ED project

Jing Yeh, Saurav Sathnarayan

2024-09-03

Contents

things need to change and edit
1. add page numbers
2. might need to add new pages/headings depending on project length.
3. Add title page?

Introduction

The goal of this experiment is to identify the programming language that delivers the fastest execution time when calculating a value of π with respect to Leibiniz formula.

$$\sum_{n=0}^{\infty}(-1)^n/(2n+1)$$

With the increasing demand for high-performance applications, understanding which programming languages offer superior speed in terms of execution is crucial for developers, especially in domains requiring real-time processing, large-scale data analysis, and resource-intensive computations.
This problem will focus on evaluating a selection of popular programming languages, including but not limited to C++, C, R, Python, Java, and Ruby. The evaluation will consider how quickly a value of pi can be calculated by applying leibiniz formula up to 100000000 terms. *We are not concerned about how accurate our value of π is, but rather how quickly a programming language computes that value. I think we don't need to mention this haha*

Compiled Language:
In a compiled language, the source code is translated into machine code by a compiler before execution. This machine code, often called an executable, can be run directly by the computer's hardware.
Compiled programs typically run faster since they are already in machine language, which the computer's processor can execute directly.
Examples: C, C++, Rust, and Go are examples of compiled languages.

Interpreted Language:
In an interpreted language, the source code is executed line-by-line by an interpreter at runtime. The interpreter reads the code, translates it into machine code, and executes it on the fly.
Interpreted programs generally run slower than compiled ones because the translation happens during execution.
Examples: Python, JavaScript, Ruby, and PHP are examples of interpreted languages.

Key Differences: Compiled languages require a compilation step that produces an executable, while interpreted languages are executed directly by an interpreter.
Compiled languages tend to have better performance due to the pre-compiled nature of the code, whereas interpreted languages are more flexible but slower due to the runtime translation.
Some languages, like Java, use a combination of both techniques, where the code is first compiled into an intermediate form (bytecode) and then interpreted just-in-time (JIT) at runtime.
still need to edit this

Objectives

I don't think we need to restate our objectives here, since we've already explained it in the Introduction

Our main objective is to test the following hypothesis:

$$H_0 : \alpha_i = 0$$

$$H_1 : \text{at least of } \alpha_i \neq 0$$

For some $i = 1, 2...6$

Planned comparsions
1. Which language is the fastest?
2. Comparsions between C and C++ and other languages
3. Do complied languages run faster than interpreted languages ?
4. Is R (vectorised Language) faster than non-vectorised language when performing calculations?

what we hope to find from out experiment.

Sources of Variation

Our treatment Factors are 6 programming languages: C, C++, Java, Python, R and Ruby. To apply each factor, will run Leibiniz formula a billion times to get an observation. Our response Y_{ij} is the amount of seconds taken to compute leibinz formula a billion times.
Our experimental units are PCs, which have taken from The following Labs Ishango, Scilab and MiddleTr.

PC Specificiations
Ishango PC
Memory: 8,0 GB
Processor: Intel® Core™ i3-9100
Graphics: Intel® UHD Graphics 630 (CFL GT2)

MidddleTR
Memory: 8,0 GB
Processor: Intel(R) Core(TM) i5-9500 CPU
Graphics:

ScilabB
Memory: 16,0 GB
Processor: 12th Gen Intel(R) Core(TM) i5-12400
Graphics:

Due to the differences in specifications in the PCs, we have decided to block for the labs, to reduce experimental error variance between the experiment units. We've selected 3 PCs from each lab (randomly). So we have 9 experimental units in total. *get gpu specs for last two pcs*

Treatment Factor *talk about treament factor*

Randomisation

randomisation and soucres of variation can be one page
discuss process of randomisation, why we using it...

Analysis of pilot experiment

As mentioned in the introduction, we aim to investigate the speed of various programming languages when used to perform large amount of computation. To start with, we conducted a small scale pilot experiment. This is based on the "first explore then confirm" strategy described in [1],

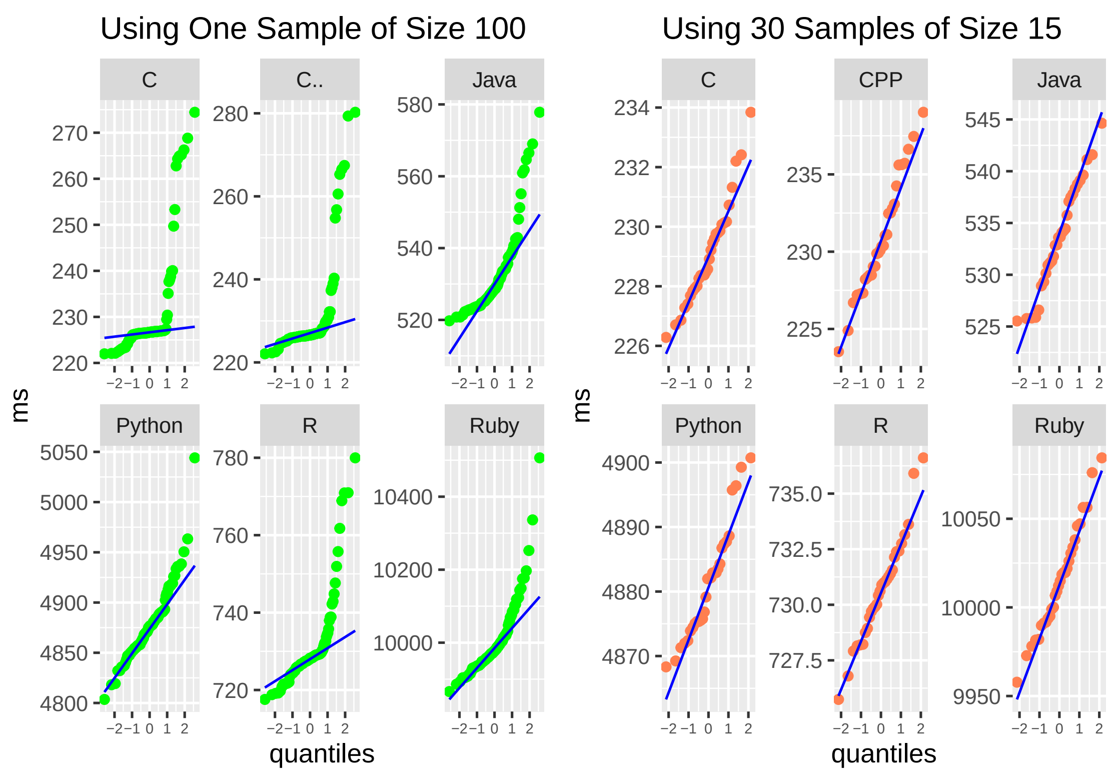
Since the number of treatments we apply in this experiment does not significantly impact the time and cost required, we apply all six treatments. However, as computers of different specifications are harder to come by, we will only use 3 different hardware setup for this pilot study.

