TCCM: Distance Programming Project

August 2022

Jeremy HARVEY

Workbook and instructions

Master Studies in
Theoretical Chemistry and
Computational Modelling

TCCM: Distance Learning Programming Project 2022

TUTOR PRESENTATION – JEREMY HARVEY

Work address: Department of Chemistry, KU Leuven, Celestijnenlaan 200F, B-3001 Leuven, Belgium
ORCID 0000-0002-1728-1596, jeremy.harvey@kuleuven.be; https://chem.kuleuven.be/en/research/qcpc/tcc

ACADEMIC QUALIFICATIONS

Doctorate in Sciences 1995, Supervisor: Heinz G. Viehe, Thesis title: *Syntheses and reactions involving thionium and sulfonium ions. The effect of electronegativity on the stability of substituted alkanes;* **Licence in Philosophy** 1994; **Licence in Chemistry** 1990. All at UC Louvain, Belgium.

PRESENT AND PREVIOUS APPOINTMENTS

- **Professor of Quantum Chemistry**, KU Leuven (Oct 2014 –)
- **Professor of Chemistry**, University of Bristol (2008 – 2014)
- **Temporary Lecturer, Lecturer, then Reader** in Chemistry, University of Bristol (1999 – 2008)
- **Postdoc**, Hebrew University of Jerusalem, and UC Irvine, with Prof. R. B. Gerber (1997 –1998).
- **Postdoc**, TU Berlin, with Prof. Helmut Schwarz (1995 –1997).

RESEARCH SUMMARY

After his PhD devoted to studying organic reaction mechanisms using experimental methods, Jeremy Harvey moved into computational chemistry as a postdoc and as a faculty member. His research focuses on the use of state-of-the-art quantum chemical methods to yield insight into various chemical problems, working often in collaboration with experimental colleagues. Most of his research uses existing quantum-chemical and molecular dynamics codes, though his work typically also involves some development of new methodology.

He is best known internationally for his research on spin-forbidden chemical reactions, combining accurate state-of-the-art quantum chemistry and advanced statistical rate theories to gain a detailed understanding of such reactions. Another major area of research concerns computational studies of catalysis and biocatalysis, including organometallic chemistry and metalloenzymes. A third notable area is the study of the dynamics of complex reactions, both in the gas phase and in solution, using the empirical valence bond (EVB) method in order to efficiently describe chemical reactivity.

Dr. Harvey has published almost 300 papers with over 14,000 citations excluding self-citations (h-index: 71). He is regularly invited to speak in international conferences, and has received several awards including the 2006 Corday-Morgan Medal of the UK Royal Society of Chemistry and the 2009 Dirac Medal of the World Association of Theoretical and Computational Chemists (WATOC), which is awarded once per year worldwide to the outstanding theoretical and computational chemist below age 40. In 2022, he was elected to membership of the International Academy of Quantum Molecular Science (IAQMS).

POSITIONS OF RESPONSIBILITY

JNH is or has been a member of several editorial advisory boards (*Chemical Science*, *Theoretical Chemistry Accounts*, *Journal of Coordination Chemistry*), a referee for many journals, and referee or panel member for grant funding bodies. He is or has been a member of external review committees for among others the Chemistry Department at the Ecole Normale Supérieure, Paris, the Max Planck Institut für Kohlenforschung, Müllheim, and the Institute of Organic Chemistry and Biochemistry, Prague. He has organised several conferences and has served as examiner for doctoral degrees in multiple European countries. Since August 2020 he is chair of the Department of Chemistry at KU Leuven.

SELECTED PUBLICATIONS

1) *Mechanism of the highly effective peptide bond hydrolysis by MOF-808 catalyst under biologically relevant conditions*, D. Conic, K. Pierloot, T. N. Parac-Vogt and J. N. Harvey, *Phys. Chem., Chem. Phys.*, 2020, **22**, 25136.
2) *Computational insight into the origin of unexpected contrast in chiral markers as revealed by STM*, A. Sanz-Matias, O. Ivasenko, Y. Fang, S. De Feyter, K. Tahara, Y. Tobe and J. N. Harvey, *Nanoscale*, 2018, **10**, 1680-1694.
3) *The Dynamics of the Reaction of FeO^+ and H_2: A Model for Inorganic Oxidation*, S. Essafi, D. P. Tew and J. N. Harvey, *Angew. Chem. Int. Ed.*, 2017, **56**, 5790 – 5794.
4) Vibrational *relaxation and micro-solvation of DF after F-atom reactions in polar solvents*, G. T. Dunning, D. R. Glowacki, T. J. Preston, S. J. Greaves, G. M. Greetham, I. P. Clark, M. Towrie, J. N. Harvey and A. J. Orr-Ewing, *Science*, 2015, **347**, 530 – 533.
5) *Assembly-line Synthesis of Organic Molecules with Tailored Shapes,* M. Burns, S. Essafi, J. R. Bame, S. P. Bull, M. P. Webster, S. Balieu, J. W. Dale, C. P. Butts, J. N. Harvey and V. K. Aggarwal, *Nature*, 2014, **513**, 183 – 188.
6) *Understanding the determinants of selectivity in drug metabolism through modeling of dextromethorphan oxidation by cytochrome P450,* J. Oláh, A. J. Mulholland and J. N. Harvey, *Proc. Natl. Acad. Sci. USA,* 2011, **108**, 6050 – 6055.

Distance Learning Module: Coding of Monte Carlo Simulation of Simple Liquids: TCCM

Jeremy Harvey, August 2021

Overall Goal. The aim of this course is to improve your skills for coding relevant for various computational chemistry problems by writing a code for modelling simple atomic liquids using Monte Carlo simulation. During the class, you will learn or reinforce your knowledge of a programming language (Fortran, C, C++, or Julia, or…), and you will implement and test one or more programs for performing Monte Carlo simulation of liquid argon and silicon. For this exercise, the basic philosophy is that you should **write your own code from start to finish**, *i.e.* starting from nothing (**no copying** of even basic code from elsewhere), and that you should *progress* your skills. More experienced students should try to write more advanced code, but less experienced students can just write 'basic' code and still get a good grade.
The recommendation is that students should develop their code in four phases: (i) a code that can carry out Monte Carlo simulations for the Lennard-Jones liquid (a model of liquid argon). In a second phase (ii), this code should be made more efficient through the use of neighbour lists. The third problem to be addressed is (iii) simulation of liquid silicon using the Stillinger-Weber model potential. Finally (iv), this too should be made more efficient by using neighbour lists. All the required theory and all the key equations needed are provided as a document that is part of this manual. This document should be read with care, in particular because it also specifies more precisely exactly what the expected functionality of the code should be. Students who only accomplish the first one, two or three of the above coding tasks will be able to pass the unit, though with a lower grade than those who complete the whole set of tasks.

Working Method. This course is taught by distance learning, meaning you will be working on your own, with support through videoconferencing, an online exchange forum, and email. Like the other modules, the course is supposed to correspond to **6 ECTS**. One ECTS is supposed to correspond to something between 25 and 30 hours, so in principle, you can expect to invest **roughly** 150-180 hours on the course. This includes all reading, coding, videoconferencing, use of the forum, email. I know, this is a lot of time – roughly four full weeks of 40h/week or 20 days of 8h/day. Another way of measuring is that the total amount of work done should be about the same as for each of the other optional modules, or half the effort invested in the main Intensive Course. I suggest that it can be helpful to keep a **diary** listing the hours you have worked on the course, and a brief description of what you were doing in each case. In any case, I suggest you make a note every week of the progress you have made during that week. After an introductory videoconference, you will largely work on your own. However, I will be there to talk to you by videoconferencing if you hit problems that cannot be addressed in other ways. There will be occasional online code feedback/troubleshooting sessions.

I suggest that the first avenue for seeking help should be by asking questions addressed to the **Forum**, and all students are encouraged to participate actively by providing answers as well as questions in a constructive and supportive way. Be kind! Not everyone has the same knowledge, experience or insight. Also, be honest: there are no stupid questions. If you don't know or understand something, make an effort to find out or understand, but then if you still do not understand, ask a question – most likely the other students do not know either. I certainly will not give 'bad' grades to students who ask simple questions. Almost the opposite: **engagement with the forum** will be one (small) criterion for marking, meaning that non-participation will be considered a weak point. However, *quality* of contributions is much more important than *quantity*, and *respect for others* is also important: you should not aim to answer every question, leaving spaces for others is important also. A few thoughtful and useful contributions is much better than a flood.

