Benchmarking of Different Programming Languages

White Paper

December, 2012
www.mu-sigma.com

Benchmarking of Different Programming Languages

Avinash Joshi
Labs -Innovation and Development
Mu Sigma Business Solutions Pvt Ltd

Abstract

This paper details the procedure followed in choosing the programming language to be used for writing agents for agent based modelling(referred to as ABM in future). Faster computations by an agent results in quicker response and hence, more efficiency which is crucial in this world of real-time analytics. Different programming languages are built differently and follow different protocols. Each programming language is created for a specific purpose and has its share of advantages and disadvantages. The goal of this paper is to find a programming language which will be the most efficient in terms of the computation time, learning curve, ability to run the code in parallel, interfacing with R and availability of libraries. A list of languages made on the basis of some intuition, will have a set of benchmark tests run on them to decide on the language for building agents.

Contents

1 Introduction **3**

2 Field of study **4**
2.1 Hypothesis/Intuition 4

3 Challenger method **4**

4 Approach to tackle the problem **4**

5 Experiment environment **4**
5.1 Experiments 4
5.2 Evaluation Criteria 5

6 Outcome of benchmark test on various languages **6**

7 Conclusion **8**

8 Reference material **9**

9 Appendix **9**
9.1 Bench Mark results (Actual time in seconds) 9
9.2 Link to the test scripts 9

1 Introduction

Programming language is an artificial language designed to execute instructions on a machine, particularly a computer. Programming languages can be used to create programs that control the behavior of a machine and/or to express algorithms precisely. Each language has a unique set of keywords and a special syntax for organizing program instructions. High-level programming languages are simple to use as they are similar to human languages. However, they are far more complex than the languages the computer actually understands, called machine languages. Each different type of CPU has its own unique machine language.
Lying in the spectrum between machine languages and high-level languages, are the 'Assembly Languages'. Assembly languages are similar to machine languages, but are easier to program in, as they allow a programmer to substitute names for numbers. Machine languages consist of numbers only. However, regardless of the language used, the program eventually needs to be converted into machine language for the computer to understand it. There are two ways to do this:

- Compile the program
- Interpret the program

An interpreter translates high-level instructions into an intermediate form, which it then executes. In contrast, a compiler translates high-level instructions directly into machine language. Compiled programs generally run faster than interpreted programs. The advantage of an interpreter, however, is that it does not need to go through the compilation stage during which machine instructions are generated. This process can be time-consuming if the program is long. The interpreter, on the other hand, can immediately execute high-level programs.

The question that consumes a lot of time and energy of computer professionals is regarding which language is the best. Every language has its strengths and weaknesses. For example, FORTRAN is a particularly good language for processing numerical data, but it does not lend itself very well to organizing large programs. Pascal is very good for writing well-structured and readable programs, but it is not as flexible as the C programming language. C++ embodies powerful object-oriented features, but is complex and difficult to learn.

For building agents, the major computations involve linear algebra using matrices and optimization. Keeping the above pointers in mind, a set of languages that need to be bench-marked and tested upon, have been identified. The intuition behind choosing these languages is explained later in the paper.

- Julia
- Incanter/Clojure
- Python + SciPi
- Fortran
- C++/Rcpp
- Scala

2 Field of study

Research for finding the best possible language for a specific set of operations. Find the accurate set of tests needed to be run so as to identify a language that is not only efficient but also simple enough to learn.

2.1 Hypothesis/Intuition

Programming Language	Reasons
1.Julia	•Faster than R owing to JIT compiler and better parallel computing •Easier to learn as it is similar to R •Not much community support and very few libraries •Easy to interface with C and its libraries
2.Incanter/Clojure	•Based on Java •Faster than R •Java numerical libraries available •Should be easy to pick up for anyone who has worked on java before •Eclipse can be used as an IDE
3.Python + SciPi	•Faster than R under certain condtions(Memory clearance etc) •High level language hence easy to pick up •Has decent set of libraries for numerical computation though not many libraries for stats compared to R
4.C++/Rcpp	•Amazing computational speed. •But needs to learn C++(which will take time) •Good interface with R •Good set of libraries
5.Fortran	•Should have very good computational speed. •But needs to learn a low level compiled language(which will take time) •Good set of libraries
5.Scala	•Similar to Incanter and Clojure •Java based

3 Challenger method

Multiple benchmarking functions are available. Adequate tests have been chosen to measure different aspects of a programming language that would be required to build agents.

4 Approach to tackle the problem

A set of tests which require different aspects of a programming language, have been devised. These tests need to be run on different languages and the time required needs to be tabulated. Also, the amount of time required to learn the language needs to be monitored to have a qualitative measure of each language.

