Journal of Economic Dynamics & Control 58 (2015) 265–273

Contents lists available at ScienceDirect

Journal of Economic Dynamics & Control

journal homepage: www.elsevier.com/locate/jedc

A comparison of programming languages in macroeconomics

S. Borağan Aruoba [a], Jesús Fernández-Villaverde [b]

[a] University of Maryland, United States
[b] University of Pennsylvania, NBER and CEPR, United States

ARTICLE INFO

Article history:
Received 12 May 2015
Accepted 15 May 2015
Available online 22 May 2015

JEL classification:
C63
C68
E37

Keywords:
Dynamic equilibrium economies
Computational methods
Programming languages

ABSTRACT

We solve the stochastic neoclassical growth model, the workhorse of modern macroeconomics, using C++14, Fortran 2008, Java, Julia, Python, Matlab, Mathematica, and R. We implement the same algorithm, value function iteration, in each of the languages. We report the execution times of the codes in a Mac and in a Windows computer and briefly comment on the strengths and weaknesses of each language.

1. Introduction

Computation has become a central tool in economics. From the solution of dynamic equilibrium models in macroeconomics or industrial organization, to the characterization of equilibria in game theory, or in estimation by simulation, economists spend a considerable amount of their time in coding and running fairly sophisticated software. And while some effort has been focused on the comparison of different algorithms for the solution of common problems in economics (see, for instance, Aruoba et al., 2006), there has been little formal comparison of programming languages. This is surprising because there is an ever-growing variety of programming languages and economists are often puzzled about which language is best suited to their needs.[1] Instead of a suite of benchmarks, researchers must rely on personal experimentation or on "folk wisdom."

In this paper, we take a first step at correcting this unfortunate situation. The target audience for our results is younger economists (graduate students, junior faculty) or researchers who have used the computer less often in the past for numerical analysis and who are looking for guideposts in their first incursions into computation. We focus on a macroeconomic application but we hope that much of our conclusions and insights carry over to other fields such as industrial organization or labor economics, among others.

We solve the stochastic neoclassical growth model, the workhorse of modern macroeconomics, using C++, Fortran, Java, Julia, Python, Matlab, Mathematica, and R. We implement the same algorithm, value function iteration, in each of the languages, and measure the execution time of the codes in a Mac and in a Windows computer. The advantage of our

E-mail addresses: aruoba@econ.umd.edu (S.B. Aruoba), jesusfv@econ.upenn.edu (J. Fernández-Villaverde).

[1] This also stands in contrast to work in other fields, such as Prechelt (2000) and Lubin and Dunning (2013), or web projects, such as *The Computer Language Benchmarks Game* (see http://benchmarksgame.alioth.debian.org/).

http://dx.doi.org/10.1016/j.jedc.2015.05.009
0165-1889/

algorithm, value function iteration, is that it is "representative" of many economic computations: expensive loops, large matrices to store in memory, and so on. Thus, while our investigation does not entail a full suite of benchmarks, both our model and our solution method are among the best available choices for our investigation. In addition, our two machines, a `Mac` and a `Windows` computer, are perhaps the two most popular environments for software development for economists.

The key take-aways of our analysis are as follows:

1. `C++` and `Fortran` are considerably faster than any other alternative, although one needs to be careful with the choice of compiler. The many other strengths of `C++` in terms of capabilities (full object orientation, template meta-programming, lambda functions, large user base) make it an attractive language for graduate students to learn. On the other hand, `Fortran` is simple and compact – and, thus, relatively easy to learn – and it can take advantage of large amounts of legacy code.
2. `Julia` delivers outstanding performance, taking only about 2.5 times longer to execute than `C++`, while `Matlab` takes about 10 times longer. Given how close `Julia`'s syntax is to `Matlab`'s and the fact that it is open-source and that the language has been designed from the scratch for easy parallelization, many economists may want to learn more about it. However, `Julia`'s standard is still evolving (causing potential backward incompatibilities in the future) and there are only a few libraries for it at the moment.
3. While `Python` and `R` are popular in economics, they do not perform well in our application, taking 44 to 491 times longer to execute than `C++`.
4. Hybrid programming and special approaches can deliver considerable speed-ups. For example, when combined with `Mex` files, `Matlab` takes only about 1.5 times longer to execute than `C++` and when combined with `Rcpp`, `R` takes about 4 times longer to execute. Similar numbers hold for `Numba` (a just-in-time compiler for `Python` that uses decorators) and `Cython` (a static compiler for writing `C` extensions for `Python`) in the `Python` ecosystem. While `Mex` files were faster, we found `Rcpp` to be elegant and easy to use. These numbers suggest that a researcher can use the friendly environment of `Matlab` or `R` for everyday tasks (data handling, plots, etc.) and rely on `Mex` files or `Rcpp` for the heavy computations, especially those involving loops.
5. The baseline version of our algorithm in `Mathematica` is very slow, unless we undertake a considerable rewriting of the code to take advantage of the peculiarities of the language.

Some could argue that our results are not surprising as they coincide with the guesses of an experienced programmer. But we regard this comment as a point of strength, not weakness. It is a validation that our exercise was conducted under reasonably fair conditions. We do not seek to overturn the experience of knowledgeable programmers, but to formalize such experience under well-described and explicitly controlled conditions and to report the information to others.

We also present some brief comments on the difficulty of implementation of the algorithm in each language and on the additional tools (integrated development environments or IDEs, debuggers, etc.) existing for each language. While this is a treacherous and inherently subjective exercise, perhaps our pointers may be informative for some readers. Since the codes are posted at our `github` repository, the reader can gauge our results and remarks for himself.[2]

The rest of the paper is structured as follows. First, in Section 2, we introduce our application and algorithm. In Section 3 we motivate our selection of programming languages. In Section 4, we report our results. Section 5 concludes.

2. The stochastic neoclassical growth model

For our exercise, we pick the stochastic neoclassical growth model, the foundation of much work in macroeconomics. We solve the model with value function iteration. In that way, we compare programming languages for their ability to handle a task such as value function iteration that appears everywhere in economics and within a well-understood economic environment.

In this model, a social planner picks a sequence of consumption c_t and capital k_{t+1} to solve

$$\max_{\{c_t, k_{t+1}\}} \mathbb{E}_0 \sum_{t=0}^{\infty} (1-\beta)\beta^t \log c_t \tag{1}$$

where $\mathbb{E}_0$ is the conditional expectation operation, β the discount factor, and the resource constraint is given by $c_t + k_{t+1} = z_t k_t^{\alpha} + (1-\delta)k_t$, where productivity z_t takes values in a set of discrete points $\{z_1, \ldots, z_n\}$ that evolve according to a Markov transition matrix Π. The initial conditions, k_0 and z_0, are given. While, in the interest of space, we have written the model in terms of the problem of a social planner, this is not required and we could deal, instead, with a competitive equilibrium.

For our calibration, we pick $\delta = 1$, which implies that the model has a closed-form solution $k_{t+1} = \alpha\beta z_t k_t^{\alpha}$ and $c_t = (1-\alpha\beta) z_t k_t^{\alpha}$. This will allow us to assess the accuracy of the solution we compute. Then, we are only left with the need to choose values for β, α, and the process for z_t. But since $\delta = 1$ is unrealistic, instead of targeting explicit moments of the data, we just pick conventional values for these parameters and processes. For β we pick 0.95, 1/3 for α, and for z_t we

[2] https://github.com/jesusfv/Comparison-Programming-Languages-Economics