Programming language and computing framework. The code can *a priori* be written in any programming language of your choice. However it needs to be rapidly testable on my computer, so in practice I recommend using one of either Fortran (90 or later), C, C++ or Julia. You may use another language if you really want to, but you should first consult with me. In case you use C++ you should avoid the use of unnecessary libraries, especially those with multiple dependencies that might be difficult to install. Again here, if in doubt, check with me. With Fortran or C you will typically not need to use any non-standard functions. I **do not recommend using a non-compiled programming language such as Python**, as it is difficult to make the code rapid enough to perform the intended simulations at an adequate speed with such languages.
You should program, compile and test code on machines available to you locally. It is anticipated that most development can be done on a laptop, but you are encouraged to test on at least one other machine so as to ensure that your code is not dependent on some particular feature of your local environment. We are not seeking to develop a very high-performance code, so there is no need for access to very high-performance clusters. There is no need for parallelization.

Personal work. The course relies in part on an honour code: the work you present should be **yours**, so you should be able to **fully explain every single line of code**. Of course, you will end up making extensive use of resources such as Stack Exchange. You may also end up getting suggestions from other students and tutors. This is normal and completely accepted. However, the code should ultimately be **yours**: you should be able to **fully explain all of it**. Note that in case more than one student from the same university takes the course, you should **not** turn this into a group project so your codes should end up being different (of course you can talk to each other). Too great similarity between the codes of students (whether in the same university or not) will be considered a negative point.

Assessment. At the end of the course, **your code should be handed in (= emailed to me)**: a code (iv above) for Monte Carlo simulation of liquid silicon using the Stillinger-Weber potential, that first simulates the liquid then outputs the radial distribution function to a text file that can then be plotted. In case this task is not completed, the assessment can instead be done for code that carries out one of the preparatory tasks (i) – (iii) mentioned above. Students should keep me informed if they feel they are not going to complete the full task. The course will be assessed based on the work during the working period, and on the code. You will be asked to **explain the code in a one-to-one video call** with me, at the end of the course. As well as from the forum and the coordinator and instructors, you may seek support from others and from your tutors. But remember: *every single line of code should be written by you and fully understood*.
The work in the course will receive a single grade expressed as a score out of 10, with 10 representing a perfect piece of work (quite rare), 9 being an excellent grade, 8 very good, 7 good, 6 acceptable, and 5 is the lowest pass grade. The mark will be awarded based on (a) engagement with the forum: did you participate with high-quality (not quantity!!) questions and answers? Were you respectful to other students? Did you demonstrate learning?; (b) mainly: the program produced: does it do what is asked? Can you **fully** explain it?

Timeframe. Given the lack of classroom sessions, I can in principle be flexible about the timeframe. However, the basic rule is: you should try to finish the module by the end of December 2022. You are **very** strongly encouraged to finish by the end of January 2023. Any student wishing to finish later should contact me and explain their planned timetable. The aim is that you fit the work on this module around work you do for other modules, and work on your thesis.

TCCM: Distance Learning Unit
Notes on Programming Languages

Introduction

In this course, one of the main outcomes is writing some of your own code, in Fortran, C++ and/or another language, perhaps Julia. The intention is to build a code starting from zero, so you also need to conceive the architecture of the code yourself. These notes provide a very short summary concerning programming in Fortran, C++, Julia (and Python). On the next pages, there are some quite general tips about how to tackle the larger task of writing Monte Carlo simulation code.

Programming Language

There are many different programming languages. These are of variable levels of usefulness for carrying out numerical computations relevant to chemistry. One of the most commonly used languages nowadays is Python. The versatility and ease of use of this language make it very attractive. For many programming tasks where efficiency is not critical, it is an excellent choice: modern computers are so fast that there are a huge number of problems which are too demanding to be solved numerically using paper and pencil or semi-automated techniques like spreadsheets, but which upon suitable coding become trivial to deal with using even very modest computer resources. Hence Python is probably the most commonly used language in scientific programming nowadays. However, in its native form it is rather inefficient for heavy numerical computations. In some cases, the high-level tasks can be written in Python, and numerically intensive tasks can be performed by executing calls to functions written in standard libraries such as NumPy, and itis possible in this way to write efficient code. For the present problem, though, the computationally demanding tasks cannot easily be separated like this. For this reason, **I do not recommend using Python for the present course**. Students who wish to do so and who feel they have the expertise to write efficient enough code in Python should consult with me prior to committing to this approach.

Three other programming languages have been used very heavily in scientific computing in chemistry and other fields: Fortran, C and C++. These are compiled languages, where a text **source code** is written, then **compiled** to yield a (binary) executable code. Provided the algorithm of interest is implemented in a reasonable way, Fortran, C or C++ yield quite efficient code. There is a lot of knowledge about how to obtain very efficient code, but this is not the focus of the present exercise.

You should pick a language for your coding yourself. My suggestion is one of C, C++, Fortran (in its modern dialects Fortran 90 or later, **not** Fortran 77), or Julia (see https://julialang.org/). You can also pick another language if you wish, but please in that case consult with me first. There are **many** tutorials available for programming in different languages, available online. Hence I do not attempt here to provide any detailed introduction to programming.

Distance Learning Module
TCCM, J Harvey, August 2022

Some Tips on Programming

A medium-sized programming project like the one involved in this course can seem a bit overwhelming at first sight, since you need to learn the programming language, to understand the problem, *and* design then write a program to carry out the calculations. Fortunately, the task is not *too* large – my own code, that has all the required functionality, has just 532 lines of Julia code. Still, this is not nothing, and some careful strategy is needed in order to approach the task in the best way (or at least in a reasonably good way – there are many good approaches, but there are even more bad ones). Here I give some tips that I find helpful.

1. **Small Steps**
   The best way to keep up momentum and insight for a programming project, especially at the beginning, is to make *small steps*. Write your first code – something like "Hello, World", compile it, then check that it works. If it does, great!! Otherwise see points 7 and 8, below. Once it works, think about the next step. You might try to write a code that reads in two numbers and prints out their sum. Write that, compile it, check it. Then maybe write a code that reads in (*x*,*y*,*z*) coordinates for two points in space, calculates the distance between them, then writes it out. Compile that, check it. Then perhaps extend this so that it computes the Lennard-Jones potential for the two atoms. Then your next code should perhaps do the same task but for a collection of ten atoms, summing up the pairwise Lennard-Jones terms between each of the unique pairs of atoms. With these small tasks, each of which could **take less than an hour**, you can build up your confidence and your skills. If you instead set yourself a big goal, you may find it hard to work out how to start, and end up losing a lot of time. My very rough rule of thumb is that when starting out you should have a successful compile-test step at least **every two hours or so**. If you spend more than this between successful compile-test steps, you are trying to make too big steps.

2. **Keep your work**
   I recommend frequently storing examples of successful code. This can be formalized using tools such as github, but at this stage, you can do it almost as well by just making frequent copies of your code (every time something works, keep it, and make subsequent changes in a new copy of the source file), making frequent new directories, giving source code files informative names, and *not throwing files away or overwriting them*.

3. **Plan**
   For each piece of code, write down (or think through carefully) what you want it to do, then sketch (in your head or on paper) the steps it needs to take. This can be formalized by writing down a logic flowchart for your algorithm (see https://en.wikipedia.org/wiki/Flowchart) but equally you can just keep it simple as a set of bullet points or even a mental plan. Another thing to think about is to list the variables and arrays that you will need to use in your code.

4. **Modularize**
   Remember that the computer is there to help you. If you find yourself typing the same thing multiple times, consider that perhaps you should write a loop, a function, or a subroutine, so as to streamline the algorithm. Code that is a bit modular in this way will be easier to read by others (and yourself). It may also be more efficient.

5. **Keep it simple**
   It is very tempting to go overboard in terms of modularizing, making everything very beautiful in terms of coding technique. But this course has a fairly modest goal, and in practice, it is unlikely that the code you write is going to be used and extended by other users and programmers – so remember that you do not need to do things perfectly. For example, the code needs some flexibility (e.g. for the Lennard-Jones example you should be able to vary the number density when executing the program), but it does not require flexibility in every respect. For example, the Lennard-Jones potential should be expressed in reduced units, and there is no need to allow for the use of other units. Another example: you should use simple cubic periodic boundary conditions. There is no need to plan for using a rhombic octahedron or other more complicated periodic models. Each year, in this module, some students make the mistake of neglecting this principle and end up investing way too much time in the course – remember, there are other courses to follow, the masters thesis to do, and so on…

6. **Learn the basics**
   Every programming language provides tools to do a number of basic things. Make sure you have learned about these concepts in tutorials or books: variables, types and declarations; arrays; loops; flow control ("if" statements); functions and subroutines, including *intrinsic* functions. For compiling, you may also want to learn about compiler flags and code optimization, and about the use of makefiles. Do a bit of reading on the important topic of generating random numbers.