5 Experiment environment

5.1 Experiments

The following tests are being run on a virtual machine on a local system. Following are the specs of the system.

- OS - Ubuntu 12.04
- Number of cores - 4
- RAM - 12 GB

Benchmark tests run: Basic matrix operations and optimization. This helps to determine how effective each language is, while dealing with matrices. Each test is run 15 times and the average time is taken as the measure to avoid anomalies in the execution time.
Test 1 : *Creation and deformation. Changing the dimensions to* 1250×5000 *(1,000,000 times) of a* 2500×2500 *matrix.*

Test 2 : *Creating a* 2400×2400 *matrix and taking the 1000th power of each element.*
Test 3 : *Sort a vector of* $7,000,000$ *elements in ascending order.*
Test 4 : *Creating a* 2800×2800 *matrix and calculating the product of the transpose of the matrix with the original matrix.*
Test 5 : *Linear regression on a* 3000×3000 *matrix.*
Test 6 : *Fft over 2,400,000 random values*
Test 7 : *Compute eigen values of a* 640×640 *matrix.*
Test 8 : *Compute determinant of a* 2500×2500 *matrix.*
Test 9 : *Find the cholesky decomposition of a* 3000×3000 *matrix.*
Test 10 : *Find the inverse of a* 1600×1600 *matrix.*
Test 11 : *Generate 3,500,000 numbers of a fibbonaci sequence*
Test 12 : *Creating a* 3000×3000 *Hibbert matrix.*
Test 13 : *Creation of a 500x500 Toeplitz matrix(Test for nested loops)*
Test 14 : *Sample 20000 points using Gibbs sampler for a joint distribution of Gamma and Normal.*
Test 15 : *Optimization of Rastrigin Function using Simulated Annealing algorithm.*

5.2 Evaluation Criteria

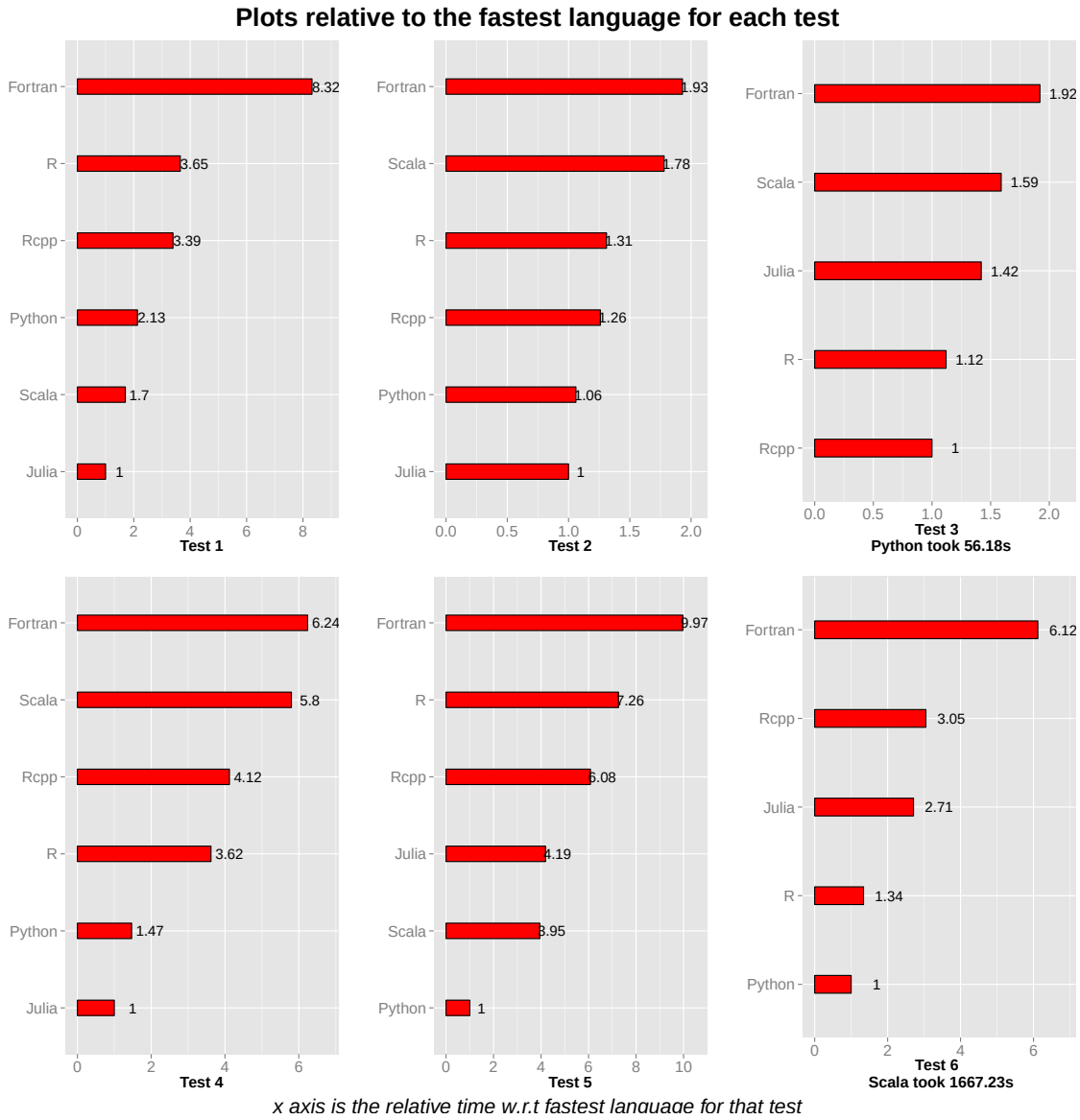
Following is a table depicting ratings of each language on different parameters for qualitative analysis. **Higher the number, more efficient is the language i.e if the parameter is testing for ease of process, then 5 means it is easiest while 0 means it is hardest.**

