

Scientific Computing Languages

(Lectures on High-performance Computing for Economists V)

Jesús Fernández-Villaverde,¹ Pablo Guerrón,² and David Zarruk Valencia³

October 22, 2018

¹University of Pennsylvania

²Boston College

³ITAM

Programming languages for scientific computation

- General-purpose languages (GPL):
 1. C++.
 2. Python.
- Domain-specific languages (DSL):
 1. Julia.
 2. R.
 3. Matlab.
- If you want to undertake research on computational-intensive papers, learning a GPL is probably worthwhile.
- Moreover, knowing a GPL will make you a better user of a DSL.

C++

- C/C++ is the infrastructure of much of the modern computing world.
- If you know `Unix` and C/C++, you can probably master everything else easily (think of Latin and Romance languages!).
- In some sense, C++ is really a “federation” of languages.
- What is the difference between C and C++?
- C++ introduced full OOP support.

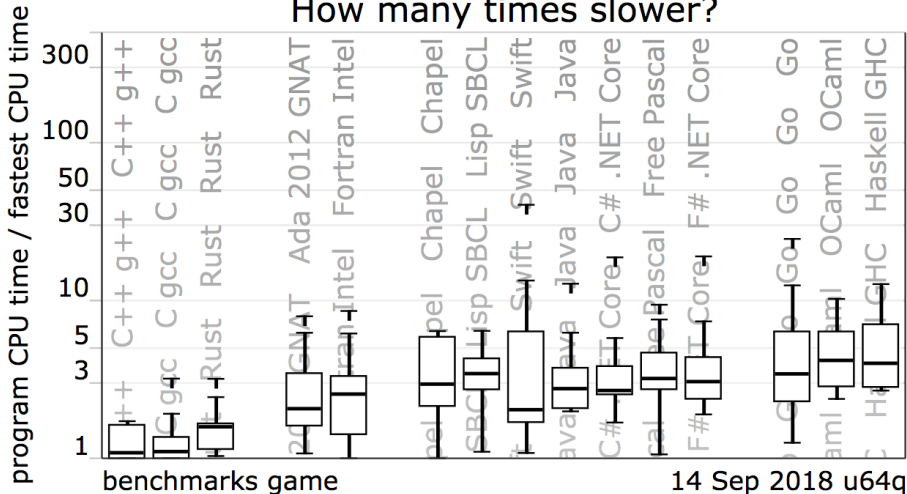
- General-purpose, multi-paradigm, static partially inferred typed, compiled language.
- Current standard is C++17 (published in December 2017).
- Developed by Bjarne Stroustrup at Bells Labs in the early 1980s.
- C++ has a more modern design and approach to programming than predecessor languages. Enormously influential in successor languages.
- If you know C++, you will be able to read C legacy code without much problem.
- In fact, nearly all C programs are valid C++ programs (the converse is not true).
- But you should try to “think” in C++17, not in C or even in older C++ standards.



C++: advantages

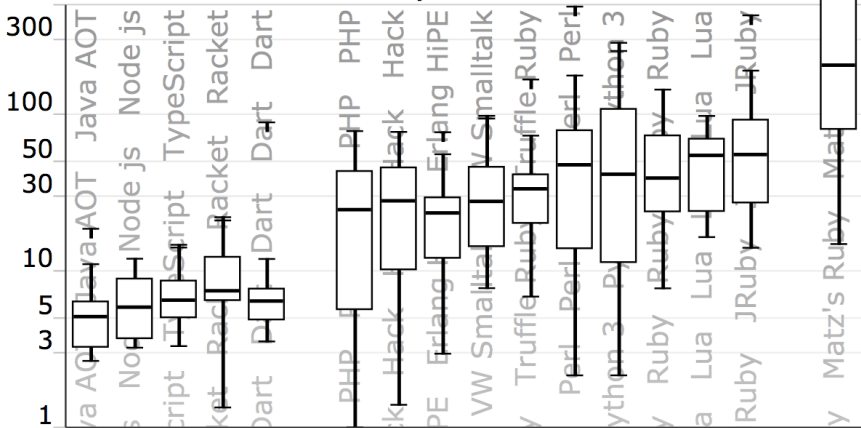
1. Powerful language: you can code anything you can imagine in C++.
2. Continuously evolving but with a standard: new implementations have support for meta-programming, functional programming,...
3. Rock-solid design.
4. Top performance in terms of speed.
5. Wide community of users.
6. Excellent open-source compilers and associated tools.
7. One of most detailed object-orientation programming (OOP) implementation available.
8. Easy integration with multiprocessor programming (OpenMP, MPI, CUDA, OpenCL, ...).
9. Allows low-level memory manipulation.