7. **Embrace debugging**
   Debugging is an essential part of coding and has been ever since the early days (see https://www.computerhistory.org/tdih/september/9/). In fact, despite all the accumulated expertise related to programming techniques, in many (or most!) projects, more time will be spent on debugging code than on writing new code. So don't get frustrated if your code does not work the first time, and instead learn to enjoy the challenge! There are many ways to debug, but they all rely on the same basic techniques: looking at what has changed since your code last worked, checking that the code actually does what you intend it to do (it can be very instructive simply to check that variables contain the things that you think they should, for which purpose you can add lots of 'print' statements that output variables at key points in the code; you can also use an integrated development environment that allows you to check the content of variables during execution), and that it flows in the way you think it should (a print statement saying "I just entered function xxxxx" can be really revealing, if it turns out that your code never actually does enter that function….). Some people prefer to use IDEs (https://en.wikipedia.org/wiki/Integrated_development_environment) for code development; these do indeed streamline some of the debugging procedure, so they can be quite helpful (though for a medium project like this one, my view is that they are not *necessary*). If you cannot solve a bug after some period of time, consider that you may have done something conceptually wrong, so look back at tip 3 on planning.

8. **Ask for help, and give it.**
   If you don't know how to solve a given problem, ask! It could be a fellow-student, or a professor, or the forum. You can in principle post anonymously on the forum, but there is little point because the aim of the forum is not to penalize ignorance at the start of the course, but instead to help everyone make progress. Also, if someone gives outstanding advice to you, don't hesitate to also put it on the forum and thereby share it with others.

Writing a Code to Model Atomic Liquids Through Monte Carlo Simulation

Jeremy Harvey

August 24, 2022

Abstract

This document lays out the basic background needed to construct a simple Monte Carlo simulation code for atomic liquids, using both the simple Lennard-Jones model for non-bonded liquids (e.g. liquid argon) and the Stillinger model for liquids involving directional bonding, such as liquid silicon. To speed up simulation, a neighbour list will be used. The structure of the liquids will be probed by computing the radial distribution function $g(r)$. All the basic equations needed to perform the corresponding coding are provided for both cases.

1 Monte Carlo Simulations

Monte Carlo simulations provide a straightforward way to generate ensembles of structures for a given system at a given temperature T. The basic idea is to make random trial changes in structure, and to accept or reject these changes using as criterion the Boltzmann distribution for the corresponding temperature. Consider two structures $\mathbf{r}^{(i)}$ and $\mathbf{r}^{(j)}$ of a system with N atoms. We use here the vector notation $\mathbf{r}^{(i)}$ to refer to a particular structure adopted by the system, comprising all $3N$ coordinates of the particles $1 \ldots N$ in the system for that given structure, i.e. $\mathbf{r}^{(i)} \equiv \{\mathbf{r}_1^{(i)}, \mathbf{r}_2^{(i)}, \ldots \mathbf{r}_N^{(i)}\}$. In this notation, $\mathbf{r}_k^{(i)}$ refers to the three Cartesian coordinates of the k-th atom for the i-th structure of the whole system. At a given temperature T, the relative probability of observing the two structures should be given by:

$$\frac{\mathcal{P}(\mathbf{r}^{(i)})}{\mathcal{P}(\mathbf{r}^{(j)})} = \exp\left(-\frac{V(\mathbf{r}^{(i)}) - V(\mathbf{r}^{(j)})}{k_B T}\right) \tag{1}$$

Where $V(\mathbf{r}^{(i)})$ is the potential energy for that structure. The above relation can be satisfied by choosing a method to update the structure of the system such that the ratio of the probabilities for changing from structure $\mathbf{r}^{(i)}$ to structure $\mathbf{r}^{(j)}$, $\mathcal{P}(\mathbf{r}^{(i)} \to \mathbf{r}^{(j)})$ and for the reverse change from $\mathbf{r}^{(j)}$ to structure $\mathbf{r}^{(i)}$ is also given by the Boltzmann factor:

$$\frac{\mathcal{P}(\mathbf{r}^{(j)} \to \mathbf{r}^{(i)})}{\mathcal{P}(\mathbf{r}^{(i)} \to \mathbf{r}^{(j)})} = \exp\left(-\frac{V(\mathbf{r}^{(i)}) - V(\mathbf{r}^{(j)})}{k_B T}\right) \tag{2}$$

The Monte Carlo method relies on updating the structure of the system being simulated in a way that satisfies the condition of eq. 2. Specifically, in your code, you will implement the method as described here:

(a) Choose an initial structure $\mathbf{r}^{(0)}$ (the way to do this will be described below), then repeatedly:

(b) Generate a random change in the structure, $\Delta\mathbf{r}$. In the jargon of Monte Carlo simulation, this is called a **trial move**. Generating the trial move can in principle be done in many different ways, for example it can involve a change in position of all atoms in the system, of several atoms or of just one atom. In most cases, the most efficient approach involves a displacement of just *one* atom and this is what your code should do in the present case. While many procedures to generate the trial moves can be used, they must satisfy the following important requirement: the probability of generating a given trial move that would convert $\mathbf{r}^{(i)}$ into $\mathbf{r}^{(j)}$ must be the same as the probability of generating the reverse trial move, that would convert $\mathbf{r}^{(j)}$ into $\mathbf{r}^{(i)}$.

(c) Calculate the change in potential energy of the system associated with the change, $\Delta V(\Delta\mathbf{r})$. It is the *change* in energy that is important – the algorithm does not require the potential energy corresponding to the initial structure $V(\mathbf{r}^{(i)})$ or of the modified structure, $V(\mathbf{r}^{(i)} + \Delta\mathbf{r})$, and it may be more efficient to calculate the difference rather than both overall energies.

(d) Decide whether or not to accept the change in structure, based on the value of the energy change that would result from this change, using the following rules:

   i) If $\Delta V(\Delta\mathbf{r})$ is negative or zero, the change should be accepted.

   ii) If $\Delta V(\Delta\mathbf{r})$ is positive, then one should compute the Boltzmann factor $\alpha = \mathcal{P}(\Delta\mathbf{r}) = \exp(-\Delta V(\Delta\mathbf{r})/k_B T)$. This is close to 1 if the energy change is small (or the temperature high), and close to 0 for larger energy changes or lower temperature.

   iii) One then chooses a random number w from a uniform distribution between 0 and 1.

   iv) If $\mathcal{P}(\Delta\mathbf{r}) \geq w$ then the change in structure should be accepted. This is more likely to occur if the Boltzmann factor is close to 1.

   v) If however $\mathcal{P}(\Delta\mathbf{r}) < w$ then the change in structure should be rejected.

   Note that the procedure described here for deciding whether to accept the move or not means that the probability of *accepting* a given trial move is certainly **not** symmetric. Instead, it obeys eq. 2. In point (b) above, it is specified that the procedure for *generating* trial moves must be symmetric – but the procedure for accepting them is not.

(e) If the change is accepted, the structure should be updated, with $\mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} + \Delta\mathbf{r}$. If the change is rejected, one keeps the current structure, $\mathbf{r}^{(i+1)} = \mathbf{r}^{(i)}$.

As mentioned above, the steps (b) to (e) need to be repeated many times, after which the series of structures $\{\mathbf{r}^{(i)}\}$ with $i = 1, \ldots n_{\text{step}}$ can be analyzed and used to make predictions concerning the properties of the system. In case the initial structure $\mathbf{r}^{(0)}$ is a very unlikely structure, it may be wise to discard some initial proportion of structures $\mathbf{r}^{(0)} \ldots \mathbf{r}^{(n_{\text{eq}})}$ of the sampled structures, in order to perform an *equilibration* of the system, allowing it to relax to a more typical structure. The number of steps n_{eq} needed for this typically needs to be chosen by trial and error. Given that in most cases only one atom moves in a given trial move, on average, after having considered N trial moves (this is sometimes call a 'sweep' of trial moves), then a proportion $\overline{\alpha}$ of the atoms will have changed position, where $\overline{\alpha}$ is the mean *success rate* of the trial moves, and lies between 0 and 1. In general, in order to obtain an efficient Monte

Carlo simulation, the trial moves need to be selected in such a way that $\overline{\alpha}$ is not too small (otherwise, very few changes in structure are performed), but $\overline{\alpha}$ should not be too large either, as that would indicate that the trial moves being considered are not large enough, so one is not rapidly covering the range of possible structures. As a ballpark, a value of $\overline{\alpha}$ of the order of 1/3 to 1/2 will represent the best compromise between these two requirements.

