A Comparison of Programming Languages in Economics: An Update

S. Borağan Aruoba* University of Maryland

Jesús Fernández-Villaverde† University of Pennsylvania

March 25, 2018

Abstract

We recompute the experiment in Aruoba and Fernández-Villaverde (2015) with the latest version of each programming language. The central conclusions of our original paper remain unaltered: `C++` is the fastest alternative, `Julia` offers a great balance of speed and ease of use, and `Python` is too slow. As the main changes, `Matlab` and `R` have considerably improved their performance, in the case of Matlab to make it competitive, for example, with `Rcpp`.

Key words: Dynamic Equilibrium Economies, Computational Methods, Programming Languages.
JEL classifications: C63, C68, E37.

*University of Maryland, <aruoba@econ.umd.edu>.
†University of Pennsylvania, NBER and CEPR <jesusfv@econ.upenn.edu>.

1. Motivation

This note updates the results in Aruoba and Fernández-Villaverde (2015; hereafter AFV2015) with the new versions of several programming languages. The motivation for the updated results come from the efforts by several languages to improve their numerical performance. `Julia` continues maturing as a programming language with new, important improvements in syntax and performance. `Matlab` has a new, improved just-in-time (JIT) execution architecture. And starting with `R 3.4.0`, `R`'s byte-code compiler is enabled by default at its level 3. The differences in performance are sufficiently important as to justify re-running our codes.

2. Changes in the Experiment

The model is the same than in AFV2015. We change, however, four details of the computations:

1. We double the number of capital grid points to 35,640.
2. We tighten the convergence criteria to 1.0^{-8} (instead of 1.0^{-7}).
3. We only print the results of one of each 20 iterations (plus the first and last), instead of one each 10.
4. Our `Mac` machine is now an Intel Core i7 @2.8 GHz processor, with 4 physical cores, and 16 GB of RAM. It runs OSX 10.13.3.

We introduce changes 1. and 2. because the computation time for the best languages was becoming too fast, complicating accurate measurement. With these changes, we need 302 iterations to convergence (instead of the former 257). We halve the number of print outs because of the very fast languages; printing 32 lines distorted the measurement of effective computing time of the fastest languages. At the same time, having 35,640 grid points of capital keeps computations sufficiently short as to let easy replicability (as we do, for example, while teaching). The original Mac machine in AFV2015 is no longer with us. All the other details of the computation are the same as in AFV2015.

3. Changes in Programming Languages

Beyond using the most recent versions of each language as of March 22, 2018, we made some small changes in the set of languages (and hybrid versions):

1. We drop `Pypy`. The user community of `Pypy` seems to have stagnated and the high-performance `Python` ecosystem is clustering around `Numba` and `Cython`.

2. We drop `R compiled`. With the `R`'s byte-code compiler, the basic script performs better than the compiled one.

3. We introduce a new version of `Julia (fast)` with some optimizations suggested by users after the paper first circulated. `Julia` benefits a lot from a mild investment on optimization and makes it easier to compare with `Numba` and `Cython`, which require some extra work as well with respect to basic `Python`.

4. We report results both for `Python 2.7.14` and `3.6.4`.

4. New Results

We summarize our new results in Table 1. As in AFV2015, we show the average run time and the relative performance of each code in terms of the best performer, still `C++` with `GCC`. `C++` with `GCC` runs in 1.60 seconds, a bit more than twice as long as before (we have twice as many grid points and a slightly tighter convergence criterion). The `Intel C++` and `Clang` run in roughly the same time (with a 3-4% penalty in time). GCC Fortran also generates executable code of the same speed than `C++` with `GCC`. `Intel Fortran` suffers some deterioration in performance of around 9%. But, in comparison with AFV2015, all five compilers deliver basically the same speed.

`Java` is now 2.0 times slower than `C++`, somewhat reducing the difference three years ago. `Julia` has also improved to 1.47 times the speed of `C++`, and, its fast version, to 1.34 times, truly a fantastic performance given how easy is to code in `Julia`. `Matlab` is one of the great improvers: now it is only 3 times slower than the best `C++` executable. But the best improvement is in `R`, now only 36 times slower than the best `C++` executable, instead of 281 (compiled) and 475 (script). `Python` also has speeded-up to 91-104 times slower than the best `C++` executable.

In terms of the hybrid and special cases, a `Mex` file written in `C++`, `Matlab`, `Rcpp` in `R`, `Numba`'s decorated code runs and `Cython` runs approximately at the same relative speed than in AFV2015. Idiomatic `Mathematica` is 2.76 times slower than `C++`, showing an stagnation of the language.

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
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

5. Concluding Remarks

The central conclusions of AFV2015 remain unaltered: `C++` is the fastest alternative, `Julia` offers a great balance of speed and ease of use, and `Python` is too slow. As the main changes, `Matlab` and `R` have considerably improved their performance, in the case of Matlab to make it competitive, for example, with `Rcpp`, without the need to learn any `C++`.

References

[1] Aruoba, S.B., and J. Fernández-Villaverde (2015). "A Comparison of Programming Languages in Macroeconomics." *Journal of Economic Dynamics and Control* 58, 265–273.