How many times slower?



program CPU time / fastest CPU time

How many times slower?



14 Sep 2018 u64q

C++: disadvantages

1. Hard language to learn, even harder to master (although for scientific computation –and if you start with control structures, arrays, and functions before pointers, objects and classes– you can get away with a small subset).
2. Large specification: C++17, Standard Template Library, ...
3. This causes, at times, portability issues.
4. Legacies from the past (you even have a goto!).
5. Minimal optimization of OO features.
6. Notation is far away from standard mathematical notation (you can fix much of this with libraries, but then you need to learn them).
7. (Personal preference): matrix indexing starts at zero.

Additional resources I

Books with focus on C++11 and later implementations and on scientific computing.

- Basic:

1. *Starting Out with C++ from Control Structures to Objects (9th Edition)*, by Tony Gaddis.
2. *Guide to Scientific Computing in C++ (2n Edition)*, by Joe Pitt Francis and Jonathan Whiteley.

- Intermediate:

1. *Discovering Modern C++: An Intensive Course for Scientists, Engineers, and Programmers* by Peter Gottschling.
2. *The C++ Programming Language (4th Edition)*, by Bjarne Stroustrup.

Additional resources II

- Advanced:

1. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*, by Scott Meyers.
2. *The C++ Standard Library: A Tutorial and Reference (2nd Edition)*, by Nicolai M. Josuttis.
3. *C++ Templates: The Complete Guide (2nd Edition)*, by David Vandevoorde and Nicolai M. Josuttis.

- References:

1. <http://www.cplusplus.com>.
2. <http://www.cprogramming.com>.
3. <https://isocpp.org/>

Python

- General-purpose, multi-paradigm, dynamically typed, interpreted language.
- Designed by Guido von Rossum.
- Name inspired by *Monty Python's Flying Circus*.
- Open source.
- Comes in two flavors: Python 2.7.15 and 3.7.0.
- Simple and with full OOP support.
- Elegant and intuitive language.
- Ideal, for instance, to teach high school students or a class in introductory CS.



Python: advantages

1. Great for prototyping: dynamic typing and REPL.
2. Rich ecosystem:
 - 2.1 Scientific computation modules: NumPy, SciPy, and SymPy.
 - 2.2 Statistics modules: Pandas.
 - 2.3 Plotting modules: matplotlib and ggplot.
3. Easy unit testing: doctest is a default module.
4. Manipulates strings surprisingly well (regular expressions) ⇒ natural language processing, artificial intelligence, big data.
5. Excellent interaction with other languages.
6. If you have a Mac, it is already preinstalled (although, likely, an older version)

Python: disadvantages

- Considerable time penalty.
- Reference distribution by Guido van Rossum: CPython with IDLE and Python Launcher at <http://www.python.org/>.
- Better distribution for economists: Anaconda, with plenty of extras, at <https://www.anaconda.com/>.
- Jython is an implementation in the JVM, at <https://www.jython.org>.
- High-performance routes:
 1. Numba is an just-in-time specializing compiler which compiles annotated Python and NumPy code to LLVM.
 2. Pypy is an implementation with a JIT compiler.
 3. Cython is a superset of Python with an interface for invoking C/C++ routines.
- Limitations of these routes.



