Science-2 Project Part A

Lennard Jones Argon Atom-Normal Analysis

- Ayush Sharma
- 2019101004

Table of Content:

Intro

Analysing the Normal Modes of an Argon System of 108 atoms following the Lennard Jones Potential. The code generates a initial random configuration of 108 atoms based on the given conditions, implements Periodic Boundary Conditions, Reduces the system of the random configuration using the Steepest Descent Algorithm for minimisation, then it generates a Hessian Matrix , and the eigen values and eigen vectors for it. It also plots a histogram of the frequencies.

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

For detailed report refer `Report.pdf`
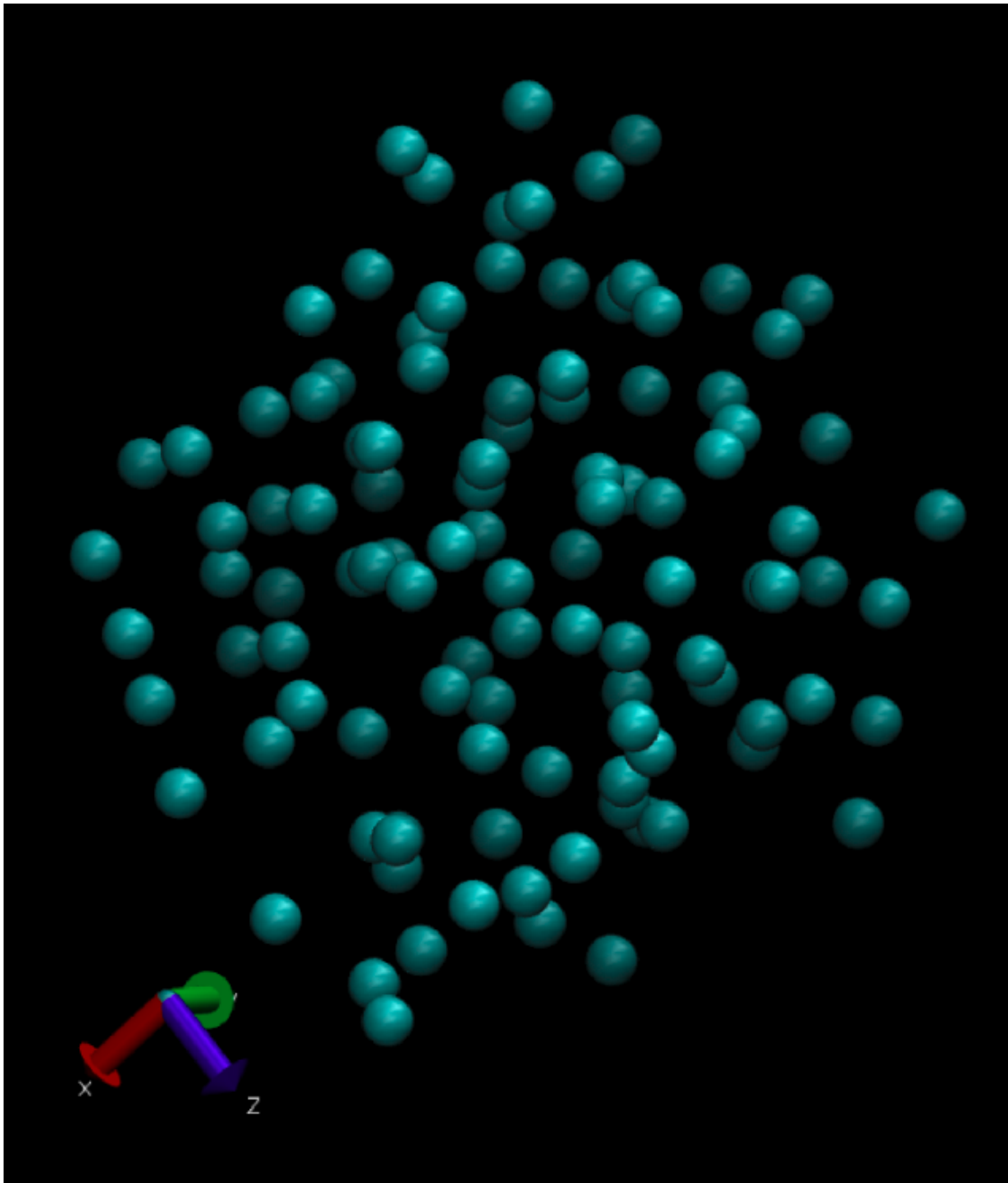
Random initial configuration generation.

