STA2005S - Experimental Design Assignment

Jing Yeh
??University of Cape Town
yhxjin001@myuct.ac.za

Saurav Sathnarayan
??University of Cape Town
yhxjin001@myuct.ac.za

2024-09-12

Abstract

This is the abstract.

It consists of two paragraphs.

Keywords: key; dictionary; word

1 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.

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

2 Reference example

Here are two sample references:. Bibliography will appear at the end of the document.

3 Materials and methods

An equation with a label for cross-referencing:

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

3.1 A subsection

A numbered list:

1) First point
2) Second point
    - Subpoint

A bullet list:

- First point
- Second point

4 Results

computers of different specifications are harder to come by, we will only use 3 different hardware setup for this pilot study.

4.1 Pilot Study

4.1.1 Problem: Runtimes of Programming Langugaes are not Normally Distributed

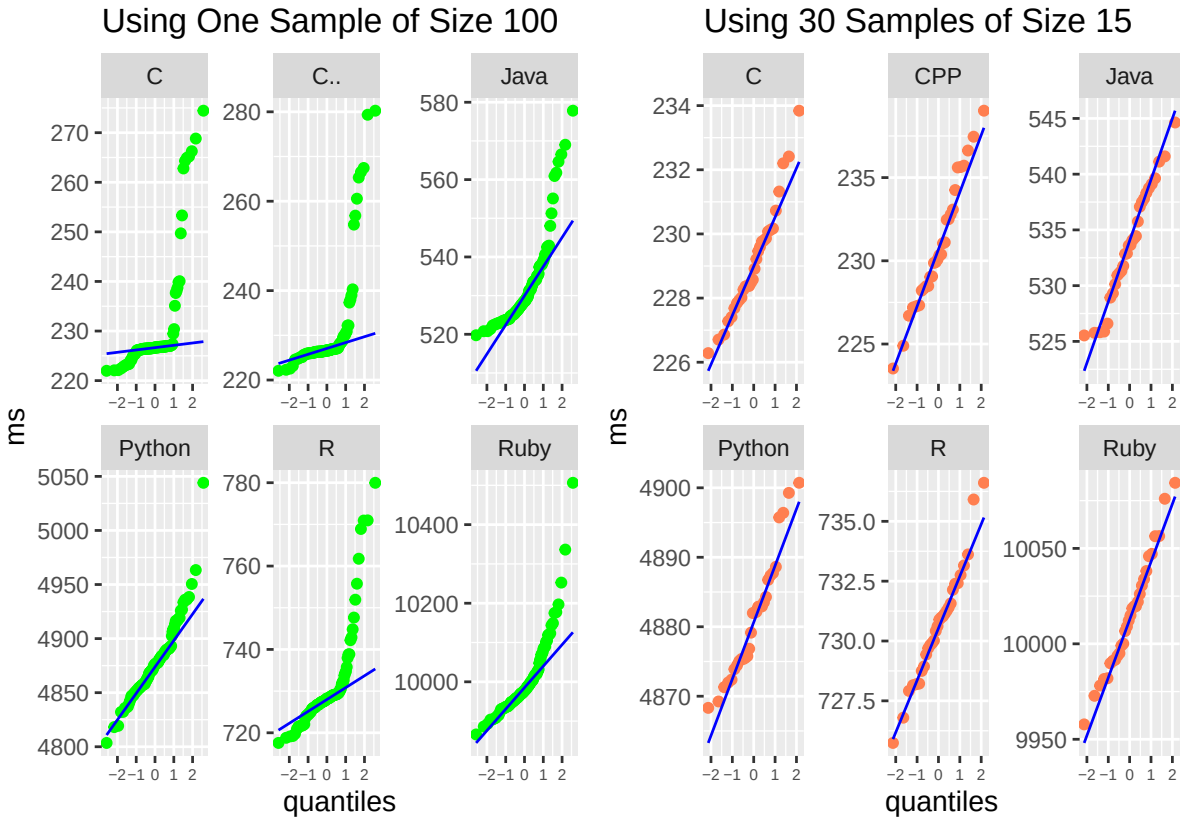
Upon inspecting the qqnorm plots of the runtime of all programming languages on the same machine, we discovered that the runtimes of languages clearly do not follow a normal distribution, even when the sample size is fairly large (n = 100)

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:

