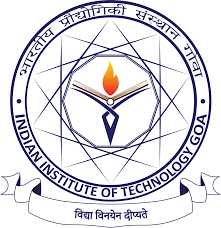
**Indian Institute Of Technology, Goa**



**Lab Assignment 01**

* **Submitted by:**

1. **Kanishk Chugh (2004215)**
2. **Prem Swarup (2003318)**
3. **Harshvardhan Gupta (2003310)**

* **Team Code: HaKaPr**
* **Course Instructor: Dr. Satyanath Bhat**
* **Course: Machine Learning (CS-331)**

**Analysis Lab\_01 Assignment 1:**

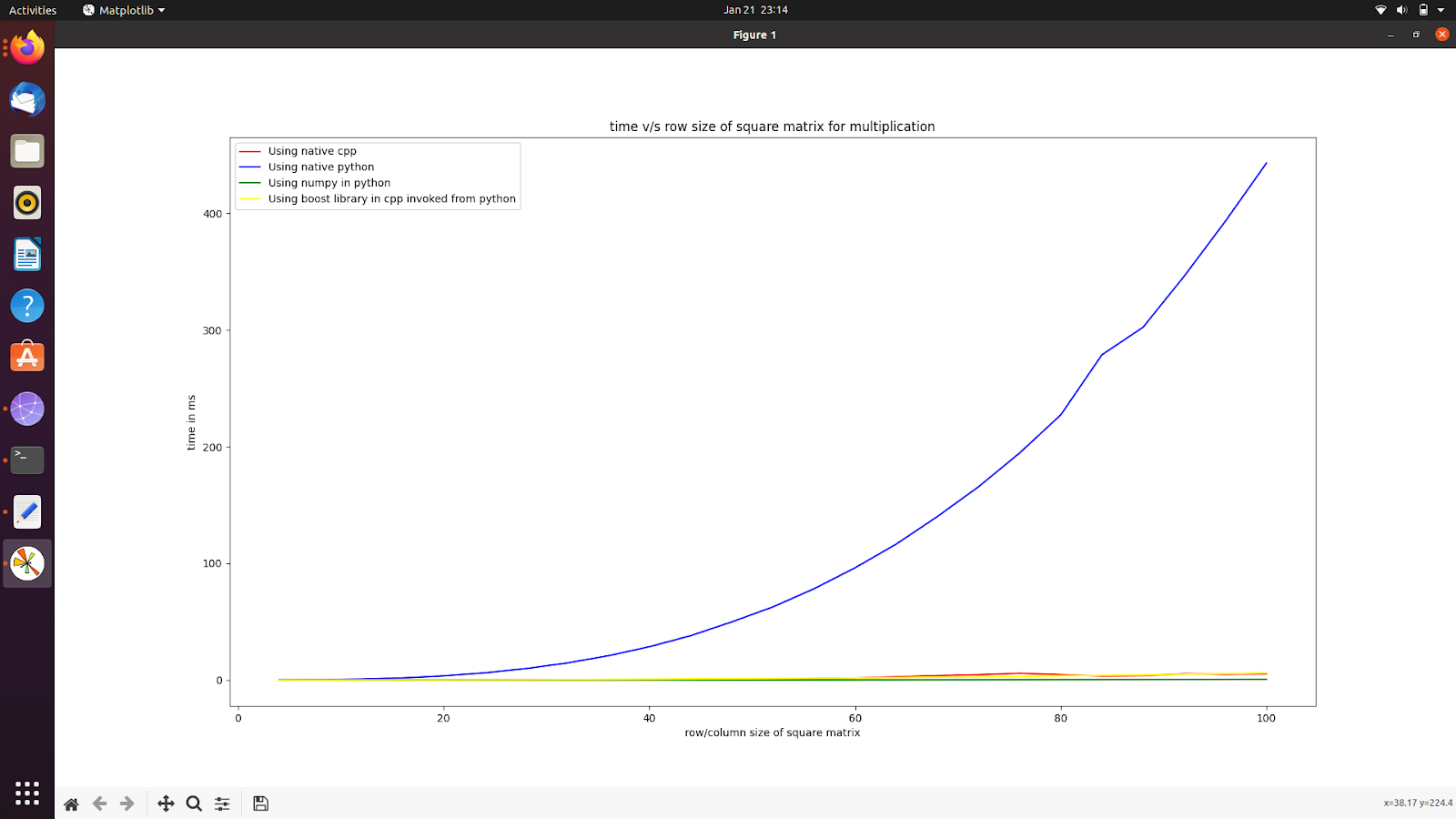
**Expectations from the run time:**-

**Time taken by native python > Time taken by boost library > Time taken by C++ program > Time taken by numpy in python**

*The reasons for the above expectations:-*

* Numpy is faster than C++ over here as numpy uses C libraries and compiler optimization to run any programs so until and unless we do compiler optimization for C++ it will be slower than Numpy in Python.
* C++ is faster than Python as it is a statically typed and compiled language, whereas Python is an interpreted language that is dynamically typed and it takes more time to interpret the code.
* Boost Python is a library that allows C++ code to be called from Python. It is slower than native C++ code because it involves additional overhead in the form of data conversion and function call overhead. Additionally, the Python interpreter itself may not be as fast as native C++ code.