- Books:
 1. *Learning Python (5th Edition)*, by Mark Lutz.
 2. *Think Python (2nd Edition)*, by Allen Downey.
 3. *High Performance Python*, by Micha Gorelick and Ian Ozsvald.
- Web page of the language: <http://www.python.org/>

Julia

- Modern, expressive, high-performance programming language designed for scientific computation and data manipulation.
- Open-source.
- LLVM-based just-in-time (JIT) compiler.
- Lisp-style macros.
- Designed for parallelism and cloud computing.
- Syntax close to Matlab and R, but **not** a faster Matlab.
- External packages.
- Easy to integrate with C++ and Python.
- However, at early stages of development (some packages are unstable).

Additional resources

- Julia: <http://julialang.org/>.
- JuliaPro: <https://juliacomputing.com/products/juliapro.html>.
- Books:
 1. *Mastering Julia*, by Malcolm Sherrington.
 2. *Getting Started with Julia*, by Ivo Balbaert.
 3. *Julia High Performance*, by Avik Sengupta.

R

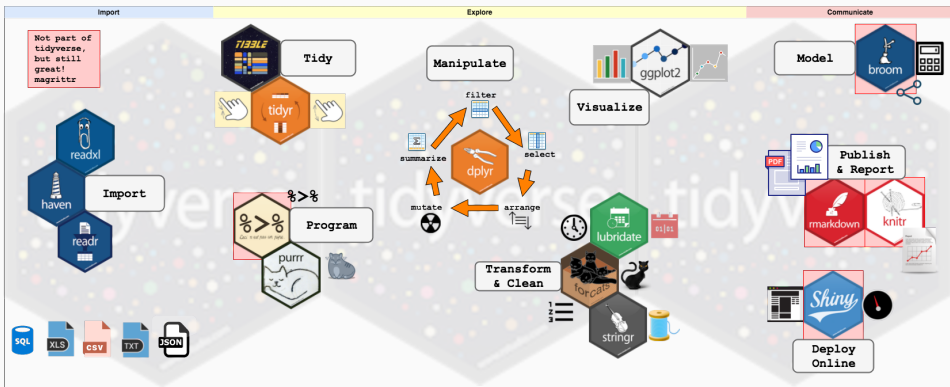


- High level, open source language for statistical computation.
- Developed by Ross Ihaka and Robert Gentleman as an evolution of S, programmed by John Chambers in 1975–1976 (at Bells Labs).
- S was itself created to substitute a Fortran library for statistics.
- It was soon moved into the public domain: <http://cran.r-project.org/>.
- Key for success.
- S has another descendent: S-Plus, a commercial implementation that is much less popular.
- Elegant and intuitive syntax.
- Advanced OOP implementation: each estimator is a class.



R: advantages

1. Rich and active community of users:
 - Around 12k packages (`ggplot2`, `dplyr`, `tidyr`, `readr`, `knitr`, `markdown`, `tidyr`,...).
 - <https://www.r-bloggers.com>
 - Dozens of books.
 - Dozens of free tutorials.
2. Widely used for big data (often with Hadoop and Spark).
3. Allows for multiprogramming: `lambda.r` and `purrr` for functional programming.
4. You can easily compile functions into byte code with package compiler.
5. Interacts well with other languages: `Rcpp`, `Reticulate`.
6. Easy to parallelize (`parallel`).



- I recommend the Microsoft R Open distribution.
- RStudio is a simple and powerful IDE:
 1. GIT integration.
 2. markdown support (different versions, including `knitr`).
 3. Package manager.
 4. Install `tidyverse` right away.

Matlab



- Matrix laboratory.
- Started in the late 1970s, released commercially in 1984.
- Widely used in engineering and industry.
- Plenty of code around (both for general purposes and in economics).
- Recent versions use Java extensively in the background⇒allows you to call Java libraries.
- Many useful toolboxes:
 1. By Mathworks.
 2. By third parties: Dynare.
- Clones available: Octave, Scilab,...

