# HW 1 write-up

## Prereq

Before starting the coding portion of the assignment, I learned the general structure and format of C++. More explicitly, I learned that for projects like this, a .hpp header is needed so that functions can be cross-referenced in different parts of the assignment, and .cpp files themselves should only include the main function. I also learned the basic ways to declare a constant, function, how to include libraries, and how if, for, and while functions are formatted in C++.

## Hw1\_1

I started off hw1\_1 by declaring the constants sigma, epsilon, and atomic number n for Au that was used directly in all the other functions. Then, I made a structure called Atom that stores the atomic number (which should be 79 for Au for all the input cases, but if not, it should output error), and the x, y, z coordinates. Then, I created a function Rij to calculate the distance between atoms. Eij uses the Lennard-Jones potential at Rij formula provided on the pset. ELJ is a summation of Eij over all atoms, coded with a for loop restricting I < j. Lastly, I coded a load\_atoms function to read the files line by line and assigning proper numbers to each item in struct Atom. In the main function, output was formatted specifically according to the sample\_output provided.

## Hw1\_2

Hw1\_2 has three functions to calculate force. Fi\_analytical takes in atoms and computes the derivative of their LJ energy (this part is computed manually by me) as a force magnitude. Then, it distributes the force along the x, y, and z direction according to their change in x, y, z, and the distance between atoms. Fi\_forward and Fi\_central similarly are introduced a new variable, h (step size), and uses a trick where whether a change is in x, y, or z direction depends on the dimension, and dimension is looped over 0, 1, and 2 for all atoms. In Fi\_forward, an elj is computed before and after step size is added to the coordinates to take the difference. In Fi\_central, an elj is computed after the step size is added and subtracted from the coordinates. Like hw1\_1, main function reads the input file and output according to the format in sample\_output.

## Hw1\_3a and b

Hw1\_3a calculates the steepest descent by calculating the energy after every step and moving towards the direction of gradient. It was hard to understand what the direction of gradient means in the beginning until I realized the gradient of energy is just force. So minimum energy is updated after every step calculated by passing the new coordinates into ELJ function from hw1\_1. I also had to decide how to gradually decrease the step size and by how much to come up with reasonable number of iterations and accurate answers. I chose 0.9 times the previous step size. For hw1\_3b, it is the same concept but now for the best\_step\_size it is based on golden ratio instead of gradual decrease. Another thing I had to debug here was that my function didn’t stop iterating until it reached the max iteration, but most of the iterations produce energies very similar to each other. I realized I had to 1) normalize my forces so that it doesn’t just fluctuate around the minimum energy, and 2) set a new break case where if delta energy is too small from the previous iteration, it should assume it was close enough to the answer.