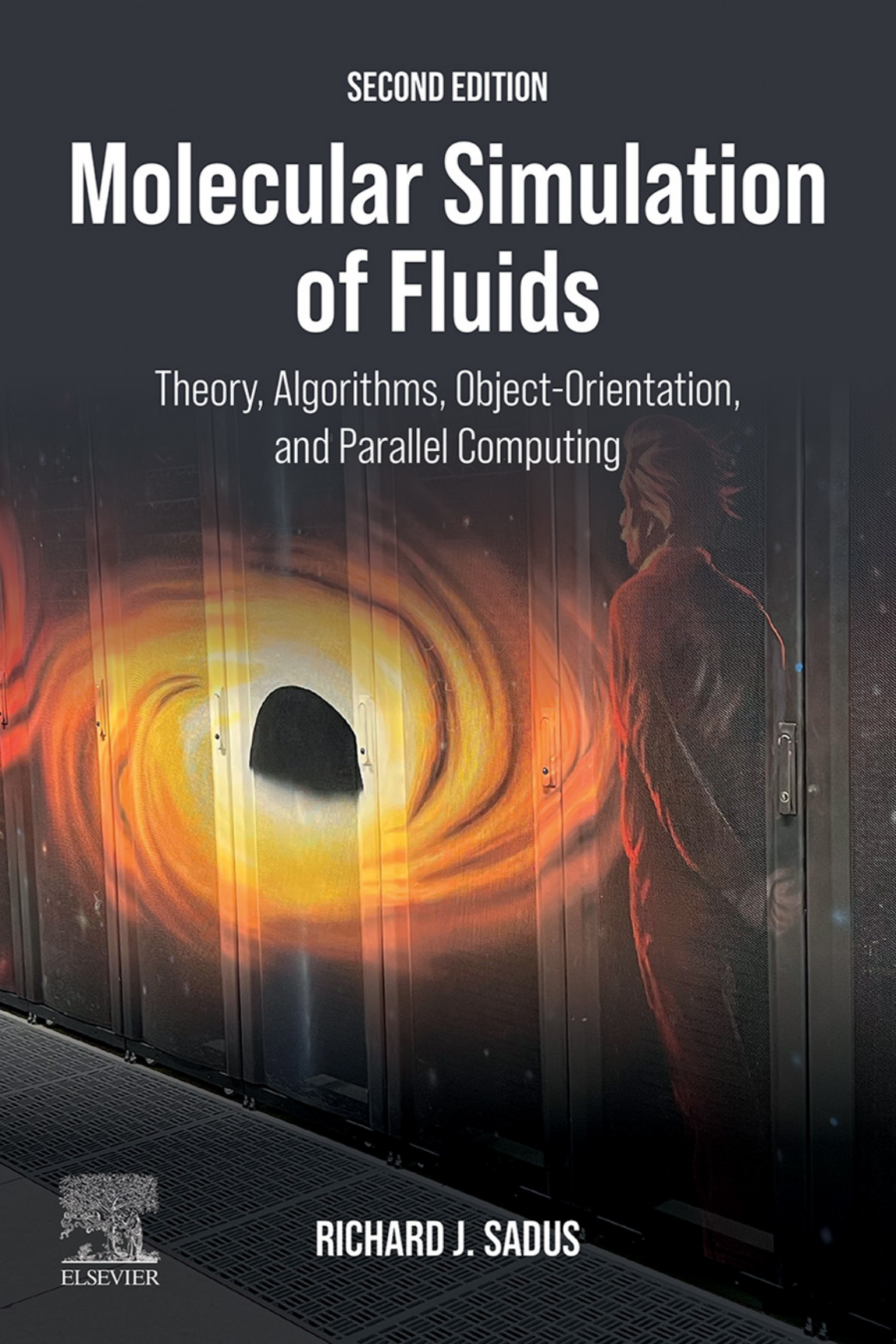
SECOND EDITION

Molecular Simulation of Fluids

Theory, Algorithms, Object-Orientation, and Parallel Computing



RICHARD J. SADUS

ELSEVIER

Molecular Simulation of Fluids

Theory, Algorithms, Object-Orientation, and Parallel Computing

This page intentionally left blank

Molecular Simulation of Fluids

Theory, Algorithms, Object-Orientation, and Parallel Computing

Second Edition

Richard J. Sadus
Department of Computing Technologies, Swinburne University of Technology, Wurundjeri Country, Australia

ELSEVIER

Elsevier
Radarweg 29, PO Box 211, 1000 AE Amsterdam, Netherlands
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, United Kingdom
50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

ISBN: 978-0-323-85398-9

Publisher: Candice Janco
Acquisitions Editor: Amy Shapiro
Editorial Project Manager: Leticia M. Lima
Production Project Manager: Kumar Anbazhagan
Cover Designer: Miles Hitchen

Typeset by MPS Limited, Chennai, India

To Angelica

This page intentionally left blank

Contents

This page intentionally left blank

Preface to the second edition

The intervening years since the publication of the first edition of this book have witnessed an enormous growth in popularity of molecular simulation. It is now arguably the preferred computational approach for both the evaluation of molecular properties and the prediction of macroscopic properties. The areas of theory, algorithms, and object-orientations covered by the first edition have greatly expanded. Examples include advances in computational chemistry, resulting in highly accurate two-body ab initio potentials; new methods for simulating solid-liquid equilibria; and the widespread use of object-orientation for simulation. The latter is particularly significant, as it has permitted the continuous development of sophisticated molecular software that can be used to tackle highly complicated situations involving proteins and other large macromolecules.

The emergence of parallel computing is the most important contributor to the growing success of molecular simulation that was not covered by the first edition. Initially, this simply involved the use of multiple central processing units (CPUs) but its impact has been supercharged by the introduction of graphics processing units (GPUs). The combination of object-orientation and parallel computing provides an important route for the development of adaptable software for future computationally challenges.

The second edition reflects advances made in key areas of molecular simulation. Each chapter has been updated, providing details of new algorithms and theory, while maintaining most of the original content that still remains both relevant and useful. Readers have often commented that the algorithms in pseudo-code from the first edition could be easily transformed into working programs. This feature has been retained and improved by adopting greater modularization via the use of subalgorithms. Case studies have been added to provide detailed explanations of some challenging aspects. The C++ code provided in Chapter 11 has also been updated.

The most significant changes are the addition of two new chapters. Chapter 4 covers the use of ab initio potentials and two-and three-body interactions. Advances in theory mean that the interactions between both atoms and molecules can be reliably obtained from first principles with increasingly less reliance on empirical models. Chapter 12 covers parallel computing using both multiple CPUs and GPUs. It is written specifically to assist readers who would like to quickly take advantage of parallel computing, perhaps

by modifying existing serial code. Therefore it is a starting point for further learning rather than a comprehensive examination of the topic. It may also be useful for readers who require an improved understanding of some of the key underlying technical aspects of large scale parallel-enabled molecular simulation software packages. Using detailed case studies, Chapter 12 shows how the C++ codes from Chapter 11 can be simply modified to execute in parallel on both multiple CPUs and GPUs. MPI is used for the multi-CPU code. CUDA integrates very well with C++ code and facilitates the near-seamless transition to GPUs. The codes are available at the book's website (https://www.elsevier.com/books-and-journals/book-companion/9780323853989) that is maintained by the publisher.

I thank Ulrich K. Deiters, Karl. P. Travis, and Billy D. Todd for reading drafts of various chapters and providing both valuable comments and insights. I also thank Angelica M. Vecchio-Sadus for proofreading. The molecular simulations were conducted on the Swinburne University supercomputer (OzSTAR), which is located on the traditional lands of the Wurundjeri people. It is illustrated on the front cover.

Richard J. Sadus
August, 2023

Preface to the first edition

Molecular simulation is being used increasingly to study a widening range of both molecular systems and fluid phenomena. Today, the goal of many simulators is to study complicated molecules such as proteins, whereas attention was formerly confined almost exclusively to simple atoms and molecules. Similarly, the simulation of phase equilibria is now common, whereas just over a decade ago, it was a difficult and rare undertaking. The impetus for the increasing use of molecular simulation can be attributed to many factors such as improvements in theory, algorithms, and computer hardware. These new developments have generated enormous growth in the simulation literature. A simple search on the keywords "Monte Carlo" and "molecular dynamics" in any journal database is likely to yield thousands of entries for the past five years alone. An aim of this book is to examine some of the important aspects of recent progress in molecular simulation.

The other important aims of this book are reflected in the title. The topic covered by the book is the molecular simulation of fluids which encompasses both Monte Carlo and molecular dynamics methods. Although the principles behind these two simulation methods are quite different, they are often complementary rather than competing approaches. The practice of molecular simulation involves three overlapping aspects. First, the theoretical basis of molecular simulation is underpinned by the fundamental concepts of statistical mechanics and particle dynamics. Second, informed by theory, algorithms are developed to calculate the properties of fluids. Third, theory and algorithms are "brought to life" on a computer via a simulation program.

There are many simulation algorithms in the literature with varying levels of usage. Some algorithms such as the "leap-frog" algorithm for molecular dynamics are well known and used widely. There are also algorithms at intermediate stage of usage, and many other potentially useful algorithms are relatively "new" and as such, they are not known widely. In this book, we have opted to present a mixture of new, intermediate, and well-established algorithms. However, the mix is biased deliberately to favor new algorithms many of which have not yet been used widely or tested exhaustively. In many cases, to facilitate implementation, the algorithms are presented in pseudo code.

Historically, the implementation of simulation algorithms has been achieved using a structured programming approach typified by programming

languages such as C and FORTRAN. In a deliberate break with tradition, there are few references in this book to either FORTRAN or C code. Instead, attention is focused on the concepts of object-oriented analysis, design, and programming. The value of object-orientation to molecular simulation remains largely untried; however, it offers potential benefits particularly for the development of simulation code involving complicated molecules such as large polymers and proteins. Simulation code is predominantly developed and used by physicists, chemists, and chemical engineers and not by computer scientists. However, advances in computer science such as object-orientation can be of potential benefit to molecular simulation practitioners. This book is in part, an effort to bridge this gap between practitioners and advances in computer science. Sample object-oriented simulation code written in C++ is included on the accompanying disk.

This book has benefited from the contributions of others. I thank Ross Daws for his valuable assistance with the object-oriented analysis and Gianluca Marcelli for reading the manuscript carefully. Thanks are also due to Peter Cummings, Denis Evans, and Rolf Lustig for access to their work prior to publication. Most importantly, I thank my wife Angelica for contributing her good judgment to many aspects of both the appearance and contents of the book.

Richard J. Sadus
January, 1999

Chapter 1

Introduction

This chapter introduces the concept of molecular simulation and provides a brief survey of developments in the field. Many of the topics introduced here are examined in greater detail in subsequent chapters. Various aspects of molecular simulation have also been reviewed elsewhere (Ciccotti et al., 1987; Haile, 1992; Allen and Tildesley, 1993; Heyes, 1998; Leach, 2001; Frenkel and Smit, 2023; Rapaport, 2004; Allen and Quigley, 2013; Allen and Tildesley, 2017; Raabe, 2017). Battimelli et al. (2020) have reported a historical narrative for some aspects of the development of molecular simulation in the twentieth century.

1.1 What is molecular simulation?

Molecular simulation is a generic term encompassing both Monte Carlo (MC) and molecular dynamics (MD) computing methods. The primary motivation for molecular simulation is to provide exact results for statistical mechanical problems in preference to approximate solutions. The feature that distinguishes molecular simulation from other computing methods and approximations is that the molecular coordinates of the system are evolved in accordance with a rigorous calculation of intermolecular energies or forces.[1] Molecular simulation can be aptly described as a computational statistical mechanics. It allows us to determine macroscopic properties by evaluating exactly a theoretical model of molecular behavior using a computer program. The keyword in the earlier description is "exactly." The results of a molecular simulation are determined solely by the nature of the theoretical model used. Consequently, comparison of simulation results with experimental data is an unambiguous test of the accuracy of the model. Discrepancies between accurate experimental measurements and molecular simulation data can be attributed unambiguously to the failure of the model to represent molecular behavior.

The history of molecular simulation largely parallels the history of the digital computer. However, it is interesting to reflect that some aspects of

1. Some useful complementary methods (Klamt, 2005) that do not involve the evolution of coordinates are not considered here.

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00002-2

what we would now recognize as molecular simulation predate the computing era, when the term "molecular dynamics" simply referred to the dynamical behavior of molecules. For example, Lord Kelvin (1904) offers the following description of an experiment to obtain the distribution of velocities involved in molecular collision.

> *"For our present problem we require two lotteries to find the influential conditions at each instant, when* Q *enters* P*'s cage—Lottery I. for the velocity* (v) *of* Q *at impact: lottery II. for the phase of* P*'s motion. For lottery I. (after trying 837 small squares of papers with velocities written on them and mixed in a bowl, and finding the plan unsatisfactory) we took nine stiff cards, numbered 1, 2 . . . 9, of size of ordinary playing-cards with round corners, with one hundred numbers written on each in ten lines of ten numbers. The velocities on each card are shown on the following table. The number of times each velocity occurs was chosen to fulfill as nearly as may be the Maxwellian law . . ."*

This quaint description includes aspects such as obtaining a Maxwell–Boltzmann velocity distribution and random sampling that are routinely performed by computer-based MD and MC molecular simulations, respectively.

The first computer-era molecular simulation of a liquid was performed by Metropolis et al. (1953) on the MANIAC computer at Los Alamos. Metropolis et al. introduced the Monte Carlo method. During a MC simulation, different trial configurations are generated randomly. The intermolecular interactions in the trial configuration are evaluated and probabilities are used to either accept or reject the change. Later, Alder and Wainwright (1957, 1959) introduced the molecular dynamics (MD) method (Chapter 6). In a MD simulation, the equations of motion (Chapter 2) for the system of molecules are solved. The molecular coordinates and momenta change according to the intermolecular forces experienced by the individual molecules. As the name implies, MD enables us to obtain the dynamic properties of the system. Unlike MC simulations, MD is totally deterministic and chance plays no role.[2] Another important distinction between MC and MD methods is that MD uses intermolecular forces to evolve the system, whereas MC simulation involves primarily the calculation of changes in intermolecular energy.

The practice of molecular simulation has continued to benefit enormously from subsequent improvements in the evolution of computer hardware (Fig. 1.1) and software (Chapters 11 and 12). Today, researchers in many disciplines use both simulation techniques widely. The popularity of molecular simulation as a research tool is evident from the many thousands of entries for the keywords "Monte Carlo" and "molecular dynamics" found in *The Web of Science* and *Scopus* databases for each successive year. The choice between the two alternatives depends often on the type of property

2. Sometimes it is convenient to combine both approaches resulting in a hybrid-MD algorithm.

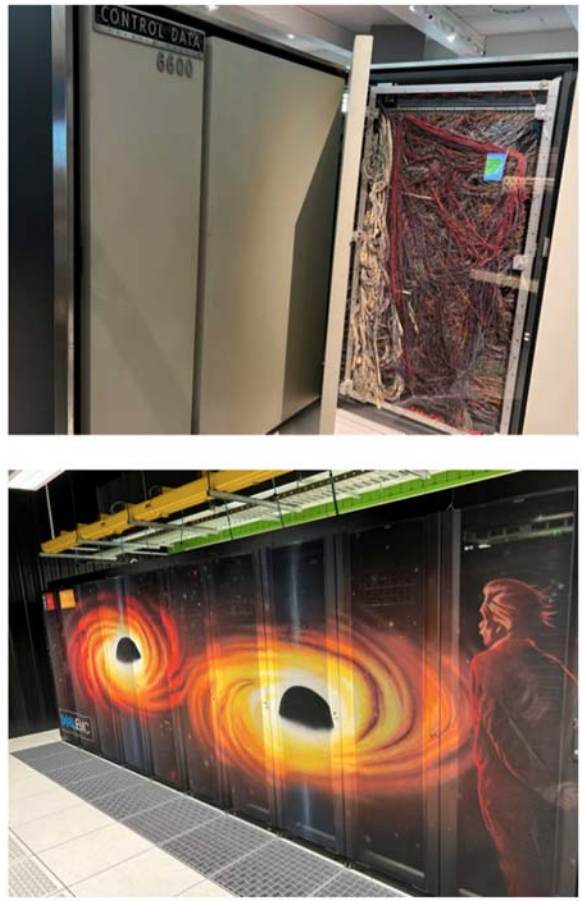


FIGURE 1.1 Computers then and now. The CDC 6600 computer (top, Science Museum, London) with its "spaghetti wiring" was the workhorse for many computational tasks in the 1960s. In contrast, today's supercomputers (bottom, OzSTAR Swinburne University of Technology) are compact racks of thousands of CPUs and GPUs (Chapter 12).

being investigated. For example, a MD strategy is required to obtain time-dependent dynamic properties of the system, whereas the MC method can be used in situations where the systems have no intrinsic dynamics. Either MC or MD can be used for equilibrium properties.

Molecular simulation is increasingly providing valuable insights into fluid properties. An example of the value of molecular simulation is illustrated by its role in the correct interpretation of neutron scattering data for water (Chialvo et al., 1998). Postorino et al. (1993) reported neutron scattering data for supercritical water, which indicated the absence of hydrogen bonding in contradiction to molecular simulation studies that predicted hydrogen bonding.

Based on their analysis, Postorino et al. (1993) concluded that the models of water used in the MD studies were inadequate. However, further careful molecular simulation studies (Chialvo and Cummings, 1994) with revised models for water continued to show the presence of hydrogen bonding. It became apparent subsequently (Soper et al., 1997) that the original neutron scattering data had been analyzed incorrectly. The revised analysis (Soper et al., 1997) showed the presence of hydrogen bonding as predicted by molecular simulation. Furthermore, there was good qualitative agreement between the revised analysis and molecular simulation predictions. Molecular simulation can also be used increasingly to resolve conflicts between experimental data for difficult-to-measure properties such as the critical densities of thermally unstable high-molecular-weight alkanes (Siepmann et al., 1993). Molecular simulations can be used to make genuine predictions from first principles that are either close to or exceed experimental accuracy (Deiters and Sadus, 2022a,b, 2023).

1.1.1 The MC method

The MC method is a stochastic strategy that relies on probabilities, which explains the association with the eponymous casino in the principality of Monaco (Fig. 1.2). To simulate fluids, transitions between different states or configurations are achieved by (1) generating a trial configuration randomly;

FIGURE 1.2 Casino de Monte-Carlo on December 29, 2013. In the absence of certainty, there is a nonzero probability (chance) that the white substance in the foreground could be real snow.

(2) evaluating an "acceptance criterion" by calculating the change in energy and other properties in the trial configuration; and (3) comparing the acceptance criterion to a random number and either accepting or rejecting the trial configuration. It is important to realize that not all states will make a significant contribution to the configurational properties of the system. To determine accurately the properties of the system in the finite time available for the simulation, it is important to sample those states that make the most significant contributions. This is achieved by generating a Markov chain (Chapters 6 and 9), which is a sequence of trails in which the outcome of successive trails depends only on the immediate predecessor. In a Markov chain, a new state will only be accepted if it is more "favorable" than the existing state. In the context of a simulation using an ensemble, this usually means that the new trial state is lower in energy. The MC algorithm is summarized by Algorithm 1.1.

In the MC process, a new configuration is obtained typically by displacing, exchanging, removing, or adding a molecule. Probabilities are also often involved in determining the nature and extent of the attempted move. The exact nature of the transition probability depends on the ensemble chosen (Chapter 9), but it always involves evaluating the energy of the new configuration and comparing it with the energy of the existing (old) state. Notice, that if the attempted change is rejected, then the old state is counted as the new state. Details of MC simulations are examined in Chapter 6.

ALGORITHM 1.1 General outline of a MC process.

```
Part 1   Create an initial configuration (config).
Part 2   Generate a Markov Chain for Ncycles:
         loop i ← 1 ... NCycles
            Create a new configuration (configTrial).
            Find the transition probability w(config, configTrial).
            Generate a uniform random number (R) between 0 and 1.
            if (w > R) then
               Accept move (config = configTrial).
            else
               Reject move (config is unaltered).
            end if
         end loop
```

1.1.2 MD

In contrast to the MC method, which relies on transition probabilities, MD solves (Chapters 7–9) the equations of motion of the molecules to generate

new configurations. Consequently, MD simulations can be used to obtain time-dependent properties of the system. A general MD algorithm is presented in Algorithm 1.2.

ALGORITHM 1.2 General outline of a MD process.

```
Part 1   Create an initial configuration.
Part 2   Simulation for a duration tMax:
         loop
            Create a new configuration by solving the equation
            of motions.
            Calculate quantities to be time-averaged.
         while (time ≤ tMax)
```

Algorithm 1.2 hides a considerable degree of complexity. There are several possible algorithms for solving the equations of motions, which are examined in detail in Chapter 7. The practical implementation of MC and MD methods have many features in common, which are apparent from the considerations discussed in Chapters 3–5.

1.2 Progress in molecular simulation

1.2.1 Early work on hard spheres and atoms

The very early work on molecular simulation was confined to either evaluating the properties of hard spheres (HS) or using the Lennard-Jones (LJ) potential (Chapter 3). The LJ potential remains the most widely used intermolecular potential for molecular simulation. Metropolis et al. (1953) showed how the MC method could be used to calculate the properties of two-dimensional rigid spheres. Most significantly, they introduced the concepts of importance sampling (Chapter 6) and periodic boundary conditions (Chapter 5). Later, Wood and Parker (1957) reported MC simulations using the LJ potential. They found evidence for a solid–fluid phase transition, which was later confirmed by Hansen and Verlet (1969). Their paper also discusses the minimum image convention (Chapter 5) to account for the truncation of the potential required by the periodic boundary conditions.

Alder and Wainwright (1957) reported the first MD simulation for HS. Their data was found to be in good agreement with MC results (Wood and Jacobson, 1957), thereby demonstrating the equivalence of the two molecular simulation methods. The first MD simulation of a LJ fluid was reported by Rahman (1964). Verlet (1967) also performed MD simulations for the LJ potential using an improved integration scheme (Chapter 7) for the equations

of motion and introducing the concept of a neighbor list (Chapter 5) to reduce computation time. Verlet's algorithm and subsequent modifications (Chapter 7) are widely used today. The predictor–corrector integration method (Chapter 7) proposed by Gear (1971) is also used extensively.

1.2.2 Small nonpolar molecules

Small diatomic or polyatomic molecules represent a natural extension of the application of molecular simulation. Harp and Berne (1968) applied the MD method for diatomic molecules. It is common to consider the intermolecular interactions between two diatomic molecules via a site–site approximation (Singer et al., 1977), in which the interaction sites correspond to the position of the nuclei of the real molecules Consequently, diatomic molecules such as nitrogen can be depicted as two LJ atoms separated by a fixed bond length. This approach was used by Cheung and Powles (1975) to model the nitrogen molecule.

The MD simulation of polyatomic molecules requires us to consider how to account for vibrational and rotational degrees of freedom in the integration of the equations of motion (Chapter 2). In most cases, it is safe to ignore molecular vibration and treat the molecule as a rigid body. The kinematics of rigid bodies is well developed (Goldstein et al., 2002); however, the time evolution of the Euler angles results in an awkward singularity. For small molecules, this problem is commonly avoided by using the quaternions approach proposed by Evans and Murad (1977), whereas the method of constraint (Ciccotti et al., 1982) is particularly advantageous for larger, more flexible molecules. In contrast, MC simulations of polyatomic molecules address the issue of rotation by proposing special moves (Chapter 6).

1.2.3 Polar molecules and ions

In contrast to interactions between nonpolar molecules, interactions between dipolar molecules or ions are long-range. A long-range force is defined as one which falls off no faster than r^{-d}, where r is the intermolecular separation and d is the dimensionality of the system. Typically, ion–ion and dipole–dipole potentials are proportional to r^{-1} and r^{-3}, respectively. Consequently, the effect of either interaction extends well past half the length of the simulation box, and special techniques (Chapter 5) are required to incorporate these long-range forces. Rigorous formulas are available (Chapter 3) to determine the contribution of various nonpolar interactions, but the most commonly used approach is to assign partial charges to the constituent atoms and evaluate the Coulombic contributions.

For ionic systems, the Coulombic energies and forces are calculated commonly using either an Ewald sum (Nosé and Klein, 1983) or a reaction field (Barker and Watts, 1969). A particle–particle and particle–mesh (PPPM) algorithm (Eastwood et al., 1980) is also effective. Fast multipole methods (FMM) have also been developed (Greengard and Rokhlin, 1987),

which are potentially useful in simulations involving thousands of molecules. Barker (1994) combined elements of Debye–Hückel theory for electrolytes with the reaction field method used for dipolar simulation resulting in a relatively simple algorithm for including long-range effects. Examples of ionic systems studied by molecular simulation include simple molten salts (Hansen and McDonald, 1975) and electrolyte solutions (Valleau and Cohen, 1980). Molecular simulations have been reported for the vapor–liquid phase equilibria of both sodium chloride (Guissani and Guillot, 1994) and binary mixtures containing sodium chloride (Strauch and Cummings, 1993).

The Ewald sum (Chapter 5) can also be applied to calculate dipole–dipole interactions (Adams and McDonald, 1976; de Leeuw et al., 1980). A relatively simple reaction field method (Barker and Watts, 1969) or the method proposed by Ladd (1977, 1978) can also be used.

The most commonly studied dipolar molecule is water. Water has been investigated by both MC (Barker and Watts, 1969; Ladd, 1977) and MD (Rahman and Stillinger, 1971) methods. The investigation of the properties of water by molecular simulation remains currently of topical interest because of its both technological importance and unique physical properties (Shvab and Sadus, 2016). Several different models of the water molecule have been investigated using molecular simulation (de Pablo and Prausnitz, 1989; de Pablo et al., 1990). Chialvo and Cummings (1999) have reported a detailed review of models for water at supercritical conditions. It is increasingly apparent that the accurate prediction of the properties of water (Chapter 4) requires interaction potentials that incorporated the effects of polarization (Shvab and Sadus, 2016).

1.2.4 Chain molecules and polymers

Long-chain molecules, highly branched molecules, and polymers impose considerable difficulties for molecular simulation when compared with either atoms or polyatomic molecules. The computational difficulties are particularly acute for MD methods for which the effect of molecular rotation must be incorporated into the equations of motion. Consequently, early examples of MD simulations were confined largely to chains of moderate complexity such as butane (Ryckaert and Bellemans, 1975; Daivis and Evans, 1995) and smaller alkanes (Ryckaert and Bellemans, 1978). Later, MD simulations were reported for eicosane (Morriss et al., 1991), branched alkanes (Daivis et al., 1992), entangled hard chains (Smith et al., 1996) and dendrimers (Bosko et al., 2004). However, the advent of improved computing architectures (Abraham et al., 2015) (Chapter 12) means that proteins and other large biomolecules can be handled using MD.

MC simulation has been applied to chain molecules of considerable size and complexity (Siepmann et al., 1993; Smit et al., 1995). A configurational-bias

Monte Carlo method (Siepmann and Frenkel, 1992) (Chapter 5) based on the work of Rosenbluth and Rosenbluth (1955) has been developed to facilitate the simulation of long chains. The vapor−liquid coexistence of alkanes as large as octatetracontane (C_{48}) (Siepmann et al., 1993; Smit et al., 1995) and other large chains (Escobedo and de Pablo, 1996) have been studied. Monte Carlo simulations have been reported for the properties of star polymers (Ohno et al., 1996; Tobita, 1996), dendrimers (Mansfield and Klushin, 1993) and large cyclic alkanes (Lee et al., 2005).

1.2.5 Ensembles for molecular simulation

The original MC simulations of Metropolis et al. (1953) sampled a canonical ensemble for which the number of particles, temperature, and volume are constant. However, there are many physical processes that occur under different conditions, for example, constant temperature and pressure. Consequently, it is desirable to conduct simulations in a variety of different ensembles depending on the nature of the property being investigated.

It is relatively easy to extend the MC method to other ensembles (Chapter 9) because the importance of sampling technique can be applied in any problem for which it is possible to construct a Markov chain. McDonald (1972) reported Monte Carlo calculations for binary mixtures in an isothermal−isobaric (*NpT*) ensemble. Valleau and Cohen (1980) conducted Monte Carlo calculations in the grand canonical (μVT) ensemble. The transformation to other ensembles is achieved by sampling the variable that is conjugate to the new fixed parameter in addition to sampling the particle coordinates. For the isothermal−isobaric ensemble, the new variable to be sampled is volume, whereas the number of particles is sampled for the grand canonical ensemble. Techniques have also been developed for MC simulations in the microcanonical (Ray, 1991) and canonical (Lustig, 2011) ensembles. The calculation of thermodynamic properties from different ensembles is examined in Chapter 2.

Generalizing ensembles for MD is complicated by the fact that the equations of motion lead naturally to the microcanonical (*NVE*) ensemble. Consequently, there is no well-defined method for modifying the equations of motion. Andersen (1980) conducted MD simulations at constant pressure and/or temperature by introducing additional degrees of freedom. Nosé (1984) introduced temperature control by adding additional terms to the Hamiltonian of the system. Later, Hoover (1985) re-structured Nosé's equations of motion, interpreting the additional variables as dynamical friction coefficients. MD algorithms for various ensembles are outlined in Chapter 9.

1.2.6 Phase equilibria

The traditional Monte Carlo and molecular techniques based on the canonical ensemble and constant pressure ensemble, respectively, do not yield reliable

chemical potential data required for phase equilibria calculations. This limitation was partly addressed by test particle (Adams, 1974; Widom, 1963, 1982; Shing and Gubbins, 1982) and grand canonical ensemble (Adams, 1975; Mezei, 1980; Yao et al., 1982) methods. Computer simulation was most commonly restricted to pure substances (Adams, 1976, 1979) with only limited applications to binary mixtures (Fincham et al., 1986; Panagiotopoulos et al., 1988).

In the last three decades, considerable progress has been made in the application of molecular simulation to phase equilibria. The main impetus has been the formulation of a Gibbs ensemble MC method (Panagiotopoulos,1987; Panagiotopoulos et al., 1988). In the Gibbs ensemble, the problem of obtaining reliable chemical potentials and uncertainties caused by interfaces are eliminated by representing the coexisting phases as separate simulation boxes. Equilibrium between the boxes (phases) is achieved by attempting molecular displacements with each box, volume changes, and particle interchanges between the boxes. The Gibbs ensemble has been used successfully to calculate phase equilibria in pure fluids (Smit et al., 1995), binary mixtures (Guo et al., 1994), and ternary mixtures (Tsang et al., 1995). The range of application of the Gibbs ensemble includes simple molecules, ions (Orkoulas and Panagiotopoulos, 1994), and polymers (de Pablo, 1995). Comprehensive reviews of the application of the Gibbs ensemble are available (Panagiotopoulos, 1992; Gubbins and Sandler, 1994).

Kofke (1993a,b) proposed a Gibbs–Duhem integration (GDI) scheme for the simulation of phase equilibrium that combines elements of thermodynamic integration with the Gibbs ensemble method. The Gibbs ensemble is used to locate one point on the phase envelope from which the rest of the curve can be traced. Accurate phase equilibria calculations have been reported (Panagiotopoulos et al., 1998) by applying histogram re-weighting techniques to the grand canonical ensemble.

A MD implementation of the Gibbs ensemble has been reported (Palmer and Lo, 1994). Möller and Fischer (1990) determined the phase coexistence of pure fluids using a MD NpT + test particle approach. Their approach has been applied to predict the vapor–liquid phase behavior of both pure fluids (Kriebel et al., 1995) and binary mixtures (Möller et al., 1992). Successful calculations of phase equilibria have also been reported using other MD methods (Schaink and Hoheisel, 1992; Guissani and Guillot, 1993).

The application of molecular simulation has been largely confined to vapor–liquid equilibria, with solid–liquid equilibria (SLE) proving much more challenging. The low probability of particle insertion into dense phases usually defeats Gibbs ensemble MC. GDI has been used very successfully for SLE but its accuracy depends on the availability of a reliable starting point. This mean SLE prediction often requires very specialized simulation method that suffer from limitations from such as the loss of precision when evaluating the chemical potential, unwanted surface effects, and difficulty in applying the method to molecules. However, progress in SLE predictions has been made

with the advent of new algorithms (Deiters and Sadus, 2022a,b). Details of algorithms for phase equilibria are given in Chapter 10.

1.2.7 Interatomic interactions

Molecular simulation methods are almost invariably implemented assuming that intermolecular interactions are confined to pairs of molecules. The calculations are usually achieved by using effective pair-wise potential for which the LJ potential is the most common example. By making this approximation, it is relatively straightforward to design force fields (Chapter 3) that can be applied to complicated molecular systems (Martin, 2006).

Effective potentials are computationally convenient, but are largely empirical without a solid theoretical background. In the last decade, advances in computational quantum chemistry (Chapter 4) have permitted the evaluation of two-body interactions from principles, resulting in highly accurate ab initio potentials for the noble gases (Hellmann et al., 2017; Deiters and Sadus, 2019). The advantage of such ab initio potentials is that they allow us to unambiguously determine the effect of two-body interactions on macroscopic properties.

1.2.8 Many-body interactions

The effect of three- or higher-body interactions has not been investigated extensively by molecular simulation. There is a very high computational impediment to implementing many-body interactions for an N-component systems. The simulation algorithm is of order N^m, where $m \geq 2$ is the number of molecules contributing to the interaction. In contrast to the two-body case, there are few three-body ab initio potentials and their accuracy is much less certain.

Simulations of the effect of three-body ($m = 3$) interactions have been reported using the Axilrod–Teller–Mutō (ATM) potential (Axilrod and Teller, 1943; Mutō, 1943). Barker et al. (1971) demonstrated that the ATM potential makes a significant contribution (5%–10%) to the overall energy of liquid argon and Monson et al. (1983) reported that three-body interactions are important for diatomic fluids. Rittger (1990a) included three-body interactions in a simulation for the chemical potential of liquid xenon. Following an examination of the thermodynamic data of xenon, Rittger (1990b) found that three-body dispersion interactions alone were insufficient to explain xenon's behavior. The profound effect of ATM interactions on phase equilibria and other thermophysical properties (Chapter 4) is well documented (Marcelli and Sadus, 1999; Sadus and Prausnitz, 1996; Wang and Sadus, 2006; Deiters and Sadus, 2019).

Despite advances in computer technology (Chapter 12), the use of three-body interaction is far from a routine undertaking, highlighting the need for careful approximations (Chapter 4) (Marcelli and Sadus, 2000). Calculating

three-body interactions is particularly beneficial when coupled with two-body ab initio two-body potentials (Deiters and Sadus, 2022a,b).

1.2.9 Nonequilibrium MD simulation

There are some properties of a fluid that are not in equilibrium. These properties can be described by nonequilibrium statistical mechanics (Evans and Morriss, 2008) and are calculated from nonequilibrium molecular dynamics (NEMD) simulation. Several NEMD techniques (Chapter 8) have been developed (Evans and Morriss, 1984). Typically, NEMD methods involve applying a perturbation to the usual equations of motion. The perturbation can be constant throughout the simulation, it can evolve with time, or alternatively, a sinusoidally oscillating perturbation can be used.

Gosling et al. (1973) used NEMD to determine shear viscosity. Evans and Morriss (1984) and Gillan and Dixon (1983) provided the theoretical basis for determining heat flow. Diffusion has also been investigated (Ciccotti et al., 1979). The application of NEMD continues to expand (Chapter 8). Examples include the study rheology (Morriss, et al., 1991) and elongational flow (Hunt et al., 2010). Many new and promising NEMD algorithms has been summarized by Todd and Daivis (2017).

1.2.10 Molecular simulation and object orientation

Historically, code for molecular simulation has been developed either prior to or during the era of structured procedural programming. Structured procedural programming emphasizes the decomposition of a problem into algorithms that are represented in languages such as FORTRAN or C as functions and/or subroutines. Structured procedural programming has "evolved" into object orientation. Object orientation has many benefits for code design, simplification, and re-use that can potentially benefit molecular simulation. It is now arguably the preferred method for code design, which has been exploited in many sophisticated simulation codes (Phillips et al., 2020; Thompson et al., 2022). Chapter 11 explores the potential benefits of object orientation.

1.2.11 Parallel computing

Arguably the most important development that has facilitated the growth in molecular simulation is the advent of parallel computing (Kirk and Hwu, 2010) (Chapter 12). Initially this simply involved distributing the computational load over multiple processors. However, the introduction of the graphics processing unit (GPU) has enabled (Kohnke et al., 2020) an enormous acceleration of MD code, making simulations with very large number of atoms or molecules computationally feasible (Hou et al., 2013; Heinecke et al., 2015). Object orientation

and parallel computing are a powerful combination that will greatly facilitate future scientific discoveries. This is examined in Chapter 12.

References

Abraham, M.J., Murtolad, T., Schulz, R., Szilárd, P., Smith, J.C., Hess, B., Lindahl, E., 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1–2, 19–25.

Adams, D.J., 1974. Chemical potential of hard sphere fluids by Monte Carlo methods. Mol. Phys. 28, 1241–1252.

Adams, D.J., 1975. Grand canonical ensemble Monte Carlo for a Lennard-Jones fluid. Mol. Phys. 29, 307–311.

Adams, D.J., 1976. Calculating the low temperature vapour line by Monte Carlo. Mol. Phys. 32, 647–657.

Adams, D.J., 1979. Calculating the high-temperature vapour line by Monte Carlo. Mol. Phys. 37, 211–221.

Adams, D.J., McDonald, I.R., 1976. Thermodynamic and dielectric properties of polar lattices. Mol. Phys. 32, 931–947.

Alder, B.J., Wainwright, T.E., 1957. Phase transition for a hard sphere system. J. Chem. Phys. 27, 1208–1209.

Alder, B.J., Wainwright, T.E., 1959. Studies in molecular dynamics. I. General method. J. Chem. Phys. 31, 459–466.

Allen, M.P., Quigley, D., 2013. Some comments on Monte Carlo and molecular dynamics methods. Mol. Phys. 111, 3442–3447.

Allen, M.P., Tildesley, D.J. (Eds.), 1993. Computer Simulation in Chemical Physics. Kluwer, Dordrecht.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Andersen, H.C., 1980. Molecular dynamics simulations at constant pressure and/or temperature. J. Chem. Phys. 72, 2384–2393.

Axilrod, B.M., Teller, E., 1943. Interaction of the van der Waals' type between three atoms. J. Chem. Phys. 11, 299–300.

Barker, J.A., 1994. Reaction field, screening, and long-range interactions in simulations of ionic and dipolar systems. Mol. Phys. 83, 1057–1064.

Barker, J.A., Watts, R.O., 1969. Structure of water: a Monte Carlo calculation. Chem. Phys. Lett. 3, 144–145.

Barker, J.A., Fisher, R.A., Watts, R.O., 1971. Liquid argon: Monte Carlo and molecular dynamics calculations. Mol. Phys. 21, 657–673.

Battimelli, Ciccotti, G., Greco, 2020. Computer Meets Theoretical Physics: The New Frontier of Molecular Simulation. Springer, Cham.

Bosko, J.T., Todd, B.D., Sadus, R.J., 2004. Viscoelastic properties of dendrimers in the melt from nonequilibrium molecular dynamics. J. Chem. Phys. 121, 12050–12059.

Cheung, P.S.Y., Powles, J.G., 1975. The properties of liquid nitrogen. IV. A computer simulation. Mol. Phys. 30, 921–949.

Chialvo, A.A., Cummings, P.T., 1994. Hydrogen bonding in supercritical water. J. Chem. Phys. 101, 4466–4469.

Chialvo, A.A., Cummings, P.T., 1999. Molecular-based modeling of water and aqueous solutions at supercritical conditions. Adv. Chem. Phys. 109, 207–433.

Chialvo, A.A., Cummings, P.T., Simonson, J.M., Mesmer, R.E., Cochran, H.D., 1998. Interplay between molecular simulation and neutron scattering in developing new insights into the structure of water. Ind. Eng. Chem. Res. 37, 3921–3025.

Ciccotti, G., Ferrario, M., Ryckaert, J.P., 1982. Molecular dynamics of rigid systems in Cartesian coordinates. A general formulation. Mol. Phys. 47, 1253–1264.

Ciccotti, G., Jacucci, G., McDonald, I.R., 1979. Thought experiments' by molecular dynamics. J. Stat. Phys. 21, 1–22.

Ciccotti, G., Frenkel, D., McDonald, I.R. (Eds.), 1987. Simulation of Liquids and Solids. Molecular Dynamics and Monte Carlo Methods in Statistical Mechanics. North-Holland, Amsterdam.

Daivis, P.J., Evans, D.J., 1995. Temperature dependence of the thermal conductivity for two models of liquid butane. Chem. Phys. 198, 25–34.

Daivis, P.J., Evans, D.J., Morriss, G.P., 1992. Computer simulation of the comparative rheology of branched and linear alkanes. J. Chem. Phys. 97, 616–627.

de Leeuw, S.W., Perram, J.W., Smith, E.R., 1980. Simulation of electrostatic systems in periodic boundary conditions. I. Lattice sums and dielectric constant. Proc. R. Soc. Lond. A373, 27–56.

de Pablo, J.J., 1995. Simulation of phase equilibria for chain molecules. Fluid Phase Equilib. 104, 195–206.

de Pablo, J.J., Prausnitz, J.M., 1989. Phase equilibria for fluid mixtures from Monte Carlo simulation. Fluid Phase Equilib. 53, 177–189.

de Pablo, J.J., Prausnitz, J.M., Strauch, H.J., Cummings, P.T., 1990. Molecular simulation of water along the liquid-vapor coexistence curve from 25°C to the critical point. J. Chem. Phys. 93, 7355–7359.

Deiters, U.K., Sadus, R.J., 2019. Two-body interatomic potentials for He, Ne, Ar, Kr, and Xe from ab initio data. J. Chem. Phys. 150, 134504.

Deiters, U.K., Sadus, R.J., 2022a. Accurate determination of solid-liquid equilibria by molecular simulation: behavior of Ne, Ar, Kr, and Xe from low to high pressures. J. Chem. Phys. 157, 204504.

Deiters, U.K., Sadus, R.J., 2022b. First-principles determination of the solid-liquid-vapor triple point: the noble gases. Phys. Rev. E. 105, 054128.