**Observations for Square Matrices Multiplication:-**

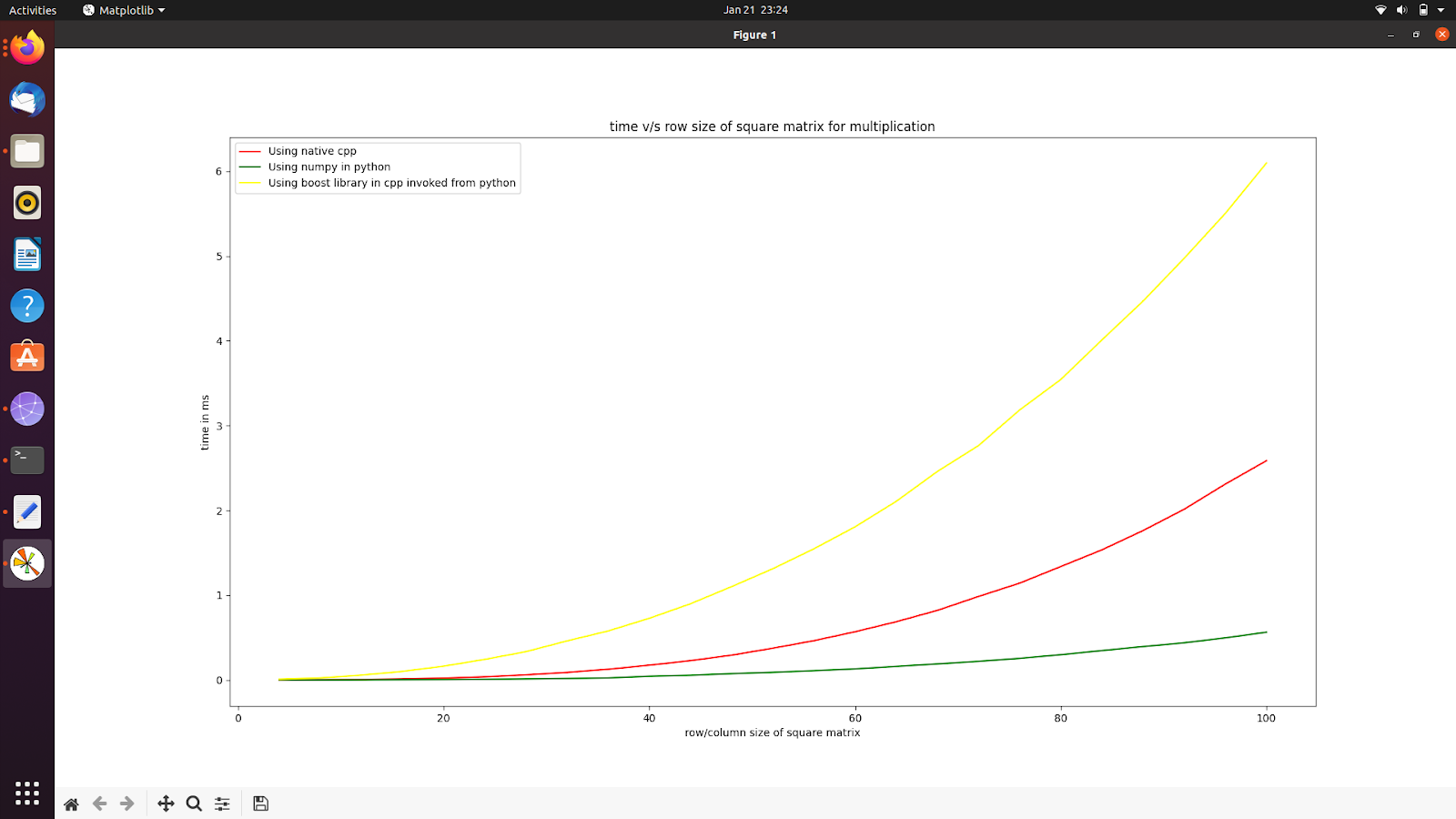


**Figure 1: Time taken to run (in ms) V/S size of square matrix**

**Observation from figure 1:-**

Time taken by native python method > Rest all programs

For comparing the run time among the other three methods we removed the plot for native python and got figure 2:

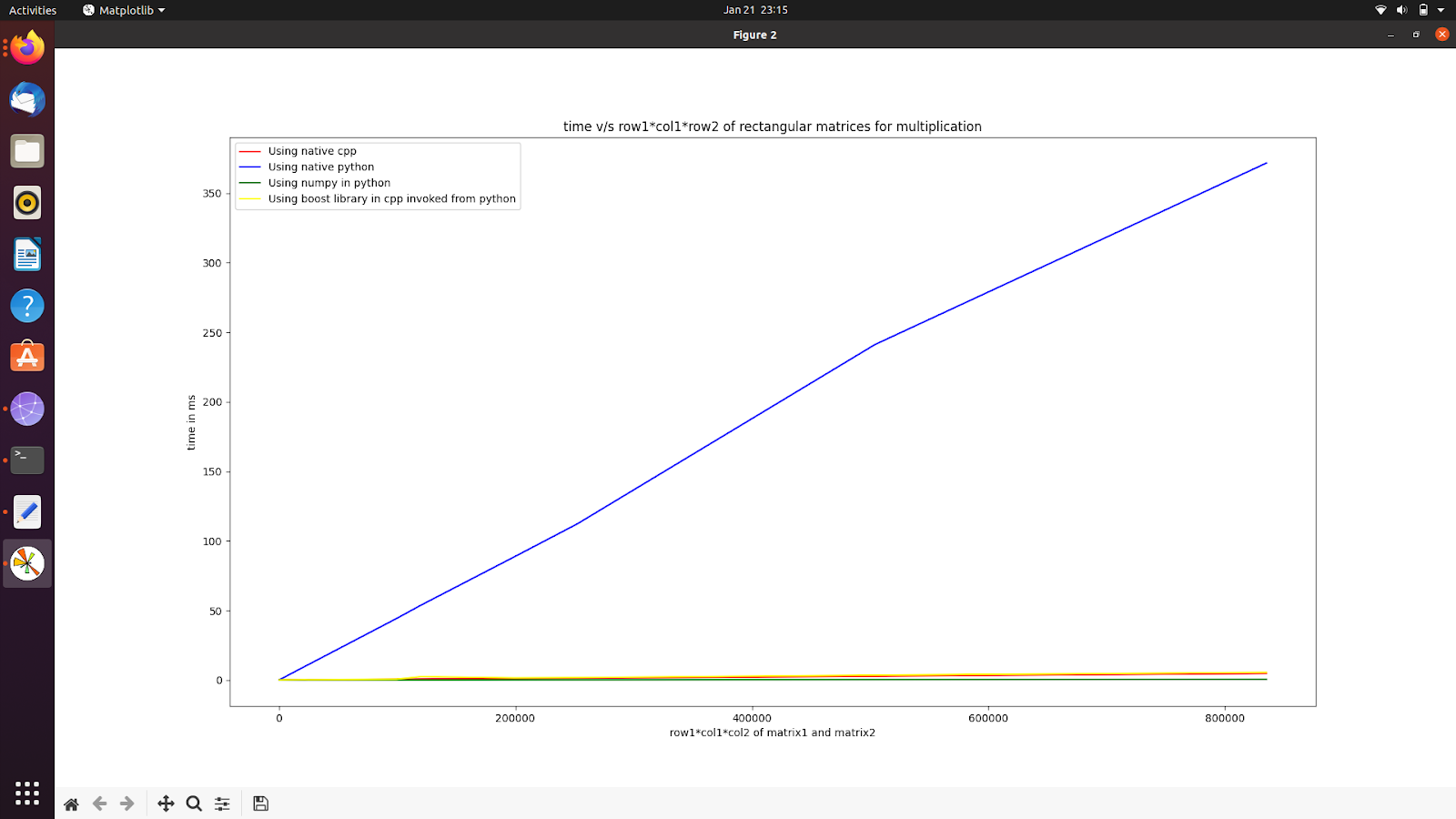


**Figure 2: Time taken to run (in ms) V/S size of square matrix (except the native python method)**

**Observations from figure 2:-**

Time taken by boost library method > Time taken by Native C++ > Time taken by numpy in python

