About these Lectures

Thomas J. Sargent and John Stachurski

March 15, 2020

1 Contents

• Overview 2
• Python 3
• Open Source 4
• How about Other Languages? 5
• Credits 6

2 Overview

Programming, mathematics and statistics are powerful tools for analyzing the functioning of economies.

This lecture series provides a hands-on instruction manual.

Topics include

• algorithms and numerical methods for studying economic problems,
• related mathematical and statistical concepts, and
• basics of coding skills and software engineering.

The intended audience is undergraduate students, graduate students and researchers in economics, finance and related fields.

3 Python

The coding language for this lecture series is Python.

Note that there’s also a related set of Julia lectures.

In terms of the differences,

• Python is a general-purpose language featuring a massive user community in the sciences and an outstanding scientific ecosystem.
• Julia is a more recent language with many exciting features.
Both are modern, open-source, high productivity languages with all the key features needed for high-performance computing.

Julia has the advantage that third party libraries are often written entirely in Julia itself.

Python has the advantage of being supported by a vast collection of scientific libraries (and being a highly marketable skill).

4 Open Source

All the computing environments we work with are free and open-source.

This means that you, your coauthors and your students can install them and their libraries on all of your computers without cost or concern about licenses.

Another advantage of open source libraries is that you can read them and learn how they work.

For example, let’s say you want to know exactly how `statsmodels` computes Newey-West covariance matrices.

No problem: You can go ahead and read the code.

While dipping into external code libraries takes a bit of coding maturity, it’s very useful for

1. helping you understand the details of a particular implementation, and
2. building your programming skills by showing you code written by first-rate programmers.

Also, you can modify the library to suit your needs: if the functionality provided is not exactly what you want, you are free to change it.

Another, a more philosophical advantage of open-source software is that it conforms to the scientific ideal of reproducibility.

5 How about Other Languages?

But why don’t you use language XYZ?

5.1 MATLAB

While MATLAB has many nice features, it’s starting to show its age.

It can no longer match Python or Julia in terms of performance and design.

MATLAB is also proprietary, which comes with its own set of disadvantages.

Given what’s available now, it’s hard to find any good reason to invest in MATLAB.

Incidentally, if you decide to jump from MATLAB to Python, this cheat-sheet will be useful.
5.2 R

R is a very useful open source statistical environment and programming language. Its primary strength is its vast collection of extension packages. Python is more general-purpose than R and hence a better fit for this course. Moreover, if there are R libraries you find you want to use, you can now call them from within Python or Julia.

5.3 C / C++ / Fortran?

Isn’t Fortran / C / C++ faster than Python? In which case it must be better, right? This is an outdated view. First, you can achieve speeds equal to or faster than those of compiled languages in Python through features like a just-in-time compilation — we’ll talk about how later on. Second, remember that the correct objective function to minimize is

\[
\text{total time} = \text{development time} + \text{execution time}
\]

In assessing this trade off, it’s necessary to bear in mind that

- Your time is a far more valuable resource than the computer’s time.
- Languages like Python are much faster to write and debug in.
- In any one program, the vast majority of CPU time will be spent iterating over just a few lines of your code.

5.3.1 Last Word

Writing your entire program in Fortran / C / C++ is best thought of as “premature optimization”.

On this topic we quote the godfather:

> We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. – Donald Knuth

6 Credits

These lectures have benefited greatly from comments and suggestions from our colleagues, students and friends. Special thanks are due to our sponsoring organization the Alfred P. Sloan Foundation and our research assistants Chase Coleman, Spencer Lyon and Matthew McKay for innumerable contributions to the code library and functioning of the website. We also thank Andrij Stachurski for his great web skills, and the many others who have contributed suggestions, bug fixes or improvements. They include but are not limited to Anmol Bhandari, Long Bui, Jeong-Hun Choi, David Evans, Shunsuke Hori, Chenghan Hou, Doc-Jin Jang, Qingyin Ma, Akira Matsushita, Tomohito Okabe, Daisuke Oyama, David Pugh, Alex Olssen, Nathan Palmer, Bill Tubbs, Natasha Watkins, Pablo Winant and Yixiao Zhou.