Deiters, U.K., Sadus, R.J., 2023. An intermolecular potential for hydrogen: classical molecular simulation of pressure-density-temperature behavior, vapor-liquid equilibria, and critical and triple point properties. J. Chem. Phys. 158, 194502.

Eastwood, J.W., Hockney, R.W., Lawrence, D., 1980. P3M3DP-the three dimensional periodic particle-particle/particle-mesh program. Comp. Phys. Commun. 19, 215–261.

Escobedo, F.A., de Pablo, J.J., 1996. Simulation and prediction of vapour-liquid equilibria for chain molecules. Mol. Phys. 87, 347–366.

Evans, D.J., Morriss, G.P., 1984. Non-Newtonian molecular dynamics. Comp. Phys. Rep. 1, 297–343.

Evans, D.J., Morriss, G.P., 2008. Statistical Mechanics of Nonequilibrium Liquids, second ed. Cambridge University Press, Cambridge.

Evans, D.J., Murad, S., 1977. Singularity free algorithm for molecular dynamics simulation of rigid polyatomics. Mol. Phys. 34, 327–331.

Fincham, D., Quirke, N., Tildesley, D.J., 1986. Computer simulation of molecular liquid mixtures. I. A diatomic Lennard-Jones model mixture for CO_2/C_2H_6. J. Chem. Phys. 88, 4535–4546.

Frenkel, D., Smit, B., 2023. Understanding Molecular Simulation. From Algorithms to Applications, third ed. Academic Press, San Diego.

Gear, C.W., 1971. Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall, Englewood Cliffs.

Gillan, M.J., Dixon, M., 1983. The calculation of thermal conductivities by perturbed molecular dynamics simulation. J. Phys. C. 16, 869–878.

Goldstein, H., Poole, C., Safko, J., 2002. Classical Mechanics, third ed. Addison Wesley, San Francisco.

Gosling, E.M., McDonald, I.R., Singer, K., 1973. On the calculation by molecular dynamics of the shear viscosity of a simple fluid. Mol. Phys. 26, 1475–1484.

Greengard, L., Rokhlin, V., 1987. A fast algorithm for particle simulations. J. Comput. Phys. 73, 325–348.

Gubbins, K.E., Sandler, S.I. (Eds.), 1994. Models For Thermodynamic and Phase Equilibria Calculations. Marcel Dekker, New York.

Guissani, Y., Guillot, B., 1993. A computer simulation study of the liquid-vapor coexistence curve of water. J. Chem. Phys. 98, 8221–8235.

Guissani, Y., Guillot, B., 1994. Coexisting phases and criticality in NaCl by computer simulation. J. Chem. Phys. 101, 490–509.

Guo, M.X., Li, Y.G., Li, Z.C., Lu, J.F., 1994. Molecular simulation of liquid-liquid equilibria for Lennard-Jones fluids. Fluid Phase Equilib. 98, 129–139.

Haile, J.M., 1992. Molecular Dynamics Simulation. Elementary Methods. John Wiley & Sons, New York.

Hansen, J.P., McDonald, I.R., 1975. Statistical mechanics of dense ionized mater. IV. Density and charge fluctuations in a simple molten salt. Phys. Rev. A 11, 2111–2123.

Hansen, J.P., Verlet, L., 1969. Phase transitions of the Lennard-Jones system. Phys. Rev. 184, 151–161.

Harp, G.D., Berne, B.J., 1968. Linear and angular momentum autocorrelation functions in diatomic liquids. J. Chem. Phys. 49, 1249–1254.

Heinecke, A., Eckhardt, W., Horsch, M., Bungartz, H.-J., 2015. Supercomputing for Molecular Dynamics Simulations: Handling Multi-Trillion Particles in Nanofluidics. Springer, Heidelberg.

Hellmann, R., Jäjer, B., Bich, E., 2017. State-of-the-art *ab initio* potential energy curve for the xenon atom pair and related spectroscopic and thermophysical properties. J. Chem. Phys. 147, 034304.

Heyes, D.M., 1998. The Liquid State: Applications of Molecular Simulation. John Wiley & Sons, Chichester.

Hoover, W.G., 1985. Canonical dynamics: equilibrium phase-space distribution. Phys. Rev. A 31, 1695–1697.

Hou, Q., Li, M., Zhou, Y., Cui, J., Cui, Z., Wang, J., 2013. Molecular dynamics simulations with many-body potentials on multiple GPUs—The implementation, package and performance. Comput. Phys. Commun. 184, 2091–2101.

Hunt, T.A., Bernardi, S., Todd, B.D., 2010. A new algorithm foe extended nonequilibrium molecular dynamics simulations of mixed flow. J. Chem. Phys. 133, 154116.

Kirk, D.B., Hwu, W.-M.W., 2010. Programming Massively Parallel Processor: A Hands-on Approach. Morgan Kaufmann, Amsterdam.

Klamt, A., 2005. COSMOS-RS: From Quantum Chemistry to Fluid Phase Thermodynamics and Drug Design. Elsevier, Amsterdam.

Kofke, D.A., 1993a. Gibbs-Duhem integration: a new method for direct evaluation of phase coexistence by molecular simulation. Mol. Phys. 78, 1331–1336.

Kofke, D.A., 1993b. Direct evaluation of phase coexistence by molecular simulation via integration along the coexistence line. J. Chem. Phys. 98, 2704–2719.

Kohnke, B., Kutzner, C., Grubmüller, J., 2020. A GPU-accelerated fast multiple method for GROMAS: Performance and accuracy. J. Chem. Thoer. Comput. 16, 6938–6949.

Kriebel, C., Müller, A., Winkelmann, J., Fischer, J., 1995. Vapour-liquid equilibria of two-centre Lennard-Jones fluids from the NpT plus test particle method. Mol. Phys. 84, 381–394.

Ladd, A.J.C., 1977. Monte Carlo simulation of water. Mol. Phys. 33, 1039–1050.

Ladd, A.J.C., 1978. Long-range dipolar interactions in computer simulations of polar liquids. Mol. Phys. 36, 463–474.

Leach, A.R., 2001. Molecular Modelling: Principles and Applications, second ed. Prentice Hall, Harlow.

Lee, J.-S., Wick, C.D., Stubbs, J.M., Siepmann, J.I., 2005. Simulating the vapour-liquid equilibria of cyclic alkanes. Mol. Phys. 103, 99–104.

Lord Kelvin, 1904. Baltimore Lectures on Molecular Dynamics and the Wave Theory of Light. C. J. Clay and Sons, Cambridge University Press, London, p. 522.

Lustig, R., 2011. Direct molecular NVT simulation of the isobaric heat capacity, speed of sound and Joule-Thomson coefficient. Mol. Sim. 37, 457–465.

Mansfield, M.L., Klushin, L.I., 1993. Monte Carlo studies of dendrimer macromolecules. Macromolecules 26, 4262–4268.

Marcelli, G., Sadus, R.J., 1999. Molecular simulation of the phase behavior of noble gases using accurate two-body and three-body intermolecular potentials. J. Chem. Phys. 111, 1533–1540.

Marcelli, G., Sadus, R.J., 2000. A link between the two-body and three-body interaction energies of fluids from molecular simulation. J. Chem. Phys. 112, 6382–6385.

Martin, M.G., 2006. Comparison of the AMBER, CHARMM, COMPASS, GROMOS, OPLS, TraPPE and UFF force fields for the prediction of vapor-liquid coexistence curves and liquid densities. Fluid Phase Equilib. 248, 50–55.

McDonald, I.R., 1972. NpT-ensemble Monte Carlo calculations for binary liquid mixtures. Mol. Phys. 23, 41–58.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., 1953. Equation of state calculations by fast computing machines. J. Chem. Phys. 21, 1087–1092.

Mezei, M., 1980. A cavity-biased (TVμ) Monte Carlo method for the computer simulation of fluids. Mol. Phys. 40, 901–906.

Möller, D., Fischer, J., 1990. Vapour liquid equilibrium of a pure fluid from test particle method in combination with NpT molecular dynamics simulations. Mol. Phys. 69, 463–473.

Möller, D., Óprzynski, J., Müller, A., Fischer, J., 1992. Prediction of thermodynamic properties of fluid mixtures by molecular dynamics simulations: methane-ethane. Mol. Phys. 75, 363–378.

Monson, A.P., Rigby, M., Steele, W.A., 1983. Non-additive energy effects in molecular liquids. Mol. Phys. 49, 893–898.

Morriss, G.P., Daivis, P.J., Evans, D.J., 1991. The rheology of n alkanes: decane and eicosane. J. Chem. Phys. 94, 7420–7433.

Mutō, Y. (1943). On the forces acting between nonpolar molecules. *J. Phys.-Math. Soc. Japan* 17, 629–631. [This paper is most often incorrectly cited in the literature as Muto, Y. (1943). *Proc. Phys. Math. Soc. Japan* 17, 629. The correct paper (in Japanese) is freely available from https://doi.org/10.11429/subutsukaishi1927.17.10-11-12_629.]

Nosé, S., 1984. A unified formulation of the constant temperature molecular dynamics methods. J. Chem. Phys. 81, 511–519.

Nosé, S., Klein, M.L., 1983. Constant pressure molecular dynamics for molecular systems. Mol. Phys. 50, 1055–1076.

Ohno, K., Shida, K., Kmiura, M., Kawazoe, Y., 1996. Monte Carlo study of the second virial coefficient of star polymers in a good solvent. Macromolecules 29, 2269–2274.

Orkoulas, G., Panagiotopoulos, A.Z., 1994. Free energy and phase equilibria for the restricted primitive model of ionic fluids from Monte Carlo simulations. J. Chem. Phys. 101, 1452–1459.

Palmer, B.J., Lo, C., 1994. Molecular dynamics implementation of the Gibbs ensemble calculation. J. Chem. Phys. 101, 10899–10907.

Panagiotopoulos, A.Z., 1987. Direct determination of phase coexistence properties of fluids by Monte Carlo simulation in a new ensemble. Mol. Phys. 61, 813–826.

Panagiotopoulos, A.Z., 1992. Direct determination of fluid phase equilibria by simulation in the Gibbs ensemble: a review. Mol. Sim. 9, 1–23.

Panagiotopoulos, A.Z., Quirke, N., Stapleton, M., Tildesley, D.J., 1988. Phase equilibria by simulation in the Gibbs ensemble. Alternative derivation, generalization and application to mixture and membrane equilibria. Mol. Phys. 63, 527–546.

Panagiotopoulos, A.Z., Wong, V., Floriano, M.A., 1998. Phase equilibria of lattice polymers from histogram reweighting Monte Carlo simulations. Macromolecules 31, 912–918.

Phillips, J.C., Hardy, D.J., Maia, J.D.C., Stone, J.E., Ribeiro, J.V., Bernardi, R.C., Buch, R., Fiorin, G., Hénin, J., Jiang, W., McGreevy, R., Melo, M.C.R., Radak, B.K., Skeel, R.D., Singharoy, A., Wang, Y., Roux, B., Aksimentiev, A., Luthey-Schulten, Z., Kalé, L.V., Schulten, K., Chipot, C., Tajkhorshid, E., 2020. Scalable molecular dynamics on CPU and GPU architectures with NAMD. J. Chem. Phys. 153, 044130.

Postorino, P., Tromp, R.H., Ricci, M.-A., Soper, A.K., Neilson, G.W., 1993. The interatomic structure of water at supercritical temperatures. Nature 366, 668–670.

Raabe, G., 2017. Molecular Simulation Studies on Thermophysical Properties with Application to Working Fluids. Springer, Berlin.

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

Rahman, A., Stillinger, F.H., 1971. Molecular dynamics study of liquid water. J. Chem. Phys. 55, 3336–3359.

Rapaport, D.C., 2004. The Art of Molecular Dynamics Simulation, second ed. Cambridge University Press, Cambridge.

Ray, J.R., 1991. Microcanonical ensemble Monte Carlo method. Phys. Rev. A 44, 4061–4064.

Rittger, E., 1990a. The chemical potential of liquid xenon by computer simulation. Mol. Phys. 69, 853–865.

Rittger, E., 1990b. Can three-atom potentials be determined from thermodynamic data? Mol. Phys. 69, 867–894.

Rosenbluth, M.N., Rosenbluth, A.W., 1955. Monte Carlo calculations of the average extension of molecular chains. J. Chem. Phys. 23, 356–359.

Ryckaert, J.P., Bellemans, A., 1975. Molecular dynamics of liquid n-butane near its boiling point. Chem. Phys. Lett. 30, 123–125.

Ryckaert, J.P., Bellemans, A., 1978. Molecular dynamics of liquid alkanes. Chem. Soc. Faraday Discuss. 66, 95–106.

Sadus, R.J., Prausnitz, J.M., 1996. Three-body interactions in fluids from molecular simulation: vapor-liquid phase coexistence of argon. J. Chem. Phys. 104, 4784–4787.

Schaink, H.M., Hoheisel, C., 1992. The phase-behavior of Lennard-Jones mixtures with nonadditive hard cores: comparison between molecular dynamic calculations and perturbation theory. J. Chem. Phys. 97, 8561–8567.

Shing, K.S., Gubbins, K.E., 1982. The chemical potential in dense fluids and fluid mixtures via computer simulation. Mol. Phys. 46, 1109–1128.

Shvab, I., Sadus, R.J., 2016. Atomistic water models: aqueous thermodynamic properties from ambient to supercritical conditions. Fluid Phase Equilib. 407, 7–30.

Siepmann, J.I., Frenkel, D., 1992. Configurational-bias Monte Carlo: a new sampling scheme for flexible chains. Mol. Phys. 75, 59–70.

Siepmann, J.I., Karaborni, S., Smit, B., 1993. Simulating the critical behaviour of complex fluids. Nature 365, 330–332.

Singer, K., Taylor, A., Singer, J.V.L., 1977. Thermodynamic and structural properties of liquids modelled by 'two-Lennard-Jones centres' pair potentials. Mol. Phys. 33, 1757–1795.

Smit, B., Karaborni, S., Siepmann, J.I., 1995. Computer simulation of vapor-liquid phase equilibria of n-alkanes. J. Chem. Phys. 102, 2126–2140.

Smith, S.W., Hall, C.K., Freeman, B.D., 1996. Molecular dynamics study of entangled hard-chain fluids. J. Chem. Phys. 104, 5615–5637.

Soper, A.K., Bruni, F., Ricci, M.A., 1997. Site-site pair correlation functions of water from 25 to 400 °C: revised analysis of new and old diffraction data. J. Chem. Phys. 106, 247–254.

Strauch, H.J., Cummings, P.T., 1993. Gibbs ensemble simulation of mixed solvent electrolyte solutions. Fluid Phase Equilib. 86, 147–172.

Thompson, A.P., Aktulga, H.M., Berger, R., Bolintineanu, D.S., Brown, W.M., Crozier, P.S., 't Veld, P.J., Kohlmeyer, A., Moore, S.G., Nguyen, T.C., Shan, R., Stevens, M.J., Tranchida, J., Trott, C., Plimpton, S.J., 2022. LAMMPS – a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Comp. Phys. Commun. 271, 108171.

Tobita, H., 1996. Random degradation of branched polymers. 1. Star polymers. Macromolecules 29, 3000–3009.

Todd, B.D., Daivis, P.J., 2017. Nonequilibrium Molecular Dynamics: Theory, Algorithms and Applications. Cambridge University Press, Cambridge.

Tsang, P.C., Omar, N.W., Perigard, B.Y., Vega, L.F., Panagiotopoulos, A.Z., 1995. Phase equilibria in ternary Lennard-Jones systems. Fluid Phase Equilib. 107, 31–43.

Valleau, J.P., Cohen, L.K., 1980. Primitive model electrolytes. I. Grand canonical Monte Carlo computations. J. Chem. Phys. 72, 5935–5941.

Verlet, L., 1967. Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. Phys. Rev. 159, 98–103.

Wang, L., Sadus, R.J., 2006. Three-body interactions and solid-liquid Phase equilibria: application of a molecular dynamics algorithm. Phys. Rev. E 74, 031203.

Widom, B., 1963. Some topics in the theory of fluids. J. Chem. Phys. 39, 2808–2812.

Widom, B., 1982. Potential distribution theory and the statistical mechanics of fluids. J. Phys. Chem. 86, 869–872.

Wood, W.W., Jacobson, J.D., 1957. Preliminary results from a recalculation of the Monte Carlo equation of state of hard spheres. J. Chem. Phys. 27, 1207–1208.

Wood, W.W., Parker, F.R., 1957. Monte Carlo equation of state of molecules interacting with the Lennard-Jones potential. I. A supercritical isotherm at about twice the critical temperature. J. Chem. Phys. 27, 720–733.

Yao, J., Greenkorn, R.A., Chao, K.C., 1982. Monte Carlo simulation of the grand canonical ensemble. Mol. Phys. 46, 587–594.

Chapter 2

Ensembles, thermodynamic averages, and particle dynamics

The principles of statistical thermodynamics are fundamental to both Monte Carlo (MC) and molecular dynamics (MD) simulation methods. It can be argued that molecular simulation is computational statistical mechanics. Molecular simulation allows us to evaluate exactly the properties of molecules as described by statistical mechanics. Statistical mechanics relates macroscopic properties to the properties of individual molecules. In turn, the properties of individual molecules can be evaluated directly using molecular simulation methods. In a MC simulation, the evaluation of the properties of molecules is performed independently of the dynamics of motion. In contrast, knowledge of particle dynamics is essential for MD simulation.

Very comprehensive and detailed descriptions of statistical mechanics are available elsewhere (Fowler and Guggenheim, 1956; Garrod, 1995; Hill, 1986; Hill, 1987; Mayer and Mayer, 1946; Reed and Gubbins, 1973; Reif, 1965; Rushbrooke, 1962; Wannier, 1987). Section 2.1 provides a brief overview of the results of statistical mechanics that are used commonly in molecular simulation programs. The concept of an ensemble is reviewed and various common thermodynamic properties are defined in terms of ensemble averages. These properties can be determined via molecular simulation by evaluating the behavior of suitable intermolecular potentials. Maitland et al. (1981) have provided a detailed description of several intermolecular potentials, and this important aspect is discussed further in Chapters 3 and 4.

Obtaining ensemble averages of pressure and energy is straightforward, whereas historically averages of other thermodynamic properties were obtained via fluctuations (Section 2.2). However, a more contemporary approach (Lustig, 1994a; Meier and Kabelac, 2006) is discussed in Section 2.3, which offers considerable advantages, arguably providing the preferred method for the future determination of thermodynamic properties via simulation.

An overview of the underlying mechanics of particle motion is presented in Section 2.4. In a sense, this can be regarded as a primer for MD because an understanding of the mechanics of particle motion is the cornerstone of the implementation of MD algorithms. The motion of particles can be described in the framework of Newtonian, Lagrangian, or Hamiltonian mechanics. Here, we

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00016-2

illustrate the relationship between these mechanical frameworks. Starting from Newton's equation, first the Lagrangian equations and then the Hamiltonian equations are obtained. This section follows closely the work of Goldstein (1950). Although Newton's equations are sufficient for simple atomic systems, Lagrangian or Hamiltonian dynamics is advantageous for more complicated systems. Lagrangian and Hamiltonian dynamics are discussed comprehensively elsewhere (Calkin, 1996), and there are several books that describe particle mechanics in the context of molecular simulation (Haile, 1992; Hoover, 1986; Hoover, 1991; Rapaport, 2004; Allen and Tildesley, 2017).

2.1 Statistical mechanics, ensembles, and averaging

2.1.1 Ensembles

Molecular simulation permits the rigorous application of the principles of statistical mechanics to calculate the properties of a chosen system. A typical simulation program requires the implementation of several algorithms involving the calculation of energies, forces, molecular orientation, and the like. However, the overall simulation framework is determined by the choice of an ensemble (Chapter 9).

From the perspective of statistical mechanics, a system's macroscopic properties such as pressure or density represent averages over all possible quantum states. To calculate the properties of a real system, it is convenient to define an ensemble. An ensemble can be regarded as an imaginary collection of a very large number of systems in different quantum states with common macroscopic attributes. For example, each system of the ensemble must have the same temperature (T), pressure (p), and number of particles (N) as the real system it represents. The ensemble average (Reed and Gubbins, 1973; Hill, 1986) of any dynamic property (A) can be obtained from the relationship,

$$\langle A \rangle = \sum_i A_i \mathbb{P}_i, \tag{2.1}$$

where A_i is the value of A in a quantum state i; $\mathbb{P}_i$ represents the probability of observing the ith state, and the angular brackets denote an ensemble average. The time-averaged properties of the real system are related to the ensemble average by invoking the following assumption:

$$A_{t=\infty} = \langle A \rangle. \tag{2.2}$$

The above equivalence of the time average and ensemble average is called the ergodic hypothesis (Haile, 1992).

The form of $\mathbb{P}_i$ is determined by which macroscopic properties are common for every system of the ensemble. For example, if the volume (V), T, and N are constant, then it can be shown that,

$$\mathbb{P}_i = \frac{e^{-\beta E_i(N,V)}}{Z_{NVT}}, \tag{2.3}$$

where E is the energy, $\beta = 1/kT$ (k is Boltzmann's constant), and Z_{NVT} is the canonical partition function.

$$Z_{NVT} = \sum_i e^{-\beta E_i(N,V)} \tag{2.4}$$

Eq. (2.3) represents the canonical ensemble. Other ensembles are possible depending on the choice of constant macroscopic properties (Hill, 1986). Some commonly used ensembles are summarized in Table 2.1, where δ is Dirac's delta function.

It is apparent that the ensembles summarized in Table 2.1 occur in two broad categories. The microcanonical, canonical, and isothermal–isobaric ensembles describe closed systems for which there is no change in the number of particles. In contrast, the grand canonical ensemble is appropriate for an open system in which the number of particles can change. In the thermodynamic limit, all of the ensembles are equivalent, and it is also possible to transform between different ensembles (Allen and Tildesley, 2017; Raabe, 2017). The choice of ensemble for a simulation is often a matter of convenience. Many physical processes do not involve a change in N and occur at either constant T and V, or constant T and p. In these cases, the natural choices are the canonical and isothermal–isobaric ensembles, respectively. It is common practice to use the constraints as a shorthand notation for the ensembles.

Each ensemble is associated with a characteristic thermodynamic function. The entropy (S) can be obtained directly from the microcanonical partition function.

$$S = k\ln Z_{NVE} \tag{2.5}$$

The Helmholtz function (A) is associated with the canonical ensemble.

$$A = -kT\ln Z_{NVT} \tag{2.6}$$

The isobaric–isothermal ensemble yields the definition for the Gibbs function (G),

$$G = -kT\ln Z_{NpT}, \tag{2.7}$$

TABLE 2.1 Summary of common statistical ensembles.

Ensemble	Constraints	Z	$\mathbb{P}_i$
Microcanonical	N, V, E	$\sum_i \delta(E_i - E)$	$\frac{\delta(E_i - E)}{Z_{NVE}}$
Canonical	N, V, T	$\sum_i e^{-\beta E_i(N,V)}$	$\frac{e^{-\beta E_i(N,V)}}{Z_{NVT}}$
Grand canonical	μ, V, T	$\sum_i e^{\beta N_i \mu} Z_{NVT}$	$\frac{e^{-\beta(E_i - \mu N_i)}}{Z_{\mu VT}}$
Isothermal–isobaric	N, p, T	$\sum_i e^{\beta p V_i} Z_{NVT}$	$\frac{e^{-\beta(E_i + pV_i)}}{Z_{NpT}}$

where p is the appropriate thermodynamic function for the grand canonical ensemble.

$$pV = -kT\ln Z_{\mu VT} \tag{2.8}$$

2.1.2 Simple thermodynamic averages

2.1.2.1 Energy

The total energy (E) (or Hamiltonian) of the ensemble can be obtained from the kinetic (K) and potential energies (U)

$$E = \langle K \rangle + \langle U \rangle, \tag{2.9}$$

where the angled brackets denote ensemble averages.

The kinetic energy can be evaluated from the individual momenta of the particles. The potential energy is the sum of the interparticle interactions (Chapter 3). As discussed subsequently, the interparticle interactions are usually calculated by assuming a particular intermolecular function. Typically, the calculation involves a summation over all distinct pairs of molecules (Chapter 3), although other many-body interactions (Chapter 4) can be involved.

2.1.2.2 Temperature

The temperature can be obtained (Allen and Tildesley, 2017) from the virial theorem. In terms of generalized momenta (p_k), the theorem can be expressed as:

$$\left\langle p_k \frac{\partial E}{\partial p_k} \right\rangle = kT \tag{2.10}$$

For N atoms, each with three degrees of freedom, we obtain the relationship.

$$T = \frac{2\langle K \rangle}{3Nk} \tag{2.11}$$

Alternatively, we can consider T to be the average of instantaneous temperature (t) contributions.

$$t = \frac{2K}{3Nk} \tag{2.12}$$

2.1.2.3 Pressure

The pressure can also be obtained (Allen and Tildesley, 2017) from the virial theorem. In terms of generalized coordinates (q_k), the virial theorem can be expressed as:

$$\left\langle q_k \frac{\partial E}{\partial q_k} \right\rangle = kT \tag{2.13}$$

Starting from Eq. (2.13), it can be shown (Allen and Tildesley, 2017) that,

$$pV = NkT + \langle W \rangle, \tag{2.14}$$

where W is the internal virial (defined below). We can define an instantaneous pressure function analogous to the instantaneous temperature.

$$p = \rho kt + \frac{W}{V} \tag{2.15}$$

The internal virial is defined as,

$$W = \frac{1}{3}\sum_i \sum_{j>i} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} = -\frac{1}{3}\sum_i \sum_{j>i} w(r_{ij}), \tag{2.16}$$

where $w(r)$ is the intermolecular pair virial function and i and j denote the atoms or molecules.

$$w(r) = r\frac{du(r)}{dr} \tag{2.17}$$

Eq. (2.17) introduces the intermolecular potential function ($u(r)$) into the calculation of the ensemble pressure. The potential pressure can be identified as $p_{pot} = W/V$.

2.1.2.4 Residual chemical potential

The residual chemical potential (μ_r) is obtained typically from the insertion of a test particle (Widom, 1963, 1982). A ghost test particle is inserted randomly into the system and the resulting change in potential energy (U_{test}) is calculated. The formula for calculating μ_r is ensemble-dependent. For the canonical (Widom, 1963) and grand canonical (Henderson, 1983) ensembles we have:

$$\mu_r = -kT\ln\left\langle \exp(-\beta U_{test})\right\rangle \tag{2.18}$$

Fluctuations of the kinetic temperature mean that the corresponding formula (Frenkel, 1986) for the microcanonical ensemble requires the instantaneous temperature:

$$\mu_r = -k\langle t\rangle \ln\left[\frac{\left\langle t^{3/2}\exp\left(-\frac{U_{test}}{kt}\right)\right\rangle}{\langle t\rangle^{3/2}}\right] \tag{2.19}$$

For the isothermal−isobaric ensemble, allowance must be made for volume fluctuations (Shing and Chung, 1987).

$$\mu_r = -kT\ln\left[\frac{\left\langle V\exp(-\beta U_{test})\right\rangle}{\langle V\rangle}\right] \tag{2.20}$$

The total configurational chemical potential can be obtained by adding the ideal contribution (see below). For atoms, it is,

$$\mu_{id} = -kT\ln\left\{\frac{N}{V}\left(\frac{2\pi mkT}{h}\right)^{3/2}\right\}, \tag{2.21}$$

TABLE 2.2 Fluctuation formulas for the C_V of atoms in different ensembles.[a]

Ensemble	Fluctuation formula
NVE	$\left(\frac{2}{3Nk}-\frac{4\langle\delta Y^2\rangle}{9k^3N^2T^2}\right)^{-1}=\left(\frac{2}{3Nk}-\frac{4\left(\langle Y^2\rangle-\langle Y\rangle^2\right)}{9k^3N^2T^2}\right)^{-1}$
NVT	$\frac{\langle\delta U^2\rangle}{kT^2}+\frac{3Nk}{2}=\frac{\langle U^2\rangle-\langle U\rangle^2}{kT^2}+\frac{3Nk}{2}$
μVT	$\frac{1}{kT^2}\left(\langle\delta U^2\rangle-\frac{\langle\delta U\delta N\rangle^2}{\langle\delta N^2\rangle}\right)+\frac{3Nk}{2}$ $=\frac{1}{kT^2}\left(\langle U^2\rangle-\langle U\rangle^2-\frac{\langle\langle UN\rangle-\langle U\rangle\langle N\rangle\rangle^2}{\langle N^2\rangle-\langle N\rangle^2}\right)+\frac{3Nk}{2}$

[a] *Y can be either* K *or* U. *The formulas include the contribution for the ideal gas term for atoms that should be replaced for molecules.*

where m and h are the mass of a single atom and Planck's constant, respectively. The additional degrees of freedom of molecules means that the ideal contribution to molecular properties should also include contributions from rotation and vibration. Although well-known statistical relationships exist (Reed and Gubbins, 1973), it has become common practice to deduce empirical relationships for real molecules (Lemmon et al., 2009).

2.2 Properties from fluctuations

During the course of a simulation, fluctuations will be observed (Lebowitz et al., 1967) in various properties of the ensemble, for example, energy fluctuations occur in the canonical ensemble. These fluctuations are useful because they can be related to thermodynamic derivatives such as specific heat and isothermal compressibility. It is particularly useful to calculate the root mean square (RMS) deviation of an ensemble property,

$$\sigma^2(A)=\left\langle\delta A^2\right\rangle=\left\langle A^2\right\rangle-\langle A\rangle^2, \tag{2.22}$$

where

$$\delta A=A-\langle A\rangle \tag{2.23}$$

It should be noted that the fluctuations in different ensembles are not the same. This is exemplified in Table 2.2 for the isochoric heat capacity (C_V) of atoms.

TABLE 2.3 Thermodynamic definitions of various properties and their evaluation in the *NpT* ensemble using fluctuation formulas.[a]

Property	Thermodynamic definition	*NpT* ensemble fluctuation formula
Isochoric heat capacity (C_V)	$\left(\frac{\partial(E=U+K)}{\partial T}\right)_V$	$C_p - TV\alpha_p^2/\beta_T$
Isobaric heat capacity (C_p)	$\left(\frac{\partial(H=U+K+pV)}{\partial T}\right)_p$	$\frac{\left\langle \delta(H=U+K+pV)^2 \right\rangle}{kT^2}$
Isothermal compressibility (β_T)	$-\frac{1}{V}\left(\frac{\partial V}{\partial p}\right)_T$	$\frac{\langle \delta V^2 \rangle}{kTV}$
Adiabatic compressibility (β_S)	$-\frac{1}{V}\left(\frac{\partial V}{\partial p}\right)_S$	$\beta_T - \frac{TV\alpha_p^2}{C_p}$
Thermal pressure coefficient (γ_V)	$\left(\frac{\partial p}{\partial T}\right)_V$	$\frac{\alpha_p}{\beta_T}$
Thermal expansion coefficient (α_p)	$\frac{1}{V}\left(\frac{\partial V}{\partial T}\right)_p$	$\frac{\langle \delta V \delta(H=U+K+pV) \rangle}{kT^2V}$
Zero frequency speed of sound (w_0)	$\sqrt{-\frac{V^2}{NM}\left(\frac{\partial p}{\partial V}\right)_S}$	$w_0 = \sqrt{\frac{V}{NM\beta_S}}$
Joule–Thomson coefficient (μ_{JT})	$\left(\frac{\partial T}{\partial p}\right)_H$	$\frac{\gamma_V - 1/T\beta_T}{\gamma_V^2 + \frac{NC_V}{VT\beta_T}}$

[a]M *is the total mass of the system.*

2.2.1 Thermodynamic properties

The thermodynamic definition of various properties is given in Table 2.3. These properties involve evaluating thermodynamic derivatives, which comprise fluctuations of quantities such as energy, pressure, volume, and number of particles. The evaluation of some thermodynamic properties is "natural" in a particular ensemble because of the different constraints (Table 2.1) that determine which quantities can fluctuate. For example, C_V is easily calculated in the *NVT* ensemble, whereas there is a straightforward relationship for C_p in the *NpT* ensemble. Nonetheless, all the thermodynamic quantities can be obtained from any ensemble. That is, none are specifically limited to a particular choice ensemble, which is simply a matter of preference or convenience rather than any theoretical restriction.

An explanation of fluctuation formulas for different ensembles is given elsewhere (Adams, 1975; Allen and Tildesley, 2017; Brown, 1958; Lebowitz et al., 1967; Rowlinson, 1969), and the relationships for the *NpT* ensemble (Brown, 1958) are summarized in Table 2.3. In the *NpT* ensemble, it is

natural to calculate C_p, β_T, and α_p, which involve δV and δH. Having determined these key properties, all other thermodynamic properties can be obtained using the relationships in Eq. (2.24) (Rowlinson, 1969; Münster, 1970), which is the basis of the formulas for C_V, w_0, and μ_{JT} in Table 2.3. In general, for a given ensemble, only three key properties are required from molecular simulation to determine all of the remaining thermodynamic properties. That is, C_V, β_T and γ_V for the *NVE*, *NVT*, and μVT ensembles and C_p, β_T, and α_p for the *NpT* and *NpH* ensembles, respectively.

$$\left.\begin{aligned} C_p &= C_V + \frac{V\gamma_V^2}{Nk} \\ &= C_V + \frac{TV\alpha_p^2}{\beta_T} \\ &= C_V + TV\beta_T\gamma_V^2 \\ &= C_V + TV\alpha_p\gamma_V \\ &= C_V\left(\beta_T/\beta_s\right) \\ \beta_S^{-1} &= \beta_T^{-1} + VT\gamma_V^2/C_V \\ \mu_{JT} &= \frac{\gamma_V - 1/\beta_T T}{\gamma_V^2 + \dfrac{NC_V}{VT\beta_T}} \\ \alpha_p &= \beta_T\gamma_V \\ w_0 &= \sqrt{\frac{V}{NM\beta_S}} \end{aligned}\right\} \tag{2.24}$$

The definition of both E and enthalpy (H) involves a contribution to the kinetic energy. This can be explicitly calculated in MD but not in MC. In the latter case, the contribution is typically incorporated separately as an ideal term, and the MC calculation involves the evaluation of a residual quantity only. Therefore it is also often useful (Lagache et al., 2001) to separate the contribution of a thermodynamic property (X) into its residual (X_r) and ideal components (X_{id}) because the value of the latter will depend on the number of degrees of freedom (f) of the molecule.

$$X = X_r + X_{id} \tag{2.25}$$

This separation of contributions is evident in the alternative formulas for C_V given for atoms ($f = 3$) in Table 2.2 with the terms involving U providing the residual term. The residual term arises from interactions between particles that can be calculated via an intermolecular potential (Chapter 3). As such it can also be described as either the potential or configurational contribution. The residual contribution is sometimes misidentified in the literature as an excess property. In chemical thermodynamics, the term "excess" should be strictly reserved for the specific case

of mixture properties: the difference between the property of a mixture of two or more components and the property of an ideal *solution* at the same T and p (Prigogine and Defay, 1954; Hansen and McDonald, 2013; Prausnitz et al., 1999).

Formulas for the ensemble averages of the thermodynamic properties in the literature typically include the contributions to K from the three degrees of freedom of translational motion of an atom that arises from the equipartition theorem, that is, $K = K_{id} = 3kT/2$. The additional degrees of freedom of molecular systems mean that ideal contributions to thermodynamic properties account for nontranslational contributions such as vibration and rotation. For the properties of molecules composed of n-atoms, the contribution to K of rotation must be included. That is, for linear and nonlinear molecules, adding the contribution from rotation yields, $K_{id} = 5kT/2$ (two additional degrees of freedom) and $K_{id} = 3kT$ (three additional degrees of freedom), respectively. Separately identifying only the residual contribution to the thermodynamic property means that the molecule-specific ideal component can be added as required to obtain the full value. It reduces the likelihood of unwittingly either underestimating (by using the in-formula atomic value for a molecule) or overestimating (by adding the molecular value to the in-formula atomic value) the contribution of K for molecules.[1]

For a linear n-atom molecule, application of the equipartition theorem yields $C_{V,\,id} = 5k/2 + (3n - 5)k$, whereas the value for an n-atom nonlinear molecule is $C_{V,\,id} = 3k + (3n - 6)k$. The second term in these equations is the contribution from vibrations. However, in many cases, high temperatures are usually required to even partially observe these contributions because vibrational motions are not fully excited at low to moderate temperatures. In most cases deviations from the constant values can be exclusively attributed to vibrations. Therefore it is normally sufficient to account for only translational and rotational contributions. That is, $C_{V,\,id} = 5k/2$ and $C_{V,\,id} = 3k$ for linear and nonlinear n-atom molecules, respectively.

Alternatively, the ideal contribution for molecules is sometimes estimated (Raabe, 2017; Elliott et al., 2023) from either experimental data or theory, which is particularly useful for obtaining reliable values for large polyatomic molecules. In contrast, large deviations from the ideal gas formula for small molecules are usually only observed at high T with the onset of significant vibrational contributions. However, water is an exception. Fig. 2.1 compares the calculated (Friedman and Haar, 1954) ideal gas C_p of water ($n = 3$) to contributions from the simple equipartition formula, that is, $C_{p,\,id} = C_{V,\,id} + k = 4k$. It is apparent that significant deviations occur for $T > 200$ K.

1. An insidious problem in the literature is that formulas originally derived for atoms are applied unmodified to molecules. To add to the confusion, atoms are sometimes also incorrectly referred to as molecules and vice versa.

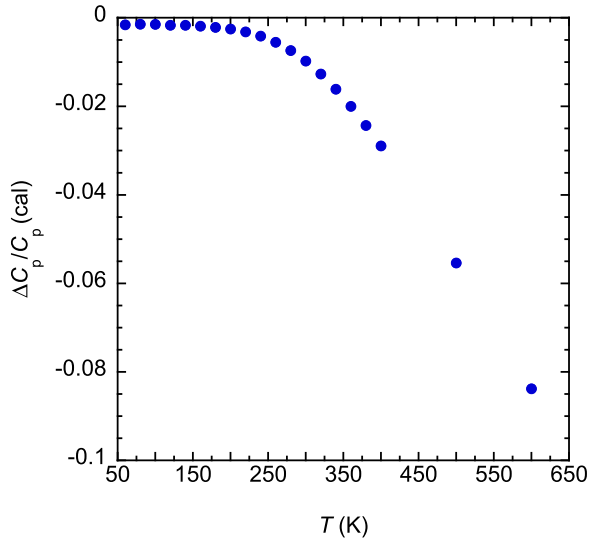


FIGURE 2.1 Relative difference between the ideal gas formula for the C_p of water and accurate calculations. The increased deviations at $T > 200$ K reflect the commencement of significant vibrational contributions.

2.2.2 Transport coefficients

In contrast to MC, MD simulation allows us to determine the time-dependent properties of fluids such as transport coefficients. Here, we simply state the results that connect molecular simulation with transport properties. A theoretical justification can be found elsewhere (Rapaport, 2004; Hansen and McDonald, 2013; Allen and Tildesley, 2017). Hoheisel (1994) has reviewed the use of MD simulation for the transport properties of molecular fluids.

2.2.2.1 Diffusion

The coefficient of diffusion (D) can be obtained (Rapaport, 2004) from,

$$D = \lim_{t\to\infty} \frac{1}{6Nt} \left\langle \sum_{j=1}^{N} \left| r_j(t) - r_j(0) \right|^2 \right\rangle, \tag{2.26}$$

where t denotes time. To evaluate correctly the right-hand side of Eq. (2.26), account must be taken of the superfluous folding back of particle coordinates common in the application of periodic boundary conditions. If we assume that the displacement per time step is small relative to the size of the system,

the true atom displacements (r') can be obtained from the following relationship (Rapaport, 2004),

$$r'_{xj}(t) = r_{xj}(t) + \text{nint}\left(\frac{r'_{xj}(t-\Delta t) - r_{xj}(t)}{L_x}\right)L_x, \tag{2.27}$$

where nint(x) is the nearest integer to x.

2.2.2.2 Shear viscosity

The shear viscosity (η) is calculated (Rapaport, 2004) from:

$$\eta = \lim_{t\to\infty}\frac{1}{6kTVt}\left\langle\sum_{x<y}\left[\sum_j m_j r_{xj}(t)v_{yj}(t) - \sum_j m_j r_{xj}(0)v_{yj}(0)\right]^2\right\rangle \tag{2.28}$$

The reason for the sum $\Sigma_{x<y}$ over the three pairs of distinct vector components xy, yz, and zx in Eq. (2.28) is to improve the accuracy of the calculation. It is not required by theory and it can be omitted.

2.2.2.3 Thermal conductivity

The thermal conductivity (λ) is given (Rapaport, 2004) by:

$$\lambda = \lim_{t\to\infty}\frac{1}{6kT^2Vt}\left\langle\sum_x\left[\sum_j r_{xj}(t)e_j(t) - \sum_j r_{xj}(0)e_j(0)\right]^2\right\rangle \tag{2.29}$$

In Eq. (2.29), Σ_x is the sum over all vector components and e is the instantaneous excess energy of atom j defined by,

$$e_j = \frac{mv_j^2}{2} + \frac{\sum_{i>j}u(r_{ij}) - \langle e\rangle}{2}, \tag{2.30}$$

where $\langle e\rangle$ is the mean energy.

2.2.3 Molecular simulation and choice of ensemble

In the thermodynamic limit all ensembles are equivalent (Hill, 1986) and should yield identical outcomes. However, molecular simulations are conducted for a finite number of particles and care should always be taken to examine the influence of finite size on a given property. In practice, this means performing simulations with a varying number of particles to gauge the influence on particle numbers on a given property.

It is sometimes observed that the pressure calculated from a MD NVT ensemble simulation using the density ($\rho = N/V$) obtained from an NpT ensemble does not exactly match the pressure of the NpT MD simulation. This problem persists irrespective of the number of particles used, and as

such at least partly reflects some deficiencies in the thermostat or barostat (Chapters 7–9). It is an issue that is arguably underreported in the literature, which can be partly attributed to the preference for the *NpT* ensemble and relatively few contemporary comparisons (Bosko et al., 2005) between different ensembles.

