But Isn't Python Slow?

It may seem like a disadvantage to write a library for doing computationally difficult tasks in a language that's got a reputation for being much slower than its competators. Python is an uncompiled language which doesn't allow it to use many of the optimizations that come with compiled languages. Raw python will always be slower than alternative languages like Java, Julia, and C.

However raw Python is rarely used for programming. Python is powerful because of its libraries which give it much more flexibility and speed. Python's libraries are typically written in C and many run just as fast. For sections of code that need to be written in python, there are specialized libraries like Cython and Numba which can dramatically improve the proformance of Python code.

The study "A comparison of programming languages in macroeconomics (2015)" found that raw python was much slower on both Windows and Mac. Using Numba, a Just In Time compiler, the code ran faster than Julia and Java on both mac and windows. In their 2018 update, Python with Numba was roughly as fast as the original Julia implimentation and Cython was as fast as their newly optimized version.

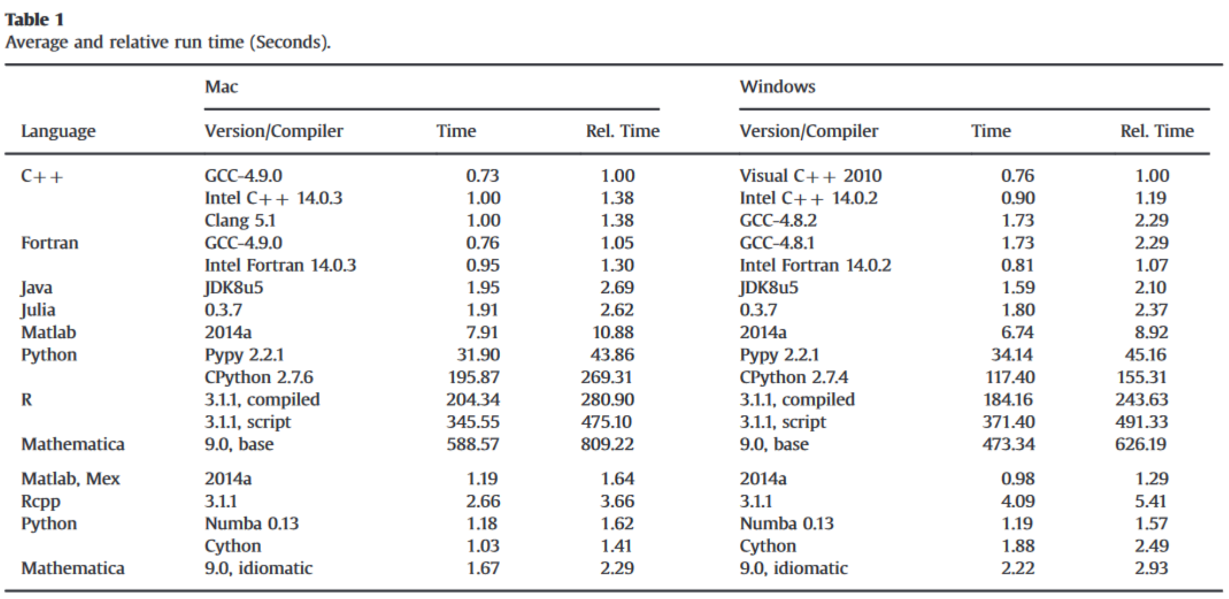


Table 1
Average and relative run time (Seconds).

	Mac			Windows		
Language	Version/Compiler	Time	Rel. Time	Version/Compiler	Time	Rel. Time
C++	GCC-4.9.0	0.73	1.00	Visual C++ 2010	0.76	1.00
	Intel C++ 14.0.3	1.00	1.38	Intel C++ 14.0.2	0.90	1.19
	Clang 5.1	1.00	1.38	GCC-4.8.2	1.73	2.29
Fortran	GCC-4.9.0	0.76	1.05	GCC-4.8.1	1.73	2.29
	Intel Fortran 14.0.3	0.95	1.30	Intel Fortran 14.0.2	0.81	1.07
Java	JDK8u5	1.95	2.69	JDK8u5	1.59	2.10
Julia	0.3.7	1.91	2.62	0.3.7	1.80	2.37
Matlab	2014a	7.91	10.88	2014a	6.74	8.92
Python	Pypy 2.2.1	31.90	43.86	Pypy 2.2.1	34.14	45.16
	CPython 2.7.6	195.87	269.31	CPython 2.7.4	117.40	155.31
R	3.1.1, compiled	204.34	280.90	3.1.1, compiled	184.16	243.63
	3.1.1, script	345.55	475.10	3.1.1, script	371.40	491.33
Mathematica	9.0, base	588.57	809.22	9.0, base	473.34	626.19
Matlab, Mex	2014a	1.19	1.64	2014a	0.98	1.29
Rcpp	3.1.1	2.66	3.66	3.1.1	4.09	5.41
Python	Numba 0.13	1.18	1.62	Numba 0.13	1.19	1.57
	Cython	1.03	1.41	Cython	1.88	2.49
Mathematica	9.0, idiomatic	1.67	2.29	9.0, idiomatic	2.22	2.93

