University of Puerto Rico
Department of Electrical and Computer Engineering
ICOM5015 Artificial Intelligence

Is Python Fast or Slow?

Group C

Manuel Alejandro Umaña Rodriguez -Undergraduate Computer Engineering
Dahyna Martínez Pérez -Undergraduate Computer Engineering
Jan Luis Pérez de Jesús -Undergraduate Computer Engineering
Christopher Hans Mayens Matías -Undergraduate Computer Engineering

For: Profesor José Fernando Vega Riveros
Date: February 21,2025

Agenda

01 Purpose of experiment

02 Hypothesis

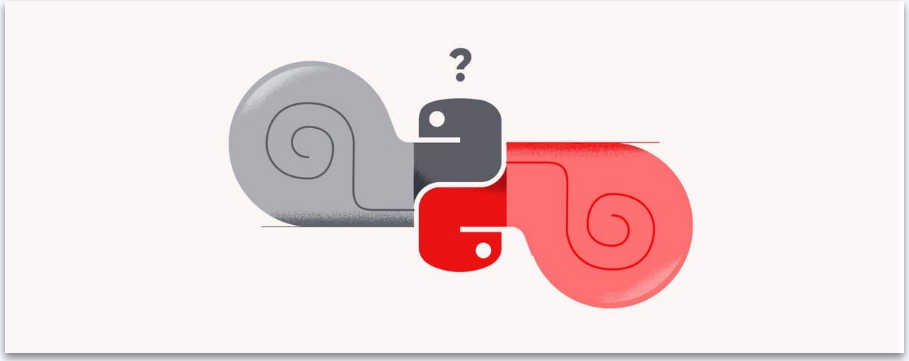
03 Concepts

04 Experiments set up

05 Information

06 Conclusion

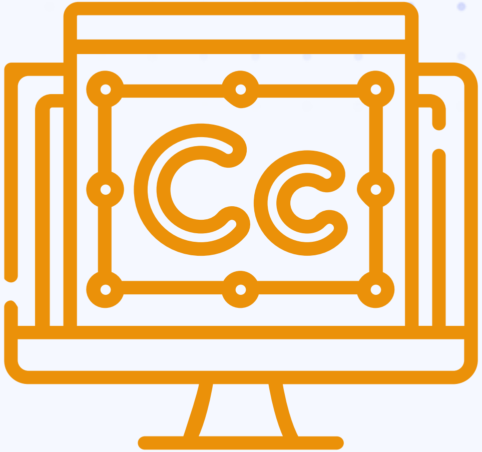
07 Credits & References



01

Purpose of experiment

- Investigate the usage of external libraries, such as Numpy, in python to compare its performance to the native language.
- Code in python for the multiplication of arrays and multidimensional arrays to compare execution times.
- What makes python libraries widely used, and are they written in python?
- Should libraries be used at all?



02

Key questions and hypothesis

Does the use of libraries affect the efficiency of python?

- How does external libraries affect performance?
- Are external libraries written in low level languages compatible with python?
- Does python on its own handle large calculations well?

- Hypothesis: Given that python is a dynamic multipurpose language, it is to be presumed that when handling larger sets of calculations its performance is relatively slow compared to other low level languages like C or C++. Since Numpy is written in these languages it performs faster calculations.

