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 JavaScript. The evaluation will consider how quickly a value of pi can be calculated using leibiniz formula, using varying number of terms, ie 100000 terms or 1000000 terms.We are not concerned about how accurate our value of π is, but rather how quickly a programming language computes that value.

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 or just-in-time (JIT) compiled at runtime.
still need to edit this

Objectives

Our main objective is to test the following hypothesis:

$$H_0 : \alpha_1 = \alpha_2$$

$$H_1 : \alpha_1 \neq \alpha_2$$

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.
come back to check if hypthesis test is right.

Sources of Variation

discuss treatment factor and different levels Our treatment factor is implementing Leibiniz Formula to the to the different programming languages.

our experiment units, maybe list pc specs
go into depth about specs of pcs, blocking for each pc, to reduce experiment error variance.

Randomisation

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

Analysis of pilot experiment

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

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.