Matlab: advantages

- Great IDE:
 1. Editor, with syntax highlight, smart indent, cells, folding nested statements, easy comparing tools.
 2. Debugger.
 3. checkcode (a linting function).
 4. Automatic TODO/FIXME Report
- Other tools:
 1. Profiler.
 2. Unit testing.
 3. Coder.
- Good OO capabilities.
- Interacts reasonably well with C/C++, Fortran, and R (type !R)

Matlab: disadvantages

- Very expensive if you do not have a university license.
- Age starts to show.
- Many undocumented features: <https://undocumentedmatlab.com/>.
- Features creep causes problems.
- Use of Java makes it prone to crashing and some numerical instabilities.
- Parallelization tools are often disappointing.
- Clones (such as Octave) are quite slow.

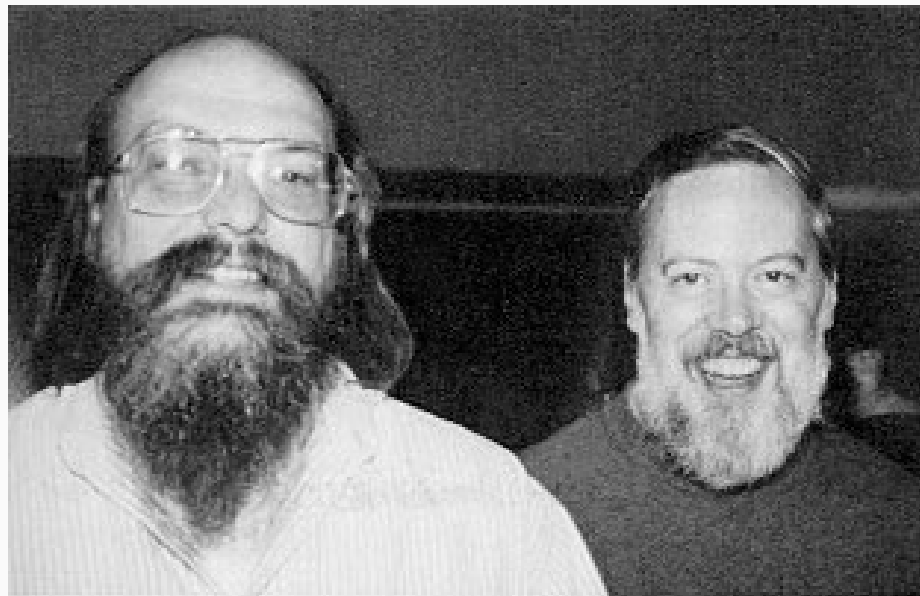
Other choices for scientific computation

- General-purpose languages (GPL):
 1. C.
 2. Fortran.
 3. Java.
 4. Scala.
- Domain-specific languages (DSL):
 1. Mathematica.
 2. Stata.

C



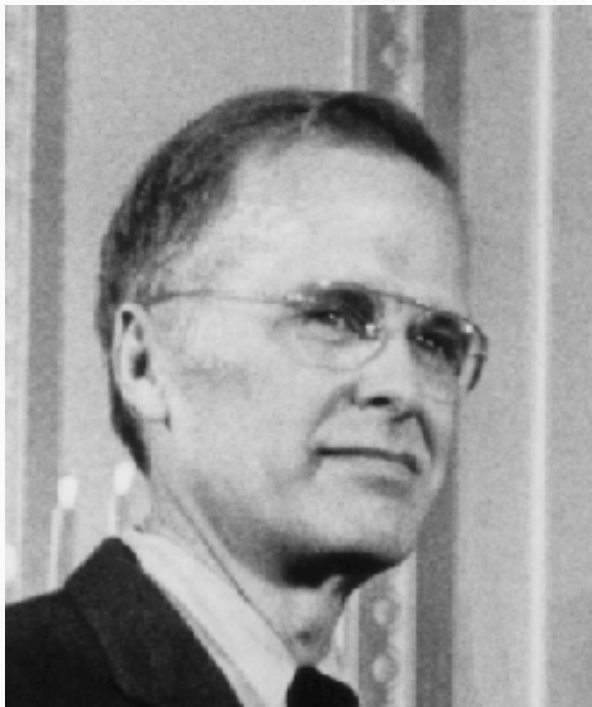
- C was created by Dennis Ritchie at Bell Labs to port UNIX to different machines.
- The name come because it was created after failing with:
 1. Fortran.
 2. With a previous language called B developed by Ken Thompson. B itself derived from a previous language called Basic Combined Programming Language (BCPL), the first brace programming language.
- C is used nowadays for mainly for systems interfaces, embedded controllers, and real-time applications.
- For instance, Linux and R source code are written in C.
- A typical project in a CS class is to code a Lisp interpreter in C.