For simple atomistic systems like those treated here, the trial move $\Delta\mathbf{r}$ is usually selected by picking four random numbers, k, and w_x, w_y and w_z.[1] k is an integer between 1 and N, and the factors w_ζ are evenly distributed between 0 and 1. Atom k is moved, with its x, y and z coordinates being changed by respectively $2\Delta r_{\max}(w_x - 1/2)$, $2\Delta r_{\max}(w_y - 1/2)$ and $2\Delta r_{\max}(w_z - 1/2)$, i.e. by up to $\Delta r_{\max}$ in either direction. The scalar quantity $\Delta r_{\max}$ defines the maximum change that can occur for each coordinate. Choosing a larger value will typically cause $\overline{\alpha}$ to be smaller, while a smaller value of $\Delta r_{\max}$ will typically lead to a larger value of $\overline{\alpha}$. The procedure described here trivially satisfies the requirement for the probability of a move $\Delta\mathbf{r}$ to be equal to that for the reverse move $-\Delta\mathbf{r}$, since movements in each direction are equally likely to be generated.

2 Periodic Boundary Conditions

The present exercise aims at simulating atomic liquids. Liquids have no structure on longer lengthscales, but at short range, the atom-atom interactions lead to noticeable structure. To be meaningful, simulations must be performed on systems with sufficient atoms such that the size of the system is larger than the lengthscale on which the structure manifests itself. For liquids, this is typically some small multiple of the typical nearest-neighbour atom distance, so some tens of Å. In principle, the system to be modelled could therefore be a sphere with diameter of this order of magnitude. However, in the case of a spherical system, unless it is quite large (diameter above 100 Å), most of its atoms will be at the surface, which is not advantageous because surface atoms tend to behave differently from those in the bulk. Accordingly, many simulations, including those that you will perform with your code, use so-called **periodic boundary conditions** with the **minimum-image convention**, as illustrated in fig. 1. The idea is as follows: the simulation system comprising N atoms (here, four atoms, labelled $A-D$) is surrounded by an infinite number of replicas of itself in each direction. In the simplest case, shown in fig. 1, each replica has a square or cubic shape with length L (there are 8 close neighbours in two dimensions, but 26 in three dimensions), though more complex periodic shapes are also possible. The length of the repeating cell L must be chosen such that it is significantly larger than the characteristic lengthscale of the problem at hand – so in our case, L must be at least a few tens of Å.

[1]The art of generating random numbers in a computer is one which plays a very important role in Monte Carlo simulations. There are a number of more or less sophisticated algorithms available to generate *pseudo*-random numbers (numbers that appear to be random but are generated in a deterministic way). Many of these algorithms can be initiated with a *seed* – a starting parameter which, if if the same value is used twice when running the code twice, will lead to *the same* sequence of 'random' numbers being generated. This can be useful for debugging purposes. For production, the seed can itself be chosen randomly or can be taken from some extrinsic property such as the clock time, so that each run generates independent results. Some of the simpler algorithms for random number generation yield results that are un-random enough that artefacts can result in the code using them; examples in the case of molecular Monte Carlo simulation are described in Ghersi, Parakh and Mezei, *J. Comput. Chem.*, 2017, **38**, 2713 – 2720 (`https://doi.org/10.1002/jcc.25065`), and in the papers referred to in that work. The standard random number generators available with the gfortran Fortran compiler and the Julia programming language are however reported to yield no noticeable artefacts.

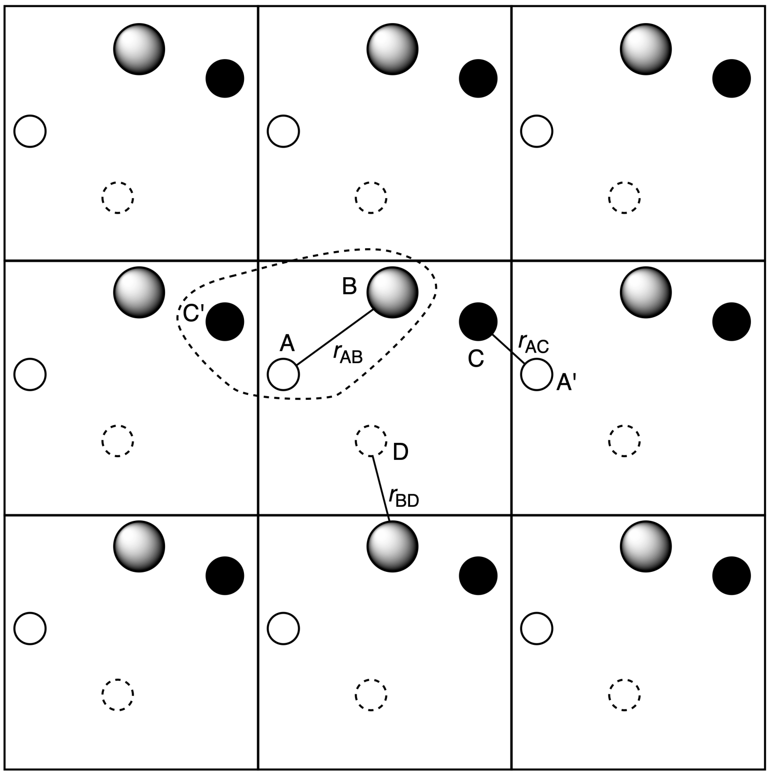

Figure 1: Schematic representation of periodic boundary conditions and the minimum-image convention in two dimensions.

In order to calculate the energy of the system, one effectively needs to take into account interactions between *all* pairs of atoms in the infinite system formed by all of the replicas. For some types of interaction, especially Coulombic interactions, this requires sophisticated approaches relying on Fourier transforms in order to evaluate the infinite number of terms that arise. Here, however, we will only be considering quite short-ranged interactions, and they can be treated using cut-offs and the *minimum-image convention*. The use of cut-offs implies that atoms are considered to interact with one another, i.e. to contribute to the overall energy expression, only if their distance is smaller than some cut-off distance r_c. This can be illustrated in terms of the energy expression of eq. 3, which includes two-body and three-body terms $V^{(2)}$ and $V^{(3)}$ (extrapolation to more complicated energy expressions is straightforward but not needed for this exercise). The two-body term for a given pair of atoms depends only on the distance r_{ij} between them, while the three-body term as written here depends on the distances r_{ij} and r_{ik} (in general the three-body term could also depend on the distance r_{jk} but this is assumed not to be the case in eq. 3). Coupled to the use of cut-offs is the fact that in the minimum-image convention, when considering the $V^{(2)}$ contribution of two atoms (e.g. atoms A and C in fig. 1, one chooses only the interaction between the two *closest* atoms chosen with one atom in the central periodic image. As shown in the figure, for r_{AB}, the shortest A–B distance involves two atoms both within the central image, but for r_{AC}, a shorter distance is obtained when atom C is in the central image, while the periodic image A' is considered for atom A. Likewise, the three-body term $V^{(3)}_{ABC}$ for this hypothetical system would involve considering the circled trio of atoms C', A and B, as these yield the shortest distances r_{AB} and r_{AC}.

$$\left\{\begin{array}{l} V = \sum_{i,j} V_{ij}^{(2)}(\mathbf{r}_i, \mathbf{r}_j) + \sum_{i,j,k} V_{ijk}^{(3)}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k) \\ |r_{ij}| > r_c \Rightarrow V_{ij}^{(2)} = 0 \\ |r_{ij}| > r_c \quad \text{or} \quad |r_{ik}| > r_c \Rightarrow V_{ijk}^{(3)} = 0 \end{array}\right. \tag{3}$$

Periodic boundary conditions with the minimum-image convention can be easily implemented in code: when computing the energy for a normal finite system, one typically needs to refer to a set of interatomic vectors $\mathbf{r}_{ij}$, which can be obtained from the Cartesian coordinates as shown in eq. 4. In the minimum-image convention, one still considers each pair of atoms (i, j) within the central image, but $\mathbf{r}_{ij}$ is replaced by the corresponding minimum-image convention vector $\mathbf{r}_{ij}^{\mathrm{MI}}$. In the simplest case of a cubic periodic image, which you will use here, this vector is constructed as shown in eq. 5. In this equation, n_x, n_y and n_z are integers, chosen so that each of the quantities $\alpha_j - \alpha_i + n_\alpha L$ $(\alpha = x, y, z)$ falls in the interval between $-L/2$ and $L/2$.