2.3 Alternative methods for thermodynamic properties

A disadvantage of the fluctuation approach is that the properties require post-simulation processing to determine the contributions of the fluctuations. This contrasts with quantities such as E and p that are simply calculated during the normal course of the simulation. Lustig (1994a,b,c) showed that, in general, all thermodynamic state variables could be obtained from derivatives determined directly from MD simulations in a classical *NVE***P** ensemble. The nature of MD and MC simulations are discussed in Chapters 6 and 7, respectively.

It should be noted that the formulas for the *NVT*, *NpT*, μVT, and *NpH* ensembles detailed below are strictly only valid for MC because the additional reduction in the degrees of freedom that occur in MD simulations have not been taken into account. However, from a practical perspective, using them for MD simulations is usually unlikely to incur any noticeable loss of accuracy above the statistical uncertainties of the simulation.

2.3.1 MD *NVE* ensemble

The *NVE* ensemble is the natural ensemble to conduct MD because the application of Newton's laws results in the conservation of energy. In addition, momentum ($\mathbf{P} = 0$) is also conserved in MD, which means that strictly speaking MD simulations are conducted in an *NVE***P** ensemble. Meier and Kabelac (2006) observed that the MD *NVE***P** ensemble also conserves another vectorial quantity (**G**) that is related to the initial position of the center of mass, which means that we are dealing with an *NVE***PG** ensemble.

The fundamental equation of state for the system in the (Meier and Kabelac, 2006) *NVE***PG** ensemble is obtained from the entropy postulate,

$$S(N, V, E, \mathbf{P}, \mathbf{G}) = k \ln \Omega(N, V, E, \mathbf{P}, \mathbf{G}), \tag{2.31}$$

where $\Omega(N, V, E, \mathbf{P}, \mathbf{G})$ is the phase-space volume. The basic phase-space functions can be expressed via abbreviations that represent the derivatives of the phase-space volume with respect to the independent thermodynamic state variables,

$$\Omega_{mn} = \frac{1}{\omega} \frac{\partial^{m+n} \Omega}{\partial E^m \partial V^n}, \tag{2.32}$$

where ω is the phase-space density. The general expression is given (Meier and Kabelac, 2006) by,

$$\Omega_{mn} = \left(\frac{N}{V}\right)^n \frac{1}{N^n}(-1)^m \frac{2}{F-d}\left(-\frac{F-d}{2}\right)_m (-1)^n(-[N-1])_n \left\langle K^{-(m-1)}\right\rangle + (1+\delta_{0n})\sum_{i=1}^{n}\binom{n}{i}(-1)^{n-i}(-[N-1])_{n-i}\left(\frac{N}{V}\right)^{n-i}\frac{1}{N^{n-i}} \times \frac{2}{F-d}\sum_{l=1}^{i}(-1)^{m+l}\left(-\frac{F-d}{2}\right)_{m+l}\left\langle K^{-(m+l-1)}\left(\sum_{k=1}^{k\max(i,l)} c_{ilk}W_{ilk}\right)\right\rangle \tag{2.33}$$

which involves the ensemble averages of products of powers of the kinetic energy $K = E - U(\mathbf{r}^N)$ and of volume derivatives of the potential energy $\partial^n U/\partial V^n$. In Eq. (2.33), the total number of degrees of freedom is $F = fN$ as used previously for the conventional fluctuation formulas; $(x)_n = x(x+1)(x+2)\ldots(x+n-1)$ represents the Pochhammer symbol with $(x)_0 = 1$ and δ_{ij} is the Kronecker delta (Abramowitz and Stegun, 1972); c_{ilk} W_{ilk} is a product of certain volume derivatives of the potential energy $W_{ilk} = (-\partial^i U/\partial V^i)(-\partial^k U/\partial V^k)\ldots$ and of multinomial coefficients c_{ilk} described in detail elsewhere (Lustig, 1994a). The $(F-d)/2$ terms accounts for the contribution of the kinetic energy minus the momentum constraint in d dimensions. For molecules it should be replaced by a term that reflects the additional degrees of freedom. The volume derivatives of nth order for the potential energy are given by,

$$\frac{\partial^n U}{\partial V^n} = \frac{1}{3^n V^n}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\sum_{k=1}^{n} a_{nk}\, r_{ij}^k\, \frac{\partial^k u}{\partial r_{ij}^k}, \tag{2.34}$$

where u is the pair potential energy and r_{ij} denotes the distance between particles i and j. A recursion relationship (Lustig, 1994a) is used to determine the a_{nk} coefficients.

The key thermodynamic quantities obtained from this approach are summarized in Table 2.4. The remaining thermodynamic quantities can be obtained by using the relationships given in Eq. (2.24). It should be noted that this approach provides us with useful relationships for both T and p.

The Ω_{nm} terms can be evaluated from Eqs. (2.35) to (2.39):

$$\Omega_{00} = kT = \frac{2}{F-d}\langle K\rangle \tag{2.35}$$

$$\Omega_{10} = 1 \tag{2.36}$$

$$\Omega_{01} = \frac{N-1}{V}kT - \left\langle\frac{\partial U}{\partial V}\right\rangle \tag{2.37}$$

TABLE 2.4 Relationships for the key thermodynamic quantities in terms of phase-space functions for the *NVE*PG MD ensemble.

Quantity	Formula in terms of Ω_{nm}
T	$\frac{\Omega_{00}}{k}$
p	Ω_{01}
C_V	$k\,(1-\Omega_{00}\,\Omega_{20})^{-1}$
γ_V	$k\frac{\Omega_{11}-\Omega_{01}\,\Omega_{20}}{1-\Omega_{00}\,\Omega_{20}}$
β_T^{-1}	$V\left[\frac{\Omega_{01}(2\,\Omega_{11}-\Omega_{01}\Omega_{20})-\Omega_{00}{\Omega_{11}}^2}{1-\Omega_{00}\Omega_{20}}-\Omega_{02}\right]$

$$\Omega_{11}=\frac{N-1}{V}+\left[1-\frac{F-d}{2}\right]\left\langle K^{-1}\left(\frac{\partial U}{\partial V}\right)\right\rangle \tag{2.38}$$

$$\Omega_{20}=-\left[1-\frac{F-d}{2}\right]\left\langle K^{-1}\right\rangle \tag{2.39}$$

For N atoms, each with 3 degrees of translational freedom, $F=3N$ and $d=3$.

This approach has been used to evaluate the thermodynamic properties for a wide variety of cases. Examples include simple potentials (Lustig, 1998; Mausbach and Sadus, 2011); polarizable water models (Shvab and Sadus, 2013); supercritical fluids (Yigzawe and Sadus, 2013); and binary mixtures (Shvab and Sadus, 2014; Stiegler and Sadus, 2015). The ease of integrating this approach into a conventional *NVE* MD algorithm (Chapter 10) provides a very compelling argument that it should be the method of choice for determining thermodynamic quantities from molecular simulation. The only significant limitation is that it requires the potential to be differentiable at all separations, which means it cannot be used for discontinuous potentials. For discontinuous potentials, the traditional fluctuation formulas continue to have a useful role. The fluctuation formulas are also independent of the simulation method. That is, the same fluctuation formulas are valid for both MC and MD simulations.

2.3.2 MC *NVT* ensemble

In the same way that the *NVE* ensemble is the natural ensemble of MD, the MC method is most easily implemented (Chapter 9) in the *NVT* ensemble. Lustig (2011, 2012) extended the approach for MC simulations in the *NVT* ensemble. The extension makes use of the Massieu−Planck system of thermodynamics (Münster, 1970), and proceeds from the entropy form of the

fundamental equation $S(E,V,N)$ to the Helmholtz energy $A/T\left(1/T,V,N\right)$ via successive Legendre transformations. The outcome, as illustrated below, is that any thermodynamic property can be obtained in terms of the partial derivatives of the Helmholtz function. The relevant statistical mechanics relationship is,

$$-\beta A(\beta,V,N)=\ln Z(\beta,V,N), \tag{2.40}$$

where Z is a partition function that depends on N, T, and V. Eq. (2.40) is the NVT ensemble analog of Eq. (2.31) and all thermodynamic properties can be obtained from combinations of partial derivatives of the partition function:

$$Z_{mn}=\frac{1}{Z}\frac{\partial^{m+n}Z}{\partial\beta^{m}\partial V^{n}} \tag{2.41}$$

The key thermodynamic relationships (Lustig, 2011) in terms of Z_{nm} are summarized in Table 2.5.

The values of Z_{nm} can be evaluated from the following relationships.

$$Z_{01}=\frac{N}{V}-\beta\left\langle\frac{\partial U}{\partial V}\right\rangle \tag{2.42}$$

$$Z_{10}=-\frac{F}{2}\beta^{-1}-\langle U\rangle \tag{2.43}$$

$$Z_{02}=\left(\frac{N}{V}\right)^{2}\left(1-\frac{1}{N}\right)+2\beta\frac{N}{V}\left\langle-\frac{\partial U}{\partial V}\right\rangle+\beta\left\langle-\frac{\partial^{2}U}{\partial V^{2}}\right\rangle+\beta^{2}\left\langle\left(\frac{\partial U}{\partial V}\right)^{2}\right\rangle \tag{2.44}$$

$$Z_{20}=\frac{F}{2}\left(\frac{F}{2}+1\right)\beta^{-2}+F\beta^{-1}\langle U\rangle+\left\langle U^{2}\right\rangle \tag{2.45}$$

TABLE 2.5 Relationships for the key thermodynamic quantities in terms of derivatives of the partition function for the MC *NVT* ensemble.

Quantity	Formula in terms of Z_{nm}
C_V	$\frac{Z_{20}-Z_{10}^{2}}{NkT^{2}}$
γ_V	$kZ_{01}-\frac{(Z_{11}-Z_{01}Z_{10})}{T}$
β_T^{-1}	$-kTV\left(Z_{02}-Z_{01}^{2}\right)$

Vlasiuk et al. (2016) observed that these relationships need to be modified by the special case for which there is a temperature dependence in the intermolecular potential. The partition function involving a temperature-dependent potential $u(q,\beta)$ is,

$$\tilde{Z}(\beta,V,N) \propto \beta^{-F/2} \int \dots \int e^{-\beta u(q,\beta)} dq, \tag{2.46}$$

where the tilde sign indicates the temperature dependence. There are no changes to the terms $\tilde{Z}_{01} = Z_{01}$ and $\tilde{Z}_{02} = Z_{02}$ that are the result of the derivatives of the partition function with respect to the volume. However, ensemble averages that involve differentiation with respect to β are affected:

$$\left.\begin{aligned} \tilde{Z}_{10} &= \frac{1}{\tilde{Z}} \frac{\partial \tilde{Z}}{\partial \beta} \\ \tilde{Z}_{11} &= \frac{1}{\tilde{Z}} \frac{\partial^2 \tilde{Z}}{\partial \beta \, \partial V} \\ \tilde{Z}_{20} &= \frac{1}{\tilde{Z}} \frac{\partial^2 \tilde{Z}}{\partial \beta^2} \end{aligned}\right\} \tag{2.47}$$

Applying the relationships in Eq. (2.47) yields:

$$\tilde{Z}_{10} = -\frac{F}{2}\beta^{-1} - \left\langle U + \beta \frac{\partial U}{\partial \beta} \right\rangle \tag{2.48}$$

$$\begin{aligned} \tilde{Z}_{11} = &-\frac{F}{2}\beta^{-1}\frac{N}{V} - \frac{N}{V}\left\langle U + \beta\frac{\partial U}{\partial \beta}\right\rangle + \left(\frac{F}{2} - 1\right)\left\langle \frac{\partial U}{\partial V}\right\rangle \\ &- \beta\left\langle \frac{\partial^2 U}{\partial V \, \partial \beta}\right\rangle + \beta\left\langle \frac{\partial U}{\partial V}\left(U + \beta\frac{\partial U}{\partial \beta}\right)\right\rangle \end{aligned} \tag{2.49}$$

$$\begin{aligned} \tilde{Z}_{20} = &\frac{F}{2}\left(\frac{F}{2} + 1\right)\beta^{-2} + F\beta^{-1}\left\langle U + \beta\frac{\partial U}{\partial \beta}\right\rangle \\ &+ \left\langle \left(U + \beta\frac{\partial U}{\partial \beta}\right)^2 \right\rangle - \left\langle 2\frac{\partial U}{\partial \beta} + \beta\frac{\partial^2 U}{\partial \beta^2}\right\rangle \end{aligned} \tag{2.50}$$

This approach has proved useful for the evaluation of the thermodynamic properties of neon (Vlasiuk et al., 2016) and other noble gases (Vlasiuk and Sadus, 2017). It is of interest to note that the formula for C_v (Table 2.2) is equivalent to the determination of this property from the fluctuation approach. However, this is not the case for the other thermodynamic properties.

2.3.3 MC *NpT* ensemble

Ströker et al. (2021) have extended Lustig's approach to the *NpT* ensemble. In common with the *NVT* case, thermodynamic quantities can be obtained in terms of partial derivatives of the phase-space functions (Z_{nm}).

$$Z_{mn} = \frac{1}{Z}\frac{\partial^{m+n} Z}{\partial \beta^m \partial p^n} \tag{2.51}$$

The key thermodynamic quantities as expressed by combinations of Z_{nm} are summarized in Table 2.6.

The Z_{nm} contributions can be evaluated from,

$$Z_{01} = -\beta\langle V\rangle \tag{2.52}$$

$$Z_{10} = -\frac{F}{2\beta} - \langle H_r\rangle \tag{2.53}$$

$$Z_{02} = \beta^2\left\langle V^2\right\rangle \tag{2.54}$$

$$Z_{20} = \frac{F^2}{4\beta^2} + \frac{F}{2\beta^2} + \frac{F\langle H_r\rangle}{\beta} + \left\langle H_r^2\right\rangle, \tag{2.55}$$

where $H_r = U + pV$ is the residual enthalpy. Inserting Eqs. (2.52) to (2.55) for the thermodynamic quantities given in Table 2.6 yields equivalent relationships to that obtained for the fluctuation approach (Brown, 1958; Hill, 1986). In contrast to both the *NVE* and *NVT* ensembles, the evaluation of the thermodynamic properties in the *NpT* ensemble do not require knowledge of the volume derivatives of *U*.

Following the procedure of Vlasiuk et al. (2016) detailed above for the NVT ensemble, Ströker et al. (2021) also reported the evaluation of Z_{nm} for

TABLE 2.6 Relationships for the key thermodynamic quantities in terms of derivatives of the partition function for the MC *NpT* ensemble.

Quantity	Formula in terms of Z_{nm}
C_p	$\frac{Z_{20} - Z_{10}^2}{kT^2}$
γ_V	$-\frac{Z_{01} - \beta(Z_{11} - Z_{10}Z_{01})}{T\left(Z_{02} - Z_{01}^2\right)}$
β_T^{-1}	$-\frac{Z_{01}}{Z_{02} - Z_{01}^2}$

temperature-dependent potentials. In common with the *NVT* case, $\tilde{Z}_{01} = Z_{01}$ and $\tilde{Z}_{02} = Z_{02}$, whereas the remaining terms can be evaluated from:

$$\tilde{Z}_{10} = -\frac{F}{2}\beta^{-1} - \left\langle U + \beta\frac{\partial U}{\partial \beta} + pV \right\rangle \tag{2.56}$$

$$\tilde{Z}_{11} = \left(\frac{F}{2} - 1\right)\langle V\rangle + \beta\left\langle \left(U + \beta\frac{\partial U}{\partial \beta} + pV\right)V \right\rangle \tag{2.57}$$

$$\begin{aligned}\tilde{Z}_{20} = {} & \frac{F}{2}\left(\frac{F}{2} + 1\right)\beta^{-2} + F\beta^{-1}\left\langle U + \beta\frac{\partial U}{\partial \beta} + pV \right\rangle \\ & + \left\langle \left(U + \beta\frac{\partial U}{\partial \beta} + pV\right)^2 \right\rangle - \left\langle 2\frac{\partial U}{\partial \beta} + \beta\frac{\partial^2 U}{\partial \beta^2} \right\rangle\end{aligned} \tag{2.58}$$

The convenience of using the *NpT* ensemble suggests that the approach will be helpful in the future (Nitzke and Vrabec, 2023; Young et al., 2023).

2.3.4 MC μVT ensemble

Using a procedure (Ströker and Meier, 2021) that is analogous to the *NVT* and *NpT* cases, all thermodynamic properties for μVT ensemble can be obtained from combinations of partial derivatives of the grand canonical partition function (ψ):

$$\psi_{nmo} = \frac{1}{\psi}\frac{\partial^{m+n+o}\psi}{\partial \beta^m \partial V^n \partial \mu^o} \tag{2.59}$$

However, in contrast to the other ensembles there are now derivatives with respect to all three independent variables, with the special case of $\psi_{000} = 1$. The key thermodynamic relationships (Ströker and Meier, 2021) in terms of ψ_{nmo} are summarized in Table 2.7.

The ψ_{nmo} contributions can be evaluated from:

$$\psi_{100} = \left(\mu - \frac{f}{2kT}\right)\langle N\rangle - \langle U\rangle \tag{2.60}$$

$$\psi_{200} = \left(\mu - \frac{f}{2kT}\right)^2\left\langle N^2\right\rangle - \frac{f\langle N\rangle}{2(kT)^2} + \left(\frac{f}{kT} - 2\mu\right)\langle NU\rangle + \left\langle U^2\right\rangle \tag{2.61}$$

$$\psi_{010} = \frac{\langle N\rangle}{V} - \frac{1}{kT}\left\langle \frac{\partial U}{\partial V} \right\rangle \tag{2.62}$$

$$\psi_{020} = \frac{\left\langle N^2\right\rangle}{V^2} - \frac{\langle N\rangle}{V^2} - \frac{2}{kTV}\left\langle N\frac{\partial U}{\partial V} \right\rangle - \frac{1}{kT}\left\langle \frac{\partial^2 U}{\partial V^2} \right\rangle + \frac{1}{(kT)^2}\left\langle \left(\frac{\partial U}{\partial V}\right)^2 \right\rangle \tag{2.63}$$

TABLE 2.7 Relationships for the key thermodynamic quantities for the MC μVT ensemble in terms of derivatives of the partition.

Quantity	Formula in terms of ψ_{nmo}
C_V	$\frac{\psi_{200}-\psi_{100}^2}{kT^2}-\frac{k\left(\psi_{001}-\left(\psi_{101}-\psi_{100}\psi_{001}\right)/kT\right)^2}{\psi_{002}-\psi_{001}^2}$
γ_V	$k\psi_{010}-\left(\psi_{110}-\psi_{100}\psi_{010}\right)/T-\frac{\left(\psi_{011}-\psi_{010}\psi_{011}\right)\left(k\psi_{001}-\left(\psi_{101}-\psi_{100}\psi_{001}\right)/T\right)}{\psi_{002}-\psi_{001}^2}$
β_T^{-1}	$-kTV\left(\psi_{020}-\psi_{010}^2-\frac{\left(\psi_{011}-\psi_{010}\psi_{001}\right)^2}{\psi_{002}-\psi_{001}^2}\right)$

$$\psi_{001}=\frac{\langle N\rangle}{kT} \tag{2.64}$$

$$\psi_{002}=\frac{\left\langle N^2\right\rangle}{(kT)^2} \tag{2.65}$$

$$\psi_{110}=\left(\mu-\frac{f}{2kT}\right)\frac{\left\langle N^2\right\rangle}{V}+\left(\frac{f}{2}-\frac{\mu}{kT}\right)\left\langle N\frac{\partial U}{\partial V}\right\rangle-\frac{\langle NU\rangle}{V}-\left\langle\frac{\partial U}{\partial V}\right\rangle+\frac{1}{kT}\left\langle U\frac{\partial U}{\partial V}\right\rangle \tag{2.66}$$

$$\psi_{101}=\left(\frac{\mu}{kT}-\frac{f}{2}\right)\left\langle N^2\right\rangle+\langle N\rangle-\frac{\langle NU\rangle}{kT} \tag{2.67}$$

$$\psi_{011}=\frac{\left\langle N^2\right\rangle}{kTV}-\frac{1}{(kT)^2}\left\langle N\frac{\partial U}{\partial V}\right\rangle \tag{2.68}$$

It should be noted that the generalization to any number of degrees of freedom in the above equations involves simply f and not $F=fN$.

2.3.5 MC *NpH* ensemble

Ströker and Meier (2022) further extended Lustig's approach for the *NpH* ensemble, for which the thermodynamic properties are obtained from derivatives of phase-space functions (Ω_{nm}).

$$\Omega_{nm}=\frac{1}{\omega}\frac{\partial^{m+n}\Omega}{\partial H^m\partial p^n} \tag{2.69}$$

TABLE 2.8 Relationships for the key thermodynamic quantities for the MC *NpH* ensemble in terms of derivatives of the phase-space function using the phase-volume definition of entropy.

Quantity	Formula in terms of Ω_{nm}
C_p	$\dfrac{k}{1-\Omega_{00}\Omega_{20}}$
γ_V	$\dfrac{k(\Omega_{11}-\Omega_{20}\Omega_{01})}{(1-\Omega_{00}\Omega_{20})(\Omega_{11}\Omega_{01}-\Omega_{02})-(\Omega_{01}-\Omega_{00}\Omega_{11})(\Omega_{20}\Omega_{01}-\Omega_{11})}$
β_T	$\Omega_{11}-\dfrac{\Omega_{02}}{\Omega_{01}}-\left(\Omega_{20}-\dfrac{\Omega_{11}}{\Omega_{01}}\right)\dfrac{\Omega_{01}-\Omega_{00}\Omega_{11}}{1-\Omega_{00}\Omega_{20}}$

Eq. (2.69) contains the special cases of $\Omega_{10}=1$ and $\Omega_{00}=\Omega/\omega$, where ω is the phase-space density. The evaluation of the thermodynamic averages can be determined in two different ways that depends on whether entropy is defined in terms of either the phase-space volume or density. The two routes are completely equivalent. However, the relationships for the phase-space volume route have the advantage of being slightly algebraically simpler and these are summarized in Table 2.8. In this case, C_p has a particularly simple definition and C_v can be easily obtained from the first relationship in Eq. (2.24).

The Ω_{nm} contributions required in Table 2.8 can be evaluated from:

$$\Omega_{00}=\frac{2\langle H-U-pV\rangle}{F} \tag{2.70}$$

$$\Omega_{20}=(F/2-1)\left\langle\frac{1}{H-U-pV}\right\rangle \tag{2.71}$$

$$\Omega_{01}=-\langle V\rangle \tag{2.72}$$

$$\Omega_{02}=(F/2-1)\left\langle\frac{V^2}{H-U-pV}\right\rangle \tag{2.73}$$

$$\Omega_{11}=(1-F/2)\left\langle\frac{V}{H-U-pV}\right\rangle \tag{2.74}$$

In Eqs. (2.70)−(2.74), the quantity $H-U-pV$ can be interpreted as the instantaneous kinetic energy. In common with the *NpT* ensemble (Eqs. 2.52−2.55), the contributions to the ensembles required in Eqs. (2.70)−(2.74) are relatively simple. Ströker and Meier (2022) observed a sizeable *N*-dependency for the Lennard-Jones potential, requiring an extrapolation procedure to obtain results for $N\rightarrow\infty$.

2.4 Particle dynamics

2.4.1 Motion of unconstrained particles

For a single particle, Newton's second law of motion is,

$$\mathbf{F} = m\ddot{\mathbf{r}} \tag{2.75}$$

where $\mathbf{F}$ is the total force acting on the particle, m is the mass of the particle, and $\ddot{\mathbf{r}}$ is the second time derivative of the position vector $\mathbf{r}$. All derivatives below are with respect to time unless otherwise indicated. In a simulation, we are normally interested in many particles and Eq. (2.75) can be easily generalized for a system containing many particles (Goldstein et al., 2002). It is useful to distinguish between external forces ($\mathbf{F}^e$) acting on the particles from outside the system (e.g., walls of a container) and internal forces ($\mathbf{F}_{ij}$) caused by interparticle interaction. In this case, the particles are atoms and molecules are treated as a constrained set of atoms. Consequently, Newton's second law of motion for particle i becomes

$$\sum_j \mathbf{F}_{ji} + \mathbf{F}_i^e = m_i\ddot{\mathbf{r}}_i \tag{2.76}$$

and summing over all particles, we obtain:

$$\sum_{j>i}\sum_i \mathbf{F}_{ji} + \sum \mathbf{F}_i^e = \sum m_i\ddot{\mathbf{r}}_i \tag{2.77}$$

The first sum in Eq. (2.77) must vanish as a consequence of Newton's second law, which requires that every action has an equal and opposite reaction, that is, each pair $\mathbf{F}_{ij} + \mathbf{F}_{ji}$ is zero. The second summation is simply the total external force ($\mathbf{F}^e$), consequently:

$$\mathbf{F}^e = \sum_i m_i\ddot{\mathbf{r}}_i \tag{2.78}$$

2.4.2 Motion of constrained particles

2.4.2.1 Holonomic and nonholonomic constraints

The above treatment assumed that the particles are unconstrained in their movement. However, in most realistic systems, constraints are typically in operation. For example, gas atoms enclosed in a container are constrained by the walls of the container to move only inside it. A rigid body, defined as a system of particles in which the interparticle distances are fixed and cannot vary with time, is another example of a constrained system. Formally, we can distinguish between holonomic and nonholonomic constraints (Goldstein et al., 2002). Holonomic systems have constraints that can be expressed as equations connecting the coordinates of the particles of the type:

$$f(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \ldots\ldots t) = 0 \tag{2.79}$$

The constraints of a rigid body are holonomic because they can be formulated as:

$$(\mathbf{r}_i - \mathbf{r}_j)^2 - c_{ij}^2 = 0 \tag{2.80}$$

For example, the implication of Eq. (2.80) for a diatomic molecule is that there is no breakage of the bond between the two atoms.

Other constrains such as the wall of a container, which cannot be expressed in terms of Eq. (2.79), are referred to as nonholonomic constraints. Strictly speaking, nonholonomic constraints can only be treated on a case-by-case basis. Fortunately, nonholonomic constraints often only apply on a macroscopic level, whereas molecular simulation is primarily concerned with the atomic or molecular scale. The wall of a container that imposes a nonholonomic constraint on a macroscopic level can be treated as a holonomic constraint at the atomic or molecular level. In reality, the container wall is composed of atoms or molecules that exert definite forces and the notion of constraint is somewhat artificial. Consequently, if a constraint is present it is typically assumed to be holonomic.

2.4.2.2 *Consequences of constraints on particles*

The presence of constraints imposes two difficulties for particle mechanics. First, the coordinates ($\mathbf{r}_i$) are not all independent because they are connected by the equations of the constraint. Therefore the equations of motion are not all independent. Second, the forces of the constraint, for example, the effect of the wall on the gas particles, are not known beforehand. Instead, they are one of the unknowns of the problem being investigated.

For holonomic constraints, the first difficulty is solved by using generalized coordinates. In the absence of constraints, a system of N particles has $3N$ independent coordinates or degrees of freedom. If there are k constraints expressed in terms of Eq. (2.79), the number of independent coordinates or degrees of freedom is reduced to $3N-k$. Consequently, a total of $3N-k$ independent variables ($q_1 \ldots q_{3N-k}$) in terms of the old coordinates ($\mathbf{r}_1 \ldots \mathbf{r}_N$) are introduced and expressed by equations of the form (Goldstein et al., 2002):

$$\left.\begin{aligned} \mathbf{r}_1 &= \mathbf{r}_1(q_1, q_2 \ldots q_{3N-k}, t) \\ &\vdots \\ \mathbf{r}_N &= \mathbf{r}_N(q_1, q_2 \ldots q_{3N-k}, t) \end{aligned}\right\} \tag{2.81}$$

To overcome the difficulty that the forces of constraint are unknown beforehand, we can re-formulate the mechanics in such a way that the forces of the constraint disappear. The resulting equations are the Lagrange and/or Hamilton equations of motion.

2.4.2.3 Lagrange equations of motion

It is useful to introduce the concept of a virtual displacement (Goldstein et al., 2002). A virtual displacement is an infinitesimal change in coordinates ($\delta\mathbf{r}$), which is consistent with the forces and constraints imposed on the system at the time. Unlike an actual displacement, there is no possibility of any change in the force of constraint. When the system is at equilibrium, $\mathbf{F}_i = 0$ and the virtual work ($\mathbf{F}_i \cdot \delta\mathbf{r}_i$) resulting in the virtual displacement also vanishes. Similarly, the sum of the virtual work over all particles is also zero.

$$\sum_i \mathbf{F}_i \cdot \delta\mathbf{r}_i = 0 \tag{2.82}$$

The force ($\mathbf{F}_i$) has contributions from both the applied force ($\mathbf{F}_i^a$) and the force of constraint ($\mathbf{f}_i$).

$$\mathbf{F}_i = \mathbf{F}_i^a + \mathbf{f}_i \tag{2.83}$$

Substituting Eq. (2.83) into Eq. (2.82), we obtain:

$$\sum_i \mathbf{F}_i^a \cdot \delta\mathbf{r}_i + \sum_i \mathbf{f}_i \cdot \delta\mathbf{r}_i = 0 \tag{2.84}$$

For many systems the virtual work of the forces of constraint is zero. In these cases, the condition for equilibrium involves only the applied forces.

$$\sum_i \mathbf{F}_i^a \cdot \delta\mathbf{r}_i = 0 \tag{2.85}$$

Eq. (2.85) is the principle of virtual work (Goldstein et al., 2002). In general, $\mathbf{F}_i^a \neq 0$ because $\delta\mathbf{r}_i$ is not completely independent but is connected by the constraints. Therefore a transformation is required such that the coefficients of $\delta\mathbf{r}$ are zero. Newton's equations of motion (Eq. (2.75)) for each particle can be rewritten as

$$\mathbf{F}_i - \dot{\mathbf{p}}_i = 0, \tag{2.86}$$

where $\mathbf{p}_i = m_i\dot{\mathbf{r}}_i$. Consequently, we can formulate the following alternative to Eq. (2.82) for the equilibrium condition.

$$\sum_i (\mathbf{F}_i - \dot{\mathbf{p}}_i) \cdot \delta\mathbf{r}_i = 0 \tag{2.87}$$

By decomposing force into applied and constraint components, we obtain:

$$\sum_i (\mathbf{F}_i^a - \dot{\mathbf{p}}_i) \cdot \delta\mathbf{r}_i + \sum_i \mathbf{f}_i \cdot \delta\mathbf{r}_i = 0 \tag{2.88}$$

If there is no virtual work from the forces of constraint, then:

$$\sum_i (\mathbf{F}_i^a - \dot{\mathbf{p}}_i) \cdot \delta\mathbf{r}_i = 0 \tag{2.89}$$

Eq. (2.89) is known as D'Alembert's principle (Goldstein et al., 2002). The forces of constraint do not appear in Eq. (2.89) and the coefficients of $\delta\mathbf{r}_i$ are zero as required. To make Eq. (2.89) useful, we can transform the virtual displacements to generalized coordinates that are independent of each other. Using the relationship,

$$\mathbf{r}_i = \mathbf{r}_i(q_1, q_2 \ldots \ldots q_n, t) \tag{2.90}$$

and assuming n independent coordinates, it can be shown (Goldstein et al., 2002; Mach, 1960) that Eq. (2.89) can be transformed to,

$$\sum_j \left[\left\{ \frac{d}{dt}\left(\frac{\partial K}{\partial \dot{q}_j}\right) - \left(\frac{\partial K}{\partial q_j}\right) \right\} - Q_j \right] \delta q_j = 0, \tag{2.91}$$

where Q_j are the components of the generalized force defined by,

$$Q_j = \sum_i \mathbf{F}_i \cdot \frac{\partial \mathbf{r}_i}{\partial q_j}, \tag{2.92}$$

where the superscript a has been omitted from the force term. If the constraints are assumed to be holonomic, the q_j terms are independent of each other. Because δq_j is independent of δq_k, Eq. (2.91) will only be satisfied if the coefficients vanish, that is,

$$\frac{d}{dt}\left(\frac{\partial K}{\partial \dot{q}_j}\right) - \frac{\partial K}{\partial q_j} = Q_j, \tag{2.93}$$

where there are n such equations.

Eq. (2.93) are the Lagrange equations (Goldstein et al., 2002; Calkin, 1996) for the general case where the forces are not derivable from a potential. However, the forces are commonly derivable from a potential function V:

$$\mathrm{F}_i = -\nabla_i V \tag{2.94}$$

Consequently, the generalized forces can be obtained from,

$$Q_j = -\sum_i \nabla_i V \cdot \frac{\partial \mathbf{r}_i}{\partial q_j} \tag{2.95}$$

which is identical to the partial derivative of a function $-V(\mathbf{r}_1 \ldots . \mathbf{r}_N)$ with respect to q_j.

$$Q_j = -\frac{\partial V}{\partial q_j} \tag{2.96}$$

Substituting Eq. (2.96) into Eq. (2.93), we obtain:

$$\frac{d}{dt}\left(\frac{\partial K}{\partial \dot{q}_j}\right) - \frac{\partial (K - V)}{\partial q_j} = 0 \tag{2.97}$$

V is completely independent of the generalized velocity because it is a function only of position. Therefore it can be included in the partial derivative with respect to $\dot{q}$.

$$\frac{d}{dt}\left(\frac{\partial(K-V)}{\partial \dot{q}_j}\right) - \frac{\partial(K-V)}{\partial q_j} = 0 \tag{2.98}$$

If we define a new function, the Lagrangian L as,

$$L = K - V \tag{2.99}$$

Eq. (2.98) becomes:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = 0 \tag{2.100}$$

Eq. (2.100) is the familiar form of the Lagrange equations (Goldstein et al., 2002; Calkin, 1996).

The advantage of the Lagrange function is that the need to use vector forces in Newton's equations of motion is replaced by the scalar functions of K and V. Any problem of mechanics can be transferred to the Lagrangian coordinates by writing K and V in generalized coordinates, hence obtain L and substitute it into Eq. (2.100). For example, for the motion of a particle in Cartesian coordinates x, y, and z, we obtain $L = m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2)/2 - V$. Substituting this into Eq. (2.100), we find that,

$$\left.\begin{aligned} F_x &= \frac{\partial V}{\partial x} = m\ddot{x} \\ F_y &= \frac{\partial V}{\partial y} = m\ddot{y} \\ F_z &= \frac{\partial V}{\partial z} = m\ddot{z} \end{aligned}\right\} \tag{2.101}$$

which are Newton's equations of motion.

2.4.2.4 Hamilton equations of motion

The Lagrangian formulation is a description of mechanics in terms of generalized coordinates and velocity, with time as a parameter. In contrast, the Hamilton equations of motion use generalized coordinates and generalized momenta as the independent variables. The generalized momentum (p_i) is defined as (Goldstein et al., 2002; Calkin, 1996):

$$p_i = \frac{\partial L(q_j, \dot{q}_i, t)}{\partial \dot{q}_i} \tag{2.102}$$

The change in independent variables can be achieved by using a Legendre transformation, which results in the definition of the Hamiltonian (H) function.

$$H(p,q,t) = \sum_i \dot{q}_i p_i - L(q,\dot{q}_i,t) \tag{2.103}$$

The differential of H as a function of only p, q, and t is:

$$dH = \sum_i \frac{\partial H}{\partial q_i} dq_i + \sum_i \frac{\partial H}{\partial p_i} dp_i + \frac{\partial H}{\partial t} dt \tag{2.104}$$

In addition, from Eq. (2.100) we can also obtain:

$$dH = \sum_i \dot{q}_i dp_i + \sum_i p_i d\dot{q}_i - \sum_i \frac{\partial L}{\partial q_i} d\dot{q}_i - \sum_i \frac{\partial L}{\partial q_i} dq_i - \frac{\partial L}{\partial t} dt \tag{2.105}$$

The terms in Eq. (2.105) involving $d\dot{q}_i$ cancel because of the following definition of generalized momentum,

$$\sum_i p_i d\dot{q}_i - \sum_i \frac{\partial L}{\partial \dot{q}_i} d\dot{q}_i = 0 \tag{2.106}$$

and from the Lagrange equations we have:

$$\frac{\partial L}{\partial q_i} = \dot{p}_i \tag{2.107}$$

Consequently, Eq. (2.105) becomes:

$$dH = \sum_i \dot{q}_i dp_i - \sum_i \dot{p}_i dq_i - \frac{\partial L}{\partial t} dt \tag{2.108}$$

Comparing Eq. (2.108) with Eq. (2.104) we obtain the following $2n + 1$ relationships (Goldstein et al., 2002):

$$\dot{q}_i = \frac{\partial H}{\partial p_i} \tag{2.109}$$

$$\dot{p}_i = -\frac{\partial H}{\partial q_i} \tag{2.110}$$

$$\frac{\partial L}{\partial t} = -\frac{\partial H}{\partial t} \tag{2.111}$$

Eqs. (2.109)–(2.111) are referred to as Hamilton's canonical equations of motion (Goldstein et al., 2002). In general, problems in mechanics can be solved by determining the Lagrangian using Eq. (2.102) to determine the momenta and hence formulate the Hamiltonian from Eq. (2.103). Substituting the Hamiltonian obtained from this procedure into Eqs. (2.109)–(2.111) yields first-order equations of motion.

If the Hamiltonian is not explicitly a function of time, it can be shown that the Hamiltonian is constant with respect to motion. Furthermore, if the potential is independent of velocity and the equations that define the generalized coordinates (Eq. (2.77)) are not explicitly time-dependent, the Hamiltonian is the total energy.

$$H = K + V \tag{2.112}$$

For the cases when Eq. (2.112) is valid, the equations of motion can be obtained directly by substituting the right-hand side of Eq. (2.112) into Eqs. (2.109)–(2.111).

2.4.2.5 Gauss's principle of least constraint

In 1829, Gauss (Mach, 1960) formulated an alternative to Newtonian dynamics by applying a least squares technique to mechanical properties. He postulated that any mechanical constraint on a system should be satisfied by using the smallest constraint force. If we have the following constrained equation of motion:

$$m\ddot{\mathbf{r}} = \mathbf{F} + \mathbf{F}_c \tag{2.113}$$

Gauss's principle of least constraint states that the constrained forces $\mathbf{F}_c$ should be as small as possible, that is, the following summation should have a minimum value.

$$\sum_i \frac{\mathbf{F}_{c,i}^2}{2m_i} \tag{2.114}$$

Evans et al. (1983) "re-discovered" Gauss's principle showing how it could be made useful for numerical calculations. For either holonomic constraints ($g(\mathbf{r},t) = 0$) or nonholonomic constraints ($g(\mathbf{r},\dot{\mathbf{r}},t) = 0$), the acceleration is confined to lie on a constant-g hypersurface by the restriction of the form:

$$n(\mathbf{r},\dot{\mathbf{r}},t)\cdot\ddot{\mathbf{r}} + w(\mathbf{r},\ddot{\mathbf{r}},t) = 0 \tag{2.115}$$

In the absence of any constraints, the unconstrained motion of the system calculated from,

$$m\ddot{\mathbf{r}}_u = \mathbf{F} \tag{2.116}$$

could leave the constraint hypersurface. This is prevented in Gauss's formulation by the addition of an acceleration normal to the surface, that is,

$$\ddot{\mathbf{r}}_c = \ddot{\mathbf{r}}_u - \lambda\frac{n(\mathbf{r},\dot{\mathbf{r}},t)}{m} \tag{2.117}$$

with the Lagrange multiplier, λ, satisfying the restriction given by Eq. (2.118).

$$\lambda = \frac{n\cdot\ddot{\mathbf{r}}_u + w}{n\cdot(n/m)} \tag{2.118}$$

The added acceleration can also be expressed in terms a constrained force ($\mathbf{F}_c$).

$$\ddot{\mathbf{r}}_c = \frac{\mathbf{F} + \mathbf{F}_c}{m} = \frac{\mathbf{F}}{m} - \lambda \frac{n}{m} \tag{2.119}$$

2.4.3 The Liouville equation

From the perspective of molecular simulation or statistical mechanics the N particles form a single mechanical system and the concept of phase space is used routinely to visualize the changing state of the system. Phase space can be defined as the Cartesian space of $6N$ dimensions. The $6N$ dimensions of phase space are composed of the $3N$ generalized configurational coordinates ($\mathbf{q}_i$) and the $3N$ momenta ($\mathbf{p}_i$). A point in phase space describes completely the dynamical state of the system. The change with time of any point of the system traces out the trajectory of the system as governed by Hamilton's canonical equations.

The change in a property of the system $F = F(\mathbf{q}_i, \mathbf{p}_i, t)$, which depends both on the dynamical state of the system and explicitly on time can be obtained from (Hirschfelder et al., 1954):

$$\frac{DF}{Dt} = \frac{\partial F}{\partial t} + \sum_i \left(\frac{\partial F}{\partial q_i} \dot{q} + \frac{\partial F}{\partial p_i} \dot{p} \right) \tag{2.120}$$

$$= \frac{\partial F}{\partial t} + \sum_i \left(\frac{\partial F}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial F}{\partial p_i} \frac{\partial H}{\partial q_i} \right) \tag{2.121}$$

$$= \frac{\partial F}{\partial t} + [F, H] \tag{2.122}$$

Eq. (2.122) introduces the use of the Poisson bracket. Any constant of motion (α) such as the total energy or total linear momentum has the property that:

$$[\alpha, H] = 0 \tag{2.123}$$

The state of the entire collection of systems, which do not interact and differ from each other only in their initial conditions, can be represented by a set of points. The set of representative points is obtained from a continuous distribution function ($\rho(\mathbf{q}_i, \mathbf{p}_i, t)$), which is referred to as the density in phase space. The distribution function must satisfy the following relationship (Hirschfelder et al., 1954) because there are neither sources nor sinks for phase points:

$$\frac{\partial \rho}{\partial t} + \sum_i \left(\frac{\partial}{\partial q_i} (\dot{q}\rho) + \frac{\partial}{\partial p_i} (\dot{p}\rho) \right) = 0 \tag{2.124}$$

From Eq. (2.124), using Hamilton's canonical equations and the definition of the Poisson bracket, we obtain the Liouville equation (Hirschfelder et al., 1954):

$$\frac{D\rho}{Dt} = \frac{\partial \rho}{\partial t} + [\rho, H] = 0 \tag{2.125}$$

Eq. (2.125) is important for the development of statistical mechanics, in particular, the definition of statistical ensembles (Section 2.1.1).

