Meta Volante 1

David Calle Gonzalez
dept. of science
EAFIT
Medellín, Colombia
dcalleg@eafit.edu.co

Santiago Gil Zapata
dept. of science
EAFIT
Medellín, Colombia
sgilz@eafit.edu.co

Sebastian Obando
dept. of science
EAM
Medellín, Colombia
sebastian.obando.8888@eam.edu.co

Juan Manuel Young Hoyos
dept. of science
EAFIT
Medellín, Colombia
jmyoungh@eafit.edu.co

***Abstract*—El presente documento tiene por objetivo demostrar-los procesos necesarios para poder ejecutar HPL en nuestro cluster de 2 nodos en Cronos.**

***Index Terms*—HPC, HPL, MPI.**

I. Introduction

Why this project? the aim of this project is to be able to analyze and draw a conclusion to the question, is it worth spending more time programming in C ++? Or is it more profitable to do this type of algorithms using Python and its libraries in terms of development time and program performance?

II. Objectives

In this project we seek to compare the efficiency of developing root search methods in python against c++, taking into account the comparison between these two languages for this we looked for the implementation of these algorithms in both languages for further analysis taking into account the speed and memory consumption of these algorithms in each of the languages.

III. Theoretical framework

The purpose of calculating the roots of an equation is to determine the values of x for which the following is true

$$f(x) = 0 \tag{1}$$

The determination of the roots of an equation is one of the oldest problems in mathematics and a great number of efforts have been made in this direction. Its importance lies in the fact that if we can determine the roots of an equation we can also determine maxima and minima, eigenvalues of matrices, solve systems of linear and differential equations[2].

The determination of the solutions of equation (1) can become a very difficult problem. If f(x) is a polynomial function of degree 1 or 2, we know simple expressions that will allow us to determine its roots. For polynomials of degree 3 or 4 it is necessary to use complex and laborious methods. However, if f(x) is of degree greater than four or is not polynomial, there is no known formula that allows us to determine the zeros of the equation (except in very particular cases).[2]

For this we have resorted to the use of technological tools that allow the calculation of these roots in a much more efficient way than doing it by traditional methods, for that we are looking for the implementation of different algorithms that facilitate the search of these roots[2].

Python is a very popular interpreted language nowadays among people looking to work with data and mathematical functions due to its ease of use and the large number of libraries that exist, as well as its great ease of development compared to other languages such as c++ or java, one of the disadvantages of python is that its performance compared to compiled languages is usually lower. In python you can use extensions like cython that allow the use of libraries and compile the code looking to optimize it over the c++ compiler, this option would seek to improve the efficiency of this same[3].

C++ is widely used in general-purpose programming languages. The language allows you to encapsulates high and low-level language features. So, it is seen as an intermediate-level language. It also used to develop complex systems where the hardware level coding requires[4].

When comparing Python vs C++, Python follows a rule of "write once, run anywhere," which means that one code will work on all operating systems. However, the C++ code needs to compile on each OS before it can execute.[5]

The biggest difference in the discussion of Python vs C++ is that the C++ source code needs to become machine code. Python follows a different tactic as it is interpreted. However, the interpretation of code is usually slower than running code directly on the hardware.[5]

IV. Algorithms

A. bisection

The bisection method is a closed method of finding a solution of an equation of the form $f(x) = 0$ within a given interval $[a, b]$.

Algorithm:

1.Choose the first interval by finding the points a and b such that there is a solution between them. That is, that $f(a)$ y $f(b)$ thave different signs such that $f(a)f(b) < 0$.

2. Calculate the first estimate of the numerical solution:

$$m = \frac{a+b}{2}$$

3. This is done by checking the sign of the product $f(a) \cdot f(m)$:

* If $f(a) \cdot f(m) < 0$, the real solution lies between x_a y m

* If $f(a) \cdot f(m > 0$, the real solution lies between m y b

4. Select a new subinterval that contains the true solution, either $[a, m]$ o $[m, b]$.

5. Steps 1 to 4 are repeated until a specified tolerance or error limit is reached.

B. Regula falsi

The false position or Regula Falsi method is a bracketed method for finding a numerical solution to an equation of the form $f(x = 0$ within a given interval $[a, b]$. solution of an equation of the form $f(x) = 0$ within a given interval $[a, b]$.

In that interval, the equation of a straight line joining a and b is:

$$y = \frac{f(b) - f(a)}{b - a}(x - b) + f(b) \tag{2}$$

From (2), the point of m where the line intersects the x-axis is determined by substituting $y = 0$ and solving for x:

$$m = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)} \tag{3}$$

The procedure (or algorithm) for finding a solution with the regula falsi method is almost the same as that of the bisection method.

Algorithm:

1.Choose the first interval by finding the points a and b such that there is a solution between them. That is, let $f(a)$ and $f(b)$ have different signs such that $f(a)f(b) < 0$.

