Exercise 2

Due: Thursday, 10/17/19

Objective: To learn how to compile Fortran libraries for Python, and to write a short Python code to perform energy minimization using the conjugate-gradient method.

The term "clusters" denotes small, stable packings of (often spherical) particles. For years, many groups have been interested in studying the geometries and energetics of clusters of objects ranging in size from atoms to small colloidal particles in the nanometer to micrometer scale. Cluster analyses are instrumental to understanding a wide range of physical phenomena, including the structure of solids, diffusion in dense liquids, aggregation of colloidal and other particles in solution, nanostructured materials, and the self-assembly behavior of many synthetic and biomolecular systems.

A cluster is characterized by the number of particles and the energetic interactions between them. For attractive spherical particles, such as those that might be modeled by the Lennard-Jones interaction, it is found that there are cluster sizes of particularly exceptional stability. The so-called **magic cluster numbers** correspond to cluster sizes where the packing of atoms is most efficient and can reach a particularly stable configuration. The most stable magic clusters are built from an **icosahedral** geometric arrangement of the particles. The first few magic cluster sizes in this series are 13, 19, 38, 55, and 75.

To find stable packings of particles, we first build an interaction model using the dimensionless Lennard-Jones (LJ) potential:

$$U^* = \sum_{i<j} 4\left(r_{ij}^{-12} - r_{ij}^{-6}\right)$$

We can then start with a random configuration of particles and use an energy-minimization algorithm to find a stable structure of low energy. However, because the number of local minima is so large, we will need to repeat this procedure for a number of times starting from different random configurations in order to locate the global minimum.

In addition, for the minimization procedure to work, we need to add a very weak biasing potential that pulls our particles towards the origin. This potential helps favor a single cluster, rather than multiple separate ones, during the minimization procedure. Without it, the LJ potential does not have sufficiently long-ranged interactions to pull together atoms that are separated by several units of distance. The biasing potential is harmonic such that our total force field is:

$$U^* = \sum_{i} \alpha |\mathbf{r}_i|^2 + \sum_{i<j} 4\left(r_{ij}^{-12} - r_{ij}^{-6}\right)$$

Here, $\alpha = 0.0001N^{-2/3}$ is a very small parameter. The scaling is chosen so that the average energy due to this term for a cluster of N particles is constant.

In this assignment, you will perform conjugate gradient minimization of Lennard-Jones particles, initially randomly distributed in space. Part of the code has already been written for you, including the Fortran library that computes the pairwise interaction energies & forces and the basic Python routine for performing the line search.

Part a

Download from the course website the files ex2lib.f90 and ex2.py. Place these files in a new path on your computer where you will store your code and perform your computations. Then, compile the pre-written Fortran library using f2py. On a Windows machine, you should use:

```
c:\mydir> f2py –c –m ex2lib ex2lib.f90 --fcompiler=gnu95 --compiler=mingw32
```

On other systems, you might delete the --compiler=mingw32 option altogether, or replace it with your favorite C compiler (e.g., --compiler=gcc).

Part b

The file ex2.py is a template for your code that already has some code written in it. Take a look at the existing code to learn what it is doing. Then, write code for the following functions:

```
function:
      InitPositions(N, L)
description:
      Returns an array of initial positions of each atom, placed
         randomly within a cubic box of dimensions L
arguments:
      N: number of atoms
      L: box width
returns:
      Pos: (N,3) array of positions

function:
      ConjugateGradient(Pos, dx, EFracTolLS, EFracTolCG)
description:
      Performs a conjugate gradient search to find an energy minimum.
arguments:
      Pos: starting positions, (N,3) array
      dx: initial step amount, a float
      EFracTolLS: fractional energy tolerance for line search
      EFracTolCG: fractional energy tolerance for conjugate gradient
returns:
      PEnergy: value of potential energy at minimum
      Pos: minimum energy (N,3) position array
```

For your conjugate gradient algorithm, keep the following points in mind:

- A line search function has already been written in the template file, and you will call this from your algorithm.
- You can get the potential energy and forces array using the library module:
  PEnergy, Forces = ex2lib.calcenergyforces(Pos)
- Your first line search direction should be the force vector. Subsequent line search directions i should be found using the conjugate gradient expressions:

$$\mathbf{d}_i^N = \mathbf{f}_i^N + \gamma_i \mathbf{d}_{i-1}^N \qquad \text{where } \gamma_i = \frac{(\mathbf{f}_i^N - \mathbf{f}_{i-1}^N) \cdot \mathbf{f}_i^N}{\mathbf{f}_{i-1}^N \cdot \mathbf{f}_{i-1}^N}$$

- You should continue successive line searches until the fractional change in energy found after successive searches is less than the tolerance EFracTolCG,
  $|U_i - U_{i-1}| <$ EFracTolCG $\times |U_i|$.