$$\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i = \left\{\begin{array}{c} \Delta x_{ij} \\ \Delta y_{ij} \\ \Delta z_{ij} \end{array}\right\} = \left\{\begin{array}{c} x_j - x_i \\ y_j - y_i \\ z_j - z_i \end{array}\right\} \tag{4}$$

$$\mathbf{r}_{ij}^{\mathrm{MI}} = (\mathbf{r}_j - \mathbf{r}_i)^{\mathrm{MI}} = \left\{\begin{array}{c} \Delta x_{ij}^{\mathrm{MI}} \\ \Delta y_{ij}^{\mathrm{MI}} \\ \Delta z_{ij}^{\mathrm{MI}} \end{array}\right\} = \left\{\begin{array}{c} x_j - x_i + n_x L \\ y_j - y_i + n_y L \\ z_j - z_i + n_z L \end{array}\right\} \tag{5}$$

In a simulation using periodic boundary conditions, one often makes the choice that if an atom i moves "out of" the primary simulation image, i.e. if one of its Cartesian coordinates $\alpha_i > L$ or $\alpha_i < 0$ (or equivalently if $\alpha_i > L/2$ or $\alpha_i < -L/2$), then it is moved "back into the box" on the "other side" by replacing α_i by $\alpha_i + L$ or $\alpha_i - L$ as needed, so that one always has $0 \leq \alpha_i \leq L$ (or $-L/2 \leq \alpha_i \leq L/2$). This makes visualization easier, but is not strictly necessary provided that eq. 5 is applied, i.e. provided that whenever $\mathbf{r}_{ij}$ (or the distance r_{ij}) appears in the energy expression, it is replaced by $\mathbf{r}_{ij}^{\mathrm{MI}}$ (or the corresponding distance r_{ij}^{MI}). Certainly the choice of whether to enforce $0 \leq \alpha_i \leq L$ or an equivalent convention whereby $-L/2 \leq \alpha_i \leq L/2$ is entirely a matter of personal preference.

3 Potential Energy Models for Atomic Liquids

For any system to be modelled, one must first choose a method to compute the potential energy for a given set of atomic coordinates. This can be done with quantum-chemical methods such as density functional theory. Another increasingly popular choice is a machine-learning method trained using quantum-chemical methods. In this work, which only addresses some quite simple systems, we will use simple empirical expressions (primitive molecular mechanics forcefields) because we want to be able to carry out calculations using very limited computational resources. For the systems we will study, the expressions we will use actually give quite good results – it is not always the case that more expensive computations yield better results.

3.1 The Lennard-Jones Fluid

The first system to be modelled here is a model for liquids comprised of rare gas atoms such as argon. The potential energy for such systems can be calculated with various *ab initio* methods, and it is found that the overall energy can be expressed quite accurately as a sum of two-body terms ($V^{(2)}$ in eq. 3). Very accurate results do require three-body correction terms, but these will be neglected here. It is found that the standard Lennard-Jones expression, eq. 6 and fig. 2, provides a fairly accurate model for $V^{(2)}$:

$$V_{ij}^{\mathrm{LJ}} = 4\epsilon \left\{ \left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right\} \tag{6}$$

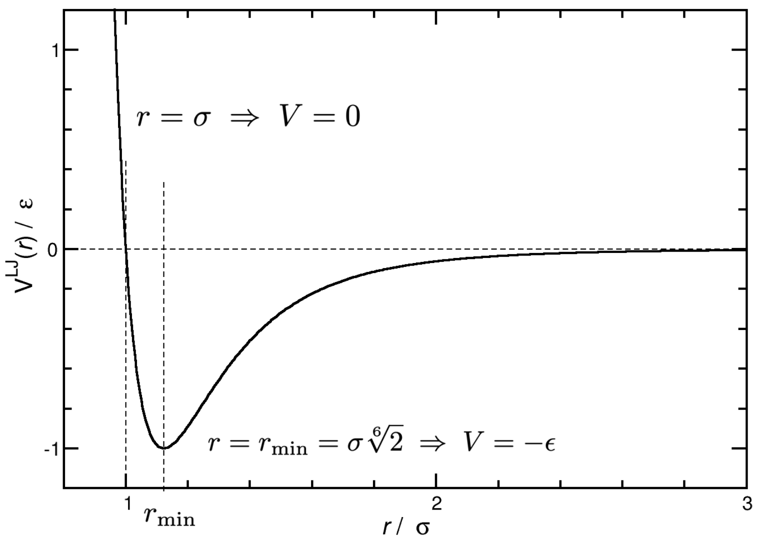

Figure 2: The two-body Lennard-Jones potential.

For a system with many argon atoms, one then has the following expression (where the distances r_{ij} should be interpreted as minimum-image convention distances r_{ij}^{MI}, the length of the vector $\mathbf{r}_{ij}^{\mathrm{MI}}$ of eq. 5 if one is using periodic boundary conditions):

$$V^{\mathrm{LJ}} = 4\epsilon \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left\{ \left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right\} \tag{7}$$

The parameters ϵ (well depth) and σ (radius of the "hard" core of the atom) can be given physical values. For example, for argon, values of 3.4 Å (σ) and 1.0 kJ mol^{-1} (ϵ) can be used. However, for the purposes of running simulations, there is no need to assign specific values to these parameters, one can instead run the simulations using so-called *reduced variables*, whereby the unit of energy is ϵ, the unit of distance σ, the unit of mass is m, the mass of the particles, and the unit of temperature is such that the Boltzmann constant $k_B = 1$. By making these choices one implicitly also chooses reduced units for other quantities. The density ρ simply becomes the same as the number density $\rho = N/V$, and the volume of the simulation cell V becomes $V' = V/\sigma^3$, so that the number density in reduced units can be expressed as $\rho^* = \rho\sigma^3$. To take an example, considering argon at its boiling point (85.5 K in SI units), and using a value of ϵ of 1.0 kJ mol^{-1}, this means that in reduced units, the boiling point $T_b^* = k_B T/\epsilon = 0.727$ in reduced units.

As mentioned above, in periodic boundary conditions, one needs to use a cut-off for the potential. The Lennard-Jones potential of eqs. 6 and 7 falls off rapidly for larger r_{ij}, so using a cut-off is not a problem. For many purposes, using $r_c = 2.5\sigma$ is sufficient and this value (or a slightly higher one, between $r_c = 3\sigma$ and 4σ) can be used here.

The Lennard-Jones potential of eq. 7 is exactly additive and involves only two-body terms. This means that when computing the change in energy for a trial move, $\Delta V(\Delta\mathbf{r})$, it is not necessary to compute the whole potential. If atom k is moved from $\mathbf{r}_k^{\text{init}}$ to $\mathbf{r}_k^{\text{trial}}$, then only the terms involving that atom need to be considered, yielding eq. 8 for $\Delta V(\Delta\mathbf{r})$ (using reduced variables). The single summation in this equation, that replaces the double sum in eq. 7, makes it much faster to evaluate.

$$\Delta V(\Delta\mathbf{r}_k) = 4\sum_{i=1;i\neq k}^{N}\left\{\left(\frac{1}{r_{ik}^{\text{trial}}}\right)^{12} - \left(\frac{1}{r_{ik}^{\text{trial}}}\right)^{6}\right\} - \left\{\left(\frac{1}{r_{ik}^{\text{init}}}\right)^{12} - \left(\frac{1}{r_{ik}^{\text{init}}}\right)^{6}\right\} \tag{8}$$

When performing Monte-Carlo simulations on the Lennard-Jones fluid, several suitable choices for the initial structure $\mathbf{r}^{(0)}$ can be made. One option is simply to place the atoms at random in the periodic box, i.e. one generates N triplets (m_x, m_y, m_z) of random numbers m uniformly distributed between 0 and 1, and then chooses $\mathbf{r}_k^{(0)} = (m_xL, m_yL, m_zL)$ for each of the atoms. This approach works well at low density, but at higher density, the chance that two atoms will be very close to one another increases, and this means the initial structure will be very high in potential energy. This can cause the simulation to take some time to equilibrate. A modified option is to generate coordinates for the atoms one at a time. The first atom is placed randomly as above, then for the following atoms random positions are proposed, but before accepting them, a test is made to check what is the shortest distance to a previously positioned atom. If this shortest distance falls below some threshold r_{min} (which could be of the order of 0.9 in reduced units), this proposed position is rejected and a new one is generated. Again this approach works well at lower density, but for higher density (especially if r_{min} is large), one may need many proposed structures for the later atoms in order to find a suitable position. A third option is to place M^3 atoms on a cubic grid, where M is the first integer larger than $\sqrt[3]{N}$. The atoms are placed on a cubic grid with dimension $M \times M \times M$, at $(L/M, L/M, L/M)$, $(L/M, L/M, 2L/M)$ and so on – and then $M^3 - N$ unneeded atoms can be selected at random and deleted.