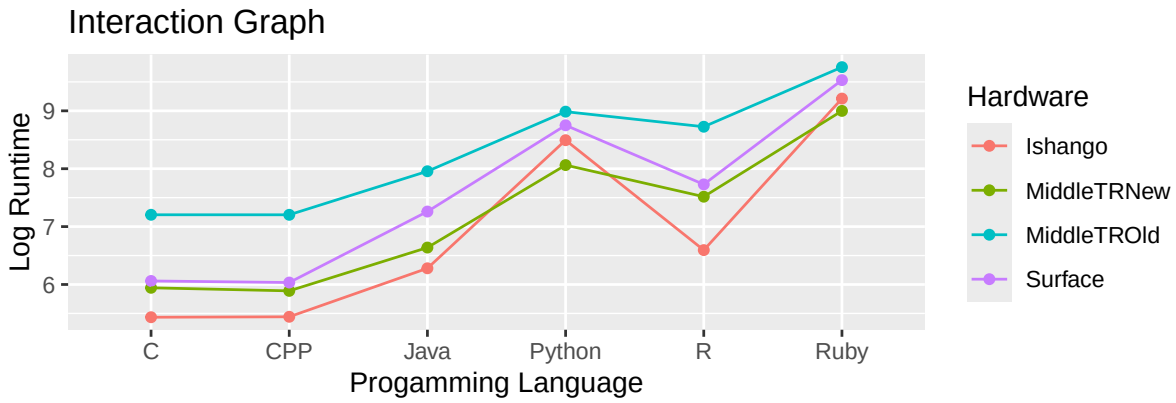
```
python3 run.py main 100000000 15 30
```



4.1.2 Problem: Ensuring no Interactions between Blocking Factors and Treatment Factors

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	7989.950	17240.590	6157.5140
3	MiddleTRNew	381.0000	361.2894	763.4666	3174.553	8095.145	1838.7730
4	Surface	428.6748	417.2392	1422.8945	6314.829	13780.423	2273.1987

From the data collected, we observed that the results collected from Ishango do not follow the general trends established by the other three setups. Firstly, the hardware setup in Ishango lab is significantly less advance than MiddleTRNew. Yet, most programming languages tend to perform better on the Ishango machine. Secondly, to add to the first observation, not all programming languages perform better on the Ishango machine.

After further investigation, we learned that programming languages perform differently on various operating systems [4]. We hypothesised that this is likely the reason for the deviation, though further studies are needed to confirm this (we lack access to machines with the same hardware setup but run on different operating system).

Therefore, we added another constraint for selecting suitable machines: the machines must all run on Windows 10, as these machines are the most widely available. ## Generate a figure.

```
plot(1:10, main = "Some data", xlab = "Distance (cm)", ylab = "Time (hours)")
```

You can reference this figure as follows: Fig. 1.

```
plot(1:5, pch = 19, main = "Some data", xlab = "Distance (cm)", ylab = "Time (hours)"
```

Reference to second figure: Fig. 2

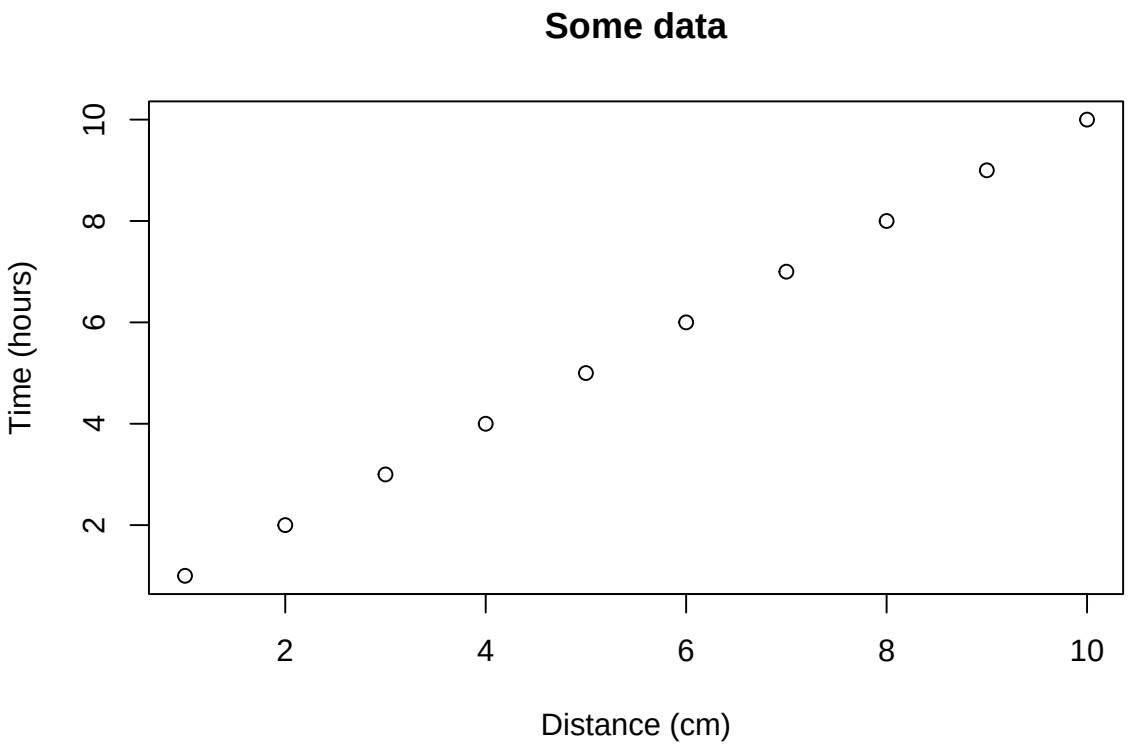


Figure 1: This is the first figure.