In debugging your code, you may want to insert print statements throughout to see the progression of the potential energy during the minimization iterations.

Part c

Write module-level code that systematically loops from $N = 2$ to $N = 25$ (inclusive). For each particle number, your code could should do the following:

- Perform K minimizations, each one starting from a different random configuration of particles. The K minimized energies should be put in a list.
- Display the minimum, average, and standard deviation of the minimized energies for the K trials.

For your simulation runs, use the following parameters (typical for this kind of system):

- EFracTolLS = 1.e-8
- EFracTolCG = 1.e-10
- the step size dx = 0.001
- For placing particles initially in a cubic volume $V = L^3$, choose L such that the average number density of particles (N/V) is equal to 0.001.

Part d

Plot the minimum and average energies as a function of N for each of $K = 100, 1000,$ and 10000. Your simulations may take several hours to finish for the last case. Can you estimate where you have been able to find the *global* minimum? Note: you can turn on real-time visualization of the minimization if you have VPython installed, which should be included with Python(x,y). Set the variable UseVisual = True in the code template and download the module atomvis.py from the course website and place it in the same folder as your script. However, visualization will slow your code by 10-20%.

Part e

Compare your results to the known global minimum energies, taken from [Leary, J. Global Optimization 11, 35 (1997)] . You can download these values from the course website. Add this curve to your graph. Why might your results be higher?

N	$-U_{\text{min}}$	N	$-U_{\text{min}}$	N	$-U_{\text{min}}$
2	1.000	10	28.4225	18	66.5309
3	3.000	11	32.7660	19	72.6598
4	6.000	12	37.9676	20	77.1770
5	9.1038	13	44.3268	21	81.6846
6	12.7120	14	47.8451	22	86.8090
7	16.5054	15	52.3226	23	92.8445
8	19.8215	16	56.8157	24	97.3488
9	24.1134	17	61.3180	25	102.3727

Part f

From simple macroscopic arguments, we would expect that the energy of a cluster would scale with both the surface area (via a surface tension) and volume (via a bulk energy density). Thus, we could model the global energy minimum as

$$U_{\text{macro}} = a + bN^{\frac{2}{3}} + cN$$

Find the constants a, b, c by fitting your minimum energy data in the $K = 10000$ case. This fitting can be done by least-squares minimization using the Solver function in Excel, or you can use any other numerical software of your choice. Then, plot the difference between the true minimum energy and the expected macroscopic energy $U - U_{\text{macro}}$ as a function of N. Can you identify the magic numbers 13 and 19, which should correspond to lower energies than expected from the macroscopic scaling estimate?

Part g – ADVANCED TRACK

For the case in which $N = 13$ and $K = 10000$, add to your Python code a routine that will make a list of all of the *unique* energy values found from the K minimizations. Because your minimization routine is finite and will not return *exact* minimum energies, you will have to test whether or not two energies are within numerical tolerance. You can use the numpy function allclose(val1, val2, rtol, atol) for this purpose. This function will return True if val1 and val2 have a relative (fractional) difference less than rtol and an absolute difference less than atol. Use atol=1.e-4 and rtol=1.e-6 in your code. Then, sort the list from minimum to maximum energy. Make a plot of the energies as a function of their rank (i.e., 0 for the minimum energy, 1 for the next highest, 2 for the next, and so on) for rank 0 to 200. Also, indicate the number of unique minima found. For comparison, Doye and coworkers [Doye, Miller, and Wales, J. Chem. Phys. 111, 8417 (1999)] used advanced minima-finding algorithms to determine that there are 1467 unique minima.

Part e – ADVANCED TRACK

How do the magic numbers change if the Lennard Jones particles are part of a polymer? In other words, consider a potential of the form:

$$U^* = \sum_{i+1<j} 4\left(r_{ij}^{-12} - r_{ij}^{-6}\right) + \sum_{i+1=j} \frac{k}{2}\left(r_{ij} - r_0\right)^2$$

where $k = 3000$ and $r_0 = 1$. Repeat part (f) for this case. Based on physical intuition, do you expect there to be more or fewer minima for the polymer versus individual LJ particles?

What to turn in

For the assignment, submit a short, typed summary of your results, clearly indicating the problem components. In addition, print a copy of your Python code for reference and attach it to the back of these.