Upon inspecting the qqnorm plots of the runtime of all programming languages on the same machine, we discovered that the runtime of languages tend to not follow a normal distribution, even when the sample size is big (n = 100)

To address this, we ran the program 15 times per sample for each programming language, and repeated the process 30 times. By the Central Limit Theorem (CLT), the distribution of sample means is approximately normal [2]. If we assume sample means to be normally distributed, the mean of the distribution of sample means is then an unbiased estimator for the true run time of each programming language[2], which we take as a single observation.

The process described above is automated using the following command:

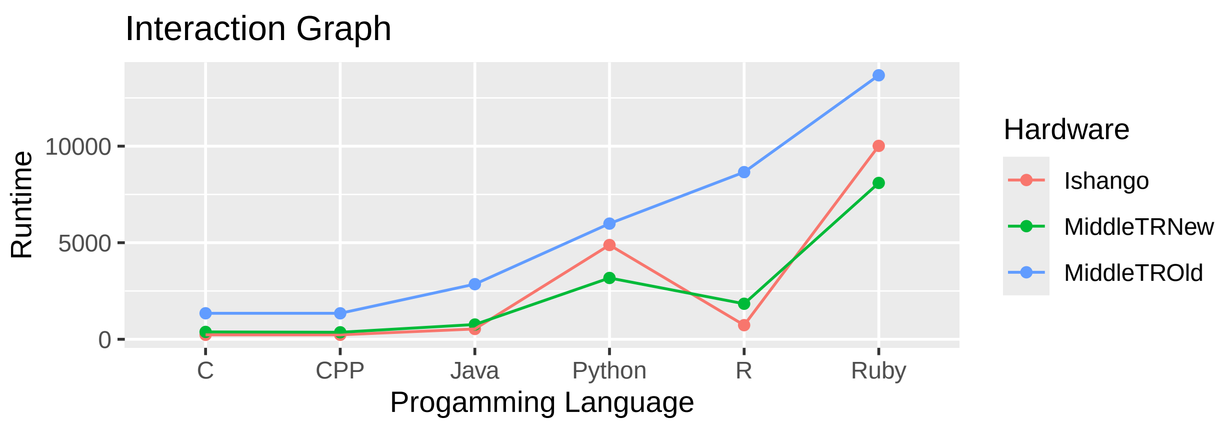
```
python3 run.py main 100000000 15 30
```



After we have applied the procedure described above, we obtained the following result:

todo: insert table for results

```
## Warning in read.table(file = file, header = header, sep = sep, quote = quote, :
## incomplete final line found by readTableHeader on 'pilotData.csv'
```



	Hardware	C	CPP	Java	Python	Ruby	R
1	Ishango	229.1316	230.8817	533.6732	4881.330	10015.010	730.6641
2	MiddleTROld	1345.2420	1344.6080	2853.2610	5989.950	13670.590	8655.5140
3	MiddleTRNew	381.0000	361.2894	763.4666	3174.553	8095.145	1838.7730

this is an exploratory study, so no need to list hypotheses (it's meant to generate hypotheses for the real study)

we need to decide on number of observations.
full analysis here
list our hypothesis here (is there a difference or not?)
list our contrasts. Figure out whether which method we are using to control type 1 error
what we hope to find from out experiment.
anova table and conclusions
contrasts and CI plus conclusions

Our Model and Analysis

Our model:

$$Y_{ij} = \mu + \alpha_i + \beta_j + e_{ij}$$
$$i = 1...a$$
$$j = 1...b$$

where

$$\sum_{i=1}^{a} \alpha_i = \sum_{j=1}^{b} \beta_i = 0$$

μ	overall mean
α_i	effect of i^{th} treatment
β_j	effect of j^{th} block
e_{ij}	random error of the observation

need to do formatting

explain model terms
repeat steps above

Summary and Conclusions

draw conclusions here
explain any challenges possibly
explain how we would do it differently, to fix errors or mistakes

References

any references, textbook, code, people wh helped us.

Apenddix

add important code here.