3.2 The Stillinger Model for Silicon

In order to describe more complex liquids, a more complicated energy expression than the simple pairwise expression of eq. 7 is needed. For *molecular* liquids (such as water or common organic solvents), a forcefield expression can be used, with interactions within each solvent molecule described by stretching, bending and torsion terms, and intermolecular interactions described by charge-charge and Lennard-Jones terms. Such systems can also be described by *ab initio* potentials (using e.g. 'first principles' DFT approaches) or using a machine-learning expression trained to reproduce DFT values. For the system to be described here, liquid silicon, and given the resources we wish to use, we need to choose the method carefully. The liquid only contains one type of atom, but it also contains atom-atom bonds which constantly form and break again as the atoms move within the liquid. These bonds impart three-body relative orientation preferences, and therefore neither the pairwise form of eq. 7 nor forcefield expressions (which require an *a priori* assignment of chemical bonds) are suitable. An *ab initio* potential would be too computationally demanding for our purposes, and generating a machine-learning potential goes beyond the scope of this work. Instead, we will use a sort of reactive forcefield potential incorporating three-body effects, which has been shown to yield quite good results at low computational expense. Specfically you will use the so-called 'Stillinger' model (described by Frank Stillinger and Thomas Weber, *Phys. Rev. B*, 1985, **31**, 5262 – 5271). The general expression for this potential is as follows (where r_{ij} is the distance between atoms i and j, and $\mathbf{r}_{ij}$ is the vector going from the position of atom i to the position of atom j, in both cases using periodic boundary conditions and the nearest-neighbour approximation):

$$V^{\text{Stillinger}} = V_2 + V_3 = \epsilon \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} f_2\left(\frac{r_{ij}}{\sigma}\right) + \epsilon \sum_{i=1}^{N-2} \sum_{j=i+1}^{N-1} \sum_{k=j+1}^{N} f_3\left(\frac{\mathbf{r}_{ij}}{\sigma}, \frac{\mathbf{r}_{ik}}{\sigma}, \frac{\mathbf{r}_{jk}}{\sigma}\right) \tag{9}$$

As for the Lennard-Jones potential, this functional form lends itself well to the use of reduced variables for energy and distance, so that in practice the ϵ and σ terms will be omitted (and you should use $T* = k_BT/\epsilon$. In the Stillinger potential, both the two- and the three-body terms are chosen to vanish exactly for larger values of r_{ij}, r_{ik} and r_{jk}. In the paper by Stillinger and Weber, the value of r at which this occurs (which functions as the cutoff distance r_c) is referred to as a. The specific functional form chosen for the two-body term is written as follows in reduced units:

$$f_2(r) = \begin{cases} A(Br^{-4} - 1)\exp\left(\frac{1}{r-a}\right) & \text{for} \quad r < a \\ 0 \quad \text{for} \quad r \geq a \end{cases} \tag{10}$$

The three-body term in reduced units can be written as (where r_{ij} is the distance between atoms i and j, and θ_{jik} is the angle formed between the vectors $\mathbf{r}_{ij}$ and $\mathbf{r}_{ik}$):

$$f_3(\mathbf{r}_{ij}, \mathbf{r}_{ik}, \mathbf{r}_{jk}) = h(r_{ij}, r_{ik}, \theta_{jik}) + h(r_{ji}, r_{jk}, \theta_{ijk}) + h(r_{ki}, r_{kj}, \theta_{ikj}) \tag{11}$$

In turn, the function $h(r_{ij}, r_{ik}, \theta_{jik})$ is defined by:

$$h(r_{ij}, r_{ik}, \theta_{jik}) = \lambda \exp\left(\frac{\gamma}{r_{ij} - a} + \frac{\gamma}{r_{ik} - a}\right)\left(\cos\theta_{jik} + \frac{1}{3}\right)^2 \quad \text{if} \quad r_{ij}, r_{ik} < a \tag{12}$$

(if either distance is larger or equal to the cut-off, this function is zero). It can be seen that this term favours a 'tetrahedral' angle of 109.5°, since $\cos 109.5° = -1/3$, so that the term $(\cos\theta_{jik} + 1/3)$ is zero when the angle at atom i adopts this value.

When choosing the parameters, Stillinger and Weber chose to require that $r_{\min}$ should be equal to $\sqrt[6]{2}$, so that $f_2(\sqrt[6]{2}) = -1$, as in the Lennard-Jones potential. Other parameters were loosely optimized under this constraint so as to reproduce a number of known properties of silicon. The resulting parameters that need to be used in the above equations are:

$$A = 7.049\,556\,277 \quad ; \quad B = 0.602\,224\,558\,4 \quad ; \quad a = 1.80 \quad ; \quad \lambda = 21.0 \quad ; \quad \gamma = 1.20 \tag{13}$$

For the purposes of coding, specially when using a neighbour list in order to avoid having to run computationally expensive loops over all triples of atoms (see below), it is convenient to re-write eqs. 9 and 11 for the three-body term as follows (in reduced units):

$$V_3 = \sum_{i=1}^{N} \sum_{j \neq i}^{N} \sum_{k \neq i, k > j}^{N} h(r_{ij}, r_{ik}, \theta_{jik}) \tag{14}$$

In the section on the Lennard-Jones potential, I described how it is never necessary to evaluate the whole potential when considering a trial move, since only the terms involving the moved atom need to be computed for the starting and trial structures (see eq. 8). A similar trick can be used for the Stillinger potential: when moving atom m, one can obtain the change in potential energy by computing the f_2 term in both structures for all (i, j) pairs where $m = i$ or $m = j$. For the three-body potential of eq. 14, one needs to take into account that moving a given atom m can affect each $h(r_{ij}, r_{ik}, \theta_{jik})$ term for which $m = i$, $m = j$ or $m = k$. Doing this properly requires some care! A description of how to do so is provided in section 5 below.

The initial structure $\mathbf{r}^{(0)}$ for the Stillinger model can be generated using the same methods as for the Lennard-Jones model, as described above. Another option is to use a initial structure that matches the crystal structure, although with multiple unit cells. Silicon has the same type of structure as diamond, with a unit cell containing eight atoms, positioned at fractional coordinates $(0, 0, 0)$, $(0.5, 0.5, 0)$, $(0.5, 0, 0.5)$, $(0, 0.5, 0.5)$, $(0.25, 0.25, 0.25)$, $(0.25, 0.75, 0.75)$, $(0.75, 0.75, 0.25)$, and $(0.75, 0.25, 0.75)$ in the unit cell, with the unit cell length of 5.431 Å. Taking three such unit cells in each of the three x, y and z directions yields a system with 216 atoms and a density of 0.0499 atoms per Å^3. In reduced units, as already mentioned, the minimum of the pair potential lies at $r = \sqrt[6]{2}$ and this is the nearest-neighbour Si–Si distance in the crystal (the three-body term is zero for the crystal, and all non-nearest-neighbour two-body terms also vanish). Based on the observed Si–Si distance in the crystal, this implies that $\sigma = 2.0951$ Å, or that the unit cell length in reduced units is 2.591. For comparison with the literature, however, the simulations address a liquid with a slightly higher density, and for that, a unit cell with a length of 2.5487 reduced length units should be used.

In my experience, though, equilibration of the silicon liquid is tricky. At lower T^*, lengthy equilibration is required before typical liquid structures are reached. Using a higher T^* is also problematic because the two-body term of eq. 10 is not highly repulsive at short distance, and accordingly simulation at higher T^* leads to population of structures with short r_{ij} which only slowly disappear at lower T^*. While these problems are not insuperable, for the purposes of your exercise here, I will therefore provide a well-equilibrated starting structure with 216 atoms.