- C is a transitional language between a high-level language and a low-level language.
- Preprocessor includes header files, macro expansions, conditional compilation, and line control.
- Lean and fast.
- Easy to do everything, easier to really screw up.
- Original philosophy behind C was that programmer needs to check for error (such as vector overflows) not the compiler. This makes C fast, but difficult to debug.
- Pointers and memory management.
- Standard reference: *The C Programming Language (2nd Edition)*, by Brian W. Kernighan and Dennis M. Ritchie (of "Hello, world!" fame).

Fortran

- Grandfather of all modern languages: “Real Programmers can write Fortran in any language.”
- Fortran (“Formula Translation”) developed by John Backus and coworkers in 1957 for the IBM 704.
- However, it has kept updating itself:
 1. Fortran 2008 (Fortran 2018 expected soon!, but only minor changes with respect to Fortran 2008).
 2. Introduction of modern features such as (limited) OO.
- Surprising resilience: still used for scientific and engineering problems (weather forecast, nuclear bombs design, chemical plants,...).
- Used by many economists.



Fortran: advantages

- Large set of best-of-pack numerical libraries.
- Small and compact language:
 1. Easy to learn.
 2. Portable.
 3. Easy to implement quality control by software engineering.
 4. Easy to debug.
- Nice array support.
- Easy to parallelize in shared-memory.

Fortran: disadvantages

- Small community of users: only used for technical computations.
- Expiration date? The 1990s foretells of its death were exaggerated. Rush toward C++ was often costly.
- No speed advantage any longer.
- Limitations of the language (limited data structures, no functional and meta-programming, next-to-none text processing,...).
- Yes, easy to learn but:
 1. Far away from modern approaches to programming.
 2. You would probably not find many classes/books on Fortran at your institution.

Java

- Created by Sun. Now Oracle and open source.
- Evolution from C++:
 1. Pure OO.
 2. Garbage collection.
 3. Cleaner structure.
- Original idea: “Write once, run anywhere (WORA)”.
- Key component: a virtual machine (VM) that performs JIT.
- You suffer a penalty in time with respect to C++.

- General-purpose, functional plus OO, strong static typed language.
- Developed by Martin Odersky.
- Syntax very close to Java, but with much syntactic sugar.
- Runs on the JVM.
- Great for interactions with Spark.
- Nice sequence of online courses on Coursera.



Mathematica

Mathematica: advantages

- Developed by Stephen Wolfram.
- Mainly oriented toward symbolic computation.
- Everything is an expression that gets manipulated.
- Functional programming and list-based (although accepts procedural programming at a performance penalty cost).
- Extensive meta-programming abilities. In particular, easy to generate Fortran and C code.
- You can compile intensive parts of the code.



Mathematica: disadvantages

- Programming approach is different from other languages.
- Difficult to write idiomatic Mathematica code if you come from other traditions.
- Nearly no OO support.
- Cryptic error messages and sparse documentation.
- Smaller community.
- Wolfram's company is high both in undeserved self-promotion and in making life difficult for the user.

Stata

- Statistical package.
- Very popular in microeconometrics.
- Old design and implementation.
- Commercial software.
- Limits the scope of what you can accomplish.
- Similar (and even more so) comments apply to other statistical/econometrics software (EViews, RATS, GAUSS,....).

Alternatives?

- Plenty of alternative GPL that we will not discuss:
 1. C#, Javascript, PHP, Perl, Swift, and Objective-C.
 2. Visual Basic .NET plus its ancestors.
 3. Ruby.
 4. Rust.
 5. Delphi/Object Pascal plus its ancestors.
 6. Go.
 7. Lua.
 8. Ada.
- These languages are not oriented toward scientific computing.
- Family of functional languages (see lecture slides on functional programming).
- Sage for symbolic computation.

