Simulating liquid Argon with your own Python MD program

Midterm Project ASU PHY494/PHY598/CHM598 (2013)

February 21, 2013 – March 13, 2013

Abstract Your project is to write, in teams, a Python molecular dynamics (MD) code to simulate liquid Argon. We want to study how varying some of the simulation parameters affects the accuracy of the simulations. Part of your task is to figure out how to assess your simulations for accuracy. You will write a short "letter"-style paper to communicate, discuss and summarize your reasoning and your results.

Due Wed, March 13, 2013, 11:59pm. E-mail to *oliver.beckstein@asu.edu* with subject *PHY598: midterm.* Each student should

- submit their *own* report in **PDF** format;
- submit **all code** that is required to reproduce the results in the report. Include a text file `README.txt` that describes the commands to run simulation 3 in Table 1. The code must run on any of the iMacs using the instructions provided. The code package may be identical for each team member.

Marking will take the following into consideration:

- code runs and produces correct output
- report clearly and succinctly describes the question, method and results and contains sufficient evidence that the requirements (see below) have been met
- thorough attribution of code; proper use of citations in the report
- evidence of teamwork
- any additional work that you want to include in an appendix to the report or additional simulations for the main report will be treated as bonus material

1 Background and materials

For background reading the class notes, cited articles, and the book by Frenkel and Smit (1) (and also Allen and Tildesley (2) and Leach (3)) are recommended. A good general article on molecular dynamics simulations is Adcock and McCammon (4) and simulations of liquid argon have been discussed in an earlier homework assignment (5). Any code that has been developed or provided throughout the class can be used, either in full or in parts, as long as proper attribution is given (see below).

2 Requirements

The following are requirements of the project and need to be demonstrated in the final report (e.g. by including appropriate graphs or mentioning in the Methods section).

2.1 MD program

Write a program to simulate the Lennard-Jones fluid (which we use as a model for liquid Argon) in Python. Atoms are treated as point masses with positions $\mathbf{x}_i$ (in 3D space, i.e. $\mathbf{x}_i = (x_i, y_i, z_i)$) that interact through the pair-wise Lennard-Jones potential

$$v_{LJ}(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], \quad r_{ij} := \sqrt{(\mathbf{x}_j - \mathbf{x}_i)^2}. \tag{1}$$

The simulations are to be performed at constant volume V, particle number N, and total energy E, for a given density ρ and initial temperature T_0. As units in the code use nm for lengths, ps for time, nm/ps for velocities, u for mass, K for temperature, and kJ/mol for energies. Your program needs to be able to do at least the following:

System setup and initialisation This stage only needs to be completed once before running the actual simulation.

1. initialise the simulation to a system of the prescribed N and ρ (e.g. by generating a cubic lattice of the particles or placing them randomly[1]) Generate a cubic simulation cell with length L. Have the code print the box dimensions.
2. assign initial velocities, corresponding to the prescribed temperature T_0, in such a way that the total linear momentum is 0. Have the code print T_0 and initial temperature $T(t = 0)$, calculated from the initial velocities.

Molecular dynamics simulation This is the "core" of the code, which performs the force evaluation and integration of motion in time steps of size Δt.

1. use periodic boundary conditions in a cubic simulation cell with length L

[1] If placing randomly you must ensure that "clashes" are avoided, i.e. no overlaps between the LJ hard cores.

2. calculate forces between particles, using the minimum-image convention; interactions are truncated at a cut-off[2] $R_{\text{cut}} = 3\sigma$ where σ is the Lennard-Jones hard core parameter in Eq. 1.[3]
3. integrate the equations of motion using the Verlet integrator
4. calculate and store for each time step
   - the potential energy $U(t)$ of the system
   - the kinetic energy $T_{\text{kin}}(t)$ and system temperature $T(t)$
   - the total energy $E(t)$
   - the total linear momentum $\mathbf{P}(t) = \sum_{i=1}^{N} m_i \mathbf{v}_i(t)$
5. write out the coordinates every 0.1 ps to a trajectory in XYZ format[4]
6. At the end of the run, your code should report timing/performance statistics: total run time ("wall time"), wall time per time step, performance in steps per h and simulated ps/h, performance in steps per day and simulated ns/d.[5]