4 The Radial Distribution Function

In order to compare the outcome of simulations with experiment or with other simulations, one needs to have some property to base the comparison on. This property could in principle be the mean energy or some other thermodynamic function, but the easiest property to compare is the mean *structure* in the form of the radial distribution function $g(r)$. This function is defined as the ratio of the density of atoms found at a given distance r from a reference atom, $\rho(r)$, and the bulk density ρ:

$$g(r) = \frac{\rho(r)}{\rho} \tag{15}$$

This function can readily be evaluated during a simulation. One defines a histogram $N_i(r_i)$ in which one records how many times during the simulation two atoms are separated by a distance falling in the interval $[r_i, r_i + \Delta r)$, where Δr is some small value (small enough to generate a smooth $g(r)$ function, but large enough that sampling does not require too many steps). The histogram is typically accumulated from $r_0 = 0$ to some maximum value $r_{\max}$, where it should be noted that when using periodic boundary conditions, the value of $g(r)$ becomes meaningless for $r > L/2$, so $r_{\max}$ should certainly be smaller than this. The histogram can be updated after every sweep (or every few sweeps) of Monte Carlo steps (updating after every step is not useful, since most atoms do not move for a given Monte Carlo step), by computing all atom-atom distances r_{ij} (with $j > i$) and incrementing the histogram appropriately, keeping track of the number of times N_{accum} that the histogram has been updated. After the simulation has finished, one can generate the radial distribution function by comparing the number of atom pairs found to have had distances in each interval $[r_i, r_i + \Delta r)$ with that expected for the given density (the volume of the shell with thickness Δr is $4/3\ \pi(r_i + \Delta r)^3 - 4\pi r^3$). In practice, one needs to use the following expression (the factor $N/2$, with N the number of atoms, is present because one has included all atom-atom distances r_{ij} (with $j > i$) when generating the histogram).

$$g(r_i) = \frac{N_i}{N_{\text{accum}} \times N/2 \times 4/3\,\pi((r_i + \Delta r)^3 - r_i^3)\rho} \tag{16}$$

For reference and help with debugging, Figure 3 shows the computed radial distribution function for both systems to be studied, the Lennard-Jones fluid and the Stillinger-Weber model of liquid silicon, with typical simulation conditions. The corresponding data files are also available.

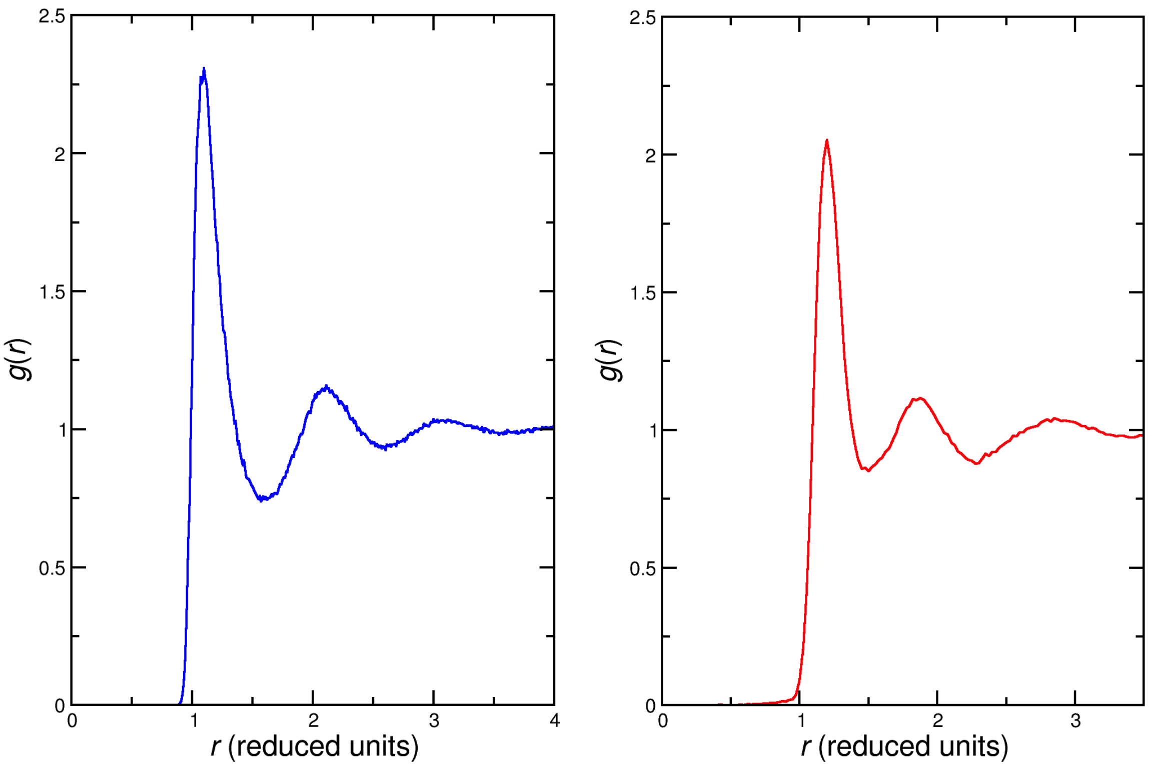

Figure 3: Sample radial distribution functions $g(r)$ for (left) the Lennard-Jones fluid, and (right) the Stillinger-Weber model of silicon. Conditions used: LJ fluid: 729 atoms, $r_c = 3.0$, $\rho^* = 0.7$, $L = 10.14$, $T^* = 1.2$, 2000 sweeps for equilibration, 500 for production, initial structure: regular cubic lattice, maximum move: 0.2, $g(r)$ sampled from $r = 0$ to $r = 4.0$, at intervals $\Delta r = 0.01$. Stillinger-Weber model: 216 atoms, $L = 7.6461$ reduced units ($\rho^* = 0.4832$), $r_{\text{skin}} = 0.4$, $T^* = 0.10$, $\Delta r_{\text{max}} = 0.15$, 2500 equilibration sweeps, 2000 sweeps for production, initial structure: random, with a minimum interatomic distance of 0.9, $g(r)$ sampled from $r = 0$ to $r = 3.5$, at intervals $\Delta r = 0.025$.

5 Efficient Coding: Neighbour Lists

Evaluating the energy expressions of eqs. 7 or 9, or the energy change expressions of eq. 8 (or the equivalent more complicated ones for the Stillinger potential, eqs. 19, 20 and 21 below) at first sight requires performing a loop over all pairs or triples of atoms in the system. Hence the computational expense of each step might be expected to scale as $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$, making the calculations quite demanding for larger N. As already mentioned, in Monte Carlo simulations, only the energy terms involving the atom which is being moved in the trial move actually need to be considered, meaning that only a single loop (Lennard-Jones) or a double loop (Stillinger) is needed, reducing the scaling to $\mathcal{O}(N)$ or $\mathcal{O}(N^2)$. Still, this can be quite computationally demanding if the system contains many atoms. It is also inefficient, since many of the atoms will be positioned at a distance from the reference atom greater than the cut-off distance r_c, and hence make no contribution to the energy.

Careful program design can to some extent alleviate this problem. For example, for each pair of atoms i and j considered, one can first compute the square[2] of the distance between them (using the minimum-image convention), $(r_{ij}^{\mathrm{MI}})^2$, and then test to see if this is larger than r_c^2, in which case one moves on to the next atom without performing any additional operations. In this way, the full computational expense is incurred only for those atoms that are close to one another, and only the value $(r_{ij}^{\mathrm{MI}})^2$ needs to be computed for all pairs (and $(r_{ij}^{\mathrm{MI}})^2$ and $(r_{ik}^{\mathrm{MI}})^2$ for triples (i, j, k)). Even so, this remains unnecessarily computationally expensive, as the computation of $(r_{ij}^{\mathrm{MI}})^2$ does not have negligible computational expense. For a fluid with number density ρ^*, on average only $\rho^* 4\pi r_c^3/3$ atoms will be within r_c of a given atom, and this is usually much smaller than N. Hence the quantity $(r_{ij}^{\mathrm{MI}})^2$ will need to be evaluated many times without making any contribution to the energy. For example, for $\rho^* = 0.7$, for a simulation cell with $L = 10$, there will be a total of 700 atoms, of which only just over 10% (79), on average, will be closer than r_c to a given atom, if $r_c = 3\sigma$.