2.Calculate the first estimate of the numerical solution using (3)

3. Determine if the true solution is between a and m, or between m and b. This is done by checking the sign of the product $f(a) \cdot f(m)$:

* If $f(a) \cdot f(m) < 0$, the real solution lies between a and m

* If $f(a) \cdot f(m) > 0$, the real solution lies between m and b

4. Select a new subinterval that contains the true solution, either $[a, m]$ or $[m, b]$.

5. Steps 1 to 4 are repeated until a specified tolerance or error limit is reached.

C. Fixed-point

The fixed-point method finds a numerical solution of an equation of the form $f(x) = 0$ by rearranging the equation such that x is on the left-hand side:

$$x = g(x) \tag{4}$$

$$x_{i+1} = g(x_i) \tag{5}$$

D. Newton-Rapshon

If the initial guess of the root is x_i, a tangent/straight line can be extended from the point $(x_i, f(x_i))$. The point where this tangent crosses the x-axis represents an improved estimate of the root.

The Newton-Raphson method can be derived from this geometric interpretation. The derivative at a point is:

$$f^{'}(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{6}$$

Setting $f(x_{i+1}) = 0$, we obtain:

$$f^{'}(x_i) = \frac{0 - f(x_i)}{x_{i+1} - x_i} \tag{7}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f^{'}(x_i)} \tag{8}$$

(1) is called the Newton-Rhapson formula and starts from an initial value, x_0.

The method starts from an initial value, x_0. It can be proven that convergence is quite likely when $g^{'}(x) < |1|$.

Although the Newton-Raphson method is usually very effective, there are situations in which it does not work well.

Its convergence depends on the nature of the function and the accuracy of the initial estimate.

* If an inflection point ($f^{''}(x) = 0$) occurs near the root, the Newton-Raphson technique may oscillate around a local maximum or minimum.

* The worst-case scenario is an extremum ($f^{'}(x) = 0$) that causes a division by zero because the solution shoots horizontally and never reaches the x-axis.

The method starts from an initial value, x_0. It can be proven that convergence is quite likely when $g^{'}(x) < |1|$.

E. Secant

One potential problem in the application of the Newton-Raphson method is the evaluation of the derivative. While this is not inconvenient for polynomials and many other functions, there are certain functions whose derivatives can be extremely difficult or inconvenient to evaluate. For these cases, the derivative can be approximated by a direct difference:

$$f^{'}(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{9}$$

Instead of using two values of x, the modified secant method introduces a small perturbation fraction, δ, such that (1) becomes:

$$f^{'}(x_i) = \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i} \tag{10}$$

Substituting (10) in the Newton-Rhapson formula,

$$f^{'}(x_i) = \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i} \tag{11}$$

The choice of a suitable value for $delta$ is not automatic. If δ is too small, the method becomes vulnerable to rounding errors caused by subtractive cancellation in the denominator of (3). If it is too large, the technique can become inefficient and even divergent. However, if chosen correctly, it provides a good alternative for cases where the evaluation of the derivative is difficult and the development of two initial guesses is inconvenient.

V. What will we test?

A. Algorithms

In this project we will only find roots of an equation using:

- Newton-Rapshon method.
- Bisection method.
- Secant method.
- Regula-Falsi method.
- Fixed-point iteration method.

The algorithms explained above were developed in python and c++, and will be tested to compare the efficiency of these in each of the cases, for this we will test different functions that seek to see the speed and resource consumption with which these methods are executed in each of the attempts.

VI. Procedures

For the execution of the methods we looked for different functions that allow the comparison of these methods for this we will run each of the methods with the corresponding functions, and we will use the *linux time* tool that allows us to measure how long it takes to run our programs.

To count the amount of time consumed by each of the tests, they were run with the linux time command, which allows us to know how long it takes to execute a process.

```
time python3 ./bisection.py
```

As you can see the execution time of a program, in this case the real time is the one that determines the execution time.

```
real    0m0.010s
user    0m0.016s
sys     0m0.000s
```

A. Bisection

For the execution in both python and c++ the following parameters will be configured:

Equation: $3x^3 - 2x - 5$
Lower end: 0
upper end: 2
Tolerance: 0.00001
Maximum number of iterations: 20

B. Regula falsi

For the execution in both python and c++ the following parameters will be configured:

Equation: $\cos(x) - xe^x$
Lower end: 0
upper end: 2
Tolerance: 0.00001
Maximum number of iterations: 20

C. Fixed point

For the execution in both python and c++ the following parameters will be configured:

Equation: $\frac{2-e^x+x^2}{3}$
Initial point: 0
Tolerance: 0.00001
Maximum number of iterations: 20

D. Newton Rapshon

For the execution in both python and c++ the following parameters will be configured:

Equation: $3x^2 - 2x$
Initial point: 0
Tolerance: 0.00001
Maximum number of iterations: 20

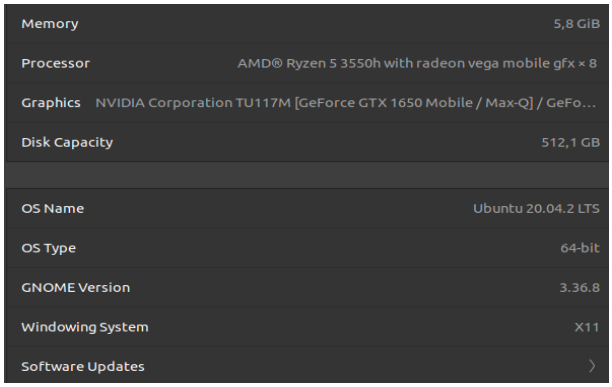
E. Secant

For the execution in both python and c++ the following parameters will be configured:

Equation: $x^3 + 1$
Lower end: 0
upper end: 2
Tolerance: 0.00001
Maximum number of iterations: 20

F. machine

Components with which each of the tests were carried out



VII. Results

A. Bisection

For the function, the method was run 10 times to observe the different changes in the method.