2.4.4 The virial theorem

An interesting consequence of the equations of motion is that they can be used to derive the virial theorem (Hirschfelder et al., 1954). The equations of motion for a system of particles with position vectors ($\mathbf{r}_i$) and applied forces ($\mathbf{F}_i$) are:

$$\dot{\mathbf{p}}_i = \mathbf{F}_i \tag{2.126}$$

We also define a quantity:

$$G = \sum_i \mathbf{p}_i \cdot \mathbf{r}_i \tag{2.127}$$

The total time derivative of G is:

$$\frac{dG}{dt} = \sum_i \dot{\mathbf{r}}_i \cdot \mathbf{p}_i + \sum_i \dot{\mathbf{p}}_i \cdot \mathbf{r}_i \tag{2.128}$$

The first term of the right-hand side of Eq. (2.128) is simply twice the kinetic energy ($2K$), and from Eq. (2.126) the second summation can be evaluated as:

$$\sum_i \dot{\mathbf{p}}_i \cdot \mathbf{r}_i = \sum_i \dot{\mathbf{F}}_i \cdot \mathbf{r}_i \tag{2.129}$$

Consequently, Eq. (2.128) becomes:

$$\frac{d}{dt} \sum_i \dot{\mathbf{p}}_i \cdot \mathbf{r}_i = 2K + \sum_i \dot{\mathbf{F}}_i \cdot \mathbf{r}_i \tag{2.130}$$

The time average of Eq. (2.130) over a time interval (τ) is obtained by integrating both sides with respect to t from 0 to τ, and dividing by τ:

$$\overline{2K} + \overline{\sum_i \mathbf{F}_i \cdot \mathbf{r}_i} = \frac{1}{\tau}[G(\tau) - G(0)] \tag{2.131}$$

As $\tau \to \infty$ we obtain (Hirschfelder et al., 1954):

$$\overline{K} = -\frac{1}{2} \overline{\sum_i \mathbf{F}_i \cdot \mathbf{r}_i} \tag{2.132}$$

Eq. (2.132) is the virial theorem, which is used frequently in statistical mechanics and applications of molecular simulation.

2.5 Summary

The statistical mechanics definition of the ensemble determines the overall nature of either a MC or MD simulation. It also determines the strategy required to determine thermodynamic averages of various properties. Traditionally, thermodynamic properties have been evaluated via ensemble fluctuations, requiring post simulation analysis. However, alternative methods are available that can be used during the course of the simulation itself. Knowledge of particle dynamics is used to determine the equations of motion required for MD simulations in various ensembles. The influence of the choice of ensemble on MC and MD simulations is examined in Chapter 9.

References

Abramowitz, M., Stegun, I.A. (Eds.), 1972. Handbook of Mathematical Functions with Formula, Graphs and Mathematical Tables. Dover, New York, p. 256.

Adams, D.J., 1975. Grand canonical ensemble Monte Carlo for a Lennard-Jones fluid. Mol. Phys. 29, 307–311.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Bosko, J., Todd, B.D., Sadus, R.J., 2005. Molecular simulation of dendrimers and their mixtures under shear: comparison of isothermal-isobaric (*NpT*) and isothermal-isochoric (*NVT*) ensemble systems. J. Chem. Phys. 123, 034905.

Brown, W.B., 1958. Constant pressure ensembles in statistical mechanics. Mol. Phys. 1, 68–82.

Calkin, M.G., 1996. Lagrangian and Hamiltonian Mechanics. World Scientific, Singapore.

Elliot, J.R., Diky, V., Knotts IV, T., Wilding, W. V. 2023. Properties of Gases and Liquids, 6th ed. McGraw Hill, New York.

Evans, D.J., Hoover, W.G., Failor, B.H., Moran, B., Ladd, A.J.C., 1983. Nonequilibrium molecular dynamics via Gauss's principle of least constraint. Phys. Rev. A 28, 1016–1021.

Fowler, R., Guggenheim, E.A., 1956. *Statistical Thermodynamics*: *A Version of Statistical Mechanics for Students of Physics and Chemistry*. Cambridge University Press, Cambridge.

Frenkel, D., 1986. In: Ciccotti, G., Hoover, W.G. (Eds.), Molecular Dynamics Simulations of Statistical Mechanical Systems. North-Holland, Amsterdam.

Friedman, A.S., Haar, L., 1954. High-speed machine computation of ideal gas thermodynamic functions. I. Isotopic water molecules. J. Chem. Phys. 22, 2051–2058.

Garrod, C., 1995. Statistical Mechanics and Thermodynamics. Oxford University Press, New York.

Goldstein, H., Poole, C., Safko, J., 2002. Classical Mechanics, third ed. Addison Wesley, San Francisco.

Haile, J.M., 1992. Molecular Dynamics Simulation. Elementary Methods. John Wiley & Sons, New York.

Hansen, J.P., McDonald, I.R., 2013. Theory of Simple Liquids, fourth ed. Academic Press, Amsterdam.

Henderson, J.R., 1983. Statistical mechanics of fluids at spherical structureless walls. Mol. Phys. 51, 741–761.

Hill, T.L., 1986. An Introduction to Statistical Thermodynamics. Dover Publications, New York.

Hill, T.L., 1987. Statistical Mechanics. Principles and Selected Applications. Dover Publications, New York.

Hirschfelder, J.O., Curtiss, C.F., Bird, R.B., 1954. Molecular Theory of Gases and Liquids. John Wiley & Sons, New York.
Hoheisel, C., 1994. Transport properties of molecular liquids. Phys. Rep. 245, 111–157.
Hoover, W.G., 1986. Molecular Dynamics. Springer-Verlag, Berlin.
Hoover, W.G., 1991. Computational Statistical Mechanics. Elsevier, Amsterdam.
Lagache, M., Ungerer, P., Boutin, A., Fuchs, A.H., 2001. Prediction of thermodynamic derivative properties of fluids by Monte Carlo simulation. Phys. Chem. Chem. Phys. 3, 4333–4339.
Lebowitz, J.L., Percus, J.K., Verlet, L., 1967. Ensemble dependence of fluctuations with applications to machine computations. Phys. Rev. 153, 250–254.
Lemmon, E.W., McLinden, M.O., Wagner, W., 2009. Thermodynamic properties of propane. III. A reference equation of state for temperatures from the melting line to 650 K and pressures up to 1000 MPa. J. Chem. Eng. Data 54, 3131–3180.
Lustig, R., 1994a. Statistical thermodynamics in the classical molecular dynamics ensemble. I. Fundamentals. J. Chem. Phys. 100, 3048–3059.
Lustig, R., 1994b. Statistical thermodynamics in the classical molecular dynamics ensemble. II. Application to computer simulation. J. Chem. Phys. 100, 3060–3067.
Lustig, R., 1994c. Statistical thermodynamics in the classical molecular dynamics ensemble. III. Numerical results. J. Chem. Phys. 100, 3068–3078.
Lustig, R., 1998. Microcanonical Monte Carlo simulation of thermodynamic properties. J. Chem. Phys. 109, 8816–8828.
Lustig, R., 2011. Direct molecular NVT simulation of the isobaric heat capacity, speed of sound and Joule-Thomson coefficient. Mol. Sim. 37, 457–465.
Lustig, R., 2012. Statistical analogues for fundamental equation of State derivatives. Mol. Phys. 110, 3041–3052.
Mach, E., 1960. McCormack, T.J. (Trans.), The Science of Mechanics, sixth ed. The Open Court Publishing Co., La Salle.
Maitland, G.C., Rigby, M., Smith, E.B., Wakeham, W.A., 1981. Intermolecular Forces: Their Origin and Determination. Clarendon Press, Oxford.
Mausbach, P., Sadus, R.J., 2011. Thermodynamic properties in the molecular dynamics ensemble applied to the Gaussian core model fluid. J. Chem. Phys. 134, 114515.
Mayer, J.E., Mayer, M.G., 1946. Statistical Mechanics. John Wiley & Sons, New York.
Meier, K., Kabelac, S., 2006. Pressure derivatives in the classical molecular-dynamics ensemble. J. Chem. Phys. 124, 064104.
Münster, A., 1970. Halberstadt, E.S. (Trans.), Classical Thermodynamics. John Wiley & Sons, London, p. 83.
Nitzke, I., Vrabec, J. 2023. Numerical discrimination of thermodynamic Monte Carlo simulations in all eight ensembles J. Chem. Theory Comput. 19, 3460–2468.
Prausnitz, J.M., de Azevedo, E.G., Lichtenthaler, R.N., 1999. Molecular Thermodynamics of Fluid-Phase Equilibria, third ed. Prentice Hall, p. 216.
Prigogine, I., Defay, R., 1954. Everett, D.H. (Trans.), Chemical Thermodynamics. Longmans Green, p. 381.
Raabe, G., 2017. Molecular Simulation Studies on Thermophysical Properties with Application to Working Fluids. Springer, Berlin.
Rapaport, D.C., 2004. The Art of Molecular Dynamics Simulation, 2nd ed. Cambridge University Press, Cambridge.
Reed, T.M., Gubbins, K.E., 1973. Applied Statistical Mechanics. Thermodynamic and Transport Properties of Fluids. Butterworth-Heinemann, Boston.
Reif, F., 1965. Fundamentals of Statistical and Thermal Physics. McGraw-Hill, Tokyo.

Rowlinson, J.S., 1969. Liquids and Liquid Mixtures, second ed. Butterworth, London, pp. 247−258.

Rushbrooke, G.S., 1962. Introduction to Statistical Mechanics. Oxford University Press, London.

Shing, K.S., Chung, S.-T., 1987. Computer simulation methods for the calculation of solubility in supercritical extraction systems. J. Phys. Chem. 91, 1674−1681.

Shvab, I., Sadus, R.J., 2013. Intermolecular potentials and the accurate predication of the thermodynamic properties of water. J. Chem. Phys. 139, 194505.

Shvab, I., Sadus, R.J., 2014. Thermodynamic properties and diffusion of water + methane binary mixtures. J. Chem. Phys. 140, 104505.

Stiegler, T., Sadus, R.J., 2015. Molecular simulation of fluids with non-identical intermolecular potentials: thermodynamic properties of 10-5 + 12-6 Mie potential binary mixtures. J. Chem. Phys. 142, 084504 (2015).

Ströker, P., Meier, K., 2021. Classical statistical mechanics in the grand canonical ensemble. Phys. Rev. E 104, 014117.

Ströker, P., Meier, K., 2022. Rigorous expressions for thermodynamic properties in the *NpH* ensemble. Phys. Rev. E 105, 035301.

Ströker, P., Hellmann, R., Meier, K., 2021. Systematic formulation of thermodynamic properties in the *NpT* ensemble. Phys. Rev. E 103, 023305.

Vlasiuk, M., Sadus, R.J., 2017. *Ab initio* interatomic potentials and the thermodynamic properties of fluids. J. Chem. Phys. 147, 024505.

Vlasiuk, M., Frascoli, F., Sadus, R.J., 2016. Molecular simulation of the thermodynamic, structural and vapor-liquid equilibrium properties of neon. J. Chem. Phys. 145, 104501.

Wannier, G., 1987. Statistical Physics. Dover Publications, New York.

Widom, B., 1963. Some topics in the theory of fluids. J. Chem. Phys. 39, 2808−2812.

Widom, B., 1982. Potential distribution theory and the statistical mechanics of fluids. J. Phys. Chem. 86, 869−872.

Yigzawe, T.M., Sadus, R.J., 2013. Intermolecular interactions and the thermodynamic properties of supercritical fluids. J. Chem. Phys. 138, 194502.

Young, J. M., Bell, I. H., Harvey, A. H. 2023. Entropy scaling of viscosity for molecular models of molten salts. J. Chem. Phys. 158, 024502.

Chapter 3

Intermolecular pair potentials and force fields

It is apparent from Chapter 2 that the calculation of any thermodynamic average requires us to determine the kinetic and potential energies of the system. In either a Monte Carlo (MC) or molecular dynamics (MD) simulation, the potential energy of the system or the force acting between molecules is calculated commonly from a pairwise additive intermolecular potential. The development of intermolecular potentials has a long history (Reed and Gubbins, 1973; Maitland et al., 1981; Stone, 2013; Rowlinson, 2002), which predates the advent of molecular simulation methods. However, except for the simplest cases, the accuracy of intermolecular potentials for the prediction of thermophysical properties can only be evaluated rigorously by molecular simulation.

The terms "intermolecular potential" and "potential model" are often used interchangeably in the literature. However, strictly speaking the latter should be reserved for situations that involve *modeling* assumptions regarding the nature of interactions describing a particular system. For example, to determine the properties of a molecule requires an overall model to describe such contributions as bond angles and geometries. The term "force field" is often used to describe the collection of contributions used to evaluate the terms required by the model. It is apparent that the different potential models or force fields can utilize either the same or different intermolecular potentials. If the contributions of the intermolecular potentials are the same then different outcomes reflect the different assumptions of the overall model. The simplest potentials involve interactions between atoms and are correctly described as interatomic potentials, although distinction between interatomic and intermolecular potential is not always made with the latter also applied to the former case.

Most intermolecular potentials are exclusively pairwise additive, whereas many-body effects are important in many cases. The terms pairwise and two-body potentials are often used interchangeably in the literature. However, although both involve only the evaluation of *pairs* of interactions, a pairwise potential often implicitly includes multi-body effects, whereas two-body potentials do not. For example, the development of the Lennard-Jones (LJ) potential was not confined exclusively to two-body interactions and as such it is an effective multi-body pairwise potential but not a two-body potential. The focus

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00017-4

of this chapter is on effective pair potentials and special consideration of two- and other multi-body potentials given in Chapter 4. The historical development of some empirical intermolecular potentials has been discussed by Fischer and Wendland (2023).

Relatively few of the large number of available intermolecular pair potentials have been comprehensively evaluated by molecular simulation. Nonetheless, the use of molecular simulation for increasingly more complicated molecular systems will inevitably require models that are not currently in widespread use. Therefore, knowledge of possible alternatives is certainly worthwhile. A nascent possibility is the use of machine learning (Jung, 2022) to determine intermolecular potentials (Gao et al., 2019; Mueller et al., 2020; Pant et al., 2020; Moradzadeh and Aluru, 2021; Perepu et al., 2023)

Water is an example of an important molecular system whose properties are significantly affected by the overall model framework and the contribution from component intermolecular potential. An overview is provided for the potential models for water. It provides a convenient case study of the application of both modeling principles and the use of intermolecular potentials.

3.1 Calculation of the potential energy

The calculation of the potential energy (or force) arising from intermolecular interaction is the most time consuming and crucial step in any molecular simulation. In general, the potential energy (U) of N interacting particles can be evaluated as,

$$U = \sum_i u_1(r_i) + \sum_i \sum_{j>i} u_2(r_i, r_j) + \sum_i \sum_{j>i} \sum_{k>j>i} u_3(r_i, r_j, r_k) + \cdots\cdots \tag{3.1}$$

where the first term represents the effect of an external field and the remaining terms represent particle interactions, that is, u_2 is the potential between pairs of particles and u_3 is the potential between particle triplets etc. Two-body interactions undoubtedly make the most significant contribution to particle interactions, and Eq. (3.1) is typically truncated after the second term. This approximation means that the simulation algorithm is of order N^m, where $m \geq 2$ denotes the number of interactions. However, except for properties involving dilute gases, the properties of real fluids involve the contribution of other multi-body interactions. It is relatively rare that accurate prediction can be accomplished by accounting only for two-body interactions. In particular, it is well-established that three-body interactions are very important in many cases (Bobetic and Barker, 1970; Klein and Venables, 1976; Monson et al., 1983; Rittger, 1990a,b,c). The dilemma is that, although three- or more-body interactions are important for accurate predictions, they impose a very large increase in computing time compared with two-body

calculations. Despite the considerable advances in computer hardware, involving systems of thousands of central and graphics processing units (Chapter 12), calculations beyond two-body interactions remain, in most circumstances, both computationally prohibitive and impractical.

The solution to this computational dilemma is to use an effective pairwise intermolecular potential (u_{eff}), that while involving only the evaluation of pairs of interactions, incorporates the effect of three- or more-body interactions. This means, that in the absence of an external field, Eq. (3.1) is simply approximated as:

$$U \approx \sum_{i} \sum_{j>i} u_{eff}\left(r_i, r_j\right) \tag{3.2}$$

As explained below, Eq. (3.2) has most commonly been applied unintentionally because the true nature of the pair potential has not been fully recognized. Therefore, the apparent success of effective potentials in predicting the properties of fluids via molecular simulation has often been a fortuitous outcome rather than the result of a deliberately implemented strategy.

The development of intermolecular potentials greatly predates the advent of molecular simulation, typically involving the combination of various contributions to intermolecular interactions (Mie, 1903; Jones, 1924) and making a comparison with experimental data. Most commonly, the empirical potentials were compared with data for the second virial coefficient, which is a genuine two-body property. This route for evaluating early pair potentials has led to the term "pair/pairwise" being incorrectly used as a synonym for two-body (Rowlinson, 2002). However, it has been known since the 1960s and confirmed by many subsequent state-of-the-art accurate ab initio calculations (Chapter 4) that genuine two-body potentials for the noble gases are much more complicated and do not accurately correspond to the simple empirical potentials. Typically, the depths of the potential minimum of the empirical pair potentials are considerably smaller than those obtained from genuine two-body potentials. Arguably, the formulas used for many of the empirical pairwise potentials have unintentionally at least partially incorporated some of the effects of multi-body interactions. Therefore, such potentials are more accurately described as effective pairwise potentials. This historical accident is quite fortuitous because it means that Eq. (3.2) is often a good approximation for the properties of real fluids.

The distinction between the terms "pair" and "two-body" might appear somewhat pedantic but it is an important distinction when molecular simulation results are used to precisely investigate the influence of underlying interactions on the macroscopic properties of fluids. No pair potential can be fully equivalent to a multi-body potential. One danger in not observing this distinction is unwittingly (Sadus and Prausnitz, 1996) double counting the same intermolecular contributions. To avoid any confusion, we will usually denote pairwise potentials that are not explicitly confined to only two-body

interactions simply as u_{eff} and the u_2 and u_3 nomenclature will be reserved for genuine two- and three-body potentials, respectively.

Although using Eq. (3.2) is the most common approach in molecular simulation, there is a growing literature involving two-body plus three-body interactions. As discussed in Chapter 4, this has been facilitated by ways of accurately approximating the contribution of three-body interactions in a computationally efficient fashion. It would be desirable to have one two-body potential that was valid for all atoms. However, the evidence from accurate ab initio calculations indicates that the intermolecular potential must be tailored to each atom. For example, different potentials are required to accurately evaluate the properties of the different noble gases (Deiters and Sadus, 2019a,b, 2020, 2021). In contrast, a common interatomic potential (Seng et al, 2020) for the noble gases is inaccurate for many of their thermodynamic properties (Deiters and Sadus, 2023).

3.1.1 Use of notation

In discussing and comparing the various intermolecular potentials in this chapter, an effort has been made to standardize the notation for various common parameters. This inevitably means that the notation will sometimes differ from that given in the original articles. Symbols are defined when they first occur and the definition is not necessarily repeated for subsequent potentials involving the same quantity. For example, irrespective of the potential, the symbol ε always denotes the depth of the potential well.

3.2 Intermolecular forces

In principle, the development of intermolecular potentials should be informed by the nature of intermolecular forces. However, in practice the role of theory often takes second place to empirical considerations. A brief background of the nature of intermolecular forces is given here, which is discussed in much greater detail elsewhere (Maitland et al., 1981; Gray and Gubbins, 1984; Stone, 2013).

3.2.1 Types of intermolecular forces

Calculation of the potential energy inevitably involves assumptions concerning the nature of attraction and repulsion between molecules. Intermolecular interaction is the result of both short- and long-range (LR) effects.

Electrostatic, induction, and dispersion effects are examples of LR interactions. In these cases, the energy of interaction is proportional to some inverse power of intermolecular separation. Electrostatic interactions result from the static charge distribution between molecules. The effect can be either attractive or repulsive, and it is exclusively pairwise additive. Induction effects are always attractive, resulting from the distortions caused by the molecular fields of neighboring molecules. However, the most important contribution is the attractive

influence of dispersion arising from instantaneous electric field caused by electron movement. Neither induction nor dispersion is pairwise additive.

Short-range (SR) interactions are characterized by an exponential decay in the interaction energy with respect to intermolecular separation. At small intermolecular separations, there is a significant overlap of the molecular wave functions causing either intermolecular exchange or repulsion. These interactions are not pairwise additive (Stone, 2013).

It is possible to calculate the intermolecular interactions from first principles. However, in practice the first principles or ab initio approach is confined to relatively simple systems. More commonly, the influence of intermolecular interaction is expressed by some type of intermolecular potential. The justification for the intermolecular potential is often entirely empirical, although, as will be discussed in Chapter 4, it is possible to determine an ab initio potential during the course of a simulation (Car and Parrinello, 1985).

3.2.2 Quantum theory of intermolecular potentials

The theoretical background for the development of intermolecular potentials is discussed in detail elsewhere (Maitland et al., 1981; Gray and Gubbins, 1984; Stone, 2013) and our discussion will be limited only to the most salient features. There are two distinct methods for developing an intermolecular potential (Pople, 1982). In the spirit of the LJ potential, the total interaction can be partitioned into LR and SR contributions. The LR part is typically proportional to inverse powers of the intermolecular distance, whereas the SR repulsive part is associated with exchange or overlap of the electronic distributions. The two contributions can be evaluated separately, and the total potential is obtained by combining the two parts.

$$\Delta U_{ab} = \Delta U_{ab}^{LR} + \Delta U_{ab}^{SR} \tag{3.3}$$

The alternative approach is to treat the interacting $a \ldots b$ complex as a single large molecule or "supermolecule." The interaction potential is obtained by calculating the energy of ab and subtracting the energies of the separated a and b molecules.

$$\Delta U_{ab} = U_{ab} - U_a - U_b \tag{3.4}$$

Both of the above methods can be considered in terms of the principle of quantum mechanics. Molecules a and b are associated with the unperturbed wave functions ψ_a and ψ_b, respectively. The interaction Hamiltonian V_{ab} resulting from Coulombic interaction between the electrons and nuclei of a and the electrons and nuclei of b can be introduced as a perturbation. If the wave functions do not overlap, the interaction energy resulting from a first-order perturbation (Pople, 1982; Stone, 2013) is:

$$\Delta_1 U_{ab}^{LR} = \langle \Psi_a \Psi_b | V_{ab} | \Psi_a \Psi_b \rangle \tag{3.5}$$

Eq. (3.5) is the rigid electronic interaction potential resulting from the Coulombic interaction of unperturbed electronic distributions of the isolated a and b molecules. This first-order treatment fails to describe the LR interaction between spherical systems. To account for dispersion interaction, it is necessary to introduce the interaction potential V_{ab} to second order. In terms of the Rayleigh−Schrödinger perturbation theory (Pople, 1982; Stone, 2013), the second-order contribution to the LR potential is,

$$\Delta_2 U_{ab}^{LR} = -\sum_{i\neq 0}\sum_{j\neq 0}\frac{\left|\left\langle \Psi_a\Psi_b|V_{ab}|\Psi_a^i\Psi_b^j\right\rangle\right|^2}{U_a^i - U_a + U_b^j - U_b}, \tag{3.6}$$

where the superscripts represent the ith excited state. From Eq. (3.6) it can be shown (Pople, 1982; Stone, 2013) that the dispersion potential for spherical atoms is,

$$\Delta U_{ab}^{disp} = \frac{C_6}{r^6} + \frac{C_8}{r^8} + \cdots\cdots, \tag{3.7}$$

where C_6, C_8 ... are coefficients that depend on relative orientations. These contributions appear explicitly in many intermolecular potentials.

A third-order perturbation treatment is required to determine the contribution of three-body interactions. Axilrod and Teller (1943) and Mutō (1943) formulated an explicit expression for triple-dipole interactions. Third-order perturbation theory can also be used to obtain the higher multipole moments that contribute to three-body interactions (Bell and Zucker, 1976). This topic is discussed in greater detail in Chapter 4.

In the supermolecule method, the interaction between molecules a and b is not treated as a perturbation. Instead, the wave function ψ_{ab} for the compound a .... b system is calculated for every required relative intermolecular configuration.

$$\Delta U_{ab} = \langle\Psi_{ab}|H_{ab}|\Psi_{ab}\rangle - \langle\Psi_a|H_a|\Psi_a\rangle - \langle\Psi_b|H_b|\Psi_b\rangle \tag{3.8}$$

Eq. (3.8) has the advantage that it takes account of repulsive interactions at short intermolecular separation. The disadvantages of the supermolecule approach center on the limitations of the wave functions that are used. Most supermolecule calculations use the Hartree−Fock procedure, which fails to account for dispersion interactions. The usefulness of supermolecule Hartree−Fock calculations is confined to polar systems dominated by electrostatic interactions. To be generally useful, the supermolecule approach must be used with a post-Hartree−Fock technique (Chapter 4).

3.3 Potentials with a hard sphere contribution

Potentials with a hard sphere(s) (HS) term have a special place (Mulero, 2008) in the application of molecular simulation because they represent the

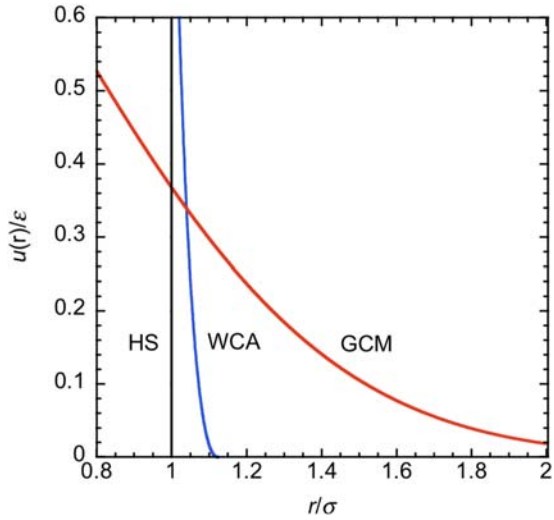


FIGURE 3.1 Comparison of some repulsive potentials, representing HS (Eq. (3.9)), WCA (Eq. (3.10)), and GCM (Eq. (3.19)) interactions.

simplest possible potential that nonetheless can at least approximately represent some of the properties of real fluids. The scope of such potentials can be systematically expanded via the addition of attractive contributions, resulting in "hard sphere + attractive term" (HSA) potentials. A comparison of some repulsive potentials is given in Fig. 3.1.

3.3.1 The HS potential

The simplest approximation is to treat atoms as impenetrable HS, that is,

$$u_{HS}(r) = \begin{cases} \infty & r \leq \sigma \\ 0 & r > \sigma \end{cases} \tag{3.9}$$

where σ is the HS diameter (Fig. 3.1). In a molecular simulation, special procedures are required (Allen and Tildesley, 2017; Wu and Sadus, 2005) to evaluate the effect of the HS potential. The properties of Eq. (3.9) are exclusively evaluated via pair interactions and there are no meaningful contributions of multibody interactions. Accurate values of the compressibility factors for the HS system have been obtained up to very high densities (Wu and Sadus, 2005), and its thermal conductivity has been investigated (Pieprzyk et al., 2020).

The HS potential remains of considerably utility as the theoretical basis for the development of HS equations of state (Sadus, 1992). The properties of the HS potential have been investigated extensively by molecular simulation (Boublík and Nezbeda, 1986; Mulero, 2008). Alder and Wainwright (1957) reported the first MD simulation for HS. Their data were found to be

in good agreement with MC results (Wood and Jacobson, 1957), thereby demonstrating the equivalence of the two molecular simulation methods. The transport coefficients of HS fluids (Alder et al., 1970) have been investigated and molecular simulation studies of mixtures of HS have been reported (Boublík and Nezbeda, 1986). The conventionally accepted view is that the HS fluid is only capable of a single solid-fluid transition, although there are some simulation data that suggest the possibility of an additional transition (Wu and Sadus, 2004). However, the absence of attractive interactions means that it cannot even predict qualitatively the properties of real fluids.

3.3.2 Non-HS repulsive potentials

The HS potential is a purely repulsive potential. However, the extent of its repulsion is unrealistically steep compared with the behavior of real fluids and its reliance on determining particle overlap makes it awkward to implement in MD compared with continuous potentials.

3.3.2.1 Weeks–Chandler–Anderson potential

The Weeks–Chandler–Anderson (WCA) potential (Weeks et al., 1971) provides a widely used non-HS alternative for purely repulsive interactions (Fig. 3.1). The WCA potential modifies the LJ potential (see Section 3.6.1):

$$u_{WCA} = \begin{cases} 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] + \varepsilon, & r \le 2^{1/6}\sigma \\ 0, & r > 2^{1/6}\sigma \end{cases} \tag{3.10}$$

In Eq. (3.10), the LJ potential is made repulsive by simply adding the value of ε at all separations below the minimum of the potential, which always yields a finite positive value. Eq. (3.10) provides a more realistic description of repulsive interactions that is easier to implement the infinitely steep HS term. It has been incorporated to study the behavior of molecular fluids such as polymers (Kröger, 2005) and dendrimers (Bosko et al., 2004, 2006), and it forms part of many theories (Chandler et al., 1983) of the liquid state. Eq. (3.10) has been useful for the simulation of chains composed of hard tangent spheres (Gao and Weiner, 1989). It yields properties that are broadly similar to the HS potential, for example, only solid–liquid equilibria (SLE) are possible (Ahmed and Sadus, 2009a; Dyre and Pedersen, 2023). It has been observed that SLE behavior do not necessarily scale inversely with the number of particles (N) and $N \geq 4000$ is recommended (Ahmed and Sadus, 2009a).

Although the WCA potential has been overwhelmingly tied to the 12–6 LJ potential, it could easily be re-formulated with many other suitable potentials. Indeed, it is generally accepted that 12-power exponent used in the 12–6 LJ version of the WCA potential, while a better alternative to the HS potential, is itself too steep to accurately represent repulsion in real fluids.

3.3.3 HSA potentials

It is well-known that properties of most real fluids require interactions from both intermolecular repulsion and attraction. In particular, vapor–liquid equilibria (VLE) invariably require the cohesion generated by intermolecular attraction. Comparisons of some HSA potentials are given in Fig. 3.2.

3.3.3.1 The Square well potential

The square well (SW) potential is the simplest HSA intermolecular potential that is capable of representing the properties of liquids,

$$u_{\mathrm{HSA}}(r) = \begin{cases} \infty & r \le \sigma \\ -\varepsilon & \sigma < r \le \lambda\sigma \\ 0 & r > \lambda\sigma \end{cases}, \tag{3.11}$$

where λ is some multiple of the HS diameter. The SW potential represents a mathematically idealized model of molecular interactions. The properties of the SW fluid have been investigated widely (Haile, 1992), and it remains a useful starting point for the development of fluid state theories (Yuste and Santos, 1994). By varying λ, Eq. (3.11) can be used as an approximation for continuous potentials, which is useful in some theoretical evaluations (Paricaud, 2006). For example $\lambda = 1.5$ (Elliott and Hu, 1999) is sometimes used as an approximation of the LJ potential (Fig. 3.2). The overall size of both the VLE phase envelope and the critical point are noticeably affected by the choice of λ (Vega et al., 1992a; Orkoulas and Panagiotopoulos, 1999; Gloor et al., 2004; Patel et al., 2005).

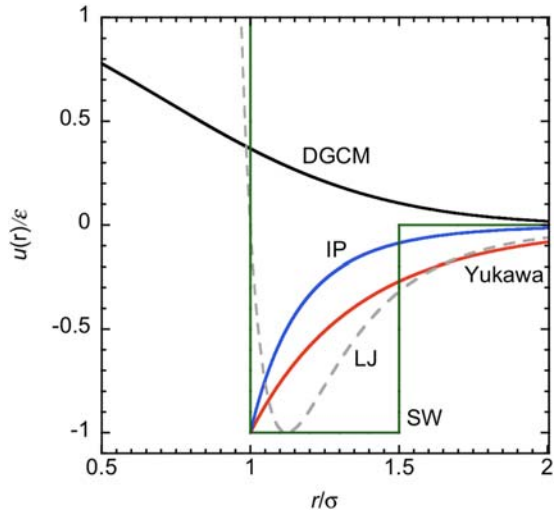


FIGURE 3.2 Comparison of repulsion + attraction represented by the DGCM ($\varepsilon_2 = 0.02\varepsilon_2$, $\zeta = 3\sigma_1$, the small attractive minimum is at $3.017\sigma_1$), Yukawa ($z = 1.8$), IP ($n = 6$), SW ($\lambda = 1.5$), and LJ (dashed line) potentials.

3.3.3.2 Penetrable SW potential

A modification of the SW potential is to allow a small degree of particle penetration. The penetrable sphere attractive (PSA) potential (Santos et al., 2008) is

$$u_{\mathrm{HSA}}(r) = \begin{cases} \varepsilon_{\mathrm{rep}} & r \le \sigma \\ -\varepsilon & \sigma < r \le \lambda\sigma \\ 0 & r > \lambda\sigma \end{cases}, \tag{3.12}$$

where the infinite repulsion is replaced by a finite value ($\varepsilon_{\mathrm{rep}}$) and the $\varepsilon_{\mathrm{rep}}/\varepsilon$ ratio can be considered as a measure of penetrability. Simulations of the phase behavior of the PSA potential have been compared (Fantoni et al., 2011) with the SW potential. Values of $\lambda = 1.5$ and $\varepsilon_{\mathrm{rep}}/\varepsilon = 1.8$ approximately coincide with the SW potential, whereas there are significant deviations from SW behavior for other choices of parameters.

3.3.3.3 Triangular well potential

There are various possible variations on the HSA concept. For example, it is possible to envisage a triangular potential (Feinberg and de Rocco, 1964; Fowler et al., 1965) of the following form:

$$u_{\mathrm{HSA}}(r) = \begin{cases} \infty & r \le \sigma \\ \dfrac{\varepsilon}{\lambda - 1}\left(\dfrac{r}{\sigma} - \lambda\right) & \sigma < r \le \lambda\sigma \\ 0 & r > \lambda\sigma \end{cases} \tag{3.13}$$

It has been used to investigate the VLE of both two- (Reyes et al., 2016) and three-dimensional (Bárcenas et al., 2014) fluids, indicating broadly similar λ-dependent behavior to that observed for the SW fluid.

3.3.3.4 Yukawa potential

Considerable interest has been demonstrated in simulating the properties of atoms interacting via the HS Yukawa potential (Rudisill and Cummings, 1989; Smit and Frenkel, 1991; Rosenfeld, 1993, 1996; Kalyuzhnyi and Cummings, 1996). The Yukawa potential can be written as

$$u_{\mathrm{HSA}}(r) = \begin{cases} \infty & r \le \sigma \\ -\dfrac{\varepsilon\sigma}{r}\exp\left[-z\left(\dfrac{r}{\sigma} - 1\right)\right] & r > \sigma \end{cases}, \tag{3.14}$$

where z is an adjustable parameter. Part of the appeal of this potential is that its behavior approximately reflects (Rudisill and Cummings, 1989) other more complicated potentials. The inverse power dependence of this potential means that it can be applied to ionic systems (Rowlinson, 1989). When

$z = 1.8$ the results obtained (Duh and Mier-Y-Terán, 1997) are similar to the LJ (12−6) potential. The phase diagram resulting from the Yukawa potential has been reported (Rudisill and Cummings, 1989; Smit and Frenkel, 1991; Galicia-Pimental et al., 2008). Kalyuzhnyi and Cummings (1996) have calculated the phase diagram for a LJ fluid modeled by the Yukawa potential. In common with the SW potential, the VLE diagram of the Yukawa potential is very sensitive to the choice of z (Lomba and Almarza, 1994; Shukla, 2000; Spear, 2000). Its VLE properties in the vicinity of the sticky sphere limit have been studied (Schöll-Paschinger et al., 2013). The Yukawa potential has been used (de Carvalho and Evans, 1997) in conjunction with the restricted primitive (see subsequent discussion) model to calculate the properties of a charged HS binary fluid. It has also been coupled (Máte et al., 2011) with dipolar interactions for the heat capacities. The potential has also proved useful for studying colloids and proteins (Valdez-Pérez et al., 2012).

3.3.3.5 Inverse power potential

A more realistic HSA potential can be obtained by using the attractive term from an inverse power (IP) law, that is,

$$u_{\mathrm{HSA}}(r) = \begin{cases} \infty & r \leq \sigma \\ -C_n r^{-n} & r > \sigma \end{cases}, \tag{3.15}$$

where n and C_n are constants. When $n = 6$, Eq. (3.15) is the Sutherland potential and C_6 is the leading dispersion coefficient (Fig. 3.2). No molecular simulations have been reported using this potential. This lack of interest can be partially explained by the greater flexibility offered by other HSA potentials via the simple expedient of varying the depth of the attractive well. It should also be noted that values of C_6 obtained by parameterization with experimental data are unrealistically large (Sherwood and Prausnitz, 1964). It nonetheless may have a useful future role in identifying the relative contribution of different dispersion coefficients on the properties of fluids.

3.3.3.6 Sticky sphere potential

Baxter (1968) proposed a HAS potential that has become known as the "sticky sphere" potential. It combines HS repulsion with an infinitely narrow potential well when $\lambda - \sigma$ is allowed to be infinitesimally small, that is,

$$u_{\mathrm{HSA}}(r) = \begin{cases} \infty & r < \sigma \\ kT \log_e \left[\dfrac{12\tau(\lambda - \sigma)}{\lambda} \right] & \sigma < r < \lambda \\ 0 & r > \lambda \end{cases}, \tag{3.16}$$

where k is Boltzmann's constant, T is temperature, and τ is a stickiness parameter that is related to the temperature. Unlike other HSA potentials,

Eq. (3.16) is temperature-dependent. As detailed by Stell (1991), Eq. (3.16) has proved useful in describing mono-dispersed systems and nonionic micelles (Rao et al., 1991).

3.3.3.7 Kihara hard core potential

The HSA approach can be generalized to any hard core (HC) plus attraction (HCA) interactions. The Kihara potential (Kihara, 1951; Maitland et al., 1981) is the best-known example of this approach,

$$u_{\mathrm{HCA}}(r) = \begin{cases} \infty & r \le d \\ 4\varepsilon\left[\left(\dfrac{\sigma-d}{r-d}\right)^{12} - \left(\dfrac{\sigma-d}{r-d}\right)^{6}\right] & r > d, \end{cases} \tag{3.17}$$

where d is the diameter of an impenetrable HC at which $u(r) = \infty$, which is used to modify the contribution of the LJ potential. The Kihara potential has the advantage that it can be applied to nonspherical molecule by using a convex core of any shape. It has been studied extensively (Sherwood and Prausnitz, 1964; Vega et al., 1992b; Tee et al., 1966), and it generally predicts the second virial coefficient more accurately than the LJ potential. The potential, which forms the theoretical basis of hard convex body (HCB) equations of state (Wei and Sadus, 2000), has not been studied widely by molecular simulation. The hardness of Eq. (3.17) can obviously be softened by removing the requirement that $u(r) = \infty$ for r $\le$ d.

3.4 Soft sphere potentials

A simple and more realistic alternative to the HS potential is the soft sphere (SS) potential, which is not infinite at interatomic separations less than the sphere diameter. SS potentials typically use an IP relationship, for example,

$$u_{\mathrm{SS}}(r) = \begin{cases} C\left(\dfrac{\sigma}{r}\right)^{n} & r \le \sigma \\ 0 & r > \sigma \end{cases} \tag{3.18}$$

In Eq. (3.18), C is a constant. In common with HS potentials, the absence of attractive interactions means that SS potentials cannot be used to model real liquids. Neither vapor−liquid nor liquid−liquid equilibria are possible for particles interacting via a SS potential. However, many simulation studies have been reported using SS potentials to determine the effect of repulsion in fluids and SLE.

MC simulations have been reported (Hoover et al., 1970, 1971) using the inverse fourth-, sixth-, ninth-, and twelfth-power SS potential to determine the thermodynamic properties of fluids and face-centered-cubic solids.

The fluid−solid melting line of particles interacting via various SS potentials has been determined (Hoover et al., 1972) by MC simulation and the influence of interatomic repulsion of the structure of liquids at melting has been investigated (Hansen and Schiff, 1973). The sixth-power potential has been used (Laird and Haymet, 1989) to study the crystal−liquid interface of a body-centered-cubic solid. Substances interacting via a sixth-power potential undergo a transition between body-centered-cubic and face-centered-cubic orientations at temperatures slightly above the freezing point (Laird and Haymet, 1992).

The general thermodynamic properties of such potentials have been extensively documented (Hamad, 1995). Investigations of the n-dependence of transport properties (Heyes and Powles, 1998) and radial distribution functions (Heyes et al., 2004) are also available.

3.5 Fully interpenetrable potentials

The defining feature of the HS and HSA potentials is that the particles are impenetrable below a separation of σ. This requirement is partially relaxed for the SS potentials (Eq. (3.18)) but repulsion nonetheless rises very steeply at small interparticle separations. An interesting alternative to this conventional approach is to completely relax this barrier and allow the particles to be fully interpenetrable at all interparticle separations.

3.5.1 Gaussian core model potential

The Gaussian core model (GCM) potential (Stillinger, 1976) is an example of a fully interpenetrable potential.

$$u_{\mathrm{GCM}}(r) = \varepsilon \exp\left[-\left(\frac{r}{\sigma}\right)^2\right] \tag{3.19}$$

It is apparent from Eq. (3.18) that the value of the potential increases in a controlled fashion for $r < \sigma$, attaining a maximum value of ε (Fig. 3.1). This means that complete interparticle penetration is possible without a very steep increase in energy. The potential rapidly decays to 0 for $r > \sigma$. These behaviors may appear to be physically unrealistic, but it is nonetheless useful (Likos, 2001) for phenomena involving intra-particle penetrations, such as networks of polymers or gels. The GCM potential also reproduces (Ahmed et al., 2009, 2010; Mausbach and Sadus, 2011) some of the anomalies (Shvab and Sadus, 2016) exhibited by water, which are attributed to the extensive network formed by hydrogen bonds. In common with water, the thermal expansion coefficient of the GCM potential is negative and the isochoric heat capacity has a minimum. Only SLE are observed (Prestipino et al., 2005; Mausbach et al., 2009) for the GCM potential with the unusual feature of a re-entrant solid phase.