Evaluation Criteria	Julia	Python	R	Rcpp	Fortran	Scala
Ease of learning	4	4	5	4	3	3
Documentation	4	4	4	4	3	3
Amount of libraries	2	4	5	5	5	4
Online support	3	5	5	3	2	4
Integration with R	0	4	5	5	0	2
Available IDE's	0	5	5	5	0	5
Setting it up on Linux/Windows	4	5	4	5	5	3
Integration with Java	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
Type of programming	Scripting	Scripting	Scripting	OOP	Low level functional	OOP/functional
Maturity of the language	1	5	4	4	5	4
Compiled/Interpreted	Interpreted	Interpreted	Interpreted	Compiled	Compiled	Interpreted

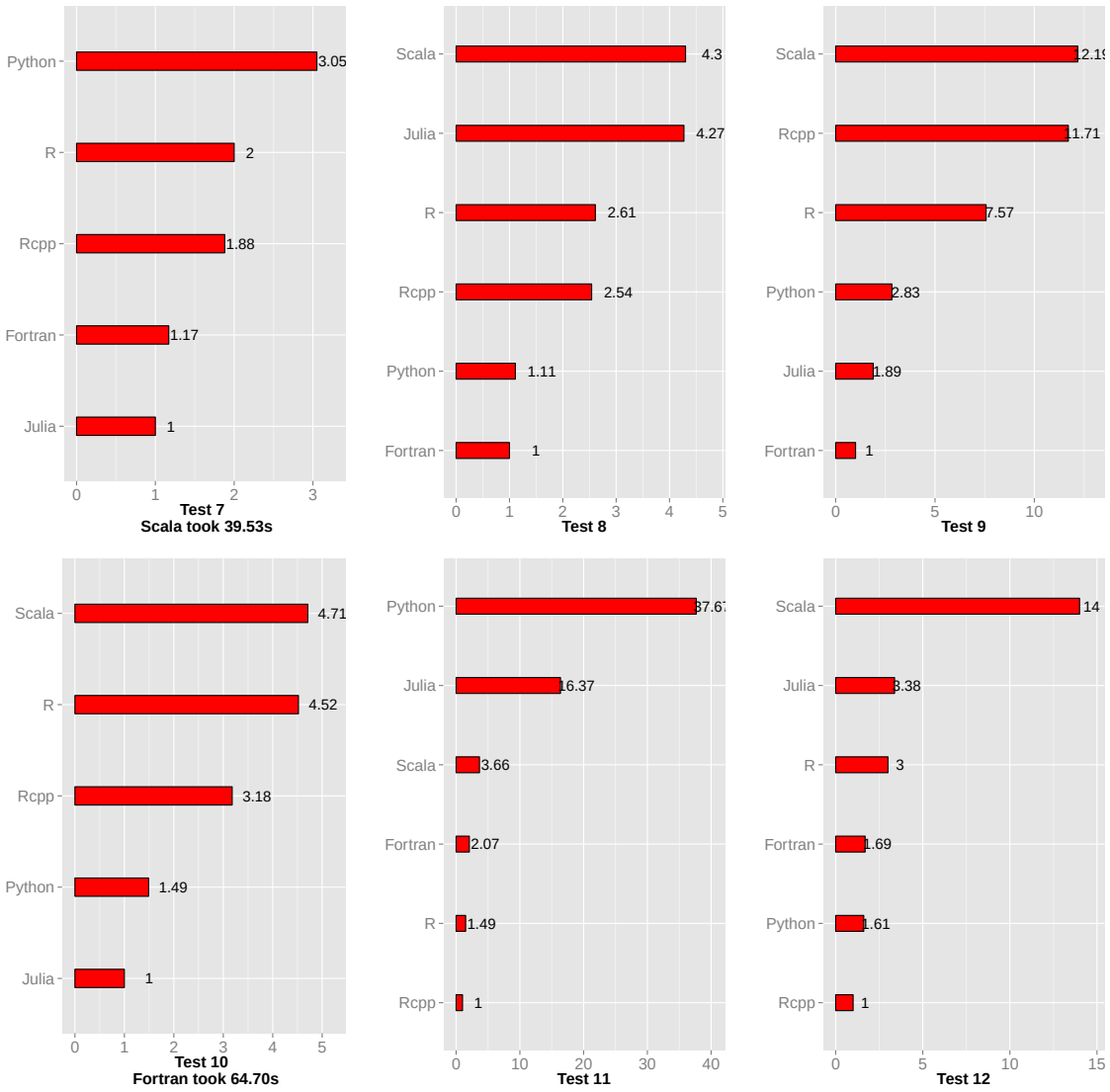
Note: Incanter/Clojure was considered for benchmarking the languages. However, it was discontinued after a certain point as the performance was extremely bad and for improving it, interfacing with Java would be required. As this language is meant to be utilized by analysts, it was dropped from the benchmarking excercise.