$3x^3 - 2^x - 5$		
# Results	Python	C++
1	0.229	0.104
2	0.206	0.012
3	0.216	0.016
4	0.207	0.021
5	0.204	0.023
6	0.208	0.017
7	0.209	0.016
8	0.204	0.023
9	0.207	0.022
10	0.205	0.023

From this, the average of these 10 executions is kept looking for the average execution time of the method, finding that the method executed with c++ is around 10 times more efficient than the one made in python.

Average		
Results	Python	C++
	0.2095	0.0187

B. Regula falsi

For the function, the method was run 10 times to observe the different changes in the method.

$cos(x) - x * e^x$		
# Results	Python	C++
1	0.209	0.039
2	0.218	0.042
3	0.244	0.039
4	0.214	0.041
5	0.212	0.057
6	0.222	0.044
7	0.214	0.048
8	0.215	0.053
9	0.224	0.049
10	0.215	0.047

From this, the average of these 10 executions is kept looking for the average execution time of the method, finding that the method executed with c++ is around 5 times more efficient than the one made in python.

Average		
Results	Python	C++
	0.216	0.045

C. Fixed-point

For the function, the method was run 10 times to observe the different changes in the method.

$2 - e^x + x^2$		
# Results	Python	C++
1	0.212	0.031
2	0.222	0.044
3	0.214	0.042
4	0.215	0.041
5	0.235	0.052
6	0.214	0.043
7	0.217	0.041
8	0.213	0.034
9	0.222	0.058
10	0.212	0.051

From this, the average of these 10 executions is kept looking for the average execution time of the method, finding that the method executed with c++ is around 5 times more efficient than the one made in python.

Average		
Results	Python	C++
	0.2176	0.0437

D. Newton-Rapshon

For the function, the method was run 10 times to observe the different changes in the method.

$x^3 - x^2 + 2$		
# Results	Python	C++
1	0.655	0.028
2	0.748	0.024
3	0.717	0.019
4	0.812	0.021
5	0.697	0.017
6	0.721	0.022
7	0.694	0.023
8	0.686	0.017
9	0.707	0.024
10	0.729	0.028

From this we take the average of these 10 executions looking for the average execution time of the method, finding that the method executed with c++ is more or less 20 to 32 times more efficient than Python.

Average		
Results	Python	C++
	0.7166	0.0223

E. Secant

For the function, the method was run 10 times to observe the different changes in the method.

$x^3 + 1$		
# Results	Python	C++
1	0.777	0.011
2	0.735	0.012
3	0.682	0.010
4	0.680	0.011
5	0.705	0.010
6	0.670	0.011
7	0.690	0.010
8	0.670	0.011
9	0.680	0.012
10	0.673	0.010

From this we take the average of these 10 executions looking for the average execution time of the method, finding that the method executed with c++ is more or less 50 times more efficient than the one made in python.

Average		
Results	Python	C++
	0.6962	0.0108

VIII. Conclusion

In conclusion the construction of root equation methods in both python and c++ are relatively simple, the comparison was based on discovering the performance of these numerical methods in a interpreted language vs a compiled language, in this case the comparison between python and c++ gives us some pretty clear results, c++ is much more efficient executing the algorithms, and not constituting a very high development complexity with respect to python, because normally due to the convenience of the libraries in python usually the development of these methods are faster, but if you are looking for efficiency in the results should be developed with c++.

With respect to the level of knowledge that you must have to develop these algorithms should be much higher in the case of c++, compared to python.

A topic that was not treated in this project and that is raised as future work is the development of these methods in python but to use an extension of c++ that allows the execution of functions and declaration of variables with c++ allowing the facility of python and the great power of the compiler of c++ and its optimizations.

References

[1] Burden, Richard L. and Faires, Duglas. Análisis Numérico. Editorial Thomson. 9 Edición 2011.

[2] Chapra, S. and Canale, R., 2003. Numerical methods for engineers. Boston: McGraw-Hill.

[3] L. Dalcin, et al.,"Cython: The Best of Both Worlds" in Computing in Science Engineering, vol. 13, no. 02, pp. 31-39, 2011.doi: 10.1109/MCSE2010.118

[4] A Brief Description - C++ Information", Cplusplus.com, 2021. [Online]. Available: https://www.cplusplus.com/info/description/.

[5] Python vs C++ Comparison: Compare Python vs C++ Speed and More", BitDegree.org Online Learning Platforms, 2021. [Online]. Available: https://www.bitdegree.org/tutorials/python-vs-c-plus-plus/.