Your code should be able to run in Python 2.7 and should make use of NumPy. You can use any other standard Python modules if you want to [6]. Your code must run and produce output when run on any machine in the iMac lab.[7]

2.2 Analysis

After the simulation has finished you should

1. plot the time series $E(t)$, $E(t)/N$, $U(t)/N$, $T_{\text{kin}}(t)/N$ (average energies per particle), $T(t)$, total linear momentum $|\mathbf{P}(t)|$
2. plot histograms of T and U/N
3. report the time-averaged values $\langle T\rangle$, $\langle U/N\rangle$, $\langle E/N\rangle$ and standard deviations over a trajectory[8]

[2]If the simulation cell is too small, i.e. $L/2 < 3\sigma$ then set $R_{\text{cut}} = L/2$.

[3]You may implement shifted potentials and long range corrections to the energy (1) but this is not a requirement.

[4]When running longer simulations then we do not necessarily want to keep every time step because little changes between steps and the trajectories quickly become very big and fill up the disk.

[5]The Python `time.time()` function from the **time** module could be helpful.

[6]"standard Python modules" refers to the packages installed on the iMacs in the iMac lab, which means the Enthought Python Distribution 2.7.2 and the Python Standard Library 2.7.

[7]There is *no C or FORTRAN compiler available* on the iMacs so you will have to work with Python/NumPy alone.

[8]The time-average of a quantity A is defined as

$$\langle A\rangle = \frac{1}{\tau}\int_0^{\tau} A(t)\,dt = \frac{1}{N_{\text{steps}}}\sum_{i=0}^{N_{\text{steps}}} A_i \tag{2}$$

4. calculate the *energy drift* (6)

$$\Delta E = \frac{1}{N_{\text{steps}}} \sum_{i=1}^{N_{\text{steps}}} \left| \frac{E(0) - E(i\Delta t)}{E(0)} \right| = \left\langle \left| 1 - \frac{E(t)}{E(0)} \right| \right\rangle \tag{3}$$

   ($E(i\Delta t)$ is simply $E(t)$)

5. visualize the trajectory in VMD (7) and prepare an image of the *first* and *last* frame of the trajectory

6. plot the radial distribution function $g(r)$ (You can write your own code or use VMD[9].)

2.3 Input parameters

In order to simulate liquid argon we are using the Lennard-Jones parameters introduced by Rahman (5). Simulate liquid Ar ($m = 39.948$ u) at a density $\rho = 1.374\ \text{g} \cdot \text{cm}^{-3}$ and an initial temperature of $T_0 = 94.4$ K; for further parameters see Table 1. For the Lennard-Jones potential Eq. 1 choose[10] $\epsilon = 120\ \text{K} \cdot k_B = 0.99774\ \text{kJ} \cdot \text{mol}^{-1}$ and $\sigma = 0.34$ nm.

2.4 Simulations

Perform the simulations with the parameters listed in Table 1. Simulations 7*—9* are optional and you can do them to explore how system size affects the energies. You can of course do further simulations if you think they are necessary or if you want to add more data points to a graph.

3 Report

Prepare a "letter-style" paper in which you report what you accomplished. The report should contain a brief introduction, including a description of the problem, and overview over the methods used and implemented, the results obtained, and what the results mean (e.g. comment on the question of some of the simulation parameters affect the accuracy of the simulations).

- maximum 4 pages, use 12 pt font Cambria/Times New Roman/Helvetica/Arial, single spaced, minimum 1 inch margins; *including figures and tables* and excluding references, acknowledgements or appendices

where the second equality assumes that A is computed for each time step i over a trajectory of total length $\tau = N_{\text{steps}}\Delta t$. The standard deviation is as usual $\sigma_A = \sqrt{\langle (A - \langle A \rangle)^2 \rangle}$.

