Project 1: Atomistic Monte Carlo Simulation

This 3-part project will have you:

1. Complete a Monte Carlo (MC) simulation code
2. Run MC simulations of argon to determine thermodynamic properties
3. Develop and apply a radial distribution function (RDF) code to quantify structural changes in argon at different state points.

Grading:

Your project is due back in two weeks. If you do not think you can meet the deadline, speak with me immediately!

The project is worth a total of 25 points toward your final grade, i.e., a 100% in this assignment will add 25 points to your final grade, an 80% 20, and so on. Assignment percentage points are provided next to each task.

The following opportunities exist for bonus (percentage) points on this assignment:

- +1% if you find a bug in my template code
- +5% if you write your code(s) from scratch – note, if you choose to do so, I encourage you to copy the trajectory file printing block – this is a very specific format that is readable by simulation visualization software
- +5% if you implement hardware acceleration and show timing data

Please refrain from using libraries for your computations (e.g., numpy for histogram generation). The goal of this project is to understand complete command of course materials though **implementation**!

Tips:

- This assignment will take time, both in terms of coding and running the actual simulations. Do not wait to get started!

- All of these calculations can be run on a desktop computer, however I **strongly** recommend running on the lab computers, which can be accessed remotely)

- Unit tests/sanity checks will save you time and headaches, e.g.:

    - Unit testing example: If you are completing the code for the get_LJ_eij function, you can test that your implementation is correct by have the code print out sigma, epsilon, rij, and the value of each term, and then quit (“exit(0);”). Using the printed rij, you can then compute the expected value by hand and verify if it matches up. Remember, you can also comment out as much code as you’d like when testing!
    - Validation data for these simulations can be found online (see part 2 of the project) – use these to check your work, But recognize that your data may not match *exactly* (see part 2)

- Save/backup your files frequently. This would be a good time to practice your Git skills.

- If you're having compilation bug issues, UM-GPT is a great tool for debugging!

Documents required for submission:

Don't forget to indicate units and to add axis labels/legends to plots, where applicable!

- ☐ Your MC software source code
- ☐ Your RDF source code
- ☐ A report containing:
    - Your simulation equilibration plots with a requested discussion of results
    - Your plotted simulation/literature data with errors and discussion of results
    - Your timing data plot with discussion of results
    - Your RDF and number integral plots with discussion of results
- ☐ If seeking bonus points for finding a bug: Documentation and explanation of the bug
- ☐ If seeking bonus points for hardware acceleration:
    - Your hardware accelerated code
    - An overview of your parallelization strategy
    - An overview of how you ran it
    - Timing data, e.g., change in computational efficiency versus requested computational resources – for this I suggest you increase the number of atoms in your simulation.
    - Another table of thermodynamics data for one of your accelerated simulations

---

Part 1: Completing the Monte Carlo Code

Your task is to complete a Monte Carlo code capable of canonical ensemble simulations of monoatomic species.

Template codes (i.e, Monte Carlo codes with some lines missing) are available on Canvas. Using the instructions that follow, you'll complete the code so that it compiles and runs properly. You'll find a few versions of the template code available to accommodate differing levels of programming experience – you may choose to work from any of those templates without penalty. If you find and document a bug in one of my template codes, you will earn +1% to your project grade. If you choose to create your own code, you will earn +5% to your project grade. If you implement hardware acceleration (e.g., MPI or GPU) and provide timing data, you will earn +5% to your project grade. Thus, a maximum of 11% bonus points are available.

If you are developing your own code, note that you can use the Mersenne Twister library (Mersenne_twister.h) as your random number generator – see the template code for implementation details.

Instructions:

Note: Line numbers are given based on the inline code.

1. Complete lines 232 and 233 which give the simulation's reduced density and temperature

2. Complete the get_dist function. It should:
   - Compute the x-, y- and z- distances vector components from atom a1 to atom a2 using the minimum image convention (i.e., updating the rij_vec object)
   - Return the Euclidean (scalar) distance between the two atoms

3. Complete the get_LJ_eij function. It should return the Lennard Jones energy for a pair of atoms separated by distance rij, accounting for the model cutoff.

4. Complete the get_LJ_fij function. It should return the Lennard Jones force vector between a pair of atoms separated by distance vector rij_vec, accounting for the model cutoff.

5. Determine the initial system energy and virial pressure. This entails looping over all pairs of atoms with i>j and:
   - Computing distance between the two atoms
   - Accumulating the atom pair energy
   - Accumulating the atom pair stresses

6. Complete the get_single_particle_LJ_contributions function. It should compute the contribution to the system energy from a single atom. This entails:
   - Computing distance between the atom of interest and every other atom "j" in the simulation (in a loop)
   - Accumulating the pair energy between the atom of interest and atom j
   - Accumulating the stress due to forces between the atom of interest and atom j

7. Complete the code for a trial move (ignore the "Widom" related lines of code for now). This entails:
   - Generating a trial displacement within the maximum allowed displacement
   - Updating the trial atom's position, accounting for periodic boundary conditions
   - Determining the trial atom's energy contribution in its trial position