6 Outcome of benchmark test on various languages

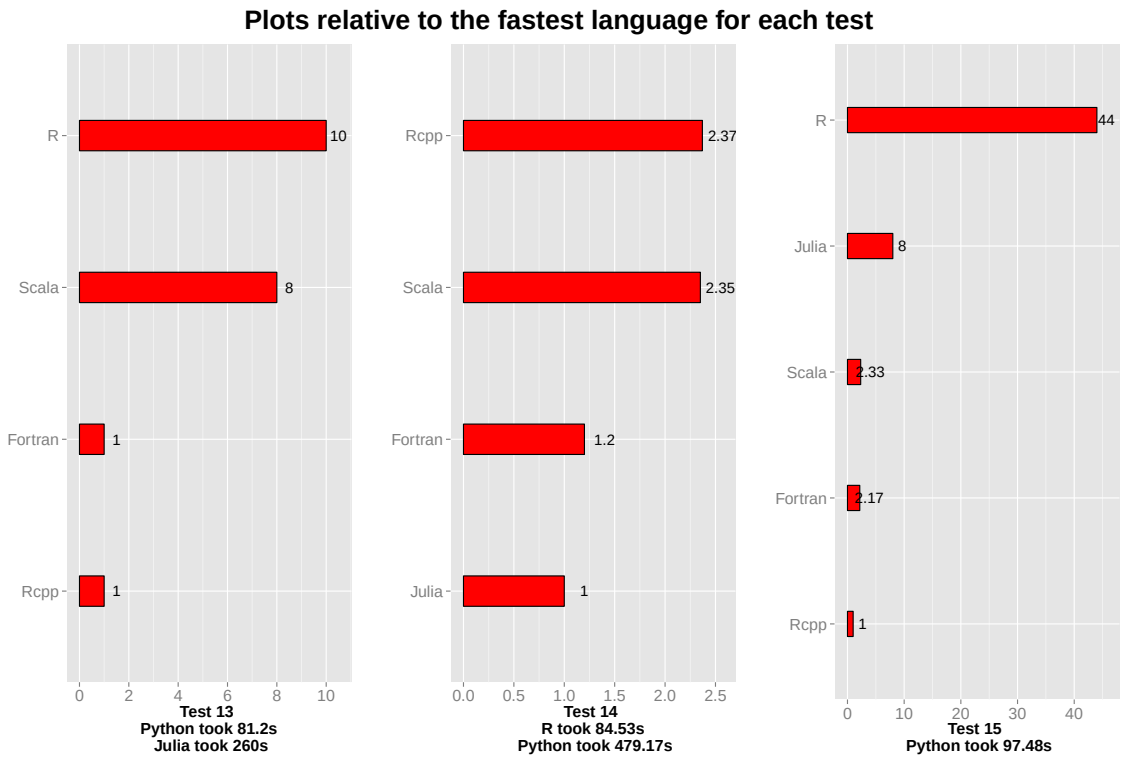
Following are the bar graph representations of the relative time taken by each language w.r.t the fastest language for each test. It means that for each test, the language having the fastest time is set to 1 and rest show how much slower they are compared to the fastest one for each test. A table of absolute results i.e actual time taken in seconds for all the languages for each test is added in appendix. A github link to the test scripts used for benchmarking is also given in the appendix.