3.5.2 Double Gaussian core model potential

The GCM potential can be modified (Prestipino, et al., 2014) to include a Gaussian shape attractive term (Fig. 3.2). The result of this modification is the double Gaussian core model (DGCM) potential.

$$u_{DGCM}(r) = \varepsilon_1 \exp\left[-\left(\frac{r}{\sigma_1}\right)^2\right] - \varepsilon_2 \exp\left[-\left(\frac{r-\zeta}{\sigma_2}\right)^2\right] \tag{3.20}$$

The DGCM potential has both maximum ($u_{DGCM}(0) = \varepsilon_1$) and minimum ($u_{DGCM}(\zeta) = -\varepsilon_2$) values, when $r = 0$ and $r = \zeta$, where $\zeta > \sigma_1$. There are also two characteristic distances, satisfying the condition that $\sigma_2 \geq \sigma_1$. A consequence of adding attractive interactions is the DGCM displays VLE (Speranza et al., 2014; Losey and Sadus, 2019). Compared to the GCM potential, the DGCM potential also exhibits more water-like anomalies in its thermodynamic properties.

3.6 Effective pairwise potentials for atoms and simple molecules

The proceeding discussion of HS, HSA, SS, GCM, and DGCM potentials have involved largely theoretical systems, which do not attempt directly to describe the behavior of real systems. In contrast, many effective pairwise potentials have been specifically developed (Maitland et al., 1981; Stone, 2013) to predict the properties of either atoms or small molecules. Much of this work has predated the advent of molecular simulation, but it is the application of molecular simulation that is decisive in evaluating both the strengths and weaknesses of intermolecular potentials.

Historically, an empirical approach was used with the parameters of the potential being obtained from experimental data such as second virial coefficients, viscosities, molecular beam cross sections, and the like. Conclusions regarding the accuracy of pair potential were made by comparing the properties predicted by the potential with experiment. In contrast, computer simulation permits the theoretically rigorous evaluation of the accuracy of intermolecular potentials. However, very few potentials have been tested extensively using molecular simulation. Notable exceptions are the HS, LJ, and exp-6 potentials.

Effective pair potentials for atoms are often incorporated into the molecular simulation of polyatomic molecules and increasingly, macromolecules. Therefore, the atomic pair potential is an important starting basis for predicting molecular properties. In view of this, the evaluation of pair potentials by molecular simulation is likely to be increasingly important for the prediction of molecular fluids.

3.6.1 Repulsion + dispersion pair potentials

3.6.1.1 Generalized n−m *Lennard-Jones/Mie potential*

Typically, an adequate representation of the intermolecular forces between nonpolar molecules can be obtained by combining repulsion and dispersion

terms to generate a pair potential. Many "repulsion + dispersion" pair potentials have been developed; however, relatively few different types of potentials are used routinely in molecular simulation.

In contrast to the HSA potentials, a more realistic representation of intermolecular interaction is given by continuous intermolecular potentials. Mie (1903) and later Lennard-Jones (Jones, 1924) proposed empirical potentials of the type,

$$u_{eff}(r) = \frac{\upsilon}{r^n} - \frac{\mu}{r^m}, \tag{3.21}$$

where υ, μ, n, and m are constants. Fowler and Guggenheim (1939) and Mayer and Mayer (1940) observed, without any derivation, that Eq. (3.21) could be generalized as,

$$u_{eff}(r) = \varepsilon\left[\left(\frac{m}{n-m}\right)\left(\frac{r_m}{r}\right)^n - \left(\frac{n}{n-m}\right)\left(\frac{r_m}{r}\right)^m\right], \tag{3.22}$$

where $m > 3$ and r_m is the separation corresponding to the minimum energy corresponding to $u_{eff}(r) = -\varepsilon$. The separation ($\sigma$) when $u_{eff}(r) = 0$, which is the HS diameter in the HS potentials, is related to r_{m} via:

$$\sigma = r_m\left(\frac{m}{n}\right)^{\frac{1}{n-m}} \tag{3.23}$$

Alternatively, making use of Eq. (3.23) to determine r_{m} from σ transforms Eq. (3.22) to,

$$u_{eff}(r) = \varepsilon\left(\frac{n}{n-m}\right)\left(\frac{n}{m}\right)^{\left(\frac{m}{n-m}\right)}\left[\left(\frac{\sigma}{r}\right)^n - \left(\frac{\sigma}{r}\right)^m\right], \tag{3.24}$$

which has the advantage of only involving a common single pre-factor term. This simplification is not obvious from Eq. (3.22).

In the literature, Eqs. (3.22) and (3.24) are interchangeably referred to as either the generalized Mie (1903) or generalized LJ (Jones, 1924) potential with the former designation being the most common. Neither Mie nor Lennard-Jones proposed the generalization of the potentials given by either Eq. (3.22) or Eq. (3.24) in their original publications. Possibly the earliest contemporaneous representation of Eq. (3.22) can be found in a 1939 textbook on statistical mechanics (Fowler and Guggenheim, 1939). Eq. (3.24) only appears in much later textbooks (Reed and Gubbins, 1973). It appears that the convenience of using Eq. (3.24) compared with Eq. (3.22) was not immediately apparent. We will refer to either Eq. (3.22) or Eq. (3.24) as the LJ/M potential to equally acknowledge both contributors in alphabetical order.

Changing the n and m exponents significantly alters the nature of the intermolecular potential as is illustrated in Fig. 3.3. When $n = 12$ and $m = 6$, we obtain the most-widely used version of the LJ/M potential, which is

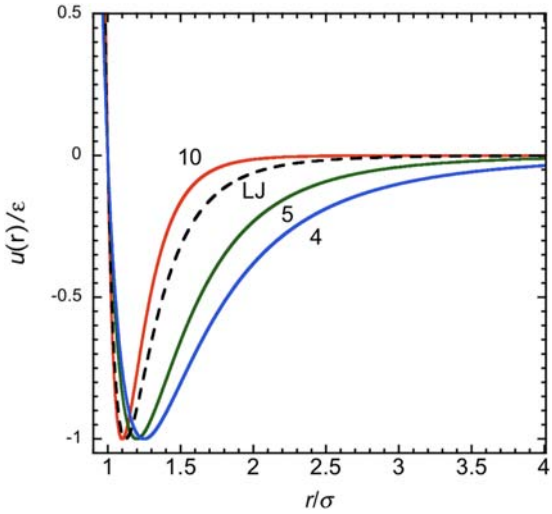


FIGURE 3.3 Comparison of the energy curve for the ($n = m + 1$, m) LJ/M potentials. Results are shown for $m = 4$, 5, and 10 and compared with the values obtained for the LJ potential (dashed line).

simply known as the LJ potential, which is the most common form of the LJ/M potential.

$$u_{eff}(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{3.25}$$

The LJ potential was previously encountered in the context of the WCA potential (Eq. (3.11)), and it is undoubtedly the most-widely used potential for molecular simulation. Wood and Parker (1957) investigated the LJ potential by MC simulations, and the first MD simulation of a LJ fluid was reported by Rahman (1964). It yields a reasonable overall qualitative representation for the properties of argon, and it has been applied to many other atoms with varying success. Its usefulness for real fluids has been discussed elsewhere (Wang et al., 2020) and, despite its many limitations, it is likely to remain in widespread use for the foreseeable future.

In the LJ potential, intermolecular repulsion and dispersion interactions are represented by different IP dependencies of intermolecular separations. The r^{-6} dependence of the attractive term is consistent with the leading term of the $1/r$ expansion of dispersion energy; however, the dispersion coefficient (i.e., $C_6 = 4\varepsilon\sigma^6$) is typically overestimated by a factor of 2 compared with experimental values. This can be partly explained partly by the absence of r^{-8} and r^{-10} terms. It should be noted that there is no theoretical justification for the r^{-12} repulsion term.

The LJ potential is often used as part of a larger potential for molecular systems. For example, it is used frequently to calculate the interactions between atoms that constitute a model of a many-atom molecule such as a polymer

(Escobedo and de Pablo, 1996; Johnson and Gubbins, 1992; Johnson et al., 1994). It is also used commonly as part of a molecular mechanics potential (see discussion below).

The appeal of the LJ potential is that it combines a realistic description of the intermolecular interaction separation with computational simplicity. The application of the LJ potential requires the evaluation of the intermolecular parameters. This is commonly done by fitting the predicted second virial coefficient (B) of the LJ potential to experimental data for real atoms or molecules. This is a convenient route because the LJ/M has an analytical solution for B, that as discussed elsewhere (Sadus, 2018, 2019) can be easily evaluated, that is,

$$B(T) = \frac{2}{3}\pi\sigma^3 F(y), \tag{3.26}$$

where

$$F(y) = y^{3/(n-m)}\left\{\Gamma\left(\frac{n-3}{n}\right) - \frac{3}{n}\sum_{i=1}^{\infty}\Gamma\left(\frac{im-3}{n}\right)y^i/i!\right\}, \tag{3.27}$$

$$y^n = \left(\frac{n}{n-m}\right)^n\left(\frac{n-m}{m}\right)^m\left(\frac{\varepsilon}{kT}\right)^{n-m}, \tag{3.28}$$

and Γ is the gamma function (Abramowitz and Stegun, 1972). Tables of parameter values for many atoms and small atoms are available in the literature (Hirschfelder et al., 1954; Maitland et al., 1981; Gray and Gubbins, 1984) and are almost universally accepted and used. For example, the often-quoted values for argon are $\varepsilon/k = 119.8$ K and $\sigma = 0.3405$ nm. The parameters for the LJ potential for noble gases have been re-evaluated (Sadus, 2018), which has resulted in better overall agreement with the experimental B data (Table 3.1). However, the

TABLE 3.1 Comparison of literature values (Maitland et al., 1981) and re-evaluated (Sadus, 2019) LJ parameters obtained from the experimental second coefficient data for noble gases. The agreement with experiment for the entire temperature range is quantified using the root mean squared deviation (RMSD).

Atom	Literature				Re-evaluated		
	T Range (K)	ε/k (K)	σ (nm)	RMSD (cm^3/mol)	ε/k (K)	σ (nm)	RMSD (cm^3/mol)
Ne	50–870	47.0	0.272	9.81	34.91	0.278	0.40
Ar	76–1000	119.8	0.341	9.33	120.32	0.342	7.10
Kr	110–870	164.0	0.383	6.48	175.34	0.366	5.81
Xe	165–970	222.2	0.401	20.30	226.51	0.408	8.16

agreement with experiment remains imperfect, which reflects the fact the LJ potential is not a two-body potential. Simply inserting an effective pair potential into the formula for a genuine two-body property such as the second virial coefficient does not transform the potential to a two-body potential. It is noteworthy that although the LJ potential is commonly attributed in the literature are being at least approximately suitable for argon, the deviation for experimental B values (Table 3.1) is significant.

The LJ potential is unarguably the most popular and widely used variant of the LJ/M potentials. However, other combinations of the n and m exponents have been investigated via molecular simulation. In particular, the n-6 LJ/M potentials have been the focus of several studies (Kiyohara et al., 1996; Okumura and Yonezawa, 2000; Gordon, 2006; Ahmed and Sadus, 2009b; Orea et al., 2008; Potoff and Bernard-Brunel, 2009). The purpose of these studies has been to investigate the effect of varying n on thermodynamic properties. The extent of both VLE (Kiyohara et al., 1996; Okumura and Yonezawa, 2000) and SLE (Ahmed and Sadus, 2009b) varies systematically with n. It has been demonstrated (Potoff and Bernard-Brunel, 2009; Potoff and Kamath, 2014; Mick et al., 2015) that experimental VLE data of both the noble gases and a range of molecules can be accurately fitted to n-6 LJ/M potentials by using n as an adjustable parameter. For example the 14−6 and 36−6 LJ/M potentials yield optimal agreement (Potoff and Bernard-Brunel, 2009) for the VLE properties of methane and tetrafluoromethane, respectively.

In contrast, studies of $n-m$ LJ/M potentials for $m \neq 6$ are much less common, which can be at least partly attributed to the fact $m = 6$ corresponds to the decay of the leading term of dispersion interactions. Sadus (2018) mapped the behavior of the second virial coefficients for many combinations of n and m values. The n-4 LJ/M potentials (Sadus, 2020a) display a high degree of cohesive behavior and the VLE diagrams in some cases are similar to that observed for dipolar systems. The (m + 1, m) LJ/M potential has also been investigated (Sadus, 2020b) systematically resulting in a range of m-dependent thermodynamic behavior.

It is apparent that there are many different combinations of the n and m exponents. Therefore, it is likely that some of the different combinations may result in potentials that are very closely related resulting in the prediction of broadly similar phenomena. That is, a given combination of n and m exponents does not necessarily yield a unique LJ/M potential. Mejía et al. (2014) and Ramrattan et al. (2015) proposed that different combinations of the n and m exponents may yield similar behavior if they share a common value of α defined by:

$$\alpha = \left(\frac{n}{n-m}\right)\left(\frac{n}{m}\right)^{\frac{m}{n-m}}\left(\frac{1}{m-3} - \frac{1}{n-3}\right) \tag{3.29}$$

Values of α for various $n-m$ potentials are compared in Fig. 3.4, which highlights cases of constant $\alpha = 0.5$ and 1.6. It is apparent that when $m \geq 6$

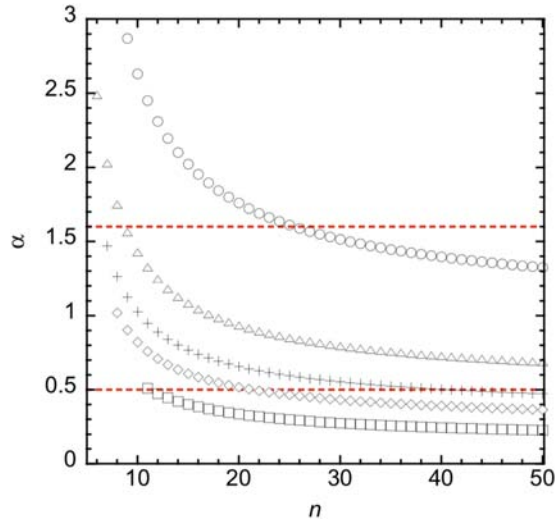


FIGURE 3.4 Comparison of α as a function of n for various n–m potentials, where $m = 4$ (○), 5 (△) 6 (+), 7, (◇), and 10(□). The dotted red lines for $\alpha = 0.5$ and 1.6 contrast the greater number of possible combinations for m ≥ 6 compared with $m = 4$, which can only intersect with $m = 5$ data.

that there are likely to be several broadly equivalent cases involving other m values. In contrast, when $m = 4$, intersection is only possible for $m = 5$ cases. It should be noted that the exact equivalence between different LJ/M potentials would generally require the use of nonintegral n and m exponents. This type of analysis is useful to avoid the unintended duplication of simulations involving broadly similar LJ/M potentials.

3.6.1.2 Egelstaff potential

Egelstaff (1987, 1994) proposed the following variant of the LJ potential,

$$u_{eff}(r) = \varepsilon\left[2\left(\frac{r_m}{r}\right)^{12} - 2\left(\frac{r_m}{r}\right)^{9} - \left(\frac{r_m}{r}\right)^{6}\right], \tag{3.30}$$

where r_m is the distance at which there is maximum attraction. In Eq. (3.30), the inclusion of the r^{-9} term means that the C_6 dispersion coefficient is closer to the experimentally determined value.

3.6.1.3 Maitland–Smith potential

There is no theoretical basis to justify the r^{-12} dependence of repulsion in the LJ, Kihara, or Egelstaff potentials. This fact motivated Maitland and

Smith (1973) to propose the following modification of the LJ potential using a generalized distance term (x).

$$u_{eff}(r) = \varepsilon\left[\left(\frac{6}{n-6}\right)x^{-n} - \left(\frac{n}{n-6}\right)x^{-6}\right] \tag{3.31}$$

Eq. (3.31) retains the r^{-6} dependence of dispersion but the value of n, which governs the distance-dependence of repulsion is obtained from

$$n = 13.0 + \gamma(x-1), \tag{3.32}$$

where γ is an adjustable parameter. This three-parameter potential more accurately (Kohler et al., 1976) describes the properties of inert gases than the LJ potential.

3.6.1.4 Born–Mayer and London potentials

The early work of Born and Mayer (1932) indicated that the repulsion between atoms should have an exponential dependence on distance because of the overlap of wave functions. If the Born–Mayer exponential term is coupled with the London dispersion formula, we obtain

$$u_{eff}(r) = A\exp(-Br) - \frac{C_6}{r^6}, \tag{3.33}$$

where A and B are empirical constants determined from experimental data. London (1937) also recognized the theoretical superiority of the exponential form proposing the following potential

$$u_{eff}(r) = b\exp\left(-\frac{r}{a}\right) - \frac{C_6}{r^6} - \frac{C_8}{r^8}, \tag{3.34}$$

where a and b are adjustable parameters.

3.6.1.5 Buckingham–Corner potential

A limitation of both the Born–Mayer and London potentials is that they go to $-\infty$ at the origin. This deficiency was addressed by Buckingham and Corner (1947), who proposed the following modification

$$u_{eff}(r) = \begin{cases} b\exp\left(-\dfrac{\alpha r}{r_m}\right) - \left(\dfrac{C_6}{r^6} + \dfrac{C_8}{r^8}\right)\exp\left[-4\left(\dfrac{r_m}{r}-1\right)^3\right] & r \le r_m \\ b\exp\left(-\dfrac{\alpha r}{r_m}\right) - \dfrac{C_6}{r^6} - \dfrac{C_8}{r^8} & r > r_m \end{cases}, \tag{3.35}$$

where r_m is the distance at which the energy is a minimum and α is the steepness of the exponential repulsion. Despite their theoretical superiority,

the earlier potentials have not been adopted widely in molecular simulation. Arguably Eq. (3.35) has been superseded by the more practical exp-6 potential (Rice and Hirschfeler, 1954) discussed below. Some applications of Eq. (3.34) have been reported (Homer et al., 1991).

A common feature of Eqs. (3.33) to (3.35) is the use of an exponential term for repulsion. Although these potentials may have not been widely used in molecular simulations, a general exponential model for repulsion has proved useful in many different contexts (Sherwood and Mason, 1965; Kooij and Lerner, 2017; Bacher et al., 2018a,b; Pedersen et al., 2019) and contributed to the development of improved equations of state (Lafitte et al., 2013).

3.6.1.6 Shaviit–Boys potential

Another interesting variation on the use of an exponential term is the potential proposed by Shavitt and Boys (1956),

$$u_{eff}(r) = \frac{4\varepsilon}{\left[\left(\frac{r}{\sigma}\right)^2 + B^2\right]^3} \sum_{i=0}^{\infty} C_{2i} \left\{ \left(\frac{r}{\sigma}\right)^{2i} \exp\left[A - A\left(\frac{r}{\sigma}\right)^2\right] - 1 \right\}, \tag{3.36}$$

where A, B, C_0, C_2, and C_4 are adjustable constants. This potential has been applied (Munn, 1964) to inert gases but no simulation data have been reported.

3.6.1.7 Modified Buckingham (exp-6) potential

Possibly the most-widely used intermolecular potential containing an exponential term is the modified Buckingham or exp-6 potential originally proposed by Rice and Hirschfeler (1954). The exp-6 potential is based on the Born–Mayer potential. For practical applications, the form of the exp-6 potential is,

$$u_{eff}(r) = \begin{cases} \infty & r \le \lambda r_m \\ \dfrac{\varepsilon}{1 - \frac{6}{\alpha}} \left\{ \dfrac{6}{\alpha} \exp\left[\alpha\left(1 - \dfrac{r}{r_m}\right)\right] - \left(\dfrac{r_m}{r}\right)^6 \right\} & r > \lambda r_m, \end{cases} \tag{3.37}$$

where λr_m is the distance at which Eq. (3.37) goes through a false maximum. Eq. (3.37) often appears in the literature without an attribution to work of Rice and Hirschfeler (1954), which is perhaps testimony to its widespread familiarity. The value of λ can be obtained (Hirschfelder et al., 1954) by finding the smallest root of the following equation:

$$\lambda^7 \exp[\alpha(1 - \lambda)] - 1 = 0 \tag{3.38}$$

The false maximum is an unsatisfactory feature of the exp-6 potential. At $r = 0$ the exponential term has a finite value allowing the dispersion term to dominate at very small intermolecular separation. Consequently, the potential

passes through a maximum and then tends to $-\infty$ as $r \to 0$. Therefore, the condition that $u(r) = \infty$ when $r \leq \lambda r_m$ must be imposed to use the potential meaningfully in a simulation. Alternatively, damping functions (see discussion below) for the dispersion term have been proposed, which overcome this problem.

It is apparent from Eq. (3.37) that $\alpha \neq 6$. A value of $\alpha = 7$ corresponds to $\lambda = 1$, which would mean an abrupt transition to infinite repulsion at r_m. Therefore, physically meaningful values commence from $\alpha > 7$. The relationship between α and λ that values that are obtained from solving Eq. (3.38) are illustrated in Fig. 3.5.

Fig. 3.5 shows that when α is reasonably large, the issue of infinite repulsion at small separations for the exp-6 potential is unlikely to be encountered for the properties of fluids under normal conditions.

For real fluids, values of α are typically in the range of 12 to 15 (Wu and Sadus, 2000). For this range of values, the following relationships have been determined (Wu and Sadus, 2000) between σ, λ, and r_{m}.

$$\left.\begin{aligned} \sigma &= \left(0.71311 + 0.01966\alpha - 5.0619 \times 10^{-4}\alpha^2\right) r_m \\ \lambda &= 1.76044 - 0.18315\alpha + 0.00514\alpha^2 \end{aligned}\right\} \tag{3.39}$$

The LJ potential is compared in Fig. 3.6 with the exp-6 potentials for different values of α, which indicates a reduction in the steepness of repulsion when $\alpha < 15$. This partially explains the improved accuracy of the exp-6 potential for the properties of real fluids.

It has been observed (Galliero and Boned, 2008) that a value of $\alpha = 13.772$ approximately corresponds to the LJ and exp-6 having the same

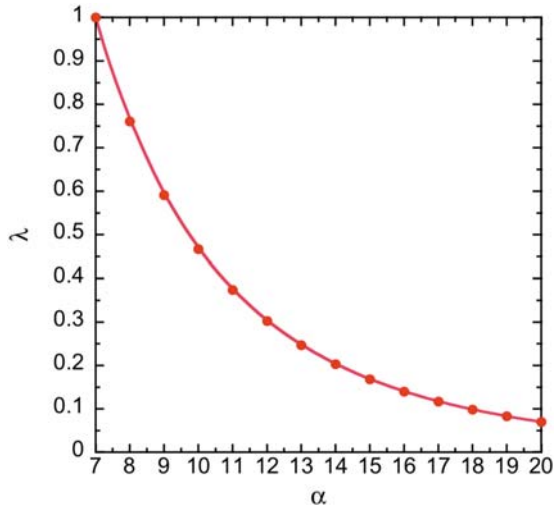


FIGURE 3.5 Values of λ as a function α for the exp-6 potential obtained from solving Eq. (3.38).

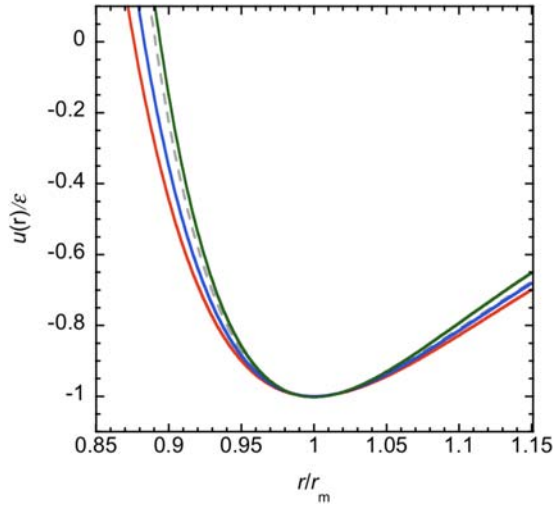


FIGURE 3.6 Comparison of the exp-6 potential with the LJ potential (dashed gray line). Results are given from right to left for $\alpha = 12$ (red line), 13 (blue line), and 15 (green line).

curvature at r_m, whereas $\alpha = 14.338$ yields equal values of energy at r_m. Although such mapping of different potentials is interesting, from the perspective of improving the accuracy of predictions, there is nothing to be gained by effectively limiting the behavior of the exp-6 potential to the inferior LJ potential. Conversely, there would be considerable merit in transforming an accurate but complicated potential to a simpler, but equally accurate, representation that greatly facilitated computations.

Molecular simulation has been used (de Kuijper et al., 1990; Sellers et al., 2015) to evaluate the properties of the exp-6 potential and it has been applied in simulations of the phase behavior of the helium + hydrogen binary mixture (Schouten et al., 1991). Zarragoicoechea and Scalise (1997) used the exp-6 potential to simulate "gas−gas" phase equilibria of nonpolar mixtures. The effect of α on VLE has been investigated (Tavares and Sandler, 1996) using Gibbs ensemble (Panagiotopoulos, 1987) simulation (Chapter 10). Increasing the value of α reduces substantially the range of temperatures for which vapor−liquid coexistence is observed. Wu and Sadus (2000) have demonstrated that the exp-6 potential yields reliable calculation of high pressure VLE of binary atomic mixtures. It has also been used to investigate SLE, involving the freezing transition (Khrapak et al., 2011), high pressure behavior (Saija and Prestipino, 2005), and the possibility of reentrant melting (Saija et al., 2010).

The exp-6 potential has been applied successfully to polar fluids (Peyrovedin and Shariati, 2020). For example, a MD study has been reported (Koshi and Matsui, 1994) for the high temperature phase separation in the water + nitrogen binary mixture. The supercritical properties of water have

been studied (Bastea and Fried, 2008) with an exp-6 potential augmented by dipolar interactions. It has also been used (Galliero and Boned, 2008) to predict transport properties.

3.6.2 General refinement of repulsion and dispersion models

It is evident from the preceding discussion that many successful intermolecular potentials treat repulsion as having an exponential dependence on intermolecular separation. It is well-documented (Stone, 2013) that even the simple potential of Born and Mayer (1932) provides a very accurate description of repulsion at moderate densities. However, as discussed in detail elsewhere (Buckingham et al., 1988; van Vleet et al., 2016), considerable efforts have been made to both improve the prediction of short-ranged repulsion and more accurately represent the contribution of dispersion interactions.

A refined model of repulsion is the exchange Coulomb model (Dham et al., 1989; Stone, 2013), for which the energy of repulsion is represented by

$$E_{rep} = \gamma(1 + ar)\exp\left[a_0 + a_1 + \frac{a_1}{r} + \frac{a_2}{r^2}\right], \tag{3.40}$$

where the constants are fitted using accurate ab initio charge densities for the atoms.

When an exponential repulsion term is coupled to a dispersion term, it is often found that the dispersion term dominates at small intermolecular separation. This can result in a unrealistic intermolecular potential as illustrated by the exp-6 potential, which has a false maximum. This problem is overcome by introducing a damping term for the dispersion term. For example, in the Hartree–Fock dispersion (HFD) model (Hepburn et al., 1975; Ahlrichs et al., 1977),

$$u_{disp}(r) = F(r)\left(\frac{C_6}{r^6} + \frac{C_8}{r^8} + \frac{C_{10}}{r^{10}}\right), \tag{3.41}$$

where

$$F(r) = \begin{cases} \exp\left[-\left(\dfrac{1.28r_m}{r} - 1\right)^2\right] & r < 1.28r_m \\ 1 & r \geq 1.28r_m \end{cases} \tag{3.42}$$

Eq. (3.41) was obtained by fitting the dispersion interactions for molecular hydrogen assuming that the same correction could be applied to all of the dispersion contributions.

Douketis et al. (1982) determined different damping terms for each contribution to dispersion, that is,

$$u_{disp}(r) = -\sum_{n=6,8,10} F_n(r)\frac{C_n}{r^n}, \tag{3.43}$$

where

$$F_n(r) = g(\rho r) f_n(\rho r). \tag{3.44}$$

In Eq. (3.43), f_n is a damping function to correct for charge overlap in the R^{-n} term; g is the correction for exchange effects in the dispersion terms; and ρ is a scaling distance. When ρ = 1, we find (Kreek and Meath, 1969):

$$g(r) = 1 - r^{1.68}\exp(-0.78r) \tag{3.45}$$

$$f_n(r) = \left[1 - \exp\left(-\frac{2.1r}{n} - \frac{0.109r^2}{\sqrt{n}}\right)\right]^n \tag{3.46}$$

Various alternative damping functions have been proposed, which improve the prediction of the potential for various properties (Koide et al., 1981; Tang and Toennies, 1984; Wheatley and Meath, 1993; Boyes, 1994). For example, the damping function for argon proposed by Boyes (1994) improves the prediction of acoustic virial coefficients.

3.6.3 Shifted-force potentials

A common feature of realistic potentials is that the intermolecular potential only asymptotes to zero at large intermolecular separation. That is zero energy is not achieved at large separation. It is sometimes expedient computationally to have a potential, which is exactly zero at distances beyond a cut-off point r_c. This can be achieved by applying the following transformation yielding a shifted-force $u_s(r)$ potential (Haile, 1992; Nicolas et al., 1979).

$$u_s(r) = \begin{cases} u(r) - u(r_c) - (r - r_c)\left(\dfrac{du}{dr}\right)_{r_c} & r \le r_c \\ 0 & r > r_c \end{cases} \tag{3.47}$$

Eq. (3.47) can be applied to any continuous intermolecular potential. The advantage of Eq. (3.47) is that long range corrections (LRC) (Chapter 5) are not required because $u_s(r) = 0$ at and after the cut-off distance (r_c). The use of LRC introduces a degree of uncertainty when comparing different simulation data that use different cut-off values. Ahmed and Sadus (2010) compared the SLE properties from the shift-force LJ potential to those obtained using the full LJ potential. A value of $r_c = 6.5\sigma$ was required to obtain consistent results between the potentials.

3.6.4 SR potentials

An alternative to the shifted-force approach is to develop a continuous potential that deliberately has a restricted range. This approach has the advantages

of both avoiding LRC and discontinuities in the force due to the shift-force approach. Hoover (2006) proposed the following 8−4 potential,

$$u_{eff}(r) = \varepsilon\left[\left(2-\left(\frac{r}{\sigma}\right)^2\right)^8 - 2\left(2-\left(\frac{r}{\sigma}\right)^2\right)^4\right], \tag{3.48}$$

where the interactions between particles are restricted to a very narrow range of separations, that is, $r \leq \sqrt{2}\sigma$. This simple potential has proved particularly useful (Travis and Sadus, 2023) in examining high pressure phase transitions.

In the same spirit of Eq. (3.49), a generalized SR potential can be devised (Wang et al., 2020) depending on the choice of n and m exponents and r_c,

$$u_{eff}(r) = \gamma\varepsilon\left(\left(\frac{\sigma}{r}\right)^{2n} - 1\right)\left(\left(\frac{r_c}{r}\right)^{2n} - 1\right)^{2m} \tag{3.49}$$

where γ is a term that ensures that the depth of the well is -ε.

$$\left.\begin{aligned} r_m &= r_c\left(\frac{1+2m}{1+2mr_c^{2m}}\right)^{1/2m} \\ \gamma &= 2mr_c^{2m}\left(\frac{1+2m}{2m(r_c^{2m}-1)}\right)^{2m+1} \end{aligned}\right\} \tag{3.50}$$

When $n = m = 1$ and $r_c = 2.5\sigma$, Eq. (3.49) generates the conventional phase diagram with VLE, SLE, and triple and critical points. In contrast, $n = m = 1$ and $r_c = 1.2\sigma$ yields a phase diagram associated with some colloidal systems. A comparison of the SR potentials with the LJ potential is given in Fig. 3.7. It is

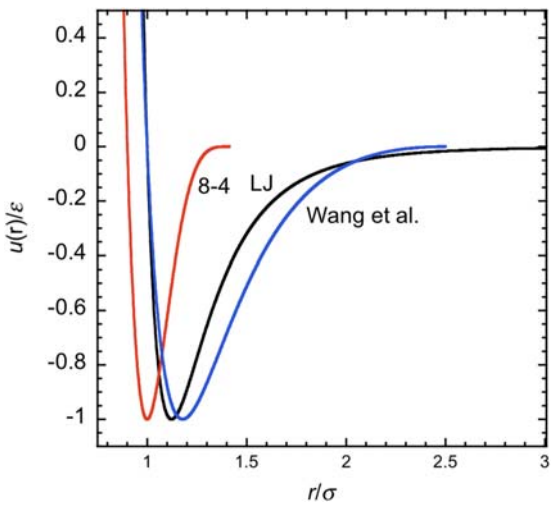


FIGURE 3.7 Comparison of the LJ potential with the short-ranged 8−4 (Eq. (3.48)) and Wang et al. (Eq. (3.49), $n = m = 1$, $r_c = 2.5\sigma$, $\gamma = 0.2915$) potentials. The 8−4 and Wang et al. potentials are exactly zero beyond a distance of $\sqrt{2}\sigma$ and 2.5σ, respectively.

evident that the shape of the potentials is distinct (Chapter 4), which will be reflected in different macroscopic properties.

3.7 Contributions to molecular interactions

Constructing an accurate pair potential for a molecule is considerably more complicated than an atomic potential. The nonspherical nature of the electron clouds of polyatomic molecules introduces many additional problems such as anisotropic overlap, dispersion, and induction forces etc. These problems can be tackled using ab initio methods such as Hartree−Fock or density function theory (Chapter 4). However, a simple alternative is to use a semi-empirical approach to construct a potential, which includes various molecular contributions. The first step in formulating a molecular potential is to identify the various molecular contributions. These contributions are commonly incorporated into an overall model of the molecule governed by a comprehensive force field (Chapter 4).

For a molecule, we can identify contributions to the potential from both inter- and intramolecular interactions. The intermolecular interactions are the interactions between the different atoms of the molecule, which occur independently of molecular bonds. In addition, a molecule will experience the effects of intramolecular interactions as a consequence of molecular bonds. The separation between intra- and intermolecular interactions is based somewhat arbitrarily on the extent of intermolecular distance. It is common to assume that atoms separated by four or more bonds are distant and interact only via nonbonding forces. Therefore, bond interactions are restricted to atoms that are bonded directly together (1,2 interactions), atoms that are joined via bonds to a common atom (1,3 interactions), and atoms with a topology involving three intermediary bonds (1,4 interactions). The potential associated with these interactions are functions of bond lengths, bond angles, proper torsional angles, and improper torsional angles.

3.7.1 Intermolecular ionic and polar potentials

Unlike atoms, molecules are associated commonly with permanent multipole moments or charges, which result in electrostatic interactions. The theoretical basis of electrostatic interactions is well known (Hirschfelder et al., 1954). The application of Coulomb's law of electrostatic interaction between charges q, dipole moments μ, and quadrupole moments Q between molecules a and b yields,

$$u^{(q,q)}(r) = \frac{q_a q_b}{4\pi\varepsilon_o r} \tag{3.51}$$

$$u^{(q,\mu)}(r) = -\frac{q_a \mu_b \cos\theta_b}{4\pi\varepsilon_o r^2} \tag{3.52}$$

$$u^{(q,Q)}(r) = \frac{q_a Q_b \left(3\cos^2\theta_b - 1\right)}{16\pi\varepsilon_o r^3} \tag{3.53}$$

$$u^{(\mu,\mu)}(r) = -\frac{\mu_a \mu_b \left(2\cos\theta_a \cos\theta_b - \sin\theta_a \sin\theta_b \cos\left(\phi_a - \phi_b\right)\right)}{4\pi\varepsilon_o r^3} \tag{3.54}$$

$$u^{(\mu,Q)}(r) = \frac{3\mu_a Q_b}{16\pi\varepsilon_o r^4}\left[\begin{array}{l}\cos\theta_a\left(3\cos^2\theta_b - 1\right) \\ -2\sin\theta_a \sin\theta_b \cos\theta_b \cos\left(\phi_a - \phi_b\right)\end{array}\right] \tag{3.55}$$

$$u^{(Q,Q)}(r) = \frac{3Q_a Q_b}{64\pi\varepsilon_o r^5}\left[\begin{array}{l}1 - 5\cos^2\theta_a - 5\cos^2\theta_b - 15\cos^2\theta_a \cos^2\theta_b \\ +2\left[\sin\theta_a \sin\theta_b \cos\left(\phi_a - \phi_b\right) - 4\cos\theta_a \cos\theta_b\right]^2\end{array}\right] \tag{3.56}$$

where the angles θ_a, θ_b, Φ_a, and Φ_b are defined in Fig. 3.8. The earlier electrostatic terms represent point−dipoles, point−quadrupoles, and the like. Polar molecules can also be modeled by placing partial changes on selected sites.

Eqs. (3.51) to (3.56) each include a contribution in the denominator of $4\pi\varepsilon_o$. This term is required when SI units are used for u(J) r(m), q(C), μ (Cm), and Q(Cm^2). The value $\varepsilon_o = 8.854188128(13) \times 10^{-12}\text{C}^2\text{N}^{-1}\text{m}^{-2}$ is the vacuum electric permittivity (Tiesinga et al., 2021) and the contribution of 4π is required to *rationalize* the units (Kennelly, 1931) for spherical symmetry.

The previous definitions assume that the interactions are completely unobstructed as would be the case in a vacuum. For many real molecules, more realistic values of the electrostatic interactions can be obtained by including the dimensionless value of relative permittivity (ε_r) in the denominator. This contribution can be quite substantial for nonpolar systems, for example, $\varepsilon_r = 78.5$ at 298 K for water (Raabe and Sadus, 2011). By definition, $\varepsilon_r = 1$ is the lower limit for a vacuum.

It should be noted that because all of the multipolar potentials decay more slowly than r^{-3} special techniques such as the Ewald sum (Allen and Tildesley, 2017) are required in the simulation to account for LR interactions (Chapter 4). However, we can also calculate an average contribution of

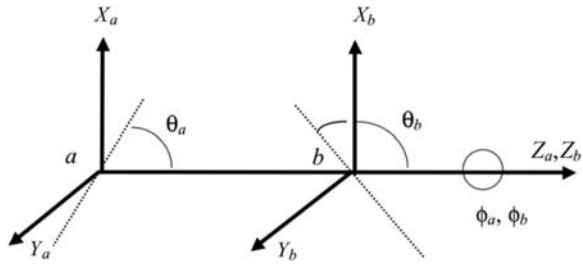


FIGURE 3.8 Definition of angles for multipole interaction of two molecules denoted by a and b. The dotted lines through the centers of a and b indicate the axis of either the dipole or quadrupole of the molecules.

multipolar interactions. The results (Hirschfelder et al., 1954) when using SI units are:

$$u_{ave}^{(q,q)}(r) = \frac{q_a q_b}{4\pi\varepsilon_o r} \tag{3.57}$$

$$u_{ave}^{(q,\mu)}(r) = -\frac{q_a^2 \mu_b^2}{3(4\pi\varepsilon_0)^2 kTr^4} \tag{3.58}$$

$$u_{ave}^{(q,Q)}(r) = -\frac{q_a^2 Q_b^2}{20(4\pi\varepsilon_0)^2 kTr^6} \tag{3.59}$$

$$u_{ave}^{(\mu,\mu)}(r) = -\frac{\mu_a^2 \mu_b^2}{3(4\pi\varepsilon_0)^2 kTr^6} \tag{3.60}$$

$$u_{ave}^{(\mu,Q)}(r) = -\frac{\mu_a^2 Q_b^2}{(4\pi\varepsilon_0)^2 kTr^8} \tag{3.61}$$

$$u_{ave}^{(Q,Q)}(r) = -\frac{7Q_a^2 Q_b^2}{40(4\pi\varepsilon_0)^2 kTr^{10}} \tag{3.62}$$

Historically, many force fields (see discussion below) have been developed using non-SI units for the charge and other multipolar interactions and as such not include the $4\pi\varepsilon_o$ term. Notice that, except for the Coulomb term (Eq. (3.58)), the contribution of the average-angled interactions will always be <0, whereas the sign of values obtained from Eqs. (3.52) to (3.56) is angle-dependent.

Except for charge−charge interaction (Eq. (3.51)) that remains unaltered, the average multipolar interactions decay faster than r^{-3}. Consequently, LR interactions are not an issue when using these potentials. Furthermore, the need to evaluate intermolecular angles and trigonometric functions is eliminated.

In simulations it is often convenient to use reduced units, which in this case means $q^* = q/\sqrt{4\pi\varepsilon_o\sigma\varepsilon}$, $\mu^* = \mu/\sqrt{4\pi\varepsilon_o\sigma^3\varepsilon}$, $Q^* = Q/\sqrt{4\pi\varepsilon_o\sigma^5\varepsilon}$, $r^* = r/\sigma$, and $T^* = kT/\varepsilon$. These definitions allow us to compare Eqs. (3.57) to (3.62) in Fig. 3.9 for the special case when all the contributions are equal to one. It is apparent from Fig. 3.9 that charge−charge interactions dominate at all separations. Charge−dipole interactions are also important but decline much more rapidly. In contrast, the contribution of charge−quadrupole interactions is noticeably less important. Dipole−dipole interactions are the most important noncharge interaction. However, at relatively small separations, dipole−quadrupole interactions takeover. Of course, the relative importance of these contributions will be affected by the charge, dipole, and quadrupole moments. In general, we can expect that charge−charge interactions will dominate with dipole−dipole interactions being the most important noncharge contribution. If both charge−charge and noncharge interactions are present, the effect of the latter will be to subtract from the repulsive nature of the former.

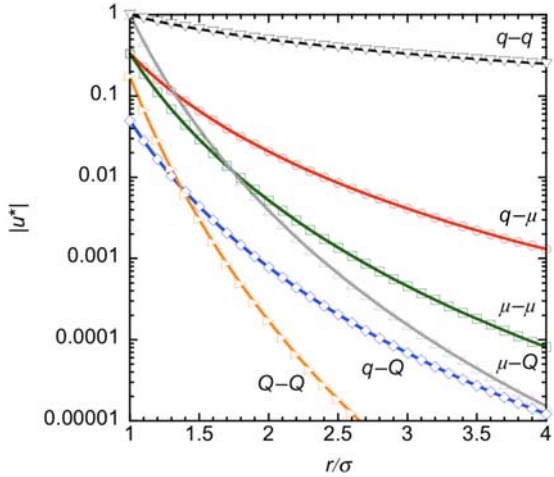


FIGURE 3.9 Comparison of the absolute value of the reduced potential energy (log scale) versus separation for charge−charge (∇), charge−dipole (○), charge−quadrupole (◇) dipole−dipole (□), dipole−quadrupole (△), and quadrupole−quadrupole (◇) interactions obtained from Eqs. (3.57) to (3.62) when $q^* = \mu^* = Q^* = T^* = 1$.