2015 language comparison [Aruoba, S.B., and J. Fern·ndez-Villaverde]

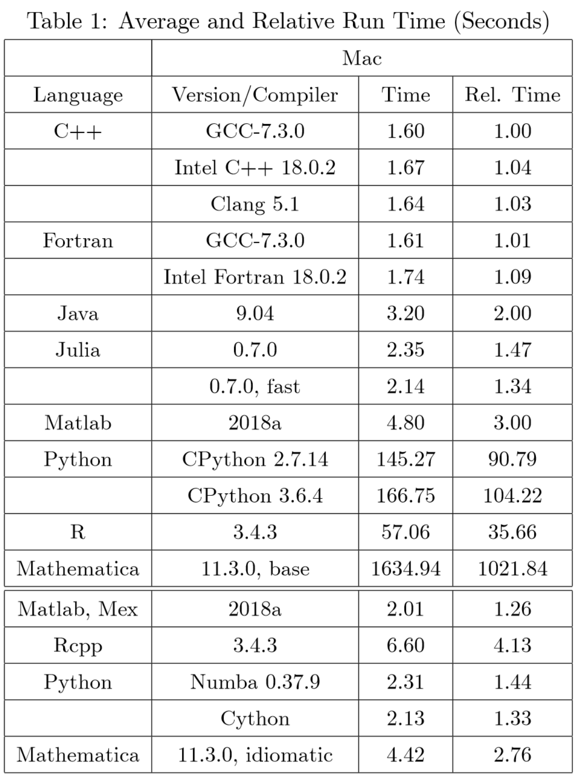
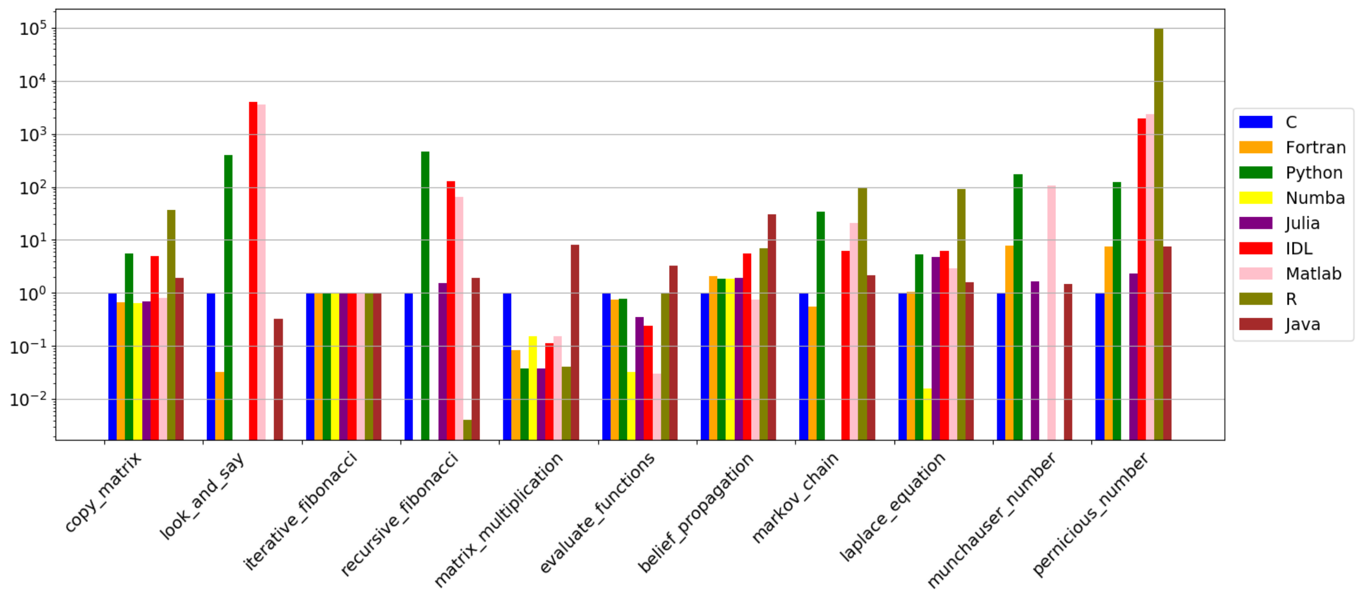


Table 1: Average and Relative Run Time (Seconds)