Plots relative to the fastest language for each test



x axis is the relative time w.r.t fastest language for that test



7 Conclusion

Programming languages are used across various fields in the world and hence there exist so many of them. Each field requires a specific set of tools and therefore each language has its advantages and a reason for being created for that particular field. In this paper, a bunch of different languages were chosen and pitted against each other. A set of tests were designed to check the computational time against each other. Also, qualitative measures were kept in mind while choosing a language. Some inferences from the above graphs are:

- 1. Different languages perform better in different tests
- 2. On an average, Rcpp comes on top with Julia being close second
- 3. On an average, Scala was relatively the slowest, although it is fastest for some tests

On qualitative measures, R, Rcpp and python perform well. All three have decent set of libraries and IDE's and relatively easy to setup and learn.

Computational time is good for Rcpp and Julia. Julia would have been a good option if not for its lack of libraries and low maturity of the language. **Hence, keeping in mind both the qualitative measure and the computational time, the language recommended to be used along with R for agent based modelling is Rcpp.**

8 Reference material

- http://docs.julialang.org/en/latest/manual/
- http://www.statalgo.com/2012/04/27/statistics-with-julia-least-squares-regression-with-direct-methods/
- http://docs.python.org/
- http://docs.scipy.org/doc/numpy/reference/
- http://docs.scipy.org/doc/scipy/reference/
- http://www.cs.mtu.edu/ shene/COURSES/cs201/NOTES/fortran.html
- http://www.nsc.liu.se/ boein/f77to90/a5.html
- http://www.scala-lang.org/docu/files/ScalaTutorial.pdf
- https://github.com/scalala/Scalala
- https://github.com/scalanlp/breeze

9 Appendix

9.1 Bench Mark results (Actual time in seconds)

	Language	Julia	Python	R	Rcmp	Rcpp	RcppCmp	Fortran	Scala
1	Test 1	0.23	0.49	0.84	0.84	0.78	0.78	1.91	0.39
2	Test 2	1.03	1.09	1.35	1.35	1.30	1.30	1.99	1.83
3	Test 3	1.45	56.19	1.14	1.13	1.02	1.01	1.96	1.62
4	Test 4	4.09	6.02	14.79	14.56	16.85	16.75	25.53	23.71
5	Test 5	13.24	3.16	22.92	23.10	19.19	19.20	31.47	12.47
6	Test 6	0.81	0.30	0.40	0.40	0.91	0.92	1.83	500.00
7	Test 7	1.13	3.45	2.26	2.30	2.13	2.12	1.32	48.12
8	Test 8	7.35	1.90	4.49	4.46	4.37	4.30	1.72	7.40
9	Test 9	5.83	8.70	23.31	23.29	36.07	36.11	3.08	37.54
10	Test 10	1.11	1.66	5.02	5.01	3.53	3.53	77.05	5.23
11	Test 11	9.66	22.23	0.88	0.87	0.59	0.60	1.22	2.16
12	Test 12	0.44	0.21	0.39	0.39	0.13	0.13	0.22	1.82
13	Test 13	0.26	0.08	0.01	0.01	0.00	0.00	0.00	0.01
14	Test 14	5.52	2645.04	466.66	343.03	13.10	13.12	6.65	12.97
15	Test 15	0.48	8.49	2.64	2.72	0.06	0.06	0.13	0.14

Table 1: *Time taken for each test in secs*

Mu Sigma is a leading provider of decision sciences and analytics solutions, helping companies institutionalize data-driven decision making. We work with market-leading companies across multiple verticals, solving high impact business problems in the areas of Marketing, Supply Chain and Risk analytics. For these clients we have built an integrated decision support ecosystem of people, processes, methodologies & proprietary IP and technology assets that serve as a platform for cross-pollination and innovation. Mu Sigma has driven disruptive innovation in the analytics industry by integrating the disciplines of business, math, and technology in a sustainable model. With over 75 Fortune 500 clients and over 2000 decision science professionals we are one of the largest pure-play decision sciences and analytics companies.

Learn more at *http://www.mu-sigma.com/contact.html* us for further information:

Mu Sigma Inc., 3400 Dundee Rd, Suite 160, Northbrook, IL – 60062 www.mu-sigma.com