Simulation studies of quadrupole−quadrupole and quadrupole−dipole interactions are also available (Smit and Williams, 1990; Dubey and O'Shea, 1994; Jiang and Pitzer, 1995; O'Shea et al., 1997). The critical temperature of a fluid is directly proportional to the strength of the quadrupole moment (Stapleton et al., 1989). The effects of charge−quadrupole interactions have not been studied by molecular simulation.

Sadus (1996a,b) reported simulations of both vapor−liquid and liquid−liquid equilibria of dipolar fluids using the Keesom potential, which combines the LJ potential with a potential similar to Eq. (3.59). Comparison with calculations using the Stockmayer potential indicates that the average dipole potential was a good approximation for relatively weak dipoles. However, for strong dipoles large differences in the predicted VLE can be expected (Gao et al., 1997) using the Keesom and Stockmayer potentials. MD calculations of the pressure−volume−temperature behavior of polarizable dipolar linear molecules and rigid polarizable molecules are available (Kriebel and Winkelmann, 1996, 1998). The averaged multipolar contributions have proved useful in coarse graining the properties of a number of real fluids (Müller and Gelb, 2003; Mognetti et al., 2008).

3.7.2 Intermolecular hydrogen bond potentials

Hydrogen bond interactions are a special class of nonbonded interactions. It is often found that conventional nonbonded potentials do not account accurately for hydrogen bonding. For example, instead of the traditional 12−6 LJ

potential, the effect of hydrogen bonding can be described more accurately using a 12−10 LJ potential

$$u_{eff}(r) = \frac{A}{r^{12}} - \frac{C}{r^{10}}, \tag{3.63}$$

where A and C are empirical constants. The 12−10 potential provides an improved description of the geometry of hydrogen bonding between the hydrogen atom and the acceptor atom. Other potentials attempt to account for deviations from the hydrogen bond geometry. Goodford (1985) proposed

$$u_{eff}(r) = \left(\frac{C}{r^6} - \frac{D}{r^4}\right)\cos^4\theta, \tag{3.64}$$

where C and D are empirical parameters and θ is the angle subtended at the hydrogen. Vedani (1988) proposed the following potential

$$u_{eff}(r) = \left(\frac{A}{r_{H-Acc}^{12}} - \frac{C}{r_{H-Acc}^{10}}\right)\cos^2\theta_{Don-H-Acc}\cos^4\omega_{H-Acc-LP}, \tag{3.65}$$

where $r_{\text{H-Acc}}$ is the distance between the hydrogen atom (H) and the acceptor atom (Acc); $\theta_{don\text{-}H\text{-}acc}$ is the inner angle formed by the donor molecule (Don), hydrogen atom and acceptor molecule configuration; and $\omega_{H\text{-}Acc\text{-}LP}$ is the angle formed between the hydrogen atom, the acceptor molecule (Acc), and the lone pair (LP) of electrons on the acceptor molecule.

3.7.3 Intramolecular bond potentials

The bond potential represents the effect of bond stretching. The Morse potential (Rappé and Casewit, 1997) is an example of an accurate bond potential,

$$u_b(b) = D_e\left\{1 - \exp[-\alpha(l - l_0)]\right\}^2 \tag{3.66}$$

where D_{e} is the bond dissociation energy, α is a constant that depends on both the masses of the atoms and the shape of the potential well, and l_0 is the reference or natural bond separation. It should be noted that the reference bond length is not necessarily the same as the equilibrium bond separation. The reference bond length is the separation when all other force terms experienced by the molecule are zero. In contrast, the equilibrium bond separation reflects the contribution of other forces on the molecule.

The Morse potential is a physically realistic representation of a bond, which allows for the breaking of bonds at large interatomic separation. However, evaluating the exponential term is expensive computationally. It also requires the evaluation of three potential parameters (D_e, $l_{0,}$ and α). If bond dissociation is excluded, a simple alternative can be obtained by expanding Eq. (3.67) about the equilibrium separation

$$u_b(l) = \frac{K_b(l - l_0)^2}{2}\left[1 - \alpha(l - l_0) + \left(\frac{7}{12}\right)\alpha^2(l - l_0)^2 + \cdots\right], \tag{3.67}$$

where $K_b = 2D_e\alpha^2$. Eq. (3.67) eliminates the evaluation of the exponential term but three parameters must be evaluated per bond stretch. If only the first term of Eq. (3.67) is used, the bond potential takes the form of Hooke's law

$$u_b(l) = \frac{K_b(l-l_0)^2}{2}, \tag{3.68}$$

where K_b is the force constant. The choice between these different levels of approximation depends on the accuracy of the desired calculation and computational expediency. Eq. (3.68) is a good approximation for bonding in ground-state molecules but it is considerably less accurate than the other alternatives at nonequilibrium separations.

3.7.4 Intramolecular angle-bending potentials

A Hooke's law approach is also used commonly for both the bond-angle potential and out-of-plane bending. The bond-angle potential can be evaluated (Dinur and Hagler, 1991) from either

$$u_b(\theta) = \frac{K_\theta(\theta-\theta_0)^2}{2}\left[1 - \alpha(\theta-\theta_0) + \left(\frac{7}{12}\right)\alpha^2(\theta-\theta_0)^2 + \cdots\right] \tag{3.69}$$

or

$$u_\theta(\theta) = \frac{K_\theta(\theta-\theta_0)^2}{2}, \tag{3.70}$$

where K_θ is the bending force constant and θ_0 is the reference or normal bond angle. Eq. (3.69) represents simple, harmonic bending, which is suitable for open and unstrained chains. In contrast, Eq. (3.69) is a more accurate alternative if anharmonicity is present.

More sophisticated alternatives for angle bending have been developed. For example, Rappé et al. (1992) proposed the following angle-bending potential.

$$u_\theta(\theta) = K_\theta\left(\frac{2\cos^2\theta_0 + 1}{4\sin^2\theta_0} - \frac{4\cos\theta_0\cos\theta}{4\sin^2\theta_0} + \frac{\cos 2\theta}{4\sin^2\theta_0}\right) \tag{3.71}$$

Unlike potentials that assume an idealized geometry, Eq. (3.71) is in principle valid for all nonlinear geometries.

3.7.5 Intramolecular out-of-plane bending potentials

Out-of-plane bending occurs typically in planar molecules with trigonal centers. The energy associated with out-of-plane bending arises from the displacement of the trigonal atom either above or below the molecular plane.

Out-of-plane bending is usually harmonic, and the potential is given (Dinur and Hagler, 1991) simply by

$$u_{\chi}(\chi) = \frac{K_{\chi}\left(\chi - \chi_0\right)^2}{2}, \tag{3.72}$$

where K_{χ} is the barrier for out-of-plane bending and χ_0 is the reference angle for out-of-plane bending.

3.7.6 Intramolecular torsional potentials

The torsional potential represents the effect of rotation about a bond and as such it is a function of the dihedral angles. It can be calculated from a Fourier series expansion (Dinur and Hagler, 1991)

$$u_{\tau}(\tau) = \frac{1}{2}\sum_{j} V_j\left[1 + (-1)^{j+1}\cos(j\tau)\right], \tag{3.73}$$

where V_j is the rotational barrier height, τ is the dihedral angle, and j is the periodicity.

3.7.7 Intramolecular cross-terms

In principle, every motion of a molecule is coupled to other motions. Consequently, an accurate description of the physics of motion sometimes requires cross-terms that represent the coupling of different types of motion. Fortunately, the coupling effect is only really significant for neighboring motions. The effect of coupling stretch−stretch interactions between bonds 1 and 2 can be represented (Dinur and Hagler, 1991) by

$$u(l_1, l_2) = \frac{K_{l1,2}\left[\left(l_1 - l_{1,0}\right)\left(l_2 - l_{2,0}\right)\right]}{2} \tag{3.74}$$

Experimental observations indicate that reducing the bond angle between bonds will induce bond stretching to minimize the strain. The effect of bond-stretching and bond-angle motion can be obtained from the following potential:

$$u(l_1, l_2, \theta) = \frac{K_{l1,2}\left[\left(l_1 - l_{1,0}\right) + \left(l_2 - l_{2,0}\right)\right](\theta - \theta_0)}{2}, \tag{3.75}$$

where the two bonds (denoted 1 and 2) are joined to a common atom and θ is the angle between these two bonds. An alternative to Eq. (3.76) is the Urey−Bradley potential, which determines the potential between the non-bonded atoms (denoted 1 and 3) as a result of bond-stretching and bond-angle motion. The Urey−Bradley potential is

$$u(r_{1-3}) = \frac{K_{r_{1-3}}\left(r_{1-3} - r_{1-3,0}\right)^2}{2}, \tag{3.76}$$

where the distances refer to the separation between the nonbonded atoms. The torsional motion can be coupled to both stretching and bond-bending motion. The coupling of torsion and stretching terms can be obtained from

$$u(l,\tau) = \frac{K_{l,\tau}[(l-l_0)(1+\cos 3\tau)]}{2}, \tag{3.77}$$

whereas the interaction between torsional motion and bond bending can be represented (Dinur and Hagler, 1991) by:

$$u(\theta,\tau) = \frac{K_{\theta,\tau}(\theta-\theta_0)(1-\cos\tau)}{2}. \tag{3.78}$$

In the case of molecules forming two angles at a common center, the effect of two bend−bend interactions must be considered.

$$u(\theta_1,\theta_2) = \frac{K_{\theta 1,2}\left(\theta_1-\theta_{1,0}\right)\left(\theta_2-\theta_{2,0}\right)}{2} \tag{3.79}$$

3.8 Simple atom-based pairwise potentials for molecules

The relative importance of the intermolecular and intramolecular terms depends on the nature of the molecule. It should be emphasized that the nonbonded interactions are of primary importance. Often, a good representation of the energy of the molecule can be obtained by ignoring bond and angle terms and only considering nonbonded and electrostatic interactions. Some examples are given below.

3.8.1 Stockmayer potential

The Stockmayer potential is obtained if Eqs. (3.25) and (3.54) are used for the contributions from repulsion, dispersion, and dipolar interactions, respectively.

$$\begin{aligned} u_{eff}(r,\theta,\phi) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \\ - \frac{\mu_a\mu_b\left(2\cos\theta_a\cos\theta_b - \sin\theta_a\sin\theta_b\cos\left(\phi_a-\phi_b\right)\right)}{4\pi\varepsilon_0 r^3} \end{aligned} \tag{3.80}$$

Several simulations have been reported using the Stockmayer potential to investigate the effect of dipole−dipole interactions on the VLE of pure fluids (Smit et al., 1989; van Leeuwen and Smit, 1993; van Leeuwen et al., 1993; Bartke and Hentschke, 2007; Jia and Hentschke, 2011). The work of van Leeuwen and Smit (1993) indicates that dispersion interactions are necessary to sustain vapor−liquid phase coexistence in a dipolar fluid. The Stockmayer potential has also been used to study polar + nonpolar binary mixtures (de Leeuw et al., 1990; Mooij et al., 1992). The excess free energy of mixing decreases with increasing polarizability and at high dipole strength, the

dipolar and nonpolar components separate into different phases. The VLE of binary mixtures interacting via the Stockmayer potential has been investigated (Gao et al., 1997). The effect of dipolar interaction is to increase the region of two-phase coexistence. A theoretical explanation of the dependence of the critical point on the dipole strength of the Stockmayer fluid has been determined (Hentschke et al., 2007). The usefulness of the Stockmayer potential to molecules is limited because molecules are not spherical.

In the force fields given below, the $4\pi\varepsilon_0$ term does not appear in the electrostatic contributions because the parameterization has been most commonly reported using non-SI units. Therefore, some care is required when converting to SI units.

3.8.2 Spurling–Mason potential

Spurling and Mason (1967) studied the virial coefficients and viscosity data of quadrupolar molecules using a potential, which explicitly incorporated the effect of quadrupole moments and other molecular factors. Their potential is

$$\begin{aligned} u_{eff}(r) = {} & 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \\ & + \frac{3Q^2}{r^5}\left[\begin{array}{l} 1 - 5\left(\cos^2\theta_a + \cos^2\theta_b\right) - 15\cos^2\theta_a\cos^2\theta_b \\ + 2\sin\theta_a\sin\theta_b\cos\left(\phi_a - \phi_b\right) \end{array}\right] \\ & - \frac{9Q^2\alpha_m}{8r^8}\left[\sin^4\theta_a + \sin^4\theta_b + 4\left(\cos^4\theta_a + \cos^4\theta_b\right)\right] \\ & + 4\varepsilon\left(\frac{\sigma}{r}\right)^6\left\{\kappa - \frac{3}{2}\left[\begin{array}{l} \left(\kappa - \kappa^2\right)\left(\cos^2\theta_a + \cos^2\theta_b\right) \\ - \kappa^2\left(\sin\theta_a\sin\theta_b\cos\left(\phi_a - \phi_b\right) - 2\cos\theta_a\cos\theta_b\right)^2 \end{array}\right]\right\} \\ & + 4D\varepsilon\left(\frac{\sigma}{r}\right)^{12}\left[3\left(\cos^2\theta_a + \cos^2\theta_b\right) - 2\right] \end{aligned} \tag{3.81}$$

where α_m, κ, and D are the mean polarizability, the anisotropy of the polarizability, and a shape parameter, respectively. The first term in Eq. (3.82) is the LJ potential. The second and third terms represent the effect of quadrupole, and quadrupole-induced dipole interactions, respectively. The potential also attempts to account for molecular anisotropy (fourth term) and shape (fifth term). When Eq. (3.82) is applied to the viscosity data of small polyatomic molecules, reasonable estimates of the quadrupole moments can be obtained (Spurling and Mason, 1967).

3.8.3 Restricted primitive model

Molten salts (Hansen and McDonald, 1975) and other ionic (Smith and Haymet, 1993) fluids have been studied using charge–charge potentials.

Potentials of the form of Eq. (3.52) have been used in simulations of aqueous electrolyte solutions. The most commonly used potential for ionic fluids is the restricted primitive model (RPM), which is composed of a HS potential in conjunction with Eq. (3.54).

$$u_{eff}(r) = u_{HS}(r) + \frac{q_a q_b}{r} \tag{3.82}$$

Orkoulas and Panagiotopoulos (1994) have reported MC calculations of the free energy and phase equilibria of an ionic fluid using the RPM potential. The dielectric constant of RPM electrolytes on the vapor branch of the coexistence line has also been reported (Caillol, 1995).

3.8.4 Fumi–Tosi potential

A more realistic intermolecular potential for ions is the Fumi and Tosi (1964) potential,

$$u_{eff}(r) = A\exp(-Br) - \frac{E}{r^6} - \frac{D}{r^8} + \frac{q_a q_b}{r} \tag{3.83}$$

where A, B, D, and E are parameters obtained from fitting experimental data. Strauch and Cummings (1993) have used the Huggins–Mayer form of this potential to calculate successfully vapor–liquid equilibria in water + NaCl and water + methanol + NaCl mixtures. Guissani and Guillot (1994) used Eq. (3.50) to determine the critical point of NaCl.

3.9 Extension to molecules

3.9.1 Nonbonded interactions in molecules

The proceeding sections have focused exclusively on atomic systems but the potentials can be directly applied within a molecular framework. Irrespective of the complexity of a molecule, the dispersion and repulsion interactions of the atoms that constitute the molecule are of primary importance. This nonbonded contribution can be calculated by using any of the potentials discussed earlier in the chapter. In practice, the most-widely used approach for calculating nonbonded interactions is to model molecules as a series of interaction sites. The interaction sites usually coincide with the constituent atoms of the molecule. Consequently, a simple atomic potential such as the LJ potential is used to evaluate the various atom–atom interactions. For example (Cheung and Powles, 1975; Monson et al., 1983), the interactions between molecules (i and j) each composed of m-atoms can be calculated from

$$u_{ij}\left(r_{\alpha\beta}\right) = \sum_{\alpha=1}^{m}\sum_{\beta=1}^{m} 4\varepsilon\left[\left(\frac{\sigma}{r_{\alpha\beta}}\right)^{12} - \left(\frac{\sigma}{r_{\alpha\beta}}\right)^{6}\right], \tag{3.84}$$

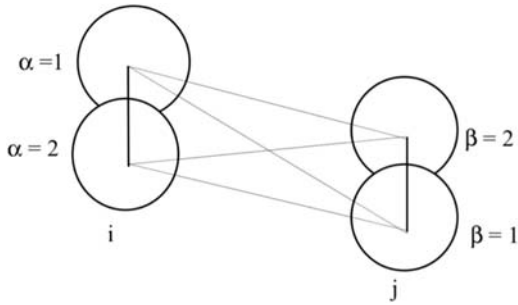


FIGURE 3.10 An atom−atom model of a diatomic molecule.

where $r_{\alpha\beta}$ is the separation of atoms α and β on two different molecules. In Eq. (3.84), the LJ is used as an example only and any other pair atomic potential can be substituted. The atom−atom model of a diatomic molecule is illustrated in Fig. 3.10.

3.9.2 Adding nonbonded interactions: the WCA + finitely extensible nonlinear elastic potential

In Section 3.9, quite complicated molecular potentials will be examined, but a simple example of a molecular system is a chain of atoms that require us to consider both nonbonded interactions and molecular constraints. Possibly the simplest model for chain or polymer molecules is the freely jointed tangent HS chain. This approach neglects variation of bond length and torsional angles but it incorporates some essential features such as molecular connectivity and intra- and intermolecular site−site excluded volume interactions. Molecular simulation of the properties of chains containing up to 200 sites have been reported (Gao and Weiner, 1989). Data for mixtures of chains are limited to relatively small chains (Chang and Sandler, 1995).

A somewhat more realistic model for molecular chains can be obtained by combining the WCA potential for the excluded volume of the HS beads (nonboded interactions) with the finitely extensible nonlinear elastic (FENE) potential (Warner, 1972; Grest and Kremer, 1986) that is responsible for maintaining the molecular topology of the beads via both a maximum intramolecular separation (r_0) and a "spring constant" (K).

$$u_{\mathrm{FENE}}(r) = \begin{cases} \dfrac{Kr_0^2 \ln\left[1 - \left(\dfrac{r}{r_0}\right)^2\right]}{2} & r < r_0 \\ \infty & r \geq r_0 \end{cases} \tag{3.85}$$

The WCA + FENE potential has proven very useful (Bosko et al., 2004) in determining the viscoelastic properties of dendrimers of different generations (Newkome et al., 2001).

3.10 Pairwise force fields from molecular mechanics

In view of the preceding considerations, the energy of a molecular system resulting from pairwise interactions can be evaluated from

$$U(r,l,\theta,\tau,\chi) = \sum_{nonbonds} u_{nb}(r) + \sum_{bonds} u_b(l) + \sum_{bending} u_\theta(\theta) + \sum_{dihedral} u_\tau(\tau) + \sum_{out-of-plane} u_\chi(\chi) + \sum u_{el} + \cdots, \tag{3.86}$$

where u_{nb} is the nonbonded potential, u_b is the bond potential, u_θ is the bond-angle potential, u_τ is the torsional potential, u_χ is the out-of-plane bending (improper torsional) potential, and u_{el} is the Coulombic potential. Eq. (3.86) only includes "diagonal terms," but it can be extended to include contributions from "cross terms" such as bond−angle or angle−angle interactions. This approach is referred to generally as molecular mechanics (Dinur and Hagler, 1991; DeKock et al., 1993). For example, to model an alkane, we can compose an overall potential, which includes a potential for nonbonded interactions; a potential for bond-bending, and a potential for angle torsion (Siepmann et al., 1993).

In effect, Eq. (3.86) attempts to capture the essential physics of the molecule. The result is a composite potential incorporating the contributions of the various possible intra- and intermolecular interactions. This combination of various contributions to describe molecular interactions is commonly referred to as a "force field." The term of force field is obviously a misnomer because Eq. (3.86) evaluates the energy not the force; however the nomenclature is in widespread use in the literature.

The accuracy of the force field depends critically on obtaining accurate values for the various potential parameters. The bond and angle parameters can be obtained from X-ray, infrared, nuclear magnetic resonance, and microwave data. The nonbonded parameters can be obtained from experimental data for second virial coefficients and molecular beam experiments. The parameters can be calculated using ab initio and semiempirical methods (Maple et al., 1994; Dinur and Hagler, 1991). A comprehensive review of molecular mechanics has been reported (Rappé and Casewit, 1997).

The primary motivation for developing force field is to realistically capture the behavior of large molecules, most notably biomolecules and proteins (van Gunsteren et al., 2006). The literature on force fields is both large and continuously expanding (Cisneros et al., 2013; Lopes et al., 2015; Rinker,

2018), and documenting all possibilities is an impossible undertaking. The difficulty in providing a comprehensive description is further compounded that many "new" force fields are arguably incremental improvements on existing ones via the use of improved parameterization rather than fundamentally novel insights into intermolecular behavior.

The complexity of force fields has increased to the point, that some like the Groningen molecular simulation (GROMOS) approach (Scott et al., 1999) have become a general software package that offer multiple possible customizations rather than a conventional standalone force field that can be realistically implemented independently by the researcher. Many researchers have become expert users of force field software packages that can be tailored to a diverse range of applications and molecular systems.

The following discussion is focused on a few key force fields that have provided a framework for many other alternatives. The optimized potential for liquid simulation (OPLS) model (Jorgensen and Tirado-Rives, 1988) is one of the simplest force fields. Other examples of force fields are the empirical conformational energy program for peptides (ECEPP/3) (Némethy et al., 1992); chemistry at Harvard macromolecular mechanics (CHARMM) (Brooks et al., 1983; Smith and Karplus, 1992); assisted model building with energy refinement (AMBER) (Weiner et al., 1984); and molecular mechanics (MM3) (Allinger et al., 1989; Lii and Allinger, 1989a,b) potentials. These force fields typically contain terms involving bond stretching, bond angle, torsional motions, and nonbonded and electrostatic interactions.

3.10.1 Optimized force fields

There are several examples of elaborate force fields that have been optimized for various molecular properties. Historically this has involved the laborious fitting of experimental data, although more recent approaches such as supervised machine learning (Behler, 2016) can be expected to an increasingly important role (Mueller et al., 2020).

3.10.1.1 OPLS/OPLS-AA force fields

The OPLS force field is of particular interest because it is specifically optimized for liquid properties (Jorgensen and Tirado-Rives, 1988). The OPLS force field has the following form:

$$U = \sum_{j>i} \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + \frac{q_i q_j}{r_{ij}} \right). \tag{3.87}$$

The interaction sites of the OPLS force field can be either nuclei or $(CH)_n$ groups. Later, Kaminski et al. (1994) added a torsional term to create an all-atom (AA) force field (OPLS-AA), which can be parameterized (Damm et al., 1997) for carbohydrates. Despite its obvious simplicity, the

OPLS-AA force field often yields results of comparable accuracy (Martin, 2006) to more sophisticated alternative models (see subsequent discussion) that are more widely used.

3.10.1.2 ECEPP models

The ECEPP/3 force field (Némethy et al., 1992) is an updated version of the ECEPP force field, which was originally developed by Momany et al. (1975) and subsequently improved (ECEPP/2) by Némethy et al. (1983). ECEPP/3 retains the same functional form as its predecessors but it uses an improved set of parameters based on accurate experimental data. The ECEPP/3 force field has the following form:

$$U = \sum_{j>i} \varepsilon_{ij} \left[\left(\frac{r^*_{ij}}{r_{ij}} \right)^{12} - \left(\frac{r^*_{ij}}{r_{ij}} \right)^{6} \right] + \sum_{j>i} \varepsilon_{ij} \left[5 \left(\frac{r^*_{ij}}{r_{ij}} \right)^{12} - 6 \left(\frac{r^*_{ij}}{r_{ij}} \right)^{10} \right] + \sum_{n} \frac{K_n}{2} (1 \pm \cos n\tau_n) + \sum_{j>i} \frac{q_i q_j}{D r_{ij}}, \tag{3.88}$$

where D is the dielectric constant and r^* is the separation corresponding to the minimum of nonboded pair interaction. Like the OPLS force field, Eq. (3.88) includes the contribution of LJ interaction between nonbonded atoms (first term) and electrostatic interactions (fourth term). However, the ECEPP potential explicitly includes a contribution from hydrogen bonding (the 12−10 term) and torsional motion (third term).

3.10.1.3 AMBER force field

The energy obtained from the AMBER force field is calculated as:

$$U = \sum_{bonds} \tfrac{K_l}{2} (l - l_0)^2 + \sum_{angles} \frac{K_\theta}{2} (\theta - \theta_0)^2 + \sum_{dihedrals} \sum_{n} \frac{V_n}{2} [1 + \cos(n\tau - \gamma)] + \sum_{i<j} 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \frac{1}{vdW_{scale}} \sum_{i<j}^{1,4terms} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \sum_{H-bonds} \left[\frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}} \right] + \sum_{i<j} \frac{q_i q_j}{\varepsilon r_{ij}} + \frac{1}{EE_{scale}} \sum_{i<j}^{1,4terms} \frac{q_i q_j}{D r_{ij}}. \tag{3.89}$$

In Eq. (3.89), the first three terms represent the bonded interaction of the molecules. These terms contain force constants for the bonds (K_l) and angles (K_θ); l_0 and θ_0 are the experimentally determined normal bond distances and angles, respectively. A truncated Fourier series represent the torsional potential with contributions from the torsional barrier (V_n), periodicity (n), the calculated dihedral angle (τ), and phase (γ). The remaining terms are the contribution of nonbonded interactions to repulsion and dispersion (LJ potential), hydrogen bonding (the 10−12 term), and electrostatic interactions between point charges (q_i). The ε parameter in the last two terms is the dielectric constant, and C and D are empirically determined parameters.

Several revisions of the AMBER force field have been made, particularly to improve the representation of dihedral angles. This involved (Hornak et al., 2006) making use of an improved quantum-level understanding of interactions. An improved backbone has been reported (Best and Hummer, 2009). Its ability to deal with an aqueous protein environment has been improved (Kirschner and Woods, 2001; Joung and Cheatham, 2008).

3.10.1.4 CHARMM *force field*

The CHARMM force field includes a contribution from out-of-plane bending

$$\begin{aligned} E = &\sum_{bonds} \frac{K_l}{2}(l-l_0)^2 + \sum_{angles} \frac{K_\theta}{2}(\theta-\theta_0)^2 \\ &+ \sum_{angles} \frac{K_\chi}{2}\left(\chi-\chi_0\right)^2 + \sum_{dihedrals}\sum_{n} \frac{V_n}{2}[1+\cos(n\tau-\gamma)] \\ &+ \sum_{i<j} 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] + \sum_{i<j}\frac{q_i q_j}{\varepsilon r_{ij}}, \end{aligned} \tag{3.90}$$

where χ is the out-of-plane angle and k_χ is the force constant for out-of-plane bending. Both the CHARMM and AMBER force fields use the same representation of nonbonded and electrostatic forces. Unlike the AMBER force field, the CHARMM force filed does not include a hydrogen-bonding term. The CHARMM force field has been developed for proteins (MacKerell et al., 1998), nucleic acids (Foloppe and MacKerell, 2000), lipids (Klauda et al., 2010), and carbohydrates (Guvench et al., 2011). Its accuracy for these systems is continuously updated (Best et al., 2012) with additional terms and new parameters. The variants are often distinguished by the addition of numerical suffixes, for example, CHARMM27.

3.10.1.5 *MM3 force field*

The MM3 force field (Allinger et al., 1989) incorporates a more elaborate description of the various bond and nonbonded potentials. MM3 eliminates the deficiencies of the earlier MM2 (Allinger, 1977) by using improved theoretical

models for bond and nonbond interactions and specifically addressing vibrational and spectroscopic data. The MM3 force field is represented by Eq. (3.91).

$$
\begin{aligned}
U = & \sum_{bonds} \frac{K_l}{2}(l-l_0)^2\left[1-2.55(l-l_0)+\left(\frac{7}{12}\right)2.55(l-l_0)^2\right] \\
& + \sum_{angles} \frac{K_\theta}{2}(\theta-\theta_0)^2\left[\begin{array}{l}1-0.014(\theta-\theta_0)+5.6\times10^{-5}(\theta-\theta_0)^2 \\ -7\times10^{-7}(\theta-\theta_0)^3+9\times10^{-10}(\theta-\theta_0)^3\end{array}\right] \\
& + \sum_{angles} \frac{K_\chi}{2}\left(\chi-\chi_{eq}\right)^2\left[\begin{array}{l}1-0.014\left(\chi-\chi_0\right)+5.6\times10^{-5}\left(\chi-\chi_0\right)^2 \\ -7\times10^{-7}\left(\chi-\chi_0\right)^3+9\times10^{-10}\left(\chi-\chi_0\right)^3\end{array}\right] \\
& + \sum_{dihedrals} \frac{1}{2}[V_{\tau1}(1+\cos\tau)+V_{\tau2}(1-\cos2\tau)+V_{\tau3}(1+\cos3\tau)] \\
& + \sum_{bonds/angles} \frac{K_{l\theta}}{2}\left[\left(l_1-l_{1,0}\right)+\left(l_2-l_{2,0}\right)\right](\theta-\theta_0) \\
& + \sum_{bonds/torsion} \frac{K_{l\chi}}{2}(l-l_0)(1+\cos3\chi) \\
& + \sum_{angles/angles} \frac{K_{\theta1\theta2}}{2}\left(\theta_1-\theta_{1,0}\right)\left(\theta_2-\theta_{2,0}\right) \\
& + \sum_{i<j} \varepsilon_{ij}\left[-2.25\left(\frac{\sigma_{ij}}{r_{ij}}\right)^6+1.84\times10^5\exp\left(-\frac{12\sigma_{ij}}{r_{ij}}\right)\right] \\
& + \sum_{i<j} \frac{\mu_i\mu_j}{\varepsilon r_{ij}^3}\left(2\cos\theta_i\cos\theta_j-\sin\theta_i\sin\theta_j\cos\left(\phi_i-\phi_j\right)\right)
\end{aligned}
\tag{3.91}
$$

In the MM3 force field, the accuracy of the bond-stretching term (the first term) is improved by incorporating more terms of the Taylor series expansion. The bond-stretching term is a better approximation of the Morse potential than the simple Hooke's law potential used in both the AMBER and CHARMM force fields. Similarly, the accuracy of the bond-angle bending (second term) and out-of-plane bending (third term) is improved by using a higher-order expansion. Three different periodicities are used to represent the effects of torsional motion (fourth term).

A feature of the MM3 force filed is the inclusion of cross-terms arising from bond stretching/bending (fifth term), bond stretching/torsional motion (sixth term), and bending/bending (seventh term) motions. The nonbonded contributions to dispersion and repulsion are modeled by an empirically modified Buckingham potential (eighth term). In contrast to either the AMBER or CHARMM force fields, the MM3 force field does not use Coulomb's law to determine the electrostatic interactions (ninth term). Instead, each bond in the molecule is assigned a bond dipole moment (μ) and the total electrostatic interaction is the sum of all possible dipole−dipole interactions. The dihedral angle made by the bond dipoles is represented by χ and θ, which are the angles made by each of the bond dipoles with the r vector.

In many cases, the MM3 force field can be used to calculate heats of formation, conformational energies, and rotational barriers to an accuracy that is close to experimental uncertainty (Allinger et al., 1989). Vibrational frequencies and thermodynamic properties are also calculated accurately using the MM3 force field (Lii and Allinger, 1989a,b). Improvements on the MM3 potential include parameters obtained from ab initio data (Shannon et al., 2019).

A common feature of many realistic force fields is that of atom type. The atom type is defined by factors such as atomic number, hybridization, and details of the local environment. For example, to apply a force field to a molecule containing carbon atoms, a distinction must be made between sp^3, sp^2, and sp hybridized carbons, which generate tetrahedral, trigonal, and linear geometries, respectively. This distinction is important because the geometry of the atom type affects the values of the potential parameters. The local environment also affects the force field parameters. The MM3 force field distinguishes between carbonyl, cyclopropane, cyclopropene, radical, and carbonium ion carbon atoms. The AMBER approach assigns a different atom type to carbon atoms at the junction between a six- and five-member ring compared with carbon atoms in an isolated five-member ring.

3.10.1.6 UFF

Complicated rules for assigning atom types means that it is often difficult to obtain parameters for different categories of molecules. Rappé et al. (1992) attempted to address this limitation by proposing a universal force field (UFF) using parameters estimated from general rules based only on the element, its hybridization, and its connectivity. The functional form of the UFF is

$$\begin{aligned} E = & \sum_{bonds} D_0[\exp(-\alpha(l-l_0)]^2 \\ & + \sum_{angles} K_\theta \left(\frac{2\cos^2\theta_0 + 1}{4\sin^2\theta_0} - \frac{4\cos\theta_0 \cos\theta_0}{4\sin^2\theta_0} + \frac{\cos 2\theta}{4\sin^2\theta_0} \right) \\ & + \sum_{\chi - angles} K_\chi (C_0 + C_1 \cos\chi + C_2 \cos 2\chi) \\ & + \sum_{dihedrals} \sum_{n} \frac{V_n}{2} [1 - \cos n\tau_0 \cos n\tau] \\ & + \sum_{j>i} \varepsilon_{ij} \left[\left(\frac{r_{ij}^*}{r} \right)^{12} - 2 \left(\frac{r_{ij}^*}{r} \right)^{6} \right] \\ & + \sum_{j>i} 322.0637 \frac{q_i q_j}{\varepsilon r_{ij}}, \end{aligned} \tag{3.92}$$

where the different terms represent bond stretching, angle bending, out-of-plane bending, torsional motion, nonbonded interaction, and electrostatic interactions, respectively. The angle-bending term is a general term valid for all linear and nonlinear geometries. The torsional term includes an improved description of torsional periodicities and minima. Parameters for the UFF

have been evaluated for elements (Rappé et al., 1992), organic molecules (Casewit et al., 1992a), and main group compounds (Casewit et al., 1992b). In common with other force fields, it requires knowledge of the charge distribution of the interaction sites of the molecule (Rappé and Goddard, 1991).

3.10.1.7 CFF93

Instead of using experimental data directly, an intermolecular potential can be obtained by applying the principles of quantum mechanics. The advantage of using quantum mechanics is that there are an unlimited number of observable quantities, which can be fitted to the potential. The parameters can also be derived for any functional group that can be calculated using quantum mechanics. For example, a force field has been developed (Maple et al., 1994) that reproduces the Hartree−Fock energy surface for alkane molecules. The relative energies and the first and second Cartesian energy derivatives obtained from quantum mechanics for alkanes were fitted to an intermolecular potential. Hwang et al. (1994) extended the approach to include alkyl functional groups. The CFF93 is given by Eq. (3.93).

$$
\begin{aligned}
U = & \sum_{b}\left[K_{2b}(b-b_0)^2 + K_{3b}(b-b_0)^3 + K_{4b}(b-b_0)^4\right] \\
& + \sum_{\theta}\left[K_{2\theta}(\theta-\theta_0)^2 + K_{3\theta}(\theta-\theta_0)^3 + K_{4\theta}(\theta-\theta_0)^4\right] \\
& + \sum_{\tau}\left[K_{1\tau}(1-\cos\tau) + K_{2\tau}(1-\cos 2\tau) + K_{3\tau}(1-\cos 3\tau)\right] \\
& + \sum_{\chi} K_{\chi}\chi^2 \\
& + \sum_{b}\sum_{b'} K_{bb'}(b-b_0)(b'-b_0{}') \\
& + \sum_{\theta}\sum_{\theta'} K_{\theta\theta'}(\theta-\theta_0)(\theta'-\theta_0{}') \\
& + \sum_{b}\sum_{\theta} K_{b\theta}(b-b_0)(\theta-\theta_0) \\
& + \sum_{\tau}\sum_{b}(b-b_0)[K_{1\tau b}\cos\tau + K_{2\tau b}\cos 2\tau + K_{3\tau b}\cos 3\tau] \\
& + \sum_{\tau}\sum_{b'}(b'-b_0')[K_{1\tau b'}\cos\tau + K_{2\tau b'}\cos 2\tau + K_{3\tau b'}\cos 3\tau] \\
& + \sum_{\tau}\sum_{\theta}(\theta-\theta_0)[K_{1\tau\theta}\cos\tau + K_{2\tau\theta}\cos 2\tau + K_{3\tau\theta}\cos 3\tau] \\
& + \sum_{\tau}\sum_{\theta}\sum_{\theta'} K_{\tau\theta\theta'}(\theta-\theta_0)(\theta'-\theta_0')\cos\tau \\
& + \sum_{i>j}\varepsilon\left[2\left(\frac{r^*}{r_{ij}}\right)^9 - 3\left(\frac{r^*}{r_{ij}}\right)^6\right] \\
& + \sum_{i>j}\frac{q_i q_j}{r_{ij}}
\end{aligned}
\tag{3.93}
$$

The first three terms in Eq. (3.93) represent the contributions from bond stretching, bond bending, torsional motion, and out-of-plane bending, respectively. Similar terms have been incorporated in the other force fields discussed previously. However, unlike other models, the CFF93 emphasizes the importance of cross interactions. The fourth, fifth, and sixth terms represent the coupling of different bond stretches, different bending motions, and stretching and bending, respectively. The coupling of different bond stretching with torsional motion is represented by the seventh and eighth terms. The ninth term couples bond bending with torsional motion. The tenth term represents the effect of coupling different bond bending and torsional motions. The remaining terms are the LJ and Coulombic potentials of the nonbonded atoms, respectively. All twelve terms are necessary for the quantitative representation of the quantum mechanical data and many of the terms make an equal contribution to the energy.

3.10.2 Comparison of force fields and future improvements

The already discussed force fields introduce an increasing degree of sophistication to the description of the physics of molecules. How do the results obtained for the various force fields compare? The answer to this question is complicated by the fact that the parameters for the various potentials cannot be extended to all molecules. The large number of possible permutations within a force field software package is also a daunting impediment for comparative purposes. This makes a definitive comparison between forces fields an impossible task. Inevitably, the usefulness of a particular force field will depend on the properties for which it is applied to. Comparisons of several different types of force fields are available (Roterman et al., 1989 & b; Hwang et al., 1994; Landis et al., 1995; MacKerell, 2004; Hornak et al., 2006; Becker et al., 2013; Vanommeslaeghe and MacKerell, 2015; Sun and Deng, 2017; Rinker, 2018; Bedrov et al., 2019). There is no force field that is consistently superior for all applications.

Landis et al. (1995) reported a comparison of the parameters required for CHARMM and MM3 force fields. Their analysis indicated that there was a considerable difference in the parameter values required by the different potentials. Rappé et al. (1992) compared the results obtained for UFF and the MM3 force field for some organic molecules. The comparison indicated that MM3 force field is less prone to error than the UFF. On the other hand, the UFF can be applied to a large variety of different combinations of atoms, whereas the large number of parameters required restricts the variety of molecules supported by force fields such as MM3.

Roterman et al. (1989a,b) compared the AMBER, CHARMM, and ECEPP force fields for peptides. These force fields gave significantly different results for the minimum energy configurations. Hwang et al. (1994) compared the structures predicted by the AMBER, MM3, and CFF93 force fields

with experimental data. The bond angles for many classes of molecules were calculated more accurately with the CFF93 than the AMBER model. The MM3 and AMBER methods can be used to calculate both rotational barriers and conformational energy differences with a reasonable degree of accuracy. However, the CFF93 model provides the most accurate predictions. Asensio et al. (1995) have concluded that the AMBER, CVFF, and CFF91 force fields predict the observed conformation of methyl α-lactoside with a similar degree of accuracy.

A common feature of the AMBER, CHARMM, ECEPP and MM3, OPLS, and UFF models is the assumption that the charge distribution associated with each atom is independent of the conformation of the molecule. These models are isotropic and no account is taken of the rearrangement of valance electron found in molecular bonds. Consequently, site−site interactions are often not transferable to other molecules within the same group. Anisotropic site−site potentials (Price, 1988; Price and Stone, 1992) have been reported, which overcome this deficiency. The results obtained from anisotropic models are generally better than that can be obtained for isotropic models, particularly if hydrogen bonding is involved.

An alternative to generalized force fields is to develop force fields for either a specific class of molecules or a particular application. An AA force field for liquid ethanol has been devised (Müller-Plathe, 1996) and used in MD simulations. Nath et al. (1998a) reported successful simulations of the VLE of alkanes modeled via a potential dubbed "NERD." The NERD potential combines nonbonded interaction with elements of intramolecular interaction in the spirit of molecular mechanics. The potential yields satisfactory agreement with experiment without using adjustable parameters (Nath et al., 1998b). The transferable potentials for phase equilibria (TraPPE) was specifically developed (Martin and Siepmann, 1998) for phase behavior. It has undergone continual development (Shah et al., 2017) for phase behavior, providing a very good representation of VLE (Eggimann et al., 2020).

Historically, force fields have been developed without accounting for the effect of polarization, but the importance of this contribution is being increasingly recognized (Vanommeslaeghe and MacKerell, 2015; Bedrov et al., 2019) and incorporated into existing force fields. The statistical mechanics (Gray and Gubbins, 1984) of polarization is well understood, but it poses several computational challenges (Li et al., 2008) resulting in the use of various approximations. Apart from the effectiveness of these approximations, they are often applied to force fields with known theoretical shortcomings such as the widespread use of the LJ potential for nonbonded interactions. In principle, accounting for polarization should have an important influence on the properties of real fluids

Increasingly ab initio data are being used to inform the development (Sun and Deng, 2017) of force fields. The use of ab initio data to determine the entire force field of even a simple system is challenging (Chapter 4). Instead,

calculations from first principles are primarily used to determine key aspects, such as the constants involved in bond stretching and bending.

The determination of force field parameters is both very time consuming and subject to interrelated influences that inhibit efficient changes. For example, the parameters of some force fields in biomolecular simulations have been developed assuming a particular model for water and would need to be re-evaluated for a new water model. In this context the application of machine learning (Huang and von Lilienfeld, 2021; Keith et al., 2021) has the potential of greatly facilitating the selection of parameters for new situations.

