

Computación avanzada

Práctica 1: Determinación de la constante de Madelung



19 de febrero de 2023

Universidad Autónoma de Madrid

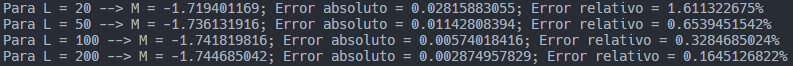
Pablo Gradolph Oliva

En el enunciado de la práctica se nos proporciona la siguiente ecuación:

Donde M es la constante de Madelung y donde los términos con *i+j+k* par son positivos y los términos con *i+j+k* impar son negativos.

Se pide hacer un programa en Python y otro en C++ para calcular dicha constante para valores de L = 20, 50 ,100 y 200, y comparar los resultados obtenidos con el valor real de M (M = -1.74756) y comparar los tiempos de ejecución entre ambos lenguajes.

Para ello, he hecho los dos programas principales “Practica1V1.py” y “Practica1V1.cpp” en los que la lógica del programa es la misma en los dos lenguajes de programación, obteniéndose así los mismos resultados:



Vemos como para valores más altos de L, la precisión en la solución obtenida es mayor y, por tanto, el error cometido es menor. Algo lógico, puesto que para longitudes más altas del cubo considerado (leer enunciado), estamos teniendo en cuenta más coordenadas del espacio lo cual mejora la aproximación (valor real cuando ).

Respecto a los tiempos de ejecución, encontramos una gran diferencia entre los dos lenguajes, y es que, aunque el tiempo de ejecución no siempre es el mismo y depende del ordenador del que se disponga o del número de tareas que esté realizando en ese momento, vemos que, en Python, el cálculo de M para los 4 valores de L toma siempre más de 1 minuto, mientras que C++ siempre está por debajo de los 2 segundos. Ejemplo de una de las medidas:

Python:



C++:



Podemos considerar que es diferencia significativa y, por esta razón, he creado los ficheros “Practica1V2.py” y “Practica1V2.cpp”, en los que he trabajado de la siguiente forma:

Dado que en C++ la ejecución es bastante rápida, en la versión 2 del programa lo que he hecho es un bucle do-while de forma que el programa finalice cuando el error en la aproximación de M esté por debajo de una tolerancia dada (a partir de una tolerancia 10e-5 puede llevar varios minutos). De esta forma, conseguimos un programa que realice una aproximación mucho más precisa para la constante de Madelung (L se incrementa de 100 en 100 por cuestión de eficiencia pero este parámetro es modificable).

Por otro lado, en Python he estudiado cuáles son las principales causas por las que la ejecución es tan lenta y he tratado de optimizar el programa al máximo. Para ello, he hecho varias pruebas y he visto que la función sqrt es similar a elevar a 1/2. Sin embargo, multiplicar una variable por sí misma es ligeramente más eficiente que elevarla al cuadrado. Y, por último, me he dado cuenta de que lo que más tiempo le lleva a Python dentro de este programa son los bucles for y, por tanto, he buscado una alternativa más eficiente.