8. Complete the code for accepting or rejecting the move. This entails:
   - Determining the change in energy
   - Completing the acceptance criteria

9. Write the code to update statistics. This includes:
   - Calculating the fraction of accepted moves
   - Calculating the maximum displacement

10. Complete the code to print out information on properties computed at that trial move. This includes:
    - Computing total pressure (i.e., virial + kinetic)
    - Computing average energy and average of squared energy
    - Computing the ∂<E>/∂T portion of heat capacity

11. Complete the Widom insertion section. This entails:
    - Uncomment the section starting with "// 5. Do a Widom insertion test"

- Generate a random position (*not* trial displacement) for the Widom particle
- Calculate the change in energy that would occur due to insertion of the Widom particle
- Update the Widom factor

12. Complete the final simulation output (i.e., starting on line 436)

Part 2: Using the Monte Carlo Code

Your task is to use your MC code to evaluate how thermodynamic properties of argon change with temperature and density. You will run simulations using 500 Ar atoms, using a particle size of 3.4 Angstroms, a well-depth of ε/k_B = 120 K, and an outer cutoff of 4 particle diameters. Simulations should be run for 5e6 MC steps, with 1e6 used for equilibration. Statistics/configurations should be printed every 2,000 steps.

- ☐ Run simulations at T = 144 K and *reduced* densities ranging from 0.1 to 0.9 by 0.1. At each density, run 4 independent simulations, for a grand total of 36 simulations.

  When running, save the timing data, standard output, and MC_traj.lammpstrj, e.g., if you compile your code with:

  ```
  g++ -O3 -o Monte_Carlo Monte_Carlo-Inline.cpp
  ```

  Then run with, e.g.:

  ```
  /usr/bin/time -o Monte_Carlo_timing.dat ./Monte_Carlo | tee Monte_Carlo.log
  ```

  Then rename the output so it is not overwritten during your next run, e.g.:

  ```
  dens=0.1; indep=1; for i in Monte_Carlo_timing.dat Monte_Carlo.log MC_traj.lammpstrj; do
  cp $i indep-${indep}.${dens}.${i}; done
  ```

  Do not delete these files until the class is over – we'll be using them again later on!

- ☐ Plot the plot reduced energy (E^*) and pressure (P^*) as a function of simulation step for all densities. All independent simulations should be in each plot. Specify at what point properties appear to be equilibrated. *Hint: Try generating two sets of plots for each property, one for the first 1e6 steps, and the other, for the final 1e4 steps.*

- ☐ Plot predicted average reduced energy (E^*), pressure (P^*), heat capacity ($C_{\mathrm{v,XS}}{}^*$), and chemical potential (μ^*_{XS}) at each density. Include error bars giving the standard error of the mean across independent simulations. Include literature values and corresponding error bars in this plot – data sources are given below. If you find differences between your data and literature values, speculate on possible sources. Discuss trends in your heat capacity error bars and possible reasons for those trends. Discuss why chemical potential goes through a minimum and exhibits a sign change.
  - Heat capacities (see supplementary information): Bearman et al., *J. Phys. Chem.* 96, 4093-4100 (1992)
  - Chemical potentials, energies, and pressures (main text): Ding & Valleau, *J. Chem. Phys.* 98, 3306–3312 (1993)

Hint: If you need to extract data from a plot rather than a table, consider using a tool like Plot Digitizer!

☐ Generate plots of runtime versus density. Discuss the trends you observe and speculate on the cause for these trends.

Part 3: Structural Analysis via Radial Pair Distribution Function (RDF)

Your final task is to write an RDF code and apply it to analyze one simulation at each temperature/pressure state point. Template codes (i.e, RDF code with some lines missing) are available on Canvas. Using the instructions that follow, you'll complete the code so that it compiles and runs properly. You'll find a template code available to accommodate differing levels of programming experience – you may choose to use the template without penalty.

☐ Complete an RDF code. If you choose to use the template code, here are the necessary steps:

1. Complete the double configuration::get_dist(int i, int) function, which computes the minimum-image-convention distance between atoms with indices i and j
2. Complete the definition of nbins. Set the number of bins based on the system box length and the requested bin width. Hint: How far out in distance should your RDF go if you're using periodic boundary conditions?
3. Complete the get_rdf_bin(double dist) function, which determines the shell bin index the current distance falls in. Hint: Make sure the bin you return will always fall within the range of the num_int vector!
4. Complete the criteria for incrementing the number of distances that fall within a given num_int bin
5. Complete the shell_vol, shell_density, shell_density_ideal, and avg_rdf. Hint: RDFs are computed based on *number* density (i.e., atoms per volume) rather than *specific* density (i.e., mass per volume)

☐ Generate two plots, one with the number integral for all densities, and the other with the radial pair distribution function for all densities. Comment on the features and trends you observe and speculate on what causes them.

Part 4: Additional Notes

- **If you collaborated with other students**: clearly state what that collaboration entailed
- **If you used UM-GPT to help with this assignment:** describe how you used the tool, what aspects you found most useful, and which were not very helpful
- **If you used additional resources to complete this project:** describe what they were, how you used them, and how you found them