3.11 Case study: Models for water

Water is arguably the most-widely studied molecule. However, despite an enormous scientific effort spanning several decades (Chialvo and Cummings, 1999; Guillot, 2002; Vega et al., 2009), our understanding of its behavior remains incomplete. There are at least 85 different models for water, many of which have been evaluated elsewhere (Shvab and Sadus, 2016). For our purposes, water provides us useful illustration of the development of models for molecular simulation.

3.11.1 Rigid models

In principle, water is a simple triatomic nonlinear molecule with a H-O-H bond angle of 104.5°. Although it is well-understood that molecules flex, a simplifying approximation is to treat the bonds as complexly rigid. These considerations suggest that the simplest realistic model would involve three sites, corresponding to the presence of the oxygen and hydrogen atoms (Fig. 3.11).

The nonlinear geometry gives rise to a dipole moment, which results in a large dipole moment. The contribution from the dipole moment can be obtained by situating partial charges of the atoms such that electro-neutrality

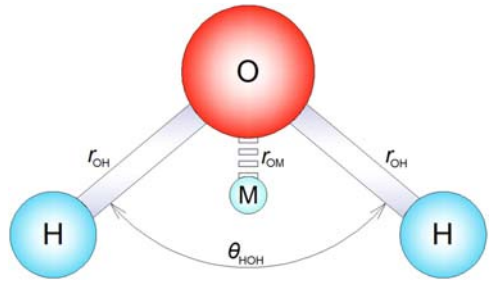


FIGURE 3.11 The structure of water, identifying intramolecular terms, constituent atoms, and an additional site (M) used to evaluate its properties.

is observed. In addition, nonbonded oxygen−oxygen, oxygen−hydrogen, and hydrogen−oxygen dispersion interactions can be envisaged between different pairs of water molecules. We would expect that the oxygen−oxygen interactions would make the dominant contribution, which could be approximated via the LJ potential. A simple potential corresponding to these considerations is

$$u_{eff}(r) = 4\varepsilon \sum_{i}^{N} \sum_{j \neq i}^{N} \left\{ \left(\frac{\sigma}{r_{ij}^{oo}} \right)^{12} - \left(\frac{\sigma}{r_{ij}^{oo}} \right)^{6} \right\} + \frac{e^2}{4\pi\varepsilon_0} \sum_{i}^{N} \sum_{j \neq i}^{N} \frac{q_i q_j}{r_{ij}}, \qquad (3.94)$$

where r_{ij}^{oo} and r_{ij} are the distances between the oxygen sites and charged sites of two molecules, respectively; $e = 1.602176634 \times 10^{-19}$ C is the charge (Tiesinga et al., 2021) such that the charges are conveniently represented as multiples of e. Notice that in Eq. (3.93), there is no contribution from the bond angle because in a ridged model it is invariant. The triangular configuration is obtained by specifying the relative positions of the oxygen and hydrogen atoms according to the bond lengths and bond angle.

Eq. (3.94) is the three-site simple point charge (SPC) potential proposed by Berendsen et al. (1981). The SPC models give a good description of the dielectric constant but it does not predict adequately phase equilibria, particularly at high temperatures (de Pablo et al., 1990). One example of the many reported improvements is to allow the charges to fluctuate (Martin et al., 1998). The SPC model also does not correctly predict the energy, self-diffusion constant, and radial distribution function. However, its accuracy can be improved (Berendsen et al., 1987) by applying a polarization correction

$$\Delta U_{pol} = \frac{\left(\boldsymbol{\mu}_l - \boldsymbol{\mu}_g \right)^2}{2\alpha}, \qquad (3.95)$$

where $\boldsymbol{\mu}_l$ is the dipole moment of the molecule, $\boldsymbol{\mu}_g$ is the dipole moment of the isolated molecule, and the α is the isotropic scalar polarizability of water molecule. Eq. (3.95) does not change Eq. (3.94), but simply results in new values of the model's parameters (Table 3.2). This extended version (SPC/E) of the SPC model reproduces the vaporization enthalpy and more accurately predicts features such as the critical behavior, pair correlation, and dielectric constant reasonably well (Guillot, 2002). Deficiencies in its accuracy include slightly high self-diffusion coefficients and specific heat capacities (Mao and Zhang, 2012).

The SPC and SPC/E potentials are three-site models with the partial charges associated with the position of the oxygen and hydrogen atoms. This is intuitively the most obvious distribution of charges. By definition, a model is a theoretical abstraction of reality and as such it does not have to conform to either our intuition or the actual situation. An improved representation of

TABLE 3.2 Summary of intermolecular bond distance, bond angle, and energy parameters for water models.

Model	q_1 (e)	q_2 (e)	r_{OH} (Å)	r_{OM} (Å)	θ (°)	ϕ (°)	σ (Å)	ε (kJ/mol)
SPC[a]	0.41	− 0.82	1.0		109.47		3.166	0.65
SPC/E[b]	0.4238	− 0.8476	1.0		109.47		3.166	0.65
SPC/Fw[c]	0.417	− 0.834	1.012		113.24		3.1655	0.6507
TIP4P[d]	0.52	− 1.04	0.9572	0.15	104.52	52.26	3.154	0.65
TIP4P/2005[e]	0.5564	− 1.1280	0.9572	0.1546	104.52	52.26	3.1589	0.7749
TIP5P[f]	0.241	− 0.241	0.9572	0.70	104.52	109.47	3.12	0.6694

[a] *Berendsen et al., 1981*.
[b] *Berendsen et al., 1987*.
[c] *Wu et al., 2006*.
[d] *Jorgensen et al., 1983*.
[e] *Abascal and Vega, 2005*.
[f] *Mahoney and Jorgensen, 2000*.

the properties of water can be obtained by introducing other partial charge sites that are not associated with the constituent atoms. Jorgensen et al. (1983) introduced a fourth charge site (Fig. 3.11) to generate a transferable intermolecular potential 4-point (TIP4P) model for water, which was later re-parametrized (Abascal and Vega, 2005) to yield TIP4P/2005. The TIP4P/2005 model accurately predicts (Shvab and Sadus, 2016) properties such as the overall phase diagram, VLE densities, and the surface tension. The addition of a fifth site (TIP5P) (Mahoney and Jorgensen, 2000) improves the prediction of densities over a wide range of temperatures and more accurately determines the configurational energy.

A common feature of the various multisite models is that the method for calculating the energy of interaction remains the same via Eq. (3.93). The only changes are (a) more sites need to be included and (b) the values of the potential parameters are different.

3.11.2 Flexible models

A simple improvement to water models can be achieved (Wu et al., 2006) by allowing the bonds to be flexible. For example, The SPC/Fw model maintains the overall three-site SPC geometry, while allowing both the bond angle and bond distance to change. Therefore, the calculation of the energy requires the following intramolecular contribution to be added to Eq. (3.93),

$$u_{\text{intra}}(r) = \frac{K_b\left[\left(r_{oh1}-r_{oh}^{o}\right)^2+\left(r_{oh2}-r_{oh}^{o}\right)^2\right]}{2}+\frac{K_\theta\left(\theta_{\angle HOH}-\theta_{\angle HOH}^{o}\right)^2}{2} \quad (3.96)$$

where r_{oh}^{o} and $\theta_{\angle HOH}^{o}$ are the equilibrium bond length and bond angle in the gas phase. This simple addition results in significant improvements to the prediction of VLE (Raabe and Sadus, 2007), dielectric behavior (Raabe and Sadus, 2011) and transport properties (Raabe and Sadus, 2012). A partial explanation for the improved agreement with experiment is that bond flexibility mimics the effects of polarization. This is particularly useful as polarization is computationally expensive (Li et al., 2008; Shvab and Sadus, 2016).

3.11.3 Polarizable models

It is becoming increasingly apparent that progress in predicting the properties of water requires the explicit inclusion of polarization (Chapter 4). A review of polarizable models is given elsewhere (Shvab and Sadus, 2016). An atomic multipole optimized energetics for biomolecular applications (AMOEBA) model has been proposed (Ren and Ponder, 2003). Evaluating polarization is computationally expensive and a simplified version (iAOMEBA, where "i" denotes inexpensive) has also been developed (Wang et al., 2013). Although initially developed for biomolecular applications, AMOEBA yields results for

thermodynamic properties (Shvab and Sadus, 2016) that are of superior accuracy to those obtained for rigid models. A systematic approach to improving water models is further examined in Chapter 4.

3.12 Application to mixtures

The proceeding intermolecular potentials and force fields have been developed exclusively for the properties of pure components. However, it is rare in nature for substances to be isolated in their pure state and the calculation of the properties of real materials requires us to account for the properties of the mixtures. Unlike conventional calculations (Wei and Sadus, 2000), which inevitably require invoking a theory of mixtures, molecular simulations can be easily extended from pure systems to mixtures by the addition of different molecular species. However, a way is required to account for interactions between the contributions of interactions between dissimilar molecules. This is almost invariably achieved via combining rules for the various intermolecular parameters. Most commonly, the Lorentz−Berthelot rules (Wei and Sadus, 2000) are used.

$$\sigma_{ij} = \frac{\sigma_i + \sigma_j}{2} \tag{3.97}$$

$$\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j} \tag{3.98}$$

Eq. (3.97) is exact only for the case of HS, and most real molecules do not behave as HS as is apparent from the realistic potentials discussed previously. It has also been known for a long time (Guggenheim, 1952) that Eq. (3.98) has no theoretical basis. Eqs. (3.97) and (3.98) do not accurately represent (Delhommelle and Millié, 2001) the properties of the relatively simple case of the noble gases, let alone molecules of greater complexity and interest such as proteins and macromolecules. Many alternative combining rules have been proposed (Wei and Sadus, 2000; Hasalm et al., 2008); however, none are sufficiently accurate to avoid the use of adjustable parameters to fully optimize agreement between theory and experiment. This means that the predictive usefulness of molecular simulation is restricted to relatively few mixtures.

An alternative to averaging the potential parameters is to average the contributions of the intermolecular potentials (Stiegler and Sadus, 2015):

$$u_{ij} = \frac{u_\alpha + u_\beta}{2}, \tag{3.99}$$

where the two distinct intermolecular potentials are denoted by u_α and u_β.

Using Eq. (3.99), the force ($\boldsymbol{f}_{ij}$) between the two particles is obtained from

$$\boldsymbol{f}_{ij} = -\frac{\mathbf{r}_{ij}}{2r_{ij}}\left(\frac{du_\alpha}{dr_{ij}} + \frac{du_\beta}{dr_{ij}}\right). \tag{3.100}$$

For a binary mixture of molecules interacting via the LJ and 10−5 LJ/M potential, Eq. (3.99) yields:

$$u(\alpha \neq \beta) = 2\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} + \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{10} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{5}\right] \tag{3.101}$$

Comparison of simulations (Stiegler and Sadus, 2015) using Eq. (3.101) with results obtained by combining the LJ/M potentials by averaging the exponents indicates good agreement for a range of thermodynamic properties. The main advantage of Eq. (3.99) is that different intermolecular potentials can be used for the individual mixture components. In contrast, the traditional combining rule approach is limited to mixtures with both common potentials and common intermolecular parameters.

3.13 Summary

Ultimately the development of intermolecular potentials and force fields should be guided by theory. Although progress is being achieved toward this goal via, for example, the application of ab initio data, most potentials are empirically based and should be regarded as effective pair potentials. Nonetheless, a wide variety of potentials are available, many of which have not been evaluated fully. The force fields are particularly useful for the properties of macromolecules and biological systems, capturing many details of the underlying physics of both inter- and intramolecular interactions. An ongoing challenge is that potentials are developed for pure systems and the predictive capability of the simulation of mixtures is restricted by the need to use combining rules to determine the interactions between dissimilar molecules. This is likely to benefit from future developments in the application of machine learning.

References

Abascal, J.L.F., Vega, C., 2005. A general purpose model for the condensed phases of water: TIP4P/2005. J. Chem. Phys. 124, 024503.

Abramowitz, M., Stegun, I.A. (Eds.), 1972. Handbook of Mathematical Functions with Formula, Graphs and Mathematical Tables. Dover, New York, p. 255.

Ahlrichs, R., Penco, P., Scoles, G., 1977. Intermolecular forces in simple systems. Chem. Phys. 19, 119−130.

Ahmed, A., Sadus, R.J., 2009a. Phase diagram of the Weeks-Chandler-Andersen potential from very low to high temperatures and pressures. Phys. Rev. E 80, 061101. Erratum: *Phys. Rev. E* 99, 029901 (2019).

Ahmed, A., Sadus, R.J., 2009b. Solid-liquid equilibria and triple points of n-6 Lennard-Jones fluids. J. Chem. Phys. 131, 174504. Erratum: *J. Chem. Phys.* 133, 229902 (2010).

Ahmed, A., Sadus, R.J., 2010. Effect of potential truncations and shifts on the solid-liquid phase coexistence of Lennard-Jones fluids. J. Chem. Phys. 133, 124515.

Ahmed, A., Mausbach, P., Sadus, R.J., 2009. Strain-rate dependent shear viscosity of the Gaussian core model fluid. J. Chem. Phys. 131, 224511.

Ahmed, A., Mausbach, P., Sadus, R.J., 2010. Pressure and energy behavior of the Gaussian core model fluid under shear. Phys. Rev. E 82, 011201.

Alder, B.J., Wainwright, T.E., 1957. Phase transition for a hard sphere system. J. Chem. Phys. 27, 1208–1209.

Alder, B.J., Gass, D.M., Wainwright, T.E., 1970. Studies in molecular dynamics. VIII. The transport coefficients for a hard-sphere fluid. J. Chem. Phys. 53, 3813–3826.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Allinger, N.L., 1977. Conformational analysis. 130. MM2. A hydrocarbon force field utilizing V_1 and V_2 torsional terms. J. Am. Chem. Soc. 99, 8127–8134.

Allinger, N.L., Yuh, Y.H., Lii, J.-H., 1989. Molecular mechanics. The MM3 force field for hydrocarbons. J. Am. Chem. Soc. 111, 8551–8575.

Asensio, J.L., Martin-Pastor, M., Jimenez-Barbero, J., 1995. The use of CVFF and CFF91 force fields in conformational analysis of carbohydrate molecules. Comparison with AMBER molecular mechanics and dynamics calculations for methyl α-lactoside. Int. J. Macromol. 17, 137–148.

Axilrod, B.M., Teller, E., 1943. Interaction of the van der Waals' type between three atoms. J. Chem. Phys. 11, 299–300.

Bacher, A.K., Schrøder, T.B., Dyre, J.C., 2018a. The EXP pair-potential system. I. Fluid phase isotherms, isochores, and quasinuniversality. J. Chem. Phys. 149, 114501.

Bacher, A.K., Schrøder, T.B., Dyre, J.C., 2018b. The EXP pair-potential system. II. Fluid phase isomorphs. J. Chem. Phys. 149, 114502.

Bárcenas, M., Odriozola, G., Orea, P., 2014. Coexistence and interfacial properties of triangular-well fluids. Mol. Phys. 112, 2114–2121.

Bartke, J., Hentschke, R., 2007. Phase behavior of the Stockmayer fluid via molecular dynamics simulation. Phys. Rev. E 75, 061503.

Bastea, S., Fried, L.E., 2008. Exp6-polar thermodynamics of dense supercritical water. J. Chem. Phys. 128, 174502.

Baxter, R.J., 1968. Percus-Yevick equation for hard spheres with surface adhesion. J. Chem. Phys. 49, 2770–2774.

Becker, C.A., Tavazza, F., Trautt, Z.T., de Macedo, R.A.B., 2013. Considerations for choosing and using force fields and interatomic potentials in material science and engineering. Curr. Opin. Sol. State. Mat. Sci. 17, 277–283.

Bedrov, D., Piquemal, J.-P., Borodin, O., MacKerell Jr, A.D., Roux, B., Schröder, C., 2019. Molecular dynamics simulations of ionic liquids and electrolytes using polarizable force fields. Chem. Rev. 119, 7940–7995.

Behler, J., 2016. Perspective: Machine learning potentials for atomistic simulations. J. Chem. Phys. 145, 170901.

Bell, R.J., Zucker, I.J., 1976. In: Klein, M.L., Venables, J.A. (Eds.), Rare Gas Solids, Vol. 1. Academic Press, London.

Berendsen, H.J.C., Postma, J.P.M., von Gunsteren, W.F., Hermans, J., 1981. In: Pullman, B. (Ed.), Intermolecular Forces. Reidel, Dordrecht.

Berendsen, H.J.C., Grigera, J.R., Straatsma, T.P., 1987. The missing term in effective pair potentials. J. Phys. Chem. 91, 6269–6271.

Best, R.B., Hummer, G., 2009. Optimized molecular dynamics force fields applied to the helix-coil transition of polypeptides. J. Phys. Chem. B 113, 9004–9015.

Best, R.B., Zhu, X., Shim, J., Lopes, P.E.M., Mittal, J., Feig, M., MacKerell Jr, A.D., 2012. Optimization of the additive CHARMM all-atom protein force field targeting improved sampling of the backbone ϕ, ψ and side-chain χ_1 χ_2 dihedral angles. J. Chem. Theory Comput. 8, 3257–3273.

Bobetic, M.V., Barker, J.A., 1970. Lattice dynamics with three-body forces: argon. Phys. Rev. B. 2, 4169–4175.

Born, M., Mayer, J.E., 1932. Zur Gittertheorie der Ionenkristalle. Z. Phys. 75, 1–18.

Bosko, J.T., Todd, B.D., Sadus, R.J., 2004. Internal structure of dendrimers in the melt: A molecular dynamics study. J. Chem. Phys. 121, 1091–1096.

Bosko, J.T., Todd, B.D., Sadus, R.J., 2006. Analysis of the shape of dendrimers under shear. J. Chem. Phys. 124, 044910.

Boublík, T., Nezbeda, I., 1986. P-V-T behaviour of hard body fluids. Theory and experiment. Coll. Czech. Chem. Commun. 51, 2301–2432.

Boyes, S.J., 1994. The interatomic potential of argon. Chem. Phys. Lett. 221, 467–472.

Brooks, B.R., Bruccoleri, R.E., Olafson, B.D., States, D.J., Swaminathan, S., Karplus, M., 1983. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. J. Comput. Chem. 4, 187–217.

Buckingham, R.A., Corner, J., 1947. Tables of second virial and low-pressure Joule-Thompson coefficients for intermolecular potentials with exponential repulsion. Proc. Roy. Soc. A 189, 118–129.

Buckingham, A.D., Fowler, P.W., Hutson, 1988. Theoretical studies of van der Waals molecules and intermolecular forces. Chem. Rev. 88, 963–988.

Caillol, J.M., 1995. A Monte Carlo study of the dielectric constant of the restricted primitive model of electrolytes on the vapor branch of the coexistence line. J. Chem. Phys. 102, 5471–5478.

Car, R., Parrinello, M., 1985. Unified approach for molecular dynamics and density-functional theory. Phys. Rev. Lett. 55, 2471–2474.

Casewit, C.J., Colwell, K.S., Rappé, A.K., 1992a. Application of a universal force field to organic molecules. J. Am. Chem. Soc. 114, 10035–10046.

Casewit, C.J., Colwell, K.S., Rappé, A.K., 1992b. Application of a universal force field to main group compounds. J. Am. Chem. Soc. 114, 10046–10053.

Chandler, D., Weeks, J.D., Andersen, H.C., 1983. Van der Waals picture of liquids, solids, and phase transformations. Science 220, 787–794.

Chang, J., Sandler, S.I., 1995. The Wertheim integral equation theory with the ideal chain approximation and a dimer equation of state: generalization to mixtures of hard-sphere chain fluids. J. Chem. Phys. 103, 3196–3211.

Cheung, P.S.Y., Powles, J.G., 1975. The properties of liquid nitrogen IV. A computer simulation. Mol. Phys. 30, 921–949.

Chialvo, A.A., Cummings, P.T., 1999. Molecular-based modeling of water and aqueous solutions at supercritical conditions. Adv. Chem. Phys. 109, 207–433.

Cisneros, G.A., Karttunen, M., Ren, P., Sagui, C., 2013. Classical electrostatics for biomolecular simulations. Chem. Rev. 114, 779–814.

Damm, W., Frontera, A., Tirado-Rives, J., Jorgensen, W.L., 1997. OPLS All-atom force field for carbohydrates. J. Comp. Chem. 18, 1955–1970.

de Carvalho, R.J.F.L., Evans, R., 1997. The screened Coulomb (Yukawa) charged hard sphere binary fluid. Mol. Phys. 92, 211–228.

de Kuijper, A., Smit, B., Schouten, J.A., Michels, J.P.J., 1990. Fluid-fluid phase separation in a repulsive α-exp-6 mixture: a comparison with the full α-exp-6 mixture by means of computer simulations. Europhys. Lett. 13, 679–682.

de Leeuw, S.W., Smit, B., Williams, C.P., 1990. Molecular dynamics studies of polar/nonpolar fluid mixtures. I. Mixtures of Lennard-Jones and Stockmayer fluids. J. Chem. Phys. 93, 2704–2714.

de Pablo, J.J., Prausnitz, J.M., Strauch, H.J., Cummings, P.T., 1990. Molecular simulation of water along the liquid-vapor coexistence curve from 25°C to the critical point. J. Chem. Phys. 93, 7355–7359.

Deiters, U.K., Sadus, R.J., 2019a. Two-body interatomic potentials for He, Ne, Ar, Kr, and Xe from ab initio data. J. Chem. Phys. 150, 134504.

Deiters, U.K., Sadus, R.J., 2019b. Fully a priori prediction of the vapor-liquid equilibria of Ar, Kr, and Xe from ab initio two-body plus three-body interatomic potentials. J. Chem. Phys. 151, 034509.

Deiters, U.K., Sadus, R.J., 2020. Ab initio interatomic potentials and the classical molecular simulation prediction of the thermophysical properties of helium. J. Phys. Chem. B 124, 2268–2276.

Deiters, U.K., Sadus, R. J., (2023), to be submitted.

Deiters, U.K., Sadus, R.J., 2021. Interatomic interactions responsible for the solid-liquid and vapor-liquid phase equilibria of neon. J. Phys. Chem. B 125, 8522–8531.

DeKock, R.L., Madura, J.D., Rioux, F., Casanova, J., 1993. In: Lipkowitz, K.B., Boyd, D.B. (Eds.), Reviews in Computational Chemistry, Vol. 4. VCH Publishers, New York.

Delhommelle, J., Millié, P., 2001. Inadequacy of the Lorentz-Berthelot combining rules for accurate predictions of equilibrium properties by molecular simulation. Mol. Phys. 99, 619–625.

Dham, A.K., Allnatt, A.R., Meath, W.J., Aziz, R.A., 1989. The Kr-Kr potential energy curve and related physical properties: the XC and HFD-B potential models. Mol. Phys. 67, 1291–1307.

Dinur, U., Hagler, A.T., 1991. In: Lipkowitz, K.B., Boyd, B. (Eds.), Reviews in Computational Chemistry, Vol. 2. VCH Publishers, New York.

Douketis, C., Scoles, G., Marchetti, S., Zen, M., Thakkar, A.J., 1982. Intermolecular forces via hybrid Hartree-Fock-SCF plus damped dispersion (HFD) energy calculations: an improved spherical model. J. Chem. Phys. 76, 3057–3063.

Dubey, G.S., O'Shea, S.F., 1994. Phase equilibria of Lennard-Jones plus quadrupolar fluids by Gibbs-ensemble Monte Carlo simulation. Phys. Rev. E 49, 2175–2183.

Duh, D.-M., Mier-Y-Terán, L., 1997. An analytical equation of state for the hard-core Yukawa fluid. Mol. Phys. 90, 373–379.

Dyre, J.C., Pedersen, U.R., 2023. Comparing zero-parameter theories for the WCA and harmonic-repulsive melting lines. J. Chem. Phys. 158, 154504.

Egelstaff, P.A., 1987. In: Price, D.L., Sköld (Eds.), Methods of Experimental Physics: Neutron Scattering. Academic Press, San Diego.

Egelstaff, P.A., 1994. An Introduction to the Liquid State, second ed. Clarendon Press, Oxford.

Eggimann, B.L., Sun, Y., DeJaco, R.F., Singh, R., Ahsan, M., Josephson, T.R., Siepmann, J.I., 2020. Assessing the quality of molecular simulations for vapor-liquid equilibra: an analysis of the TraPPE database. J. Chem. Eng. Data 65, 1330–1344.

Elliott, J.R., Hu, L., 1999. Vapor-Liquid equilibria of square-well spheres. J. Chem. Phys. 110, 3043–3048.

Escobedo, F.A., de Pablo, J.J., 1996. Simulation and prediction of vapour-liquid equilibria for chain molecules. Mol. Phys. 87, 347–366.

Fantoni, R., Malijevsky, A., Santos, A., Giacometti, A., 2011. Phase diagram of the penetrable-square-well model. Eur. Phys. Lett. 93, 26002.

Feinberg, M.J., de Rocco, A.G., 1964. Intermolecular forces: the triangle well and some comparisons with the square well and Lennard-Jones. J. Chem. Phys. 41, 3439–3540.

Fischer, J., Wendland, M., 2023. On the history of key empirical intermolecular potentials. Fluid Phase Equilib. 573, 113876.

Foloppe, N., MacKerell Jr, A.D., 2000. All-atom empirical force field for nucleic acids: I. Parameter optimization based on small molecule and condensed phase macromolecular target data. J. Comp. Chem. 21, 86–104.

Fowler, R., Guggenheim, E.A., 1939. Statistical Thermodynamics: A version of Statistical Mechanics for Students of Physics and Chemistry. Cambridge University Press, Cambridge, p. 280.

Fowler, R.H., Graben, H.W., de Rocco, A.G., Feinberg, M.J., 1965. Some additional results for the triange-well potential model. J. Chem. Phys. 43, 1083–1084.

Fumi, F.G., Tosi, M.P., 1964. Ionic sizes and Born repulsive parameters in the NaCl-type alkali halides. I. The Huggins-Mayer and Pauling forms. J. Phys. Chem. Solids 25, 31–43.

Galicia-Pimental, U.F., López-Lemus, J., Orea, P., 2008. Liquid-vapor interfacial properties of attractive Yukawa fluids. Fluid Phase Equilib. 265, 205–208.

Galliero, G., Boned, C., 2008. Molecular dynamics study of the repulsive form influence of the interaction potential on structural, thermodynamic, interfacial, and transport properties. J. Chem. Phys. 129, 074506.

Gao, J., Weiner, J.H., 1989. Contribution of covalent bond force to pressure in polymer melts. J. Chem. Phys. 91, 3168–3173.

Gao, G.T., Woller, J.B., Zeng, X.C., Wang, W., 1997. Vapour-liquid equilibria of binary mixtures containing Stockmayer molecules. J. Phys.: Condens. Matter 9, 3349–3360.

Gao, H., Wang, J., Sun, J., 2019. Improve the performance of machine-learning potentials by optimizing descriptors. J. Chem. Phys. 150, 244110.

Gloor, G.J., Jackson, G., Blas, F.J., del Rio, F.J., de Miguel, E., 2004. An accurate density functional theory for the vapor-liquid interface of associating chain molecules based on the statistical associating fluid theory for potentials of variable range. J. Chem. Phys. 121, 12740–12759.

Goodford, P.J., 1985. A computational procedure for determining energetically favorable binding sites on biologically important macromolecules. J. Med. Chem. 28, 849–857.

Gordon, P.A., 2006. Development of intermolecular potentials for predicting transport properties of hydrocarbons. J. Chem. Phys. 125, 014504.

Gray, C.G., Gubbins, K.E., 1984. Theory of Molecular Fluids, Vol. 1: Fundamentals. Clarendon Press, Oxford.

Grest, G.S., Kremer, K., 1986. Molecular dynamics simulation for polymers in the presence of a heat bath. Phys. Rev. A 33, 3628–3631.

Guggenheim, E.A., 1952. Mixtures: the theory of the Equilibrium Properties of some Simple Classes of Mixtures, Solutions and Alloys. Clarendon Press, Oxford, p. 158.

Guillot, B., 2002. A reappraisal of what we have learnt during three decades of computer simulations of water. J. Mol. Liq. 101, 219–260.

Guissani, Y., Guillot, B., 1994. Coexisting phases and criticality in NaCl by computer simulation. J. Chem. Phys. 101, 490–509.

Guvench, O., Mallajosyula, S.S., Raman, E.P., Hatcher, Vanommeslaeghe, K., Foster, T.J., Jamison II, F.W., MacKerell Jr, A.D., 2011. CHARMM additive all-atom force field for carbohydrate derivatives and its utility in polysaccharide and carbohydrate protein modelling. J. Chem. Theory Comput. 7, 3162–3180.

Haile, J.M., 1992. Molecular Dynamics Simulation. Elementary Methods. John Wiley & Sons, New York.

Hamad, E., 1995. Thermodynamic properties of molecules interacting with inverse power potentials. Z. Phys. Chem. 190, 183–191.
Hansen, J.P., McDonald, I.R., 1975. Statistical mechanics of dense ionized matter. IV. Density and charge fluctuations in a simple molten salt. Phys. Rev. A 11, 2111–2123.
Hansen, J.P., Schiff, D., 1973. Influence of interatomic repulsion on the structure of liquids at melting. Mol. Phys. 25, 1281–1290.
Hasalm, A.J., Galindo, A., Jackson, G., 2008. Prediction of binary intermolecular potential parameters for use in modeling fluid mixtures. Fluid Phase Equilib. 266, 105–128.
Hentschke, R., Bartke, J., Pesth, F., 2007. Equilibrium polymerization and gas-liquid critical behavior in the Stockmayer fluid. Phys. Rev. E 75, 011506.
Hepburn, J., Scoles, G., Penco, R., 1975. A simple but reliable method for prediction of intermolecular potentials. Chem. Phys. Lett. 36, 451–456.
Heyes, D.M., Powles, J.G., 1998. Thermodynamic, mechanical and transport properties of fluids with steeply repulsive potentials. Mol. Phys. 95, 259–267.
Heyes, D.M., Rickaysen, G., Branka, 2004. Static properties and time correlation functions of fluids with steeply repulsive potentials. Mol. Phys. 102, 2057–2070.
Hirschfelder, J.O., Curtiss, C.F., Bird, R.B., 1954. Molecular Theory of Gases and Liquids. John Wiley & Sons, New York.
Homer, J., Jenkins, J.D., Porter, K.E., Kakhu, A.I., 1991. Prediction of vapour-liquid equilibrium data for binary mixtures from molecular parameters using a generalized London potential. J. Chem. Soc. Faraday Trans. 87, 57–62.
Hoover, W.G., 2006. Smooth Particle Applied Mechanics: The State of the Art. World Scientific, Singapore.
Hoover, W.G., Ross, M., Johnson, K.W., Henderson, D., Barker, J.A., Brown, B.C., 1970. Soft-sphere equation of state. J. Chem. Phys. 52, 4931–4941.
Hoover, W.G., Gray, S.G., Johnson, K.W., 1971. Thermodynamic properties of the fluid and solid phases for inverse power potentials. J. Chem. Phys. 55, 1128–1136.
Hoover, W.G., Young, D.A., Grover, R., 1972. Statistical mechanics of phase diagrams. I. Inverse power potentials and the close-packed to body-centered cubic transition. J. Chem. Phys. 56, 2207–2210.
Hornak, V., Abel, R., Okur, A., Strockbine, B., Roitberg, A., Simmerling, C., 2006. Comparison of multiple Amber force fields and development of improved backbone parameters. Proteins: Struct. Func. BioInfo. 65, 712–725.
Huang, B., von Lilienfeld, A., 2021. Ab initio machine learning in chemical compound space. Chem. Rev. 121, 10001–10036.
Hwang, M.J., Stockfisch, T.P., Hagler, A.T., 1994. Derivation of class II force fields. 2. Derivation and characterization of a class II force field, CFF93, for the alkyl functional group and alkane molecules. J. Am. Chem. Soc. 116, 2515–2525.
Jia, R., Hentschke, R., 2011. Simulation study of the polarizable Stockmayer fluid in an external field. Phys. Rev. E. 84, 051508.
Jiang, S., Pitzer, K.S., 1995. Thermodynamic properties of mixtures of dipolar and quadrupolar hard spheres: theory and simulation. J. Chem. Phys. 102, 7632–7640.
Johnson, J.K., Gubbins, K.E., 1992. Phase equilibria for associating Lennard-Jones fluids from theory and simulation. Mol. Phys. 77, 1033–1053.
Johnson, J.K., Müller, E.A., Gubbins, K.E., 1994. Equation of state for Lennard-Jones chains. J. Phys. Chem. 98, 6413–6419.
Jones, J.E., 1924. On the determination of molecular fileds. II. From the equation of state of a gas. Proc. R. Soc. London, Ser. A 106, 463–477.

Jorgensen, W.L., Tirado-Rives, J., 1988. The OPLS potential functions for proteins. Energy minimizations for crystals of cyclic peptides and crambin. J. Am. Chem. Soc. 110, 1657–1666.

Jorgensen, W.L., Chandrachekar, J., Madura, J.D., Impey, R.W., Klein, M.L., 1983. Comparison of simple potential functions for simulating liquid water. J. Chem. Phys. 79, 926–935.

Joung, I.S., Cheatham III, T.E., 2008. Determination of alkali and halide monovalent ion parameters for use in explicitly solvated biomolecular simulations. J. Chem. Phys. B 112, 9020–9041.

Jung, A., 2022. Machine Learning: The Basics. Springer, Berlin.

Kalyuzhnyi, Y.V., Cummings, P.T., 1996. Phase diagram for the Lennard-Jones fluid modelled by the hard-core Yukawa fluid. Mol. Phys. 87, 1459–1462.

Kaminski, G., Duffy, E.M., Matsui, T., Jorgensen, W.L., 1994. Free energies of hydration and pure liquid properties of hydrocarbons from the OPLS all-atom model. J. Phys. Chem. 98, 13077–13088.

Keith, J.A., Vassilev-Galindo, V., Cheng, B., Chmiela, S., Gastegger, M., Müller, K.-R., Tkatchenko, A, 2021. Combining machine learning and computational chemistry for predictive insights into chemical systems. Chem. Rev. 121, 9816–9872.

Kennelly, A.E., 1931. Rationalised versus unrationalised practical electromagnetic units. Proc. Amer. Phil. Soc. 70, 103–119.

Khrapak, S.A., Chaudhuri, M., Morfill, G.E., 2011. Freezing of Lennard-Jones-type fluids. J. Chem. Phys. 134, 054120.

Kihara, T., 1951. The second virial coefficient of non-spherical molecules. J. Phys. Soc. Jpn. 6, 289–296.

Kirschner, K.N., Woods, R.J., 2001. Solvent interactions determine carbohydrate conformation. PNAS 98, 10541–10545.

Kiyohara, K., Spyriouni, T., Gubbins, K.E., Panagiotopoulos, A.Z., 1996. Thermodynamic scaling Gibbs ensemble Monte Carlo: A new method for determination of phase coexistence properties of fluids. Mol. Phys. 89, 965–974.

Klauda, J.B., Venable, R.M., Freites, J.A., O'Connor, J.W., Tobias, D.J., Mondragon-Ramirez, C., Vorobyov, I., MacKerell Jr, A.D., Pastor, R.W., 2010. Update of the CHARMM all-atom additive force field for lipids: validation on six lipid types. J. Chem. Phys. B 114, 7830–7843.

Klein, M.L., Venables, J.A. (Eds.), 1976. Rare Gas Solids, Vol. 1. Academic Press, London.

Kohler, F., Wilhem, E., Posch, H., 1976. Recent advances in the physical chemistry of the liquid state. Adv. Mol. Relax. Process. 8, 195–239.

Koide, A., Meath, W.J., Allnatt, A.R., 1981. Second-order charge overlap effects and damping functions for isotropic atomic and molecular interactions. Chem. Phys. 58, 105–119.

Kooij, S., Lerner, E., 2017. Unjamming in models with analytic pairwise potentials. Phys. Rev. E 95, 062141.

Koshi, M., Matsui, H., 1994. Molecular dynamics study of high temperature phase-separation in a H_2O/N_2 mixture with exp-6 interactions. Mol. Sim. 12, 227–239.

Kreek, H., Meath, W.J., 1969. Charge-overlap effects. Dispersion and induction forces. J. Chem. Phys. 50, 2289–2302.

Kriebel, C., Winkelmann, J., 1996. Thermodynamic properties of polarizable Stockmayer fluids: Perturbation theory and simulation. Mol. Phys. 88, 559–578.

Kriebel, C., Winkelmann, J., 1998. Simulation studies on mixtures of polarizable dipolar and polarizable non-polar linear molecules. Mol. Phys. 93, 347–353.

Kröger, M., 2005. Models for Polymeric and Anisotropic Liquids. Springer, Berlin.

Lafitte, T., Apostolakou, A., Avendaño, C., Galindo, A., Adjiman, C.S., Müller, E.A., Jackson, G., 2013. Accurate statistical associating fluid theory for chain molecules formed from Mie segments. J. Chem. Phys. 139, 154504.

Laird, B.B., Haymet, A.D.J., 1989. The crystal-liquid interface of a body-centered-cubic-forming substance: computer simulation of the r^{-6} potential. J. Chem. Phys. 91, 3638–3646.

Laird, B.B., Haymet, A.D.J., 1992. Phase diagram for the inverse sixth power potential system from molecular dynamics computer simulation. Mol. Phys. 75, 71–80.

Landis, C.R., Root, D.M., Cleveland, T., 1995. In: Lipkowitz, K.B., Boyd, D.B. (Eds.), Reviews in Computational Chemistry, Vol. 6. VCH Publishers, New York.

Li, J., Zhou, Z., Sadus, R.J., 2008. Parallel algorithms for molecular dynamics with induction forces. Comput. Phys. Comm. 178, 384–392 (2008).

Lii, J.-H., Allinger, N.L., 1989a. Molecular mechanics. The MM3 force field for hydrocarbons. 2. Vibrational frequencies and thermodynamics. J. Am. Chem. Soc. 111, 8566–8575.

Lii, J.-H., Allinger, N.L., 1989b. Molecular mechanics. The MM3 force field for hydrocarbons. 3. The van der Waals potentials and crystal data for aliphatic and aromatic hydrocarbons. J. Am. Chem. Soc. 111, 8576–8582.

Likos, C.N., 2001. Effective interactions in soft condensed matter physics. Phys. Rep. 348, 267–439.

Lomba, E., Almarza, N.G., 1994. Role of the interaction range in the shaping of phase diagrams in simple fluids. The hard sphere Yukawa fluid as a case study. J. Chem. Phys. 100, 8367–8372.

London, F., 1937. The general theory of molecular forces. Trans. Faraday Soc. 33, 8–26.

Lopes, P.E.M., Guench, O., Mackerell Jr., A.D., 2015. Current status of protein force fields for molecular dynamics. Methods Mol. Biol. 1215, 47–71.

Losey, J., Sadus, R.J., 2019. Thermodynamic properties and anomalous behavior of double-Gaussian core model potential fluids. Phys. Rev. E 100, 012112.

MacKerell Jr, A.D., 2004. Empirical force fields for biological macromolecules: overview and issues. J. Comp. Chem. 25, 1585–1604.

MacKerell Jr, A.D., Bashford, D., Bellott, M., Dunbrack Jr, R.L., Evanseck, J.D., Field, M.J., Fischer, S., Gao, J., Guo, H., Ha, S., Joseph-McCarthy, D., Kuchnir, L., Kuczera, K., Lau, F.T.K., Mattos, C., Michnick, S., Ngo, T., Nguyen, D.T., Prodhom, B., Reiher III, W.E., Roux, B., Schlenkrich, M., Smith, C.J., Stote, R., Straub, J., Watanabe, M., Wiórkiewicz-Kuczera, J., Yin, D., Karplus, M., 1998. All-atom empirical potential for molecular modelling and dynamics studies of proteins. J. Chem. Phys.B 102, 3586–3616.

Mahoney, M.W., Jorgensen, W.L., 2000. A five site model of liquid water and the reproduction of the density anomaly by rigid, nonpolarizable potential functions. J. Chem. Phys. 112, 8910–8922.

Maitland, G.C., Smith, E.B., 1973. Simplified representation of intermolecular potential energy. Chem. Phys. Lett. 22, 443–446.

Maitland, G.C., Rigby, M., Smith, E.B., Wakeham, W.A., 1981. Intermolecular Forces: Their origin and Determination. Clarendon Press, Oxford.

Mao, Y., Zhang, Y., 2012. Thermal conductivity, shear viscosity and specific heat of rigid water models. Chem. Phys. Lett. 542, 37–41.

Maple, J.R., Hwang, M.-J., Stockfisch, T.P., Dinur, U., Waldman, M., Ewig, C.S., Hagler, A.T., 1994. Derivation of class II force fields. I. Methodology and quantum force field for the alkyl functional group and alkane molecules. J. Comput. Chem. 15, 162–182.

Martin, M.G., 2006. Comparison of the AMBER, CHARMM, COMPASS, GROMOS, OPLS, TraPPE and UFF force fields for the prediction of vapor-liquid coexistence curves and liquid densities. Fluid Phase Equilib. 248, 50–55.

Martin, M.G., Siepmann, J.I., 1998. Transferable potentials for phase equilibria. 1. United-atom description of n-alkanes. J. Phys. Chem. B 102, 2569–2577.

Martin, M.G., Chen, B., Siepmann, J.I., 1998. A novel Monte Carlo algorithm for polarizable force fields: application to a fluctuating charge model for water. J. Chem. Phys. 108, 3383–3385.

Máte, Z., Szalai, I., Boda, D., Henderson, D., 2011. Heat capacities of dipolar Yukawa model polar fluid. Mol. Phys. 109, 203–208.

Mausbach, P., Sadus, R.J., 2011. Thermodynamic properties in the molecular dynamics ensemble applied to the Gaussian core model fluid. J. Chem. Phys. 134, 114515.

Mausbach, P., Ahmed, A., Sadus, R.J., 2009. Solid-liquid phase equilibrial of the Gaussian core model fluid. J. Chem. Phys. 131, 184507. Erratum: *J. Chem. Phys.* **132**, 019901 (2010).

