Doug Frey, CBEE, UMBC

Is it worth the effort to learn to use Julia?

Julia is a new, free and open source computer programming language that was developed at MIT. Julia was designed for science and engineering applications and especially for high-performance numerical computing and machine learning. The goal of the Julia language is to achieve an unprecedented combination of programming simplicity and computational speed. But there are already several options available for computer programming languages. Is another option really needed? Below are the cases for and against learning to use Julia.

The case for learning to use Julia.

1. Julia is fast.

Below is a graph that compares single-threaded computation times on a logarithmic scale for various computer programming languages in their native operating state with the coding implemented in the most straightforward style. Note that smaller times are better. As shown, by this comparison Julia is as fast as C or Fortran, which are classic compiled languages having very short execution times. Julia is also very easy to run in parallel on multiple CPU cores and to run on GPUs (graphics processing unit) compared to the alternative languages shown below. The comparisons shown below were taken from Bezanson et al., "Julia: A fresh approach to numerical computing", SIAM Review, 59(1), 65-98, 2017. This article was written by the developers of Julia so the comparisons could be biased. However, other independent sources confirm qualitatively the results shown below.

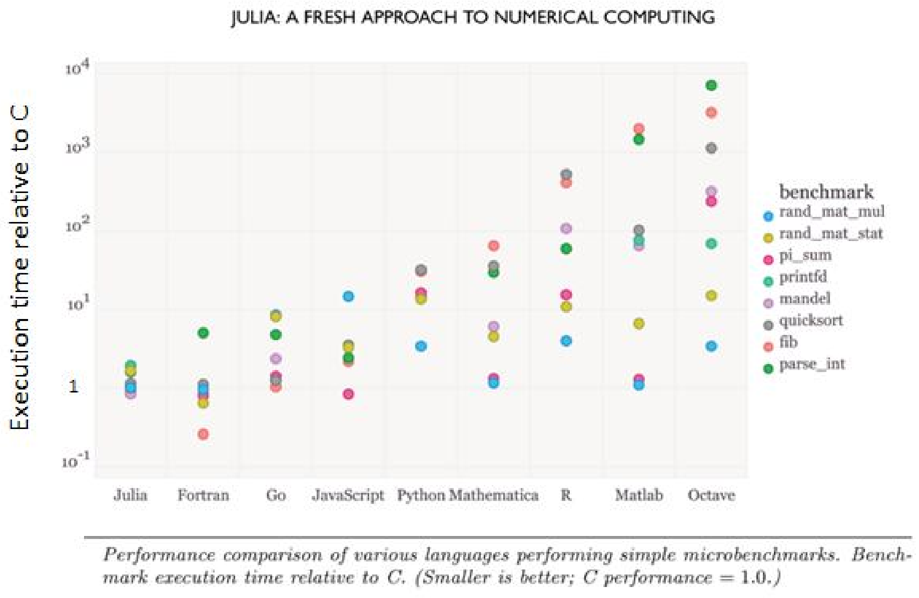

Fig. 1. Comparison of execution times for various benchmarks (loops, recursion, sorting, etc., implemented in the most straight forward style) and for several programming languages. Smaller is better. Execution times are relative to execution times for C.

An impartial description of the computation speed obtained by Julia is given in the following 2019 special feature article in the journal ***Nature***

https://www.nature.com/articles/d41586-019-02310-3 (https://www.nature.com/articles/d41586-019-02310-3)

2. Julia includes many advanced packages that are newly written.

Most Julia packages are newly written and are written entirely in Julia. As a result, several Julia packages are "best-in-class" and superior to the corresponding packages used in other computer languages. Julia packages that are widely considered to be superior to corresponding packages used in other languages include DifferentialEquations (for solving ODEs) and JuMP (for mathematical programming and optimization). Other Julia packages that are considered to be of excellent quality include Flux (for machine learning) and BioJulia (for computational biology).

The figure below gives computation times on the vertical axis (using a log scale) versus the numerical error obtained for the numerical solution of the Lotka-Voltera differential equations (the predator-prey differential equations) using various numerical packages. Note that smaller times are better. As shown, for the case illustrated the DifferentialEquations package in Julia is 10 to 100 times faster at a given error level than the comparable packages in Matlab and Python.

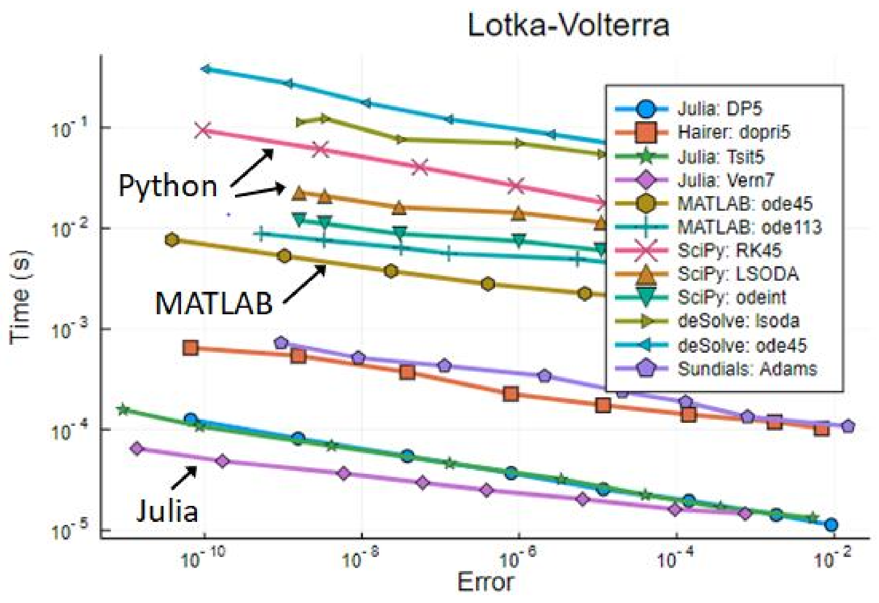

Fig. 2. Comparison of execution times for ODE packages.