[9]See menu *Extensions→Analysis→Radial Pair Distribution function g(r)* but note that VMD does *not* read the simulation box from the XYZ file and hence cannot take periodic boundary conditions into account. You can provide this information with the sub-menu *Utilities: Set unit cell dimensions*

[10]We are choosing as energy unit kJ/mol and hence $k_B = 1.3806488 \times 10^{-23} \text{J} \cdot \text{K}^{-1} = 8.3144621\ \text{J} \cdot \text{mol}^{-1} \cdot \text{K}^{-1}$

simulation	N	τ (ps)	Δt (ps)
0	64	100	0.01
1	864	10	0.001
2	864	10	0.005
3	864	100	0.01
4	864	100	0.05
5	864	10	0.01
6	864	1	0.01
7*	512	100	0.01
8*	125	100	0.01
9*	864	100	0.001

Table 1. Simulations to be performed for the project. τ is the total simulation time, Δt is the time step, N the total number of atoms. *Simulations marked with an asterisk are optional.

- title, all authors on the team ("star" ("*") the person who wrote this report), abstract
- include the following sections: Introduction, Methods, Results and Discussion, Conclusions, Acknowledgements, References (see this handout for how to format references)
- appendix for any bonus work (but your report must be readable without it)
- any figures must be properly labeled (axes, units, individual lines distinguishable)
- In the Acknowledgements section mention any help you got from outside your team. Also mention briefly who in the team contributed to which part of the project. *The Acknowledgements section should be identical for all three team reports.* For instance,

  > "All authors designed the project together. M.N. wrote code (mdInit.py) to set up the system and contributed the force calculation and performed simulations. X.Y. wrote the MD code (mdLJ.py) and performed simulations. Q.Z. wrote the analysis code (LJanalysis.py), analyzed data together with M.N. and X.Y., and contributed the overlap detection routine in mdInit.py. All authors discussed the results."

  (Many journals require attributions of this kind. You don't have to follow the example exactly but you need to spell out everyone's major contributions to the success of the project.)

You might find it difficult to keep within the page limit. Try to be concise, combine multiple graphs into one, e.g. all per-particle energies (but make sure that each line is properly labeled). Graphs can be small but must still be readable.[11] You don't have to include all graphs for all simulations in the paper but there must be sufficient data shown to support your conclusions. For instance, you could show the energy time series for a typical and a extreme case and summarize the results by plotting the averages, standard deviations, and drifts from all simulations.

4 Code re-use and collaboration

You will carry out the project in teams except for the report, which must be written by each team member individually. In the authors list, add a star "*" to the person who wrote the report.

- You can use any code that was developed or provided during class.
- You are allowed to discuss the problem with other teams, and you are allowed to share individual pieces of code, *provided that each piece of code is attributed to the original author (use full names).* However, if more than 50% of code appear to be from other sources than the team, marks will be deducted. (Code from class until the project start date is exempt from the 50% rule.)
- In particular, you can use anything that has been discussed publicly in the **Midterm Wiki** on BlackBoard (go to *Tools* → *Wikis*: *Midterm Project 2013*) although attribution is still required.
- Copying text (report and code) verbatim from other sources without attribution constitutes plagiarism. The report should be in your own words but it is perfectly acceptable to cite other works instead of explaining in detail how, for instance, periodic boundary conditions are implemented.
- You can use the Acknowledgements section to highlight major external contributions (in addition to comments in the code).

References

1. Frenkel, D. and B. Smit, 2002. Understanding Molecular Simulations. Academic Press, San Diego, 2nd edn.

2. Allen, M. P. and D. J. Tildesley, 1987. Computer Simulations of Liquids. Oxford University Press, Oxford.

3. Leach, A. R., 1996. Molecular Modelling. Principles and Applications. Longman.

[11]Hint: Generate graphs in `matplotlib` at final size by using `plt.figure(figsize=(5, 5))`; font sizes can be changed with `import matplotlib; matplotlib.rc('font', size=8)`; consider plotting graphs with `linewidth=3` to make them better visible.

4. Adcock, S. A. and J. A. McCammon, 2006. Molecular dynamics: survey of methods for simulating the activity of proteins. *Chem Rev* 106:1589–615.

5. Rahman, A., 1964. Correlations in the motion of atoms in liquid argon. *Phys. Rev.* 136:405–411.

6. Martyna, G. J., M. E. Tuckerman, D. J. Tobias, and M. L. Klein, 1996. Explicit reversible integrators for extended systems dynamics. *Molecular Physics* 87:1117–1157.

7. Humphrey, W., A. Dalke, and K. Schulten, 1996. VMD – Visual Molecular Dynamics. *J. Mol. Graph.* 14:33–38.