs(x1 - x0) / abs(x1) < tolerance:
49         terminate = True
50         break
51
52     x0 = x1
53
54     return x1
55
56
57 @jit(nopython=True)
58 def updated_function(n1, n2):
59     hn1 = h(n1)
60     hn2 = h(n2)
61     if n1 <= 0.5 and n2 <= 0.5:
62         new_n1 = delta * (0.5 * theta + (1 - theta) * n1)
63         new_n2 = delta * (0.5 * theta + (1 - theta) * n2)
64     elif n1 >= hn2 and n2 >= hn1:
65         new_n1 = delta * n1
66         new_n2 = delta * n2
67     elif n1 > 0.5 and n2 <= hn1:
68         new_n1 = delta * n1
69         new_n2 = delta * (theta * hn1 + (1 - theta) * n2)
70     else:
71         new_n1 = delta * (theta * hn2 + (1 - theta) * n1)
```

A terminal window is open in the foreground, showing the IPython REPL. The prompt is `In [1]:` and the command `run matsuyama_synchronization.py` has been entered. The output of the command is `+3785 15:58`.

Jupyter notebooks

Wald_Friedman (unsaved changes)

File Edit View Insert Cell Kernel Help Python 3

File Edit View Insert Cell Kernel Help

File Edit View Insert Cell Kernel Help

A Problem that stumped Milton Friedman

(and that Abraham Wald solved by inventing sequential analysis)

By [Chase Coleman](#) and [Thomas J. Sargent](#)

We begin by importing some packages called by the code that we will be using in this notebook.

```
In [2]: import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate as interp
import scipy.stats as st
import seaborn as sb
import quantecon as qe
from ipywidgets import interact, widgets

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```
In [3]: #-Download supporting Wald_Friedman_utils.py file from GitHub-#
qe.fetch_nb_dependencies(["Wald_Friedman_utils.py"])
from Wald_Friedman_utils import *
```

Fetching file: Wald_Friedman_utils.py

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/>