Given:

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

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) $\geq$ 3.4 $A°$ (following PBC).

It saves the generated configuration in file `init_conf.xyz` in `outputs` folder. Following is the inital VMD output for initial configuration:-

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:-

$$V_{LJ}(r) = 4 \in \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]$$

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.

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` .

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: $\qquad \mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$
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

Following is the final VMD output for final configuration:-
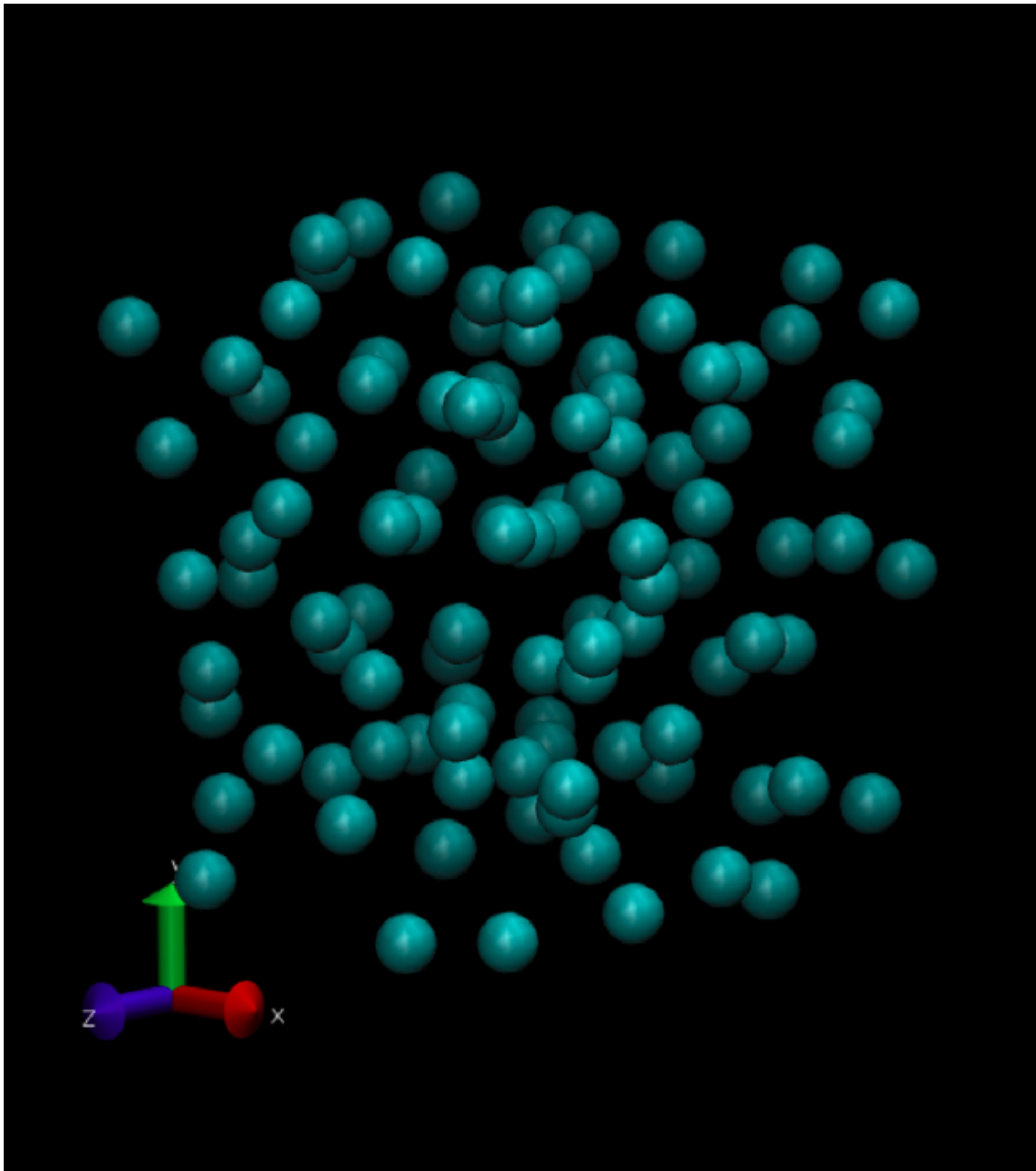
Hessian Matrix calculation with Eigen vectors & Eigen values