**Observations for Rectangular Matrices Multiplication:-**

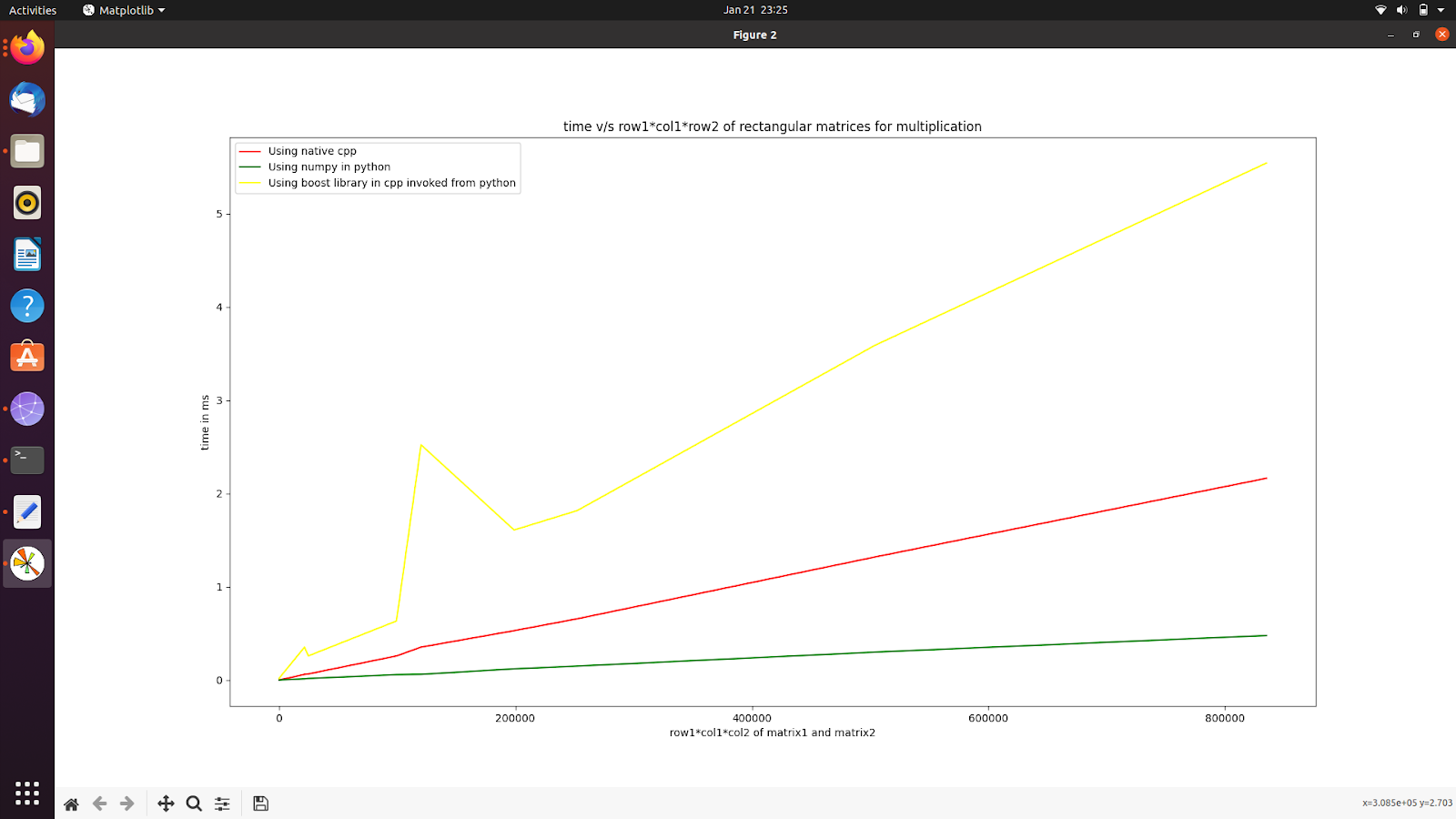


**Figure 3: Time taken to run (in ms) vs row1\*col1\*col2 of Rectangular matrix**

**Observations from figure 3:-**

Time taken by native python method > Rest all methods

For comparing among the other three we removed the plot for native python and got figure 4:

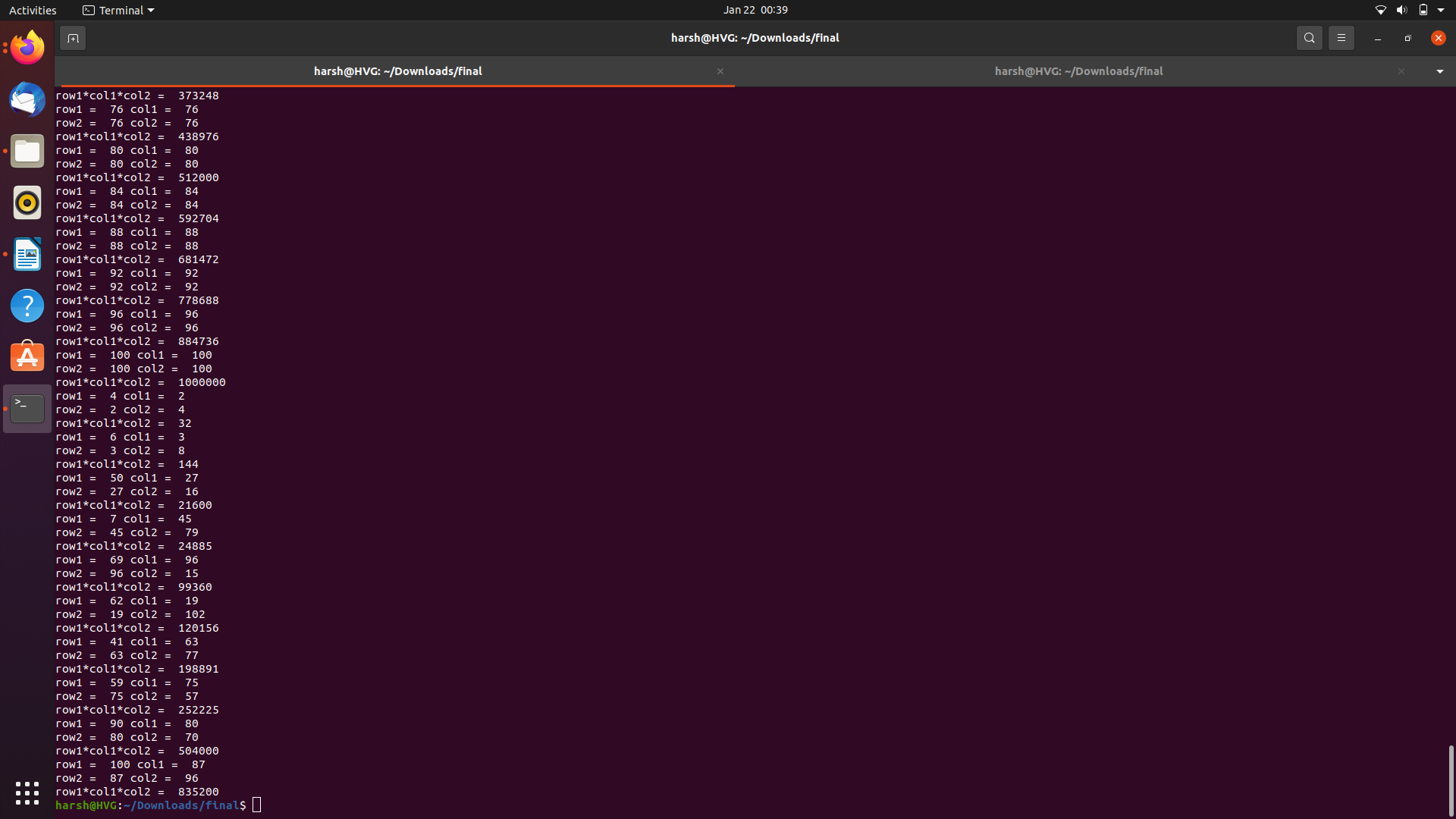


**Figure 4: Time is taken to run (in ms) vs row1\*col1\*col2 of Rectangular matrix (except the native python program)**

**Observation**:

The time taken by numpy in a python program is least followed by C++ program and then boost library program. The reason is apparent, and the expectations are met by observation. Here we observe a spike in the yellow graph (Boost python library in C++ invoked from python) the reason for this spike is:

The set of dimensions of input matrices that are being multiplied is given below:-



* The spike came at the 6th input: row1\*col1\*col2 = 120156

the auxiliary space taken: row1\*col2 = 62\*102 = 6324

* The next rectangular matrix multiplication has the following:

row1\*col1\*col2 = 198891

auxiliary space: row1\*col2 = 41\*77 = 3388

*7th multiplication takes less time as compared to 6th , even after having larger row1\*col1\*col2.*

One reason could be that in boost, we are typecasting a 2D result matrix from a 2D C++ integer array to a 2D NumPy array (which is relatively slower than native integer multiplication in C++), and it is copied twice while returning in the function call. As a result, for relatively smaller col1 in comparison to row1 and col2, the time complexity of typecasting and returning (i.e., O(row1\*col2)) becomes significantly greater than computing multiplication (i.e., O(row1\*col1\*col2)). So for cases like the 6th input where matrices being multiplied are skewed, i.e., row 1 and col 2 are quite larger than col 1, multiplication becomes reasonably slower.