	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

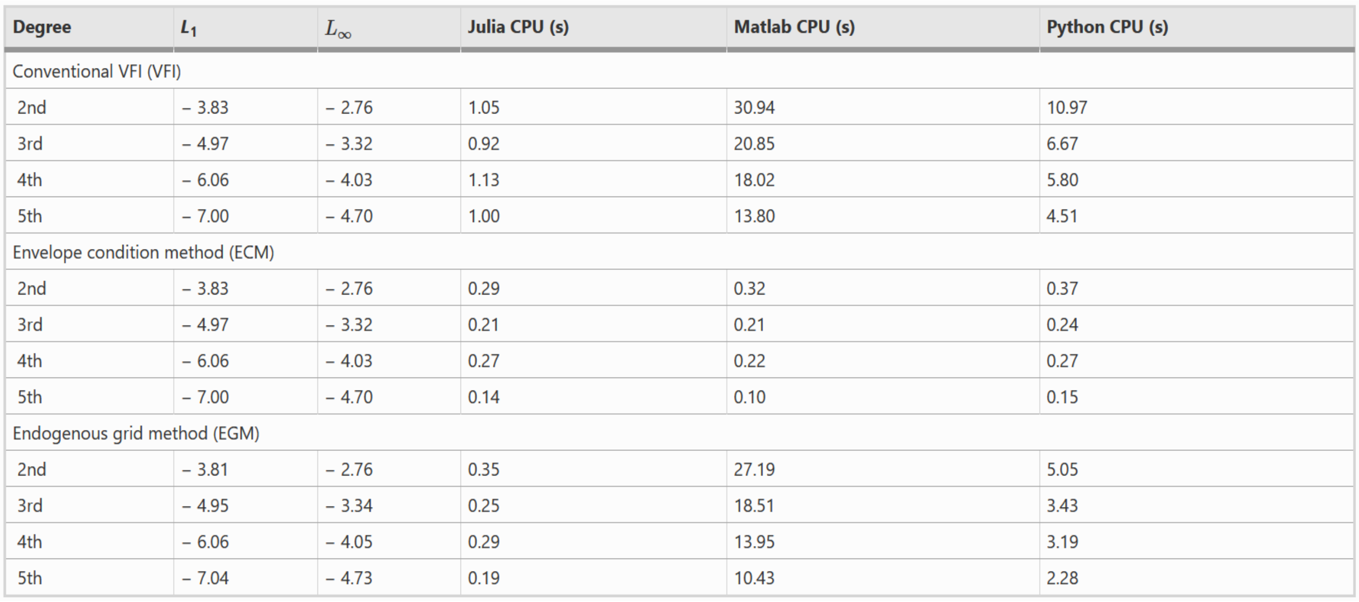
2018 language comparison [Aruoba, S.B., and J. Fern·ndez-Villaverde]

In a Nasa comparison, Python is slower for certain tasks, like string manipulations and recursive functions, while its roughly as fast, as Julia when working with arrays, especially when using Numba. This specific analysis has a number of problems with its benchmarking. Several of the Python solutions can be speed up dramatically with a small amount of tweaking, and others have bugs in them which make them appear like they're faster than they are. Occasionally the solution method will be different across languages. In the 2021 analysis, the R method of calculating fibonacci numbers recursively is much faster because it uses a different method than the other languages. If all of them used it their results would be much more similar. A Numba method for calculating the fibonacci numbers recursively isn't provided, but applying it to their raw python method, speeds it up by roughly 50x. More issues with this comparison are discussed on the issues section of the github page.



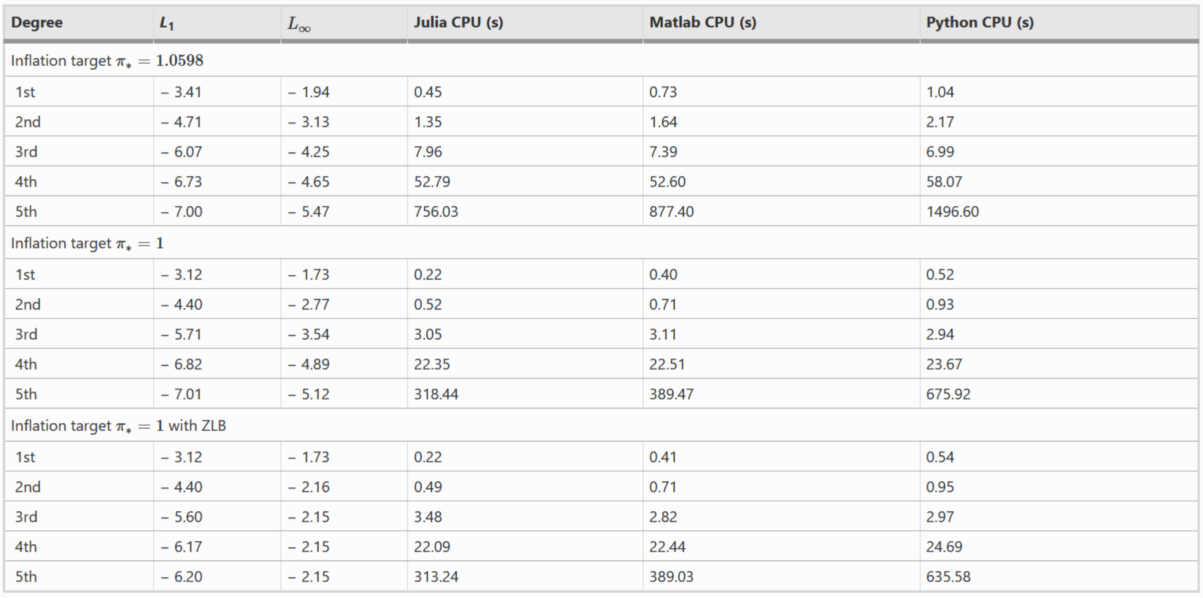
2021 language comparison [Jules Kouatchou and Alexander Medema]