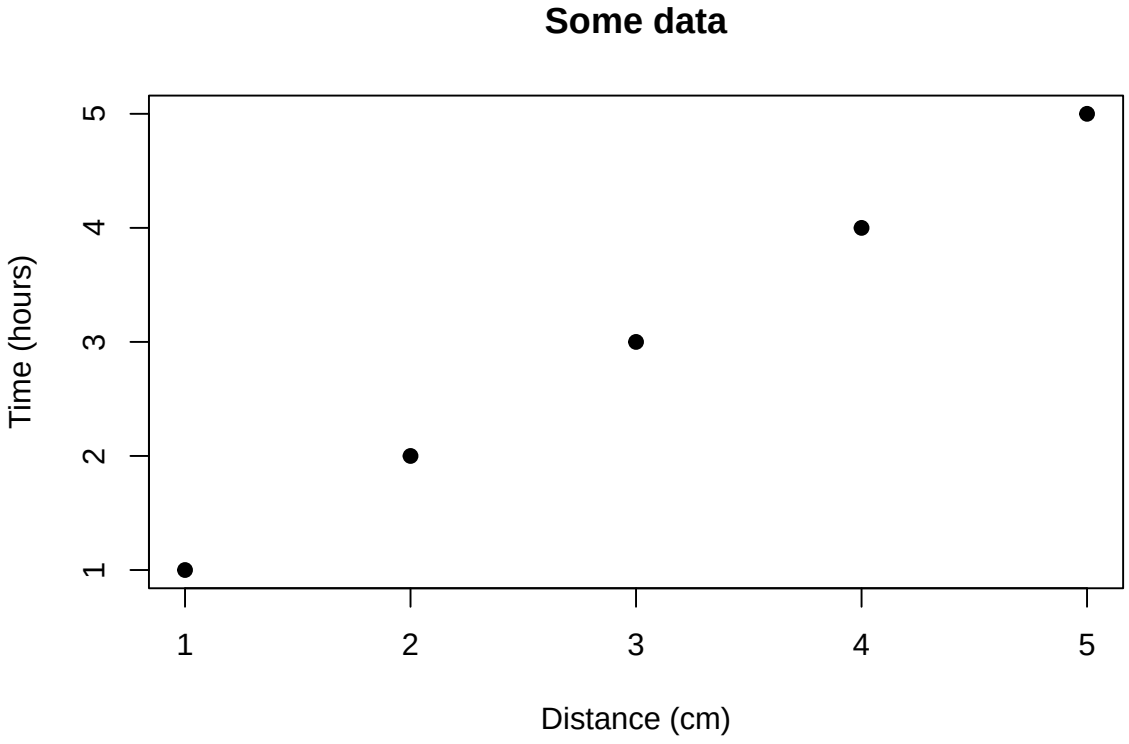


Figure 2: This is the second figure.

4.2 Generate a table using `xtable`

```
df <- data.frame(ID = 1:3, code = letters[1:3])

# Creates tables that follow OUP guidelines using xtable
library(xtable)
print(xtable(df, caption = "This is the table caption", label = "tab:tab1"),
  comment = FALSE
)
```

	ID	code
1	1	a
2	2	b
3	3	c

Table 1: This is the table caption

You can reference this table as follows: Table 1.

4.3 Generate a table using `kable`

```
df <- data.frame(ID = 1:3, code = letters[1:3])

# kable can alse be used for creating tables
knitr::kable(df,
  caption = "This is the table caption", format = "latex",
  booktabs = TRUE, label = "tab2"
)
```

You can reference this table as follows: Table 2.

5 Discussion

You can cross-reference sections and subsections as follows: Section 3 and Section 3.1.

Table 2: This is the table caption

ID	code
1	a
2	b
3	c

Note: the last section in the document will be used as the section title for the bibliography.

6 References

7 Appendix

PC Specificiations

	CPU	MEMORY	OPERATING SYSTI
Ishango PC	9th Gen Intel® Core™ i3-9100	8,0 GB	Ubuntu 22
MidddleTROld	9th Gen Intel(R) Core(TM) i5-9500 CPU	8,0 GB	Windows
MiddleTRNew	12th Gen Intel(R) Core(TM) i5-13400	16,0 GB	Windows
ScilabB	12th Gen Intel(R) Core(TM) i5-12400	16,0 GB	Windows
Surface	9th Gen Intel(R) Core(TM) i5-8250 CPU	16,0 GB	Windows
ASUS laptop	5th Gen Intel(R) Core(TM) i7-5500U CPU	6,0 GB	Windows

Acknowledgements

This is an acknowledgement.

It consists of two paragraphs.