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.

Formula used :-

Let N be the number of atoms and let (x_A, y_A, z_A) be the Cartesian coordinates of the Ath atom.

$$(\mathbf{H})_{AB} = \frac{\partial^2 E}{\partial X_A \partial X_B} \qquad (X_{3A-2}, X_{3A-1}, X_{3A-0}) = (x_A, y_A, z_A) \qquad \text{for } A \in \{1, \ldots, N\} \qquad (1)$$

$$\frac{\partial^2 E}{\partial X_A^2} = \frac{E(X_A + h) + E(X_A - h) - 2E(X_A)}{h^2} \qquad \text{for } X_A = X_B \qquad (2)$$

$$\frac{\partial^2 E}{\partial X_A \partial X_B} = \frac{1}{2h^2} \left(E(X_A + h, X_B + h) + E(X_A - h, X_B - h) - E(X_A + h, X_B) - E(X_A - h, X_B) \right.$$
$$\left. - E(X_A, X_B + h) - E(X_A, X_B - h) + 2E(X_A, X_B) \right) \qquad \text{for } X_A \neq X_B \qquad (3)$$

Normal mode & Plotting Vibrational Frequencies

For diagonal mass-weighted hessian matrix, we have sum of harmonic oscillator hamiltonians, whic can be done by choosing normal coordinates = {q1, . . . , q3N }.

As hessian is real & symmetric, ti can be orthogonalised using real orthogonal matrix as followin.

$$\tilde{\mathbf{H}}_0 = \tilde{\mathbf{Q}} \mathbf{K} \tilde{\mathbf{Q}}^T \qquad \mathbf{K} = \begin{pmatrix} k_1 & 0 & \cdots & 0 \\ 0 & k_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & k_{3N} \end{pmatrix} \qquad \tilde{\mathbf{Q}}\tilde{\mathbf{Q}}^T = \tilde{\mathbf{Q}}^T\tilde{\mathbf{Q}} = 1$$

where Q are eigen vectors.

Back-tranforming to un-mass waighted cartesian space gives the normal modes of the system.

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

```
N
COMMENT LINE 1
A1  x1  y1  z1     dx11 dy11 dz11
A2  x2  y2  z2     dx12 dy12 dz12
. . .
AN  xN  yN  zN     dx1N dy1N dz1N

N
COMMENT LINE 2
A1  x1  y1  z1     dx21 dy21 dz21
A2  x2  y2  z2     dx22 dy22 dz22
. . .
AN  xN  yN  zN     dx2N dy2N dz2N


. . .


N
COMMENT LINE N
A1  x1  y1  z1     dxN1 dyN1 dzN1
A2  x2  y2  z2     dxN2 dyN2 dzN2
. . .
AN  xN  yN  zN     dxNN dyNN dzNN
```

Histogram Plot :

Some modes are very high in number while others are very low.

Frequency Histogram