Lennard Jones Argon Atom-Normal Analysis

- Ayush Sharma
- 2019101004

Given:

- N = 108 (initial number of atoms)
- $L_x = L_y = L_z$ = 18 A° (side of the cube)
- ∈=0.238 Kcal/mol (Lennard Jones Energy Parameter)
- σ = 3.4 A°
- r_{ij} (radius) ≥ 3.4 A°(distance between any two pairs of atoms)

This project accounts for analysing Normal mode of an Argon system of 108 atoms. The project implements the following things:-

- Random initial configuration generation.
- Calculating `LJ Potential` of the generated System.
- Finding minimum energy configuration of generated system (Using Steepest Descent Algorithm)
- Hessian Matrix calculation with Eigen vectors & Eigen values
- Plotting Vibrational Frequencies

Random initial configuration generation.

A class for configuration of molecule `Configuration()` has been made in file `configuration.py` , for storing the 3-d coordinates of each of the 108 atoms and calculating their energy described in the next section.

Code for random generation has been done in file `q1.py` . Since, the code is self explanatory as it generates 3-D points for cube $L_x = L_y = L_z$ = 18 A° s.t. no two points have distance less than r_{ij} (radius) ≥ 3.4 A° (following PBC).

It saves the generated configuration in file `init_conf.xyz` in `outputs` folder.

Calculating `LJ Potential` of the generated System.

In this I calculated LJ Potential/Energy of the generated system. Code for random generation has been done in file `q2.py` . The main function which calculates energy is (in file `configuration.py`) :-

```
def calculate_potential(self):
    print("Total atoms = ",self.total_atoms)
    pairs = []
    for i in range(self.total_atoms):
        for j in range(i+1, self.total_atoms):
            pairs.append((self.config[i],self.config[j]))

    potential = 0
    for (p1, p2) in pairs:
        Rij = norm(self.pbc(p1,p2))
        if Rij!=0:
            val = 4*self.epsilon
            a = self.sigma/Rij
            val = val*( a**12 - a**6 )
```

```
            potential += val
    return potential
```

Basically I did the summation of interaction energy per pair, which is as follows:-

$$U_{LJ}(r_{ij}) = 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]$$

The output for the submitted molecule is: **-122.0238631761858**

Finding minimum energy configuration of generated system.

In this part we minimises the total energy of system w.r.t PBC using steepest descend algo.

Steepest Descend Algorithm :

- Heuristics:-
    1. iteration_heuristic = 200
    2. alpha_heuristic = 0.135

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$
2: `for` $k = 1...K$
3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$
4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

Logic of Code :

- Used library `autograd` for faster calculation than `numpy` . It also posses numpy operations.
- Used `gradient` function from `autograd` for getting slope at given point.
- In `for` loop I stored the cost i.e. energy & weight i.e. configuraton of the system.
- Finaly, printedstored final conf. in file `final_conf.xyz` and looged the energy at each step in file `gradient_descent_log.txt` .

Code for steepest descent algo in `q3.py` :-

```
iteration_heuristic = 200
alpha_heuristic = 0.135
for i in tqdm(range(iteration_heuristic)):
    new_config -= alpha_heuristic*gradient(new_config)
    weight_history.append(new_config)
    cost_history.append(calculate_potential(new_config))
```

Lenard Jones Potential of the Initial configuration = -122.0238631761858

Lenard Jones Potential of the Final configuration = -152.47234949736315

PBC(Periodic Boundary Condition) :

These are boundary condition for approximating large system by focusing on it's small part i.e. unit cell.

Following function account for the same & calculates distance to the nearest mirror image in the simulation :-

```
def pbc(point1, point2):
    L = 18
    mod_length = (point2 - point1) % L # The image in the first cube
    return ((mod_length+L/2)%L)-L/2 # MIC separation vector
```

As a result, there is no need to verify the boundary conditions during the updating of the atom's coordinates. I convert the out of bound coordinates to inside the box as required when placing the solution in `final_conf.xyz` .

Hessian Matrix calculation with Eigen vectors & Eigen values

File `q4.py` accounts for Hessian matrix calculation & eigen vectors & values with class `Hessian` solely.

The file took atleast 50 minutes to run. So used `Pool()` for multiprocessing which reduces it to around 27 minutes and got the hessian matrix.

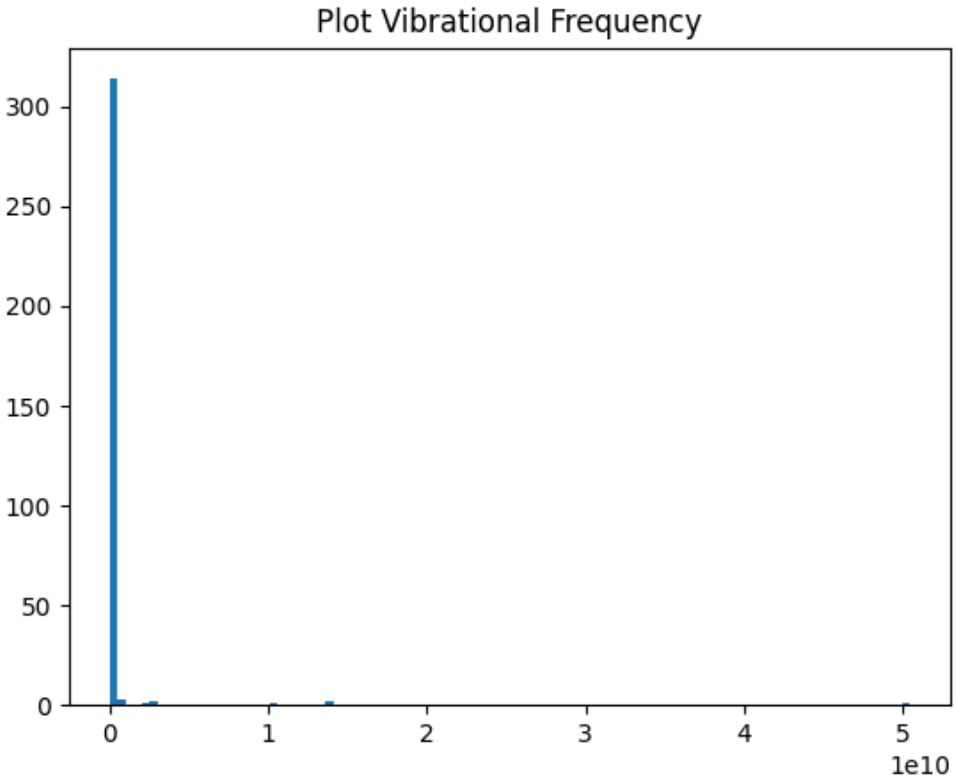
Used numpy library funtions for eigen balues & vectors.

The hassien matrix, eigen values & eigen vectorsis saved in `hassien.py` , `eigen_values.dat` and `eigen_vectors.dat` respectively.

Normal mode & Plotting Vibrational Frequencies

Normal modes are written in file `mode.xyz` . Format is:-

Histogram Plot :


Plot Vibrational Frequency
300
250
200
150
100
50
0
0
1
2
3
4
5
1e10