03

Concepts

Experimental concepts

Platform

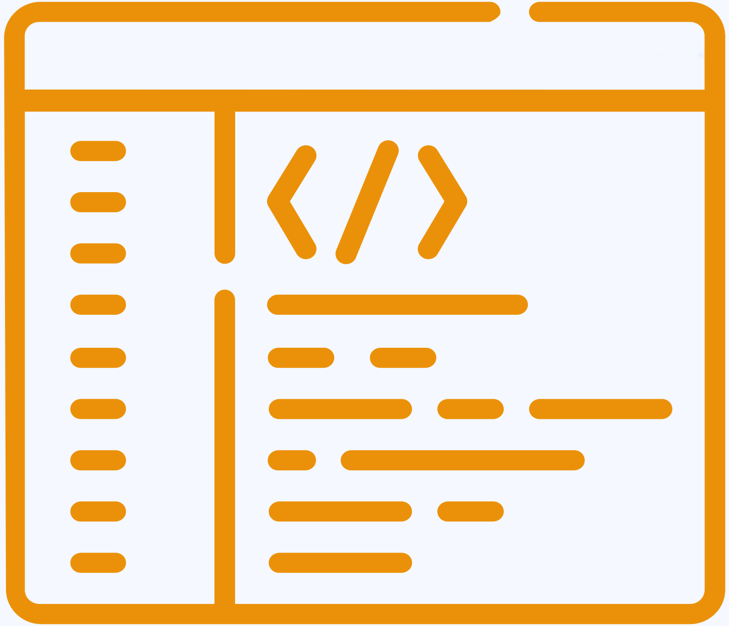
Python: Programming language of high level. Emphasizes code readability with the use of indentation.

Subject

Arrays: Ordered collection of elements.

Measure

Performance (in code): Refers to the efficiency, accuracy and speed of the software execution.



04 Experiments set up.

Tools used

Different tools used for comparison.

The imports for the experiments where:

import random: Set up for creating random integer numbers when filling arrays.

import time: Measure time taken.

import numpy as np: Main comparison tool being used.

import matplotlib.pyplot as plot: Graphing the results.

Method for comparing

Comparing times for 1 dimensional and 2 dimensional arrays.

The loop comparing the 1 dimensional arrays starts with 10 elements.

- Create both arrays with increasing size.
- Multiply arrays with python code.
- Capture time taken for multiplication.
- Multiply both arrays with NumPy.
- Capture time taken for multiplication with NumPy.
- Update element size and repeat previous steps.

The same steps are taken for the 2 dimensional array for finding time.

Method for graph creation

How the Data for the graph was collected and used.

At the end of all the dimensional multiplications, there were four lists that were created to capture the time each execution lasted.

- python_1D_times
- numpy_1D_times
- python_2D_times
- numpy_2D_times

The plots were made by importing the matplotlib.pyplot and using the provided dimensions. To create an Execution Time vs Array Size Plots for every 1D and 2D experiment.

05

Information.

Information 1D Arrays

Dimensions	Pure Python Execution Time(seconds)	NumPy Execution Time (seconds)
10	0.0075656000	0.0003246000
50	0.0000152000	0.0000092000
100	0.0000220000	0.0000052000
200	0.0000382000	0.0000140000
500	0.0001164000	0.0000086000

Table 1. Time Execution data of 1D Array Multiplications for Graph 1 for 1 attempt.