Comparison

A baby example

- A basic RBC model:

$$\max \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \log c_t$$

$$\text{s.t. } c_t + k_{t+1} = e^{z_t} k_t^\alpha + (1 - \delta) k_t, \forall t > 0$$

$$z_t = \rho z_{t-1} + \sigma \varepsilon_t, \varepsilon_t \sim \mathcal{N}(0, 1)$$

- If $\delta = 1$, by “guess and verify”:

$$c_t = (1 - \alpha\beta) e^{z_t} k_t^\alpha$$

$$k_{t+1} = \alpha\beta e^{z_t} k_t^\alpha$$

- Calibration:

Parameter	β	α	ρ	σ
Value	0.95	1/3	0.95	0.007

Table 1: Average and Relative Run Time (Seconds)

	Mac		
Language	Version/Compiler	Time	Rel. Time
C++	GCC-7.3.0	1.60	1.00
	Intel C++ 18.0.2	1.67	1.04
	Clang 5.1	1.64	1.03
Fortran	GCC-7.3.0	1.61	1.01
	Intel Fortran 18.0.2	1.74	1.09
Java	9.04	3.20	2.00
Julia	0.7.0	2.35	1.47
	0.7.0, fast	2.14	1.34
Matlab	2018a	4.80	3.00
Python	CPython 2.7.14	145.27	90.79
	CPython 3.6.4	166.75	104.22
R	3.4.3	57.06	35.66
Mathematica	11.3.0, base	1634.94	1021.84

Table 1 (cont.): Average and Relative Run Time (Seconds)

	Mac		
Language	Version/Compiler	Time	Rel. Time
Matlab, Mex	2018a	2.01	1.26
Rcpp	3.4.3	6.60	4.13
Python	Numba 0.37.9	2.31	1.44
	Cython	2.13	1.33
Mathematica	11.3.0, idiomatic	4.42	2.76

Comparison

- Value function iteration, with 35,640 points in the grid of capital and 5 points in the grid of productivity.
- Code has been written in a relatively uniform way across languages.
- Hence, it does not take advantage of language-specific features.
- Code was not optimized for maximum performance, but for clarity.
- Mathematica was a partial exception: because of its functional programming structure, a “matlabish” version is too slow.
- Hence, we may be biased in its favor.
- GCC compilers were faster than LLVM or Intel compilers.

Some (opinionated) advice I

- If you decide to learn one GPL, our advice would be to learn C++.
 1. You will have the most powerful tool existing.
 2. You will understand everything else.
 3. Good programming habits.
 4. Keeps your options open.
 5. You can start with a relatively small subset of the language and move later to OO, meta-programming, and functional programming.
- Python: it is probably a better idea to master Julia first.

Some (opinionated) advice II

- You should only learn Fortran if you need to use legacy code or it is the language that your advisor asks you to code in:
 1. No real differences in performance: back-end of GCC is the same, only front-end differs.
 2. Furthermore, once you have learned C++, you can probably pick up Fortran in a couple of days if you really need to.
- Java or Scala: little reason to pay the speed penalty with respect to C++ unless:
 1. The use of the VM is relevant for the project.
 2. You want to link Java libraries in Matlab, Mathematica, or other DSL or use Spark at a high-level.

Some (opinionated) advice III

- Even if you use a GPL for “heavy duty” computation, a DSL is convenient for:
 1. Simple tasks (editing some data).
 2. Making graphs (great packages available).
 3. Using specialized packages (for most statistical procedures there is a nice R package).
 4. Prototyping (we prototype all the time).
- If the expected computation time of your code is less than a few minutes in Julia or R, the cost of coding in a GPL language probably is too high.
- Given Julia’s current speed, one can think about it as a good “default” language.
- Julia and R are the most useful DSLs for economists. Matlab will remain useful for a long time given the large amount of legacy code on it. Mathematica is powerful, but small installed base. Stata is dominated by R.