Many simulation codes accordingly make use of advanced techniques to increase efficiency, such as the use of *neighbour list*. These are lists $\mathbf{N}^L(j) \equiv \{k\}_j$, generated for each atom j in the system at regular intervals during the simulation, and containing the atom numbers k of each of the atoms that have a distance r_{kj}^{MI} to atom j that is less than some threshold r_n. We can write that:

$$r_{kj}^{\mathrm{MI}} < r_n \Rightarrow k \in \mathbf{N}^L(j) \quad \& \quad j \in \mathbf{N}^L(k) \tag{17}$$

Once the lists $\mathbf{N}^L$ are available, one simply needs to loop over the atoms in the appropriate list, rather than over all atoms, when evaluating the energy in the case of the Lennard-Jones potential. For the case of the Stillinger potential, one also loops over neighbour lists rather than over all atoms – the details of which atoms to include in the loops are discussed below. Note that the threshold r_n must be at least equal to the cut-off distance. In fact, it needs to be *larger* than the cut-off distance, because otherwise the neighbour list would need to be reevaluated after each atomic motion. Choosing a distance $r_n = r_c + r_{\mathrm{skin}}$ as shown in fig. 4, with r_{skin} some distance such that the atoms occupying the 'skin' between the sphere of radius r_c and that of radius r_n are those which are likely to move closer than r_c to the central atom within the next few Monte Carlo moves, enables one to update the neighbour list only every few moves. The decision when to trigger the reevaluation of the neighbour list can be taken by looking to see how much each atom has moved since the list was created. In detail, one needs to store the positions of all atoms at the moment when the neighbour list is created, then after each successful Monte Carlo step, one can compute the total distance (using the minimum image convention) that this atom has moved since the neighbour list was created. In case this distance is larger than $r_{\mathrm{skin}}/2$, the neighbour list must be refreshed. A larger r_{skin} means that there are more atoms in each neighbour list, so evaluating the potential takes longer, but also means that the neighbour list needs to be renewed less frequently, so there is a trade-off in

[2]For the Lennard-Jones potential, one should only ever calculate the square of interatomic distances – the potential of eq. 6 can be evaluated in terms of the square, so computing r_{ij} via taking the square root of r_{ij}^2 is a waste of computational time.

terms of efficiency. In any case, r_{skin} needs to be more than twice as large as the maximum length of a Monte Carlo move $\Delta r_{\max}$, because otherwise a single move could bring an atom to within less than r_c of the central atom.

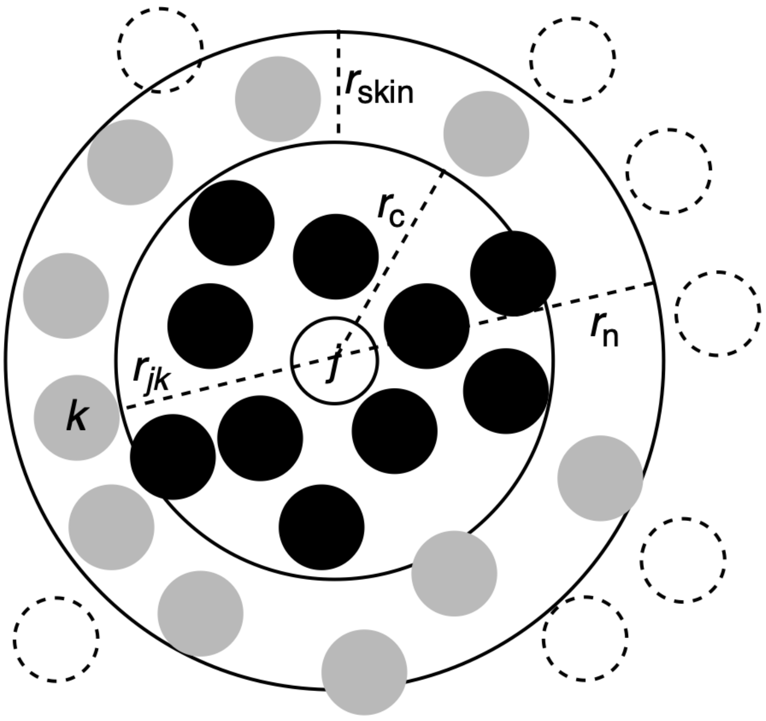

Figure 4: Principle behind the use of a neighbour list. The atoms shown as black circles have $r_{jk} < r_c$, while the grey atoms have $r_{jk} < r_n$. The atoms shown as dashed circles have $r_{jk} > r_n$ and are not included in the neighbour list.

In detail, it is recommended that the neighbour list be stored in an integer array of dimensions $n_{\max} \times N$ in Fortran (or $N \times n_{\max}$ in C++). $n_{\max}$ is the maximum number of expected neighbours per atom, and should be some multiple of $\rho 4\pi r_n^3/3$ – perhaps twice or three times, in order to allow for regions of higher density within the whole. As well as storing the identities of the atoms that are closer than r_n in this array, one needs a one-dimensional integer array to store the number of neighbours that each atom has, and a double precision array of dimension $3 \times N$ to store the atom positions at the moment the neighbour list was generated.

For the Lennard-Jones case, eq. 7 suggests that the neighbour list for atom i would only need to contain the atoms j with $j > i$, but given that the Monte Carlo code will typically use instead eq. 8, in fact, both $j > i$ and $j < i$ neighbours should be stored. Then eq. 8 can be applied by looping only over the atoms within the neighbour list of the moved atom (but still applying the cutoff, for those atoms with a distance of more than r_c but less than r_n):

$$\Delta V = \sum_{j \in \mathbf{N}^L(i)} \left\{ \left(\frac{1}{r_{ij}^{\text{trial}}} \right)^{12} - \left(\frac{1}{r_{ij}^{\text{trial}}} \right)^{6} \right\} - \left\{ \left(\frac{1}{r_{ij}^{\text{init}}} \right)^{12} - \left(\frac{1}{r_{ij}^{\text{init}}} \right)^{6} \right\} \tag{18}$$

For the Stillinger case, the neighbour list should likewise contain all neighbours.

The change in energy upon moving atom i from position $\mathbf{r}_i^{\text{init}}$ to $\mathbf{r}_i^{\text{trial}}$ can be obtained as the sum of the change in the two-body and three-body terms of eq. 9, as follows:

$$\Delta V_2 = \sum_{j \in \mathbf{N}^L(i)} f_2(r_{ij}^{\text{trial}}) - \sum_{j \in \mathbf{N}^L(i)} f_2(r_{ij}^{\text{init}}) \tag{19}$$

For the three-body term, one has:

$$\Delta V_3 = V_3^i(\mathbf{r}^{\text{trial}}) - V_3^i(\mathbf{r}^{\text{init}}) \tag{20}$$

Where the partial three-body energy associated with atom i, V_3^i, is given by the expression below. The first summation includes all three-body terms in which atom i is the central atom. The second summation includes all three-body terms in which an atom j in the neighbour list of atom i is the central atom.

$$V_3^i = \sum_{j,k \in \mathbf{N}^L(i), j<k} h(r_{ij}, r_{ik}, \theta_{jik}) + \sum_{j \in \mathbf{N}^L(i)} \sum_{k \in \mathbf{N}^L(j), k \neq i} h(r_{ji}, r_{jk}, \theta_{ijk}) \tag{21}$$

6 Programming Tips

The task in this course is to write a code that correctly simulates the behaviour of a liquid of atoms and generates the corresponding radial distribution function.

Specifically, the code should:

- Simulate liquid silicon, using the Stillinger potential wth reduced units and periodic boundary conditions.
- Be able to model a box containing an arbitrary number of atoms, with an arbitrary box length, and using an arbitrary reduced temperature T^*.
- Be able to read in these simulation parameters, together with an initial structure, from a provided input file.
- Output the computed radial distribution function for this system, for r ranging from 0 units of length to 4 units of length.
- Use a neighbour list to make evaluation of the potential more efficient.

It is suggested that prior to writing the code as described above, one should first write some more simple codes. This is not mandatory, but given the considerable overlap with the main task, seems like useful practice to me. Specifically, one might wish to write:

1. A code for Monte Carlo simulation of the Lennard-Jones liquid, for a system of 100 or more atoms, using a cutoff distance r_c of 3, an adjustable density, and with the code being able to output the radial distribution function. This code should use reduced units. No model input or starting structure is provided here: the code should create its own initial structure by randomly placing the atoms in the simulation box.
2. A code as above, but using a neighbour list in order to make evaluation of the potential more efficient.
3. A code simulating liquid silicon using the Stillinger potential, as in the main task, but *without* using neighbour lists. This code should start from the provided initial structure, so it should be able to read the provided input file.

It is recognized that different students have different levels of experience and expertise with coding, and that the above main objective may be too ambitious for some students. It will be possible to gain a passing grade based on a working code corresponding to any of the intermediate goals 1. – 3. listed above, though students are encouraged to progress as far as possible.

All of the key equations needed to accomplish the mentioned tasks are provided in the previous sections. To be specific:

1. For the first program, for Monte Carlo simulation of the Lennard-Jones potential, I suggest generating the initial structure at random. The key equation is then eq. 8.
2. The modified program using a neighbour list will require careful reading of the section on neighbour lists, and eq. 8 should be replaced by eq. 18.
3. For the initial Stillinger code, one should use eqs. 19, 20 and 21, but with the summations running over all atoms in the system.
4. The final code should also use neighbour lists.