3. Julia is a full-featured language that has capabilities not available in other languages.

Julia includes modern, advanced features that are difficult to implement in older languages (such as Matlab) including multiple dispatch, differentiable programming, a flexible and declarative type system, support for unicode characters, and homoiconic capabilities that facilitate metaprogramming (where code functions as data for other code that produces new code).

4. Julia is free and open source.

Julia is free and open source, and the underlying codes in Julia and in Julia packages are publicly available and not proprietary.

5. Julia is easy to learn and use.

The Julia language incorporates the best features of Python, Matlab, and Lisp with a unified and simplified syntax. If you know Python, Matlab, or Lisp then you are already familiar with Julia. The following link compares the syntax in Python, Matlab, and Julia for basic operations:

https://cheatsheets.quantecon.org/ (https://cheatsheets.quantecon.org/)

6. Julia provides a superior teaching environment for machine learning (and for many other topics).

Julia user-written code, Julia packages, and most of the Julia language itself are all written in Julia. As a result you can examine a program at all levels from top to bottom by knowing a single language. This makes Julia ideal from a teaching and learning perspective. In contrast, most of the packages used in Matlab and Python are written in C or Fortran, so that their content is relatively inaccessible.

7. Julia can employ Python, C and Fortran packages when needed.

You can use all your favorite Python packages from within Julia if you desire. For example, the Python package TensorFlow that was developed at Google for machine learning can be used from within Julia.

The case against learning to use Julia.

1. Julia is new and currently has a relatively small number of users.

The first experimental "beta" version of Julia was released in 2012 and version 1.0 of Julia was released in 2018. Consequently the Juila user base is currently comparatively small, and there are correspondingly a smaller number of pre-written packages available compared to other computer languages. Below is a table from the PYPL website that gives the relative number of internet searches in the USA for computer language tutorials in August 2020. As shown, Julia currently is used at a level higher than Visual Basic but at a lower level than Matlab, R, and Python.

PYPL relative rankings of computer language usage in the USA (Aug., 2020):

Python:	32.5
Java:	14.9
R:	5.6
C/C++:	4.9
Matlab:	1.9
Julia:	1.1
Visual Basic:	0.8
Haskel:	0.6
Maple:	0.2

One way to compare Python and Julia relative usage in the scientific and engineering areas is to note that the 2019 SciPy Conference (which is the annual conference on the scientific use of Python) had 720 attendees while JuliaCon 2019 had 350 attendees.

It is also useful to examine the PYPL rankings of programming language usage for languages used mainly for science and engineering applications (i.e., Matlab, Julia, and R) on a logarithmic scale for the last 15 years as illustrated below so future usage can be estimated. As shown, the usage of Matlab peaked in 2010 and has steadily declined since then, while Julia has shown a steady increase in usage since its introduction in 2012. Note also that the usage of R, which is a programming language used mainly for statistical analysis, surpassed the usage of Matlab in 2013. Further, the usage of R peaked in 2018 and is now slowly declining. If the trends shown in the figure below continue, Julia can be expected to overtake Matlab in usage by 2024 or even earlier.

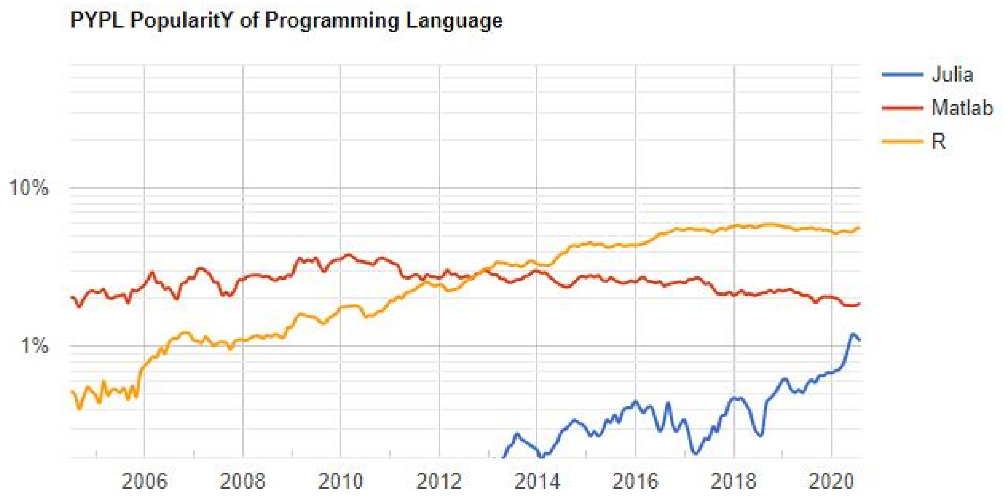

Fig. 3. Comparison of programming language usage in the USA over time for Matlab, Julia, and R.

2. Some alternative languages have features that are not available in Julia.

Some computer languages have unique capabilities, and if you need to use those capabilities then Julia is not a good choice for you. For example, Matlab can interface directly to Comsol Multiphysics, while Julia does not have this feature.

3. Computation speed is not important in certain applications.

If computation speed is not important in your applications, then possibly Julia may not be the best choice for you.