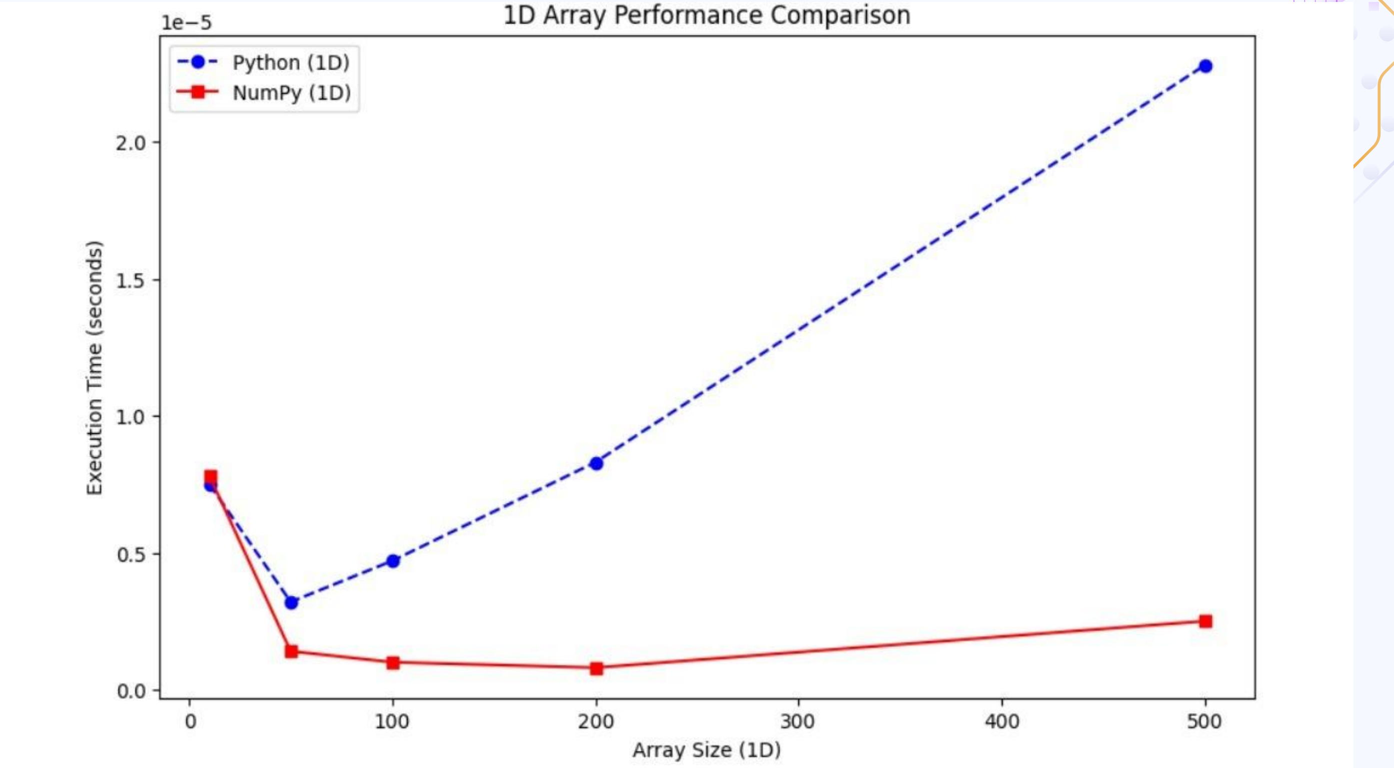
Information 2D Arrays

Dimensions	Pure Python Execution Time(seconds)	NumPy Execution Time (seconds)
10	0.0000507000	0.0000133000
50	0.0005219000	0.0000655000
100	0.0017039000	0.0001151000
200	0.0043834000	0.0000872000
500	0.0277210000	0.0004880000

Table 2. Time Execution data of 2D Array Multiplications for Graph 3 for 1 attempt.

Graphs

Figure 1.
Comparison of 1D
Array
Multiplication
Performances
utilizing Pure
Python and
NumPy