The paper "Matlab, Python, Julia: What to Choose in Economics?" compares Matlab, Python, and Julia, for two specific economic models; A stripped down version of the RBC model, and a more complicated New Keynsian model. They optimized their solutions for all three languages, and employeed Numba for python, though there may still be room for improvement. They find that Python and Matlab are slower than Julia for the Value Function ideration methods, and the Endogenous grid method used heavily by this project, because their implimentation wasn't vectorized. Python arrays run faster the larger they are because the overhead is proportionally smaller. When using a vectorized method that allows Matlab and Python to take advantage of this, they are comparably fast. In the comparison for their solution to the new keynsian model, all three were similarly fast with Python only being moderately slower for the largest approximation, though the authors didn't speculate why. During their multicore processing test, Python and Julia ran similarly fast.



Degree	L_1	L_∞	Julia CPU (s)	Matlab CPU (s)	Python CPU (s)
Conventional VFI (VFI)					
2nd	− 3.83	− 2.76	1.05	30.94	10.97
3rd	− 4.97	− 3.32	0.92	20.85	6.67
4th	− 6.06	− 4.03	1.13	18.02	5.80
5th	− 7.00	− 4.70	1.00	13.80	4.51
Envelope condition method (ECM)					
2nd	− 3.83	− 2.76	0.29	0.32	0.37
3rd	− 4.97	− 3.32	0.21	0.21	0.24
4th	− 6.06	− 4.03	0.27	0.22	0.27
5th	− 7.00	− 4.70	0.14	0.10	0.15
Endogenous grid method (EGM)					
2nd	− 3.81	− 2.76	0.35	27.19	5.05
3rd	− 4.95	− 3.34	0.25	18.51	3.43
4th	− 6.06	− 4.05	0.29	13.95	3.19
5th	− 7.04	− 4.73	0.19	10.43	2.28

Consumption function solution comparison [Coleman, C., Lyon, S., Maliar, L. et al.]



Degree	L_1	L_∞	Julia CPU (s)	Matlab CPU (s)	Python CPU (s)
Inflation target $\pi_* = 1.0598$					
1st	− 3.41	− 1.94	0.45	0.73	1.04
2nd	− 4.71	− 3.13	1.35	1.64	2.17
3rd	− 6.07	− 4.25	7.96	7.39	6.99
4th	− 6.73	− 4.65	52.79	52.60	58.07
5th	− 7.00	− 5.47	756.03	877.40	1496.60
Inflation target $\pi_* = 1$					
1st	− 3.12	− 1.73	0.22	0.40	0.52
2nd	− 4.40	− 2.77	0.52	0.71	0.93
3rd	− 5.71	− 3.54	3.05	3.11	2.94
4th	− 6.82	− 4.89	22.35	22.51	23.67
5th	− 7.01	− 5.12	318.44	389.47	675.92
Inflation target $\pi_* = 1$ with ZLB					
1st	− 3.12	− 1.73	0.22	0.41	0.54
2nd	− 4.40	− 2.16	0.49	0.71	0.95
3rd	− 5.60	− 2.15	3.48	2.82	2.97
4th	− 6.17	− 2.15	22.09	22.44	24.69
5th	− 6.20	− 2.15	313.24	389.03	635.58

New Keynsian model comparison [Coleman, C., Lyon, S., Maliar, L. et al.]

Overall Python is generally slower that Julia, C, and Fortran, though not by much. Even though raw python is much slower, the strength of Python comes from its libraries, which run at the speed of the languages they're written in with a usually insignificant amount of overhead. In situations where python needs to be speeded up, tools like numba and Cython can both bring it up to par.