Graphs

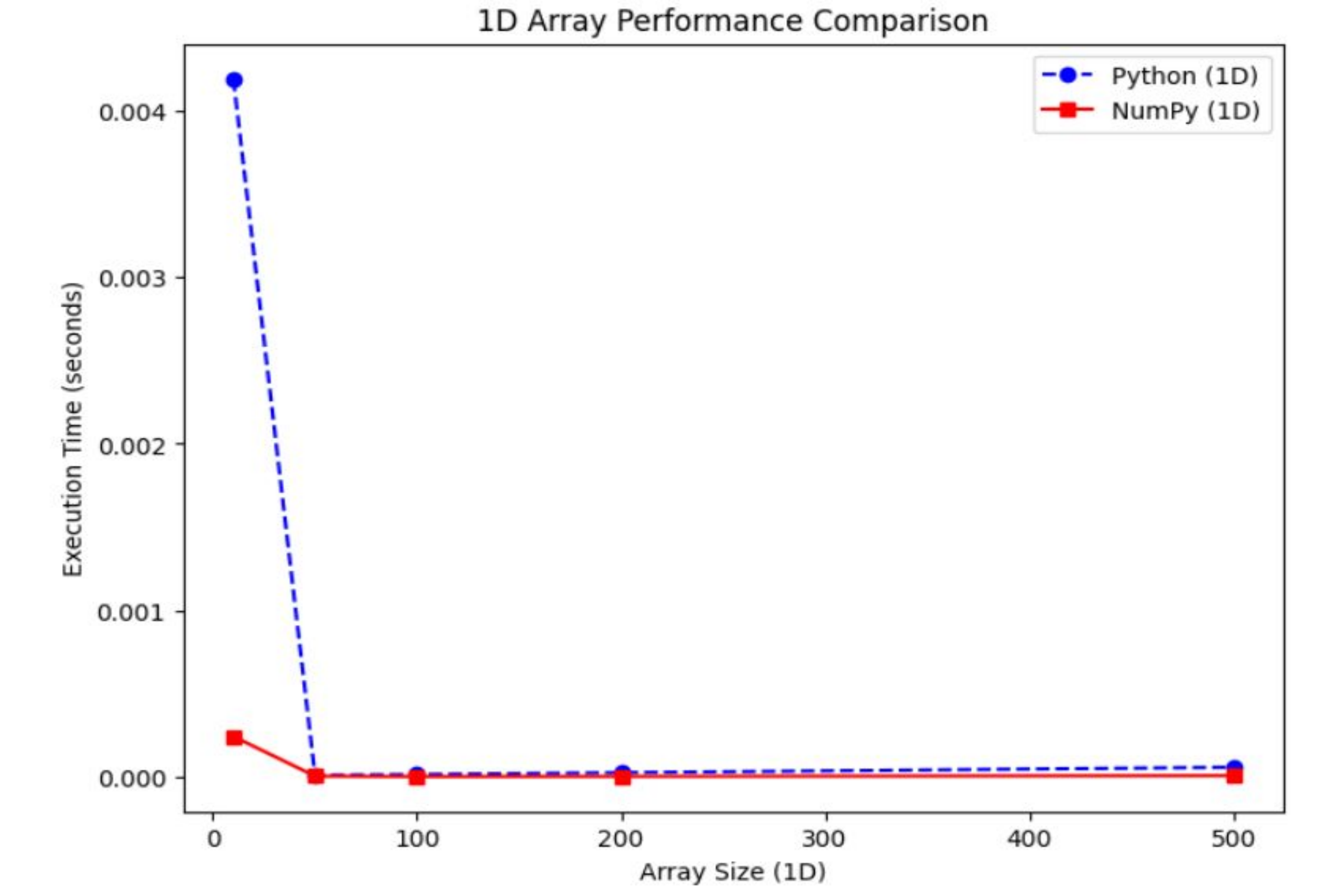
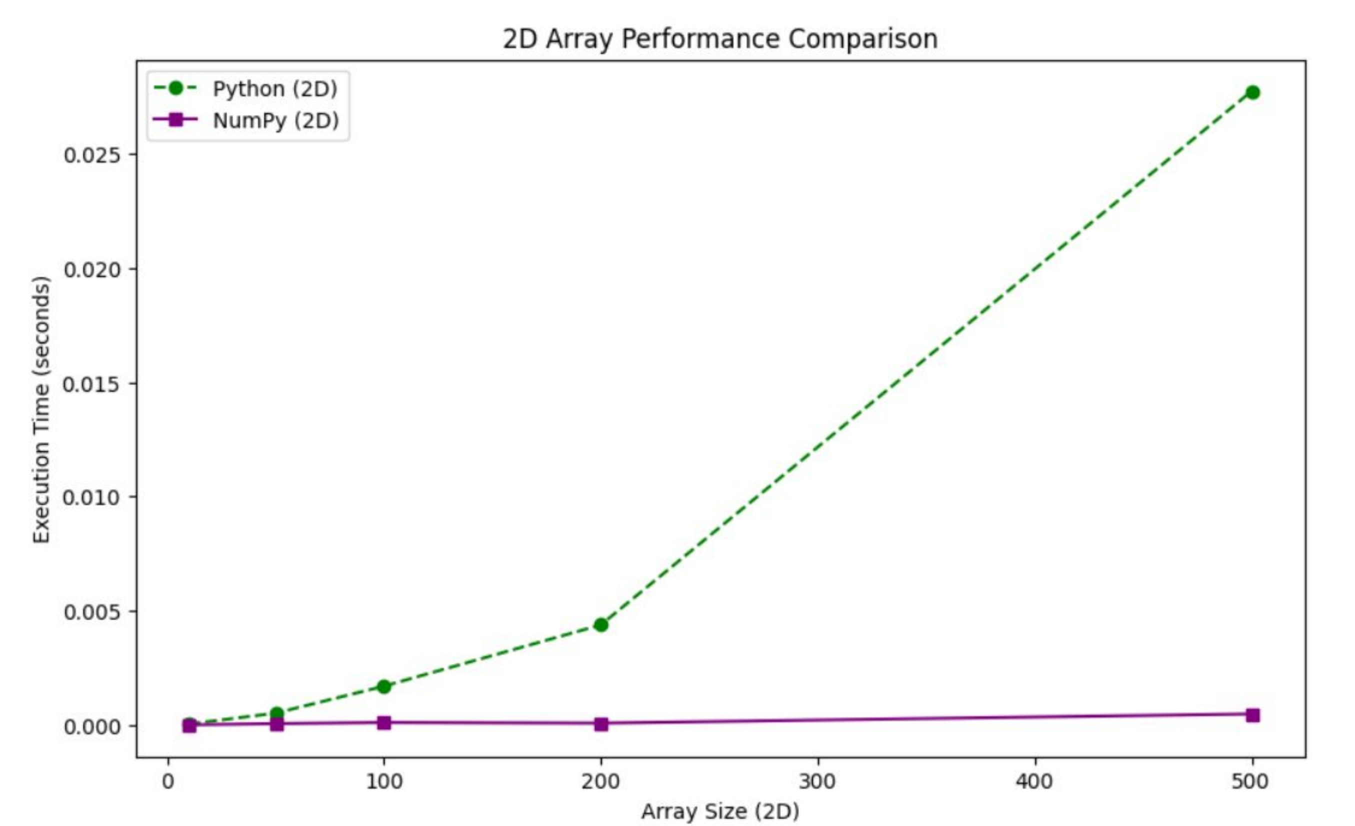


Figure 2. Comparison of 1D Array Multiplication Performances utilizing Pure Python and NumPy normalised

Graphs

Figure 3. Comparison of 2D Array Multiplication Performances utilizing Pure Python and NumPy



06

Conclusion and Lessons Learned.

Conclusions and Lessons Learned

01 — **Effectiveness of external Libraries** — The usage of external libraries can help improve upon the efficiencies of programs by providing more precise tools.

02 — **Languages of external library** — Not all libraries are required to be made in python to be compatible. Low level languages like C or C++ provide the tools to efficiently manage large sets of calculations.

03 — **Continuous memory** — NumPy packages break down to fragments the data for calculations and is able to process all those fragments parallelly with the help of multithreading.

04 — **Usages of Python** — Python is still a very readable and dynamic language that offers various compatibility feature and can process low sets of information precisely and efficiently.

07

Credits and References.

References

[1] NumPy documentation, “What id NumPy,” *numpy.org*, Dec. 8, 2024. [Online]. Available: https://numpy.org/doc/2.2/user/whatisnumpy.html . [Accessed Feb.20, 2025].

[2] W3schools, “NumPy Introduction,” *w3schools.com*, 2022. [Online]. Available: https://www.w3schools.com/python/numpy/numpy_intro.asp. [Accessed Feb.20, 2025].

[3] R. Jsaha, “Why is Numpy faster in Python,” *geeksforgeeks.org*, Dec. 8, 2024. [Online]. Available: https://www.geeksforgeeks.org/why-numpy-is-faster-in-python/. [Accessed Feb.20, 2025].

[4] I. Sushant, “What makes python a slow language,” *geeksforgeeks.org*, July. 2021. [Online]. Available: https://www.geeksforgeeks.org/what-makes-python-a-slow-language/ . [Accessed Feb.20, 2025].

[5] NumPy reference, “Thread Safety,” *numpy.org*, Dec. 8, 2024. [Online]. Available: https://numpy.org/doc/2.1/reference/thread_safety.html . [Accessed Feb.20, 2025].