Mayer, J.E., Mayer, M.G., 1940. Statistical Mechanics. John Wiley & Sons, Ney York, p. 271.

Mejía, M., Herdes, C., Müller, E.A., 2014. Force fields for coarse-grained molecular simulations from a corresponding states correlation. Ind. Eng. Chem. Res. 53, 4131–4141.

Mick, J.R., Barhahi, M.S., Jackman, B., Rushaidat, K., Schwiebert, L., Potoff, J.J., 2015. Optimized Mie potentials for phase equilibria: application to noble gases and their mixtures with n-alkanes. J. Chem. Phys. 143, 114504.

Mie, G., 1903. Zur Kinetschen Theorie der einatomigen Körp. Ann. Phys. 316, 657–697.

Mognetti, B.M., Virnau, P., Yelash, L., Paul, W., Binder, K., Müller, M., MacDowell, L.G., 2008. Coarse-graining dipolar interactions in simple fluids and polymer solutions: Monte Carlo studies of the phase behavior. Phys. Chem. Chem. Phys. 11, 1923–1933.

Momany, F.A., McGuire, R.F., Burgess, A.W., Scheraga, H.A., 1975. Energy parameters in polypeptides. VII. Geometric parameters, partial atomic charges, nonbonded interactions, hydrogen bond interactions, and intrinsic torsional potentials for the naturally occurring amino acids. J. Phys. Chem. 79, 2361–2381.

Monson, A.P., Rigby, M., Steele, W.A., 1983. Non-additive energy effects in molecular liquids. Mol. Phys. 49, 893–898.

Mooij, G.C.A.M., de Leeuw, S.W., Smit, B., Williams, C.P., 1992. Molecular dynamics studies of polar/nonpolar fluid mixtures. II. Mixtures of Stockmayer and polarizable Lennard-Jones fluids. J. Chem. Phys. 97, 5113–5120.

Moradzadeh, A., Aluru, N.R., 2021. Understanding simple liquids through statistical and deep learning. J. Chem. Phys. 154, 204503.

Mueller, T., Hernandez, A., Wang, W., 2020. Machine learning for interatomic potential models. J. Chem. Phys. 152, 050902.

Mulero, A. (Ed.), 2008. Theory and Simulation of Hard-Sphere Fluids and Related Systems. Springer, Berlin.

Müller, E.A., Gelb, L.D., 2003. Molecular modeling of fluid-phase equilibria using an isotropic multipolar potential. Ind. Eng. Chem. Res. 42, 4123–4131.

Müller-Plathe, F., 1996. An all-atom force field for liquid ethanol - properties of ethanol-water mixtures. Mol. Sim. 18, 133–143.

Munn, R.J., 1964. Interaction potential of the inert gases. I. J. Chem. Phys. 40, 1439–1446.

Mutō, Y., 1943. On the forces acting between nonpolar molecules. J. Phys.-Math. Soc. Jpn. 17, 629–631. Available from: https://doi.org/10.11429/subutsukaishi1927.17.10-11-12_629 [In the literature, this paper is often incorrectly cited as Muto, Y., 1943. Proc. Phys. Math. Soc. Japan 17, 629.].

Nath, S.K., Escobedo, F.A., de Pablo, J.J., 1998a. On the simulation of vapor-liquid equilibria for alkanes. J. Chem. Phys. 108, 9905–9911.

Nath, S.K., Escobedo, F.A., de Pablo, J.J., Patramai, I., 1998b. Simulation of vapor-liquid equilibria for alkane mixtures. Ind. Eng. Chem. Res. 37, 3195–3902.

Némethy, G., Pottle, M.S., Scheraga, H.A., 1983. Energy parameters in polypeptides. 9. Updating of geometrical parameters, nonbonded interactions, and hydrogen bond interactions for the naturally occurring amino acids. J. Phys. Chem. 87, 1883–1887.

Némethy, G., Gibson, K.D., Palmer, K.A., Yoon, C.N., Paterlini, G., Zagari, A., Rumsey, S., Scheraga, H.A., 1992. Energy parameters in polypeptides. 10. Improved geometrical parameters and nonbonded interactions for use in the ECEPP/3 algorithm, with application to proline-containing peptides. J. Phys. Chem. 96, 6472–6484.

Newkome, G.R., Moorefield, C.N., Vögtle, 2001. Dendrimers and Dendroms: Concepts, Syntheses, Applications. Wiley-VCH, Weinheim.

Nicolas, Gubbins, K.E., Streett, W.B., Tildesley, D.J., 1979. Equation of state for Lennard-Jones molecules. Mol. Phys. 37, 1429–1454.

O'Shea, S.F., Dubey, G.S., Rasaiah, J.C., 1997. Phase transitions of quadrupolar fluids. J. Chem. Phys. 107, 237–242.

Okumura, H., Yonezawa, F., 2000. Liquid-vapor coexistence curves of several interatomic model potentials (2000) J. Chem. Phys. 113, 9162–9168.

Orea, P., Reyes-Mercado, Y., Duda, Y., 2008. Some universal trends of the Mie (n, m) fluid thermodynamics. Phys. Letts A 372, 7024.

Orkoulas, G., Panagiotopoulos, A.Z., 1994. Free energy and phase equilibria for the restricted primitive model of ionic fluids from Monte Carlo simulations. J. Chem. Phys. 101, 1452–1459.

Orkoulas, G., Panagiotopoulos, A.Z., 1999. Phase behavior of the restricted primitive model and square-well fluids from Monte Carlo simulations in the grand canonical ensemble. J. Chem. Phys. 110, 1581–1590.

Panagiotopoulos, A.Z., 1987. Direct determination of phase coexistence properties of fluids by Monte Carlo simulation in a new ensemble. Mol. Phys. 61, 813–826.

Pant, S., Smith, Z., Wang, Y., Tajkhorshid, E., Tiwary, P., 2020. Confronting pitfalls of AI-augmented molecular dynamics using statistical physics. J. Chem. Phys. 153, 234118.

Paricaud, P., 2006. A general perturbation approach for equation of state development: Applications to simple fluids, ab initio potentials, and fullerenes. J. Chem. Phys. 124, 154505.

Patel, B.H., Docherty, H., Varga, S., Galindo, A., Maitland, G.C., 2005. Generalized equation of state for square-well potentials of variable range. Mol. Phys. 103, 129–139.

Pedersen, U.R., Bacher, A.K., Schrøder, T.B., Dyre, J.C., 2019. The EXP par-potential system. III. Thermodynamic phase diagram. J. Chem. Phys. 150, 174501.

Perepu, P.K., Mishra, B.K., Pandu, A.M., 2023. Prediction of interaction energy for rare gas dimers using machine learning approaches. J. Chem. Sci. 135, 12.

Peyrovedin, H., Shariati, A., 2020. Polar hard-core exponential-6 intermolecular potential function for determining the thermodynamic properties of polar gases. Ind. Eng. Chem. Res. 59, 14106–14114.

Pieprzyk, S., Branka, A.C., Heyes, D.M., Bannerman, M.N., 2020. A comprehensive study of the thermal conductivity of the hard sphere fluid and solid by molecular dynamics simulation. Phys. Chem. Chem. Phys. 22, 8834–8845.

Pople, J.A., 1982. Intermolecular binding. Faraday Discuss. Chem. Soc. 73, 7–17.

Potoff, J.J., Bernard-Brunel, D.A., 2009. Potentials for phase equilibria calculations: applications to alkanes and perfluoroalkanes. J. Phys. Chem. B. 113, 14725–14731.

Potoff, J.J., Kamath, G., 2014. Mie potentials for phase equilibria: applications to alkenes. J. Chem. Eng. Data 59, 3144–3150.
Prestipino, S., Saija, F., Giaquinta, P.V., 2005. Phase diagram of the Gaussian-core model. Phys. Rev. E 71, 050102.
Prestipino, S., Speranza, C., Malescio, G., Giaquinta, P.V., 2014. Twofold reentrant melting in a double-Gaussian fluid. J. Chem. Phys. 140, 084906.
Price, S.L., 1988. Is the isotropic atom-atom model potential adequate? Mol. Sim. 1, 135–156.
Price, S.L., Stone, A.J., 1992. Electrostatic models for polypeptides: can we assume transferability? J. Chem. Soc. Faraday Trans. 88, 1755–1763.
Raabe, G., Sadus, R.J., 2007. Influence of bond flexibility on the vapor-liquid phase equilibria of water. J. Chem. Phys. 126, 044701 (2007).
Raabe, G., Sadus, R.J., 2011. Molecular dynamics simulation of the dieletric constant of water: The effect of bond flexibility. J. Chem. Phys. 134, 235501.
Raabe, G., Sadus, R.J., 2012. Molecular dynamics simulation of the effect of bond flexibility on the transport properties of water. J. Chem. Phys. 137, 104512.
Rahman, A., 1964. Correlations in the motion of atoms in liquid argon. Phys. Rev. 136, 405–411.
Ramrattan, N.S., Avendan, C., Müller, E.A., Galindo, A., 2015. A corresponding-states framework for the description of the Mie family of intermolecular potentials. Mol. Phys. 113, 932–947.
Rao, K.S., Goyal, P.S., Dasannacharya, B.A., Menon, S.V.G., Kelkar, V.K., Manohar, C., Mishra, B.K., 1991. Application of Baxter sticky hard-sphere model to nonionic micelles. Phys. B 174, 170–173.
Rappé, A.K., Casewit, C.J., 1997. Molecular Mechanics Across Chemistry. University Science Books, Sausalito.
Rappé, A.K., Goddard III, W.A., 1991. Charge equilibration for molecular dynamics simulation. J. Phys. Chem. 95, 3358–3363.
Rappé, A.K., Casewit, C.J., Colwell, K.S., Goddard III, W.A., Skiff, W.M., 1992. UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations. J. Am. Chem. Soc. 114, 10024–10035.
Reed, T.M., Gubbins, K.E., 1973. Applied Statistical Mechanics: Thermodynamic and Transport Properties of Fluids. Butterworth-Heinemann, Boston, p. 114.
Ren, P.Y., Ponder, J.W., 2003. Polarizable atomic multipole water for molecular mechanics simulation. J. Chem. Phys. B 117, 5933–5947.
Reyes, Y., Bárcentas, M., Odriozola, G., Orea, P., 2016. Thermodynamic properties of triangle-well fluids in two dimensions: MC and MD simulations. J. Chem. Phys. 145, 174505.
Rice, W.E., Hirschfeler, J.O., 1954. Second virial coefficients of gases obeying a modified Buckingham (Exp-Six) potential. J. Chem. Phys. 22, 187–192.
Rinker, S., 2018. Fixed-charge atomistic force fields for molecular dynamics simulations in the condensed phase: An overview. J. Chem. Inf. Model. 58, 565–578.
Rittger, E., 1990a. The chemical potential of liquid xenon by computer simulation. Mol. Phys. 69, 853–865.
Rittger, E., 1990b. Can three-atom potentials be determined from thermodynamic data? Mol. Phys. 69, 867–894.
Rittger, E., 1990c. An empirical three-atom potential for xenon. Mol. Phys. 71, 79–96.
Rosenfeld, Y., 1993. Free energy model for inhomogeneous fluid mixtures: Yukawa-charged hard spheres, general interactions, and plasmas. J. Chem. Phys. 98, 8126–8148.
Rosenfeld, Y., 1996. Ewald method for simulating Yukawa systems. Mol. Phys. 88, 1357–1363.

Roterman, I.K., Gibson, K.D., Scheraga, H.A., 1989a. A comparison of the CHARMM, AMBER and ECEPP potentials for peptides. I. Conformational predictions for the tandemly repeated peptide (Asn-Ala-Asn-Pro)$_9$. J. Biomol. Struct. Dyn. 7, 391–419.

Roterman, I.K., Lambert, M.H., Gibson, K.D., Scheraga, H.A., 1989b. A comparison of the CHARMM, AMBER and ECEPP potentials for peptides. II. ϕ-ψ maps for N-acetyl Alanine N'-methyl amide: comparisons, contrasts and simple experimental tests. J. Biomol. Struct. Dyn. 7, 421–453.

Rowlinson, J.S., 1989. The Yukawa potential. Phys. A 156, 15–34.

Rowlinson, J.S., 2002. Cohesion: A Scientific History of Intermolecular Forces. Cambridge University Press, Cambridge.

Rudisill, E.N., Cummings, P.T., 1989. Gibbs ensemble simulation of phase equilibrium in the hard core two-Yukawa fluid model for the Lennard-Jones fluid. Mol. Phys. 68, 629–635.

Sadus, R.J., 1992. High Pressure Phase Behaviour of Multicomponent Fluid Mixtures. Elsevier, Amsterdam.

Sadus, R.J., 1996a. Molecular simulation of the vapour-liquid equilibria of pure fluids and binary mixtures containing dipolar components: the effect of Keesom interactions. Mol. Phys. 87, 979–990.

Sadus, R.J., 1996b. Molecular simulation of the liquid-liquid equilibria of binary mixtures containing non-polar and dipolar components interacting via the Keesom potential. Mol. Phys. 89, 1187–1194.

Sadus, R.J., 2018. Second virial coefficient properties of the n-m Lennard-Jones/Mie potential. J. Chem. Phys. 149, 074504. Erratum: *J. Chem. Phys*. 149, 079902 (2019).

Sadus, R.J., 2019. Two-body intermolecular potentials from second virial coefficient properties. J. Chem. Phys. 150, 024503. Erratum: *J. Chem. Phys*. 150, 024503 (2019).

Sadus, R.J., 2020a. Vapor-liquid equilibria and cohesive r^{-4} interactions. J. Chem. Phys. 153, 204504.

Sadus, R.J., 2020b. Effect of the range of particle cohesion on the phase behavior and thermodynamic properties of fluids. J. Chem. Phys. 153, 244502.

Sadus, R.J., Prausnitz, J.M., 1996. Three-body interactions in fluids from molecular simulation: vapor-liquid phase coexistence of argon. J. Chem. Phys. 104, 4784–4787.

Saija, F., Prestipino, S., 2005. High-pressure phase diagram of the exp-6 model: the case of Xe. Phys. Rev. B 72, 024113.

Saija, F., Malescio, G., Prestipino, S., 2010. Re-entrant melting of the exp-6 fluid: The role of repulsion softness. Phys. Chem. Liq. 48, 477–487.

Santos, A., Fanton, R., Giacometti, 2008. Penetrable square-well fluids: Exact results in one dimension. Phys. Rev. E. 77, 051206.

Seng, X., Toennies, J. P., Tang, K. T., 2020. Conformal potential for all the rare gas dimers over the full range of internuclear distances. Phys. Rev. Lett. 125, 253402.

Schöll-Paschinger, E., Valdez-Pérez, N.E., Benavides, A.L., Castañedo-Priego, R., 2013. Phase behavior of the modified-Yukawa fluid and its sticky limit. J. Chem. Phys. 139, 184902.

Schouten, J.A., de Kuijper, A., Michels, J.P.J., 1991. The critical line of He-H_2 up to 2500 Kelvin and the influence of attraction on fluid-fluid separation. Phys. Rev. B: Condens. Matter 44, 6630–6634.

Scott, W.R.P., Hünenberger, P.H., Tironi, I.G., Mark, A.E., Billeter, S.R., Fennen, J., Torda, A.E., Huber, T., Krüger, P., van Gunsteren, W.F., 1999. The GROMOS biomolecular simulation program package. J. Phys. Chem. A 103, 3596–3607.

Sellers, M.S., Lisal, M., Brennan, J.K., 2015. Mol. Phys. 113, 45–54.

Shah, M.S., Siepmann, J.I., Tsapatsis, M., 2017. Transferable potentials for phase equilibria: improved united-atom description of ethane and ethylene. AIChE J. 63, 5098–5109.

Shannon, R.J., Hornung, B., Tew, D.P., Glowacki, D.R., 2019. Anharmonic molecular mechanics: ab initio based Morse parametrizations for the popular MM3 force field. J. Phys. Chem. A 123, 2991−2999.

Shavitt, I., Boys, S.F., 1956. A General expression for intermolecular potentials. Nature 178, 1340.

Sherwood, A.E., Mason, E.A., 1965. Virial coefficients for the exponential repulsive potential. Phys. Fluids 8, 1577−1579.

Sherwood, A.E., Prausnitz, J.M., 1964. Third virial coefficient for the Kihara, exp-6, and square-well potentials. J. Chem. Phys. 41, 413−428.

Shukla, K.P., 2000. Phase equilibria and thermodynamic properties of hard core Yukawa fluids of variable range from simulations and an analytical theory. J. Chem. Phys. 112, 10358−10367.

Shvab, I., Sadus, R.J., 2016. Atomistic water models: aqueous thermodynamic properties from ambient to supercritical conditions. Fluid Phase Equilib. 407, 7−30.

Siepmann, J.I., Karaborni, S., Smit, B., 1993. Simulating the critical behaviour of complex fluids. Nature 365, 330−332.

Smit, B., Frenkel, D., 1991. Vapour-liquid equilibria of the hard core Yukawa fluid. Mol. Phys. 74, 35−39.

Smit, B., Williams, C.P., 1990. Vapour-liquid equilibria for quadrupolar Lennard-Jones fluids. J. Phys.: Condens. Matter 2, 4281−4288.

Smit, B., Williams, C.P., Henriks, E.M., de Leeuw, S.W., 1989. Vapour-liquid equilibria for Stockmayer fluids. Mol. Phys. 68, 765−769.

Smith, D.E., Haymet, A.D.J., 1993. Simulations of aqueous solutions: the role of flexibility and the treatment of long-range forces. Fluid Phase Equilib. 88, 79−87.

Smith, J.C., Karplus, M., 1992. Empirical force field study of geometries and conformational transitions of some organic molecules. J. Am. Chem. Soc. 114, 801−812.

Spear, R.P., 2000. Fluid-fluid transition of hard spheres with very-short-range attraction. Phys., Rev. E 61, 6019−6022.

Speranza, C., Prestipino, S., Malescio, G., Giaquinta, P.V., 2014. Phase behavior of a fluid with a double Gaussian potential displaying waterlike features. Phys. Rev. E. 90, 012305.

Spurling, T.H., Mason, E.A., 1967. Determination of molecular quadrupole moments from viscosities and second virial coefficients. J. Chem. Phys. 46, 322−326.

Stapleton, M.R., Tildesley, D.J., Panagiotopoulos, A.Z., Quirke, N., 1989. Phase equilibria of quadrupolar fluids by simulation in the Gibbs ensemble. Mol. Sim. 2, 147−162.

Stell, G., 1991. Sticky spheres and related systems. J. Stat. Phys. 63, 1203−1221.

Stiegler, T., Sadus, R.J., 2015. Molecular simulation of fluids with non-identical intermolecular potentials: thermodynamic properties of 10-5 + 12-6 Mie potential binary mixtures. J. Chem. Phys. 142, 084504.

Stillinger, F.H., 1976. Phase transitions in the Gaussian core system. J. Chem. Phys. 65, 3968−3974.

Stone, A.J., 2013. The Theory of Intermolecular Forces, *2nd Ed* Oxford University Press, Oxford.

Strauch, H.J., Cummings, P.T., 1993. Gibbs ensemble simulation of mixed solvent electrolyte solutions. Fluid Phase Equilib. 86, 147−172.

Sun, L., Deng, W.-Q., 2017. Recent developments of first-principles force fields. WIREs Comput. Mol. Sci. 7, 1−15.

Tang, K.T., Toennies, J.P., 1984. An improved simple model for the van der Waals potential based on universal damping functions for the dispersion coefficients. J. Chem. Phys. 80, 3726−3741.

Tavares, F.W., Sandler, S.I., 1996. Vapour-liquid equilibria of exponential-six fluids. Mol. Phys. 87, 1471–1476.

Tee, S.L., Gotoh, S., Stewart, W.E., 1966. Molecular parameters for normal fluids: The Kihara potential with spherical core. Ind. Eng. Chem. Fundam. 5, 363–366.

Tiesinga, E., Mohr, P.J., Newell, D.B., Taylor, B.N., 2021. CODATA recommended values of the fundamental physical constants: 2018. Rev. Mod. Phys. 93, 025010. Available from: https://physics.nist.gov/cuu/Constants/index.html.

Travis, K.P. and Sadus, R.J. (2023), to be submitted.

Valdez-Pérez, N.E., Benavides, A.L., Schöll-Paschinger, E., Castañedo-Priego, R., 2012. Phase behavior of colloids and proteins in aqueous suspensions: theory and computer simulations. J. Chem. Phys. 137, 084905.

van Gunsteren, W.F., Bakowies, D., Baron, R., Chandrasekhar, I., Christen, M., Daura, X., Gee, P., Geerke, D.P., Glättli, A., Hünenberger, H., Kastenholz, M.A., Oostenbrink, C., Schenk, M., Trzesniak, D., va der Vegt, N.F.A., Yu, H.B., 2006. Biomolecular modeling: Goals, problems, perspectives. Angew. Chem. Int. Ed. 45, 4064–4092.

van Leeuwen, M.E., Smit, B., 1993. What makes a polar liquid a liquid? Phys. Rev. Lett. 71, 3991–3994.

van Leeuwen, M.E., Smit, B., Hendriks, E.M., 1993. Vapour-liquid equilibria of Stockmayer fluids. Computer simulations and perturbation theory. Mol. Phys. 78, 271–283.

van Vleet, M.J., Misquitta, A.J., Stone, A.J., Schmidt, J.R., 2016. Beyond Born-Mayer: Improved models for short-range repulsion in ab initio force fields. J. Chem. Theory Comput. 12, 3851–3870.

Vanommeslaeghe, K., MacKerell Jr, A., D., 2015. CHARMM additive and polarizable force fields for biophysics and computer-aided drug design. Biochim. Biophys. Acta 1850, 861–871.

Vedani, A., 1988. YETI: An interactive molecular mechanics program for small-molecular protein complexes. J. Comput. Chem. 9, 269–280.

Vega, L., de Miguel, E., Rull, L.F., Jackson, G., McLure, I.A., 1992a. Phase equilibria and critical behavior of square-well fluids of variable width by Gibbs ensemble Monte Carlo Simulation. J. Chem. Phys. 96, 2296–2305.

Vega, C., Lago, S., de Miguel, E., Rull, L.F., 1992b. Liquid-vapor equilibria of linear Kihara molecules. J. Phys. Chem. 96, 7431–7437.

Vega, C., Abascal, J.L.F., Conde, M.M., Aragones, J.L., 2009. What ice can teach us about water interactions: A critical comparison of the performance of different water models. Farad. Discuss. 141, 251–276.

Wang, L.-P., Head-Gordon, T., Ponder, J.W., Ren, P., Chodera, J.D., Eastman, P.K., Martinez, T.J., Pande, V.S., 2013. Systematic improvement of a classical molecular model of water. J. Phys. Chem. B 117, 9956–9972.

Wang, X., Ramirez-Hinestrosa, S., Dobnikar, J., Frenkel, D., 2020. The Lennard-Jones potential: When (not) to use it. Phys. Chem. Chem. Phys. 22, 10524–10633.

Warner Jr., H.R., 1972. Kinetic theory and rheology of dilute suspensions of finitely extendible dumbbells. Ind. Eng. Chem. Fundam. 11, 379–387.

Weeks, J.D., Chandler, D., Andersen, H.C., 1971. Role of repulsive forces in determining the equilibrium structure of simple liquids. J. Chem. Phys. 54, 5237–5247.

Wei, Y.S., Sadus, R.J., 2000. Equations of state for the calculation of fluid-phase equilibria. AIChE J. 46, 169–196.

Weiner, S.J., Kollman, P.A., Case, D.A., Singh, U.C., Chio, C., Alagona, G., Profeta Jr, S., Weiner, P., 1984. A new force field for molecular mechanical simulation of nucleic acids and proteins. J. Am. Chem. Soc. 106, 765–784.

Wheatley, R.J., Meath, W.J., 1993. Dispersion energy damping functions, and their relative scale with interatomic separation, for (H, He, Li)-(H, He, Li) interactions. Mol. Phys. 79, 253–275.

Wood, W.W., Jacobson, J.D., 1957. Preliminary results from a recalculation of the Monte Carlo equation of state for hard spheres. J. Chem. Phys. 27, 1207.

Wood, W.W., Parker, F.R., 1957. Monte Carlo equation of state of molecules interacting with the Lennard-Jones potential. I. A supercritical isotherm at about twice the critical temperature. J. Chem. Phys. 27, 720–733.

Wu, G.-W., Sadus, R.J., 2000. Molecular simulation of the high pressure phase equilibria of binary atomic fluid mixtures using the exponential-6 intermolecular potential. Fluid Phase Equilib. 170, 269–284.

Wu, G.-W., Sadus, R.J., 2004. A new phase for one-component hard spheres. J. Chem. Phys. 120, 11686–11691.

Wu, G.-W., Sadus, R.J., 2005. Hard sphere compressibility factors for equation of state development. AIChE J. 51, 309–313.

Wu, Y., Tepper, H., Voth, G., 2006. Flexible simple point-charge water model with improved liquid-state properties. J. Chem. Phys. 124, 024503.

Yuste, S.B., Santos, A., 1994. A model for the structure of square-well fluids. J. Chem. Phys. 101, 2355–2364.

Zarragoicoechea, G.J., Scalise, O.H., 1997. On the gas-gas equilibria of second kind of nonpolar fluid binary mixtures from a hard-sphere exp-6 molecular model. J. Chem. Phys. 107, 4258–4363.

Chapter 4

Ab initio, two-body and three-body intermolecular potentials

In Chapter 3 details of effective pair potentials and force fields that are used in molecular simulation to evaluate the contribution to energy via Eq. (3.1) were given. In this chapter, we examine both two- and three-body intermolecular potentials used to calculate energy via Eq. (3.2). These potentials can be obtained from empirical, theory, and ab initio sources.

It is common to assume that intermolecular interactions are pairwise additive. Two-body interactions dominate other multi-body interaction and pairwise potentials alone are often sufficiently accurate. However, from a theoretical perspective, it is well known that some aspects of intermolecular interactions cannot be exclusively pairwise additive. Strictly, pairwise additivity only applies to electrostatic, magnetic, and short-range penetration interactions. To some extent, many-body effects will be present in induction, dispersion, resonance, exchange, repulsion, and charge transfer interactions. In particular, many-body effects are likely to exert a strong influence on induction energy. For example, Wilson and Madden (1994) have reported that many-body induction effects have a crucial role in stabilizing the crystal structures of $CdCl_2$ and $MgCl_2$.

Despite the evidence for deviations from pairwise additivity, molecular simulations of liquids have been confined almost exclusively to pairwise potentials. However, with the increasing sophistication of both experimental and theoretical techniques, it is likely that many-body interactions will be required for the quantitative description of liquids. This is particularly the case for large molecules and polar molecules, which are likely to be influenced substantially by many-body interactions. The increasing importance of many-body interactions with molecular size and complexity is illustrated in Fig. 4.1, which shows that the nonadditive dispersion coefficient (v) increases progressively with molecular size relative to the two-body dispersion coefficient (C_6). It has been reported (Barker et al., 1974; Rittger, 1990c) that calculating three-body interactions is required to obtain satisfactory agreement with experiment for the properties of xenon. In a dipolar system, the effect of three-body interactions can be expected to be considerably greater because of additional nonadditive effects from induction.

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00009-5

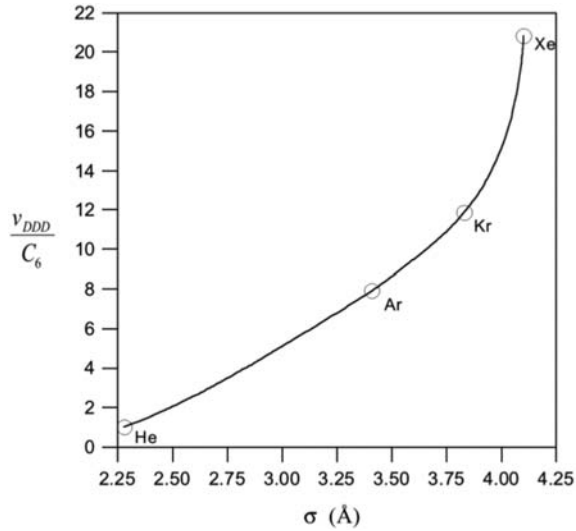


FIGURE 4.1 Relative importance of the nonadditive triple-dipole dispersion coefficient with respect to molecular size. Values are given for helium, argon, krypton, and xenon.

Advances in computational chemistry have witnessed the development of intermolecular potentials from first principles or ab initio data. Ab initio potentials can be accurate alternatives to empirically derived potentials. As discussed below, ab initio calculations can be used to determine the interaction between atoms from first principles. The information obtained from ab initio calculations can be fitted to obtain an ab initio potential that can be employed in both Monte Carlo (MC) and molecular dynamics (MD) simulations.

The current state-of-the-art for ab initio potentials is largely confined to the two-body properties of simple atoms or small polyatomic molecules. Ab initio potentials for large molecules typically include ab initio data for part of the molecule's properties (Sun and Deng, 2017), such as bond-stretching terms, rather than being fully ab initio potentials. Therefore, it is useful to make the distinction between fully ab initio potentials and potentials informed or augmented by ab initio data. Obtaining ab initio potentials for three-body interactions is more challenging than the two-body case. Consequently, most three-body potentials are obtained from conventional theory or empirical considerations. A useful strategy is to augment ab initio two-body potentials with three-body potentials. The combination facilitates the prediction of the properties of real systems.

4.1 Ab initio calculations

The objective of ab initio methods is to calculate molecular properties from the Schrödinger equation. Typically, the Born–Oppenheimer approximation is used,

which assumes that the movement of the nuclei and electron are independent of each other. The electronic energy is obtained by solving the Schrödinger equation for the electrons. This results in a wave function, which is used to calculate properties such as the molecular multipole moments. The electronic energy as a function of nuclear coordinates generates a potential energy surface, which is the potential energy term in the Schrödinger equation. The other properties of the molecule are functions of the potential energy.

Approximations are required to solve the Schrödinger equation. The most common methods used are systematic corrections to the Hartree−Fock (HF) or self-consistent field (SCF) models and density functional theory (DFT). Sandré and Pasturel (1997) have reviewed the use of ab initio calculations in the context of molecular dynamics. The description of the ab initio approach given here is limited to the key underlying principles because the topic has been discussed in detail elsewhere (Scheiner, 1997; Marx and Hutter, 2009).

The current state-of-the-art for ab initio potentials is to determine the potential energy curve using a supermolecular calculation up to the coupled cluster (CC) with single, double, and perturbative triple excitation [CCSD (T)] contributions (Raghavachari et al., 1989; Stone, 2013). In a few cases (Hellmann et al., 2021) this can be taken to up to the triple and quadruple level with pentuple excitations [CCSDTQ(P)]. Here, we will mainly focus on contemporary potentials, which have benefited from the latest advances in computational chemistry.

4.1.1 Hartree−Fock method

The HF approach commonly involves expanding molecular orbitals in a contracted Gaussian set and evaluating integrals over the contracted Gaussians. The general form of the integrals is:

$$(ij\backslash kl) = \iint \chi_i(1)^* \chi_j(1) \frac{1}{r_{12}} \chi_k(1)^* \chi_l(1) d\tau_1 d\tau_2 \tag{4.1}$$

To solve Eq. (4.1), we must first determine the basis functions (χ_I). The most-widely used basis functions are

$$\chi = R_{lk}(r) \sum_i c_i N_i \exp(-\alpha_i r^2), \tag{4.2}$$

where R_{lk} is a solid harmonic, $r \equiv (r,\theta,\varphi)$ is the position of the electron relative to the nucleus, $N_i\exp(-\alpha_i r^2)$ is a primitive Gaussian function, N_i is a normalizing factor, and c_i is an expansion coefficient, which optimizes the energy of the isolated atom. Generally, there are approximately $N^4/8$ two-electron integrals for a calculation involving N basis functions. It is common to solve these integrals using software packages such as GAUSSIAN.[1]

1. https://gaussian.com

The basic HF method cannot be used to calculate the intermolecular energy because it does not consider the effect of electron correlation. However, it can be extended (Friesner, 1991) to include electron correlation effects. Some examples include perturbation expansion (Møller and Plesset, 1934); optimized multiconfigurational wave function methods (MCSCF) (Shepard, 1987); valence bond (VB) methods (Goddard and Harding, 1978); CC, and configuration interaction (CI) methods (Bartlett, 1981). These additional refinements are at the expense of a considerable increase in computational cost. If N_b denotes the number of basis sets, the HF electron correlation algorithms are typically of order N_b^5 to N_b^8. Consequently, applications of the HF method are confined generally to relatively small molecules. The HF plus dispersion (HFD) method is an alternative to a full ab initio calculation. The HFD approach combines an ab initio calculation with an empirical expression for dispersion. The quality of both the HF and post-HF predictions depends on the basis set. Currently, the best basis sets are augmented correlation-consistent sets such as aug-cc-$pVnZ$ ($n = D,\ T,\ Q,\ 5$), which allow an extrapolation to infinitely large basis sets.

4.1.2 Density functional theory

DFT is an alternative to the HF method. Instead of calculating the wave functions, DFT evaluates the electron density profile ($n(r)$). The DFT method is based on the work of Hohenberg and Kohn (1964), which postulated that the ground-state properties of a system are functions of charge density. The total electronic energy is a function of the electron density (ρ),

$$E(\rho) = E_{KE}(\rho) + E_C(\rho) + E_H(\rho) + E_{XC}(\rho), \tag{4.3}$$

where E_{KE}, E_C, E_H, and E_{XC} are the contributions from kinetic energy, electron−nuclear interaction, electron−electron Coulombic energy, and exchange and correlation, respectively. It is apparent from Eq. (4.3) that the density determines the ground-state properties of the system.

The general procedure for a density functional calculation is to express the various contributions to Eq. (4.3) in terms of the density and attempt to optimize the energy with respect to density in accordance with any constraints on the system. Eq. (4.3) can be written as,

$$E(\rho) = 2\sum \int \psi_i \left(-\frac{\nabla^2}{2}\right)\psi_i dv + \int V_{nuclear}\rho(\mathbf{r})dv + \frac{1}{2}\iint dv dv' \frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} + E_{XC}[\rho(\mathbf{r})], \tag{4.4}$$

where ψ is the spatial orbital and $\rho(r)$ the charge density at a point $\mathbf{r}$. The charge density is the sum over occupied molecular orbitals of ψ^2.

$$\rho(\mathbf{r}) = \sum_{i=1}^{N_{occupied}} \left|\psi_i(\mathbf{r})\right|^2 \tag{4.5}$$

To complete the evaluation of Eq. (4.5), an approximation for the exchange-correlation term is required. Typically, a local density approximation is used, which assumes that the charge density varies slowly throughout a molecule. Therefore, a localized region of a molecule behaves like a uniform electron gas and

$$E_{XC}[\rho(\mathbf{r})] \cong \int \rho(\mathbf{r})\varepsilon_{XC}[\rho(\mathbf{r})]d\mathbf{r}, \tag{4.6}$$

where ε_{XC} is the exchange-correlation energy per particle in a uniform electron gas. The exchange energy of a uniform electron gas can be obtained from a variety of analytical expressions. For example, Kohn and Vashista (1983) proposed be following simple relationship:

$$\varepsilon_{XC}(\rho(\mathbf{r})) = -0.78558770[\rho(\mathbf{r})]^{1/3}. \tag{4.7}$$

Kohn and Sham (1965) showed that E could be found from the solution of single-particle equations of the form:

$$\left[-\frac{\nabla^2}{2} + V_{nuclear}(\mathbf{r}) + \int \mathrm{d}v' \frac{\rho(\mathbf{r})}{|\mathbf{r}\text{-}\mathbf{r}'|} + V_{XC}(\mathbf{r})\right]\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r}), \tag{4.8}$$

where the exchange-correlation function V_{XC} is obtained from:

$$V_{XC}(\mathbf{r}) = \frac{\mathrm{d}E_{XC}[\rho(\mathbf{r})]}{\mathrm{d}\rho(\mathbf{r})}. \tag{4.9}$$

The most important advantages of the DFT approach compared with HF-based methods are that electron correlations are included from the start and that it is considerably less expensive computationally. A DFT calculation scales typically as $N_b{}^3$ compared with $N_b{}^5$ to $N_b{}^7$ for HF calculations. The computational cost is not affected greatly by replacing the local density approximation with more accurate alternatives. However, unlike HF methods, DFT cannot be improved systematically. The HF methods are also more accurate than DFT calculations for small molecules.

DFT has been used with mixed success to obtain intermolecular potentials. Good results have been reported for water (Xantheas, 1995). However, a comparison of DFT results with experiment for hydrogen bonded complex indicated generally poor agreement (del Berne et al., 1995). Poor results have also been reported (Pérez-Jordá and Becke, 1995) for the application of DFT for some inert gases. A weakness of conventional DFT is that current density functionals do not describe intermolecular interaction energy as decaying according to r^{-6} at large separations. Further, the errors associated with numerical quadratures are similar in magnitude to the intermolecular

interaction energies. Therefore, the prediction of dispersion interaction often fails; however, some progress in improving this situation has been reported (Johnson et al., 2009).

It should be noted that the earlier issues are not fundamental limitations of the DFT method, and they may be overcome in the future. It is an area of ongoing research interest and several comprehensive reviews of DFT are available (Parr and Yang, 1990; Trickey, 1990; Friesner, 1991; March, 1992; Orio et al., 2009; Cohen et al., 2012; Kryanko and Ludeña, 2014; Bretonnet, 2017; Verma and Truhlar, 2020).

4.1.3 The Car–Parrinello method

The accuracy of conventional simulations is limited to the accuracy of the intermolecular potential used. Car and Parrinello (1985) have combined molecular dynamics with some elements of electronic-structure theory. The Car–Parrinello molecular dynamics (CPMD) method provides an alternative to the conventional development of intermolecular potentials by determining interactions "on-the-fly." The main element of their approach is the formulation of a new Lagrangian (Chapter 2) function from which the equations of motion are derived for the nuclear coordinates and electronic orbitals. The Lagrangian function is

$$\begin{aligned} L = &\sum_i \frac{1}{2}\mu \int_\Omega d^3 r \left|\dot{\psi}_i\right|^2 + \sum_I \frac{1}{2} M_I \dot{R}_I^2 + \sum_\nu \frac{1}{2}\mu_\nu \dot{\alpha}_\nu^2 \\ &- E\left[\{\psi_i\}, \{R_I\}, \{\alpha_\nu\}\right] \end{aligned}, \tag{4.10}$$

where the orbitals ψ_i are subject to holonomic constraints.

$$\int_\Omega d^3 r \psi_i^*(r,t)\psi_j(r,t) = \delta_{ij} \tag{4.11}$$

In Eq. (4.10), E is the energy; $\{R_I\}$ are nuclear coordinates; $\{\alpha_\nu\}$ are all the possible external constraints such as volume Ω; M_I are the physical ionic masses; and μ and μ_ν are arbitrary parameters and the dot indicates a time derivative.

The definition of the Lagrangian function generates a dynamics for the $\{\psi_i\}$'s, $\{R_I\}$'s, and $\{\alpha_\nu\}$'s through the equations of motion

$$\left.\begin{aligned} \mu\ddot{\psi}_i(\mathrm{r},t) &= -\frac{\delta E}{\delta \psi_i^*(\mathrm{r},t)} + \sum_k \Lambda_{ik}\psi_k(\mathrm{r},t) \\ M_I\ddot{R}_I &= -\nabla_{R_I} E \\ \mu_\nu \ddot{\alpha}_\nu &= -\frac{\partial E}{\partial \alpha_\nu} \end{aligned}\right\}, \tag{4.12}$$

where Λ_{ik} are the Lagrange multipliers (Chapter 2) introduced to satisfy the constraints in Eq. (4.10). The ion dynamics in Eq. (4.10) have a physical

meaning, whereas the dynamics associated with the $\{\psi_i\}$'s and $\{\alpha_v\}$'s are fictitious. From Eq. (4.10), the kinetic energy E_{kin} is:

$$E_{kin} = \sum_i \frac{1}{2}\mu \int_\Omega d^3r \left|\dot{\psi}_i\right|^2 + \sum_I \frac{1}{2} M_I \dot{R}_I^2 + \sum_v \frac{1}{2}\mu_v \dot{\alpha}_I^2. \tag{4.13}$$

In principle, these equations can be used for ab initio molecular dynamics simulation for which the intermolecular potential is not specified in advance.

Variations on the CPMD method have been discussed by Payne et al. (1990, 1992) and Marx and Hutter (2009). Tuckerman and Parrinello (1994a,b) have developed strategies for integrating the equations of motion in Car−Parrinello simulations. The technique has also been applied (Tuckerman et al., 1996) to path integral molecular dynamics. Woo et al. (1997) have reported an interesting combination of the CPMD method with quantum mechanics and molecular mechanics principles. CPMD has been applied to a variety of situations including material investigation (Sahoo and Nair, 2016), biomolecular chemical reactions (Hofbauer and Frank, 2012), and water (Frank, 2020).

4.2 Two-body atomic potentials

In Chapter 3, we emphasized the distinction between effective pairwise potentials and two-body potentials. The square well, Lennard-Jones (LJ), and exp-6 potentials are generic potentials, which can be applied to a wide variety of fluids with a varying degree of accuracy. In contrast, considerable effort has been made in developing highly accurate intermolecular potentials that are specific to the two-body interactions of particular atom or class of atoms. The state-of-the-art is to obtain two-body interactions form ab initio data; however, reasonably accurate two-body potentials have been obtained via the careful study of experimental two-body data such as the second virial coefficients.

4.2.1 Empirical potentials

The noble gases have been the focus of the work to obtain two-body potentials from the analysis of experimental data. The resulting potentials have been compared extensively (Klein, 1984; Klein and Venables, 1976; Rowlinson, 2002). Many of the empirical two-body potentials have the following general form:

$$u_2(r) = \sum_{i=1}^{n} f_i(r)\exp(g(r)) - \sum_{j=o}^{m} \frac{h(r)C_{2j+6}}{k + r^{2j+6}}, \tag{4.14}$$

where $f(r)$, $g(r)$, and $h(r)$ are some functions of intermolecular separation (r) and k is a constant, which is often 0. Eq. (4.14) combines an exponential contribution from repulsion with attractive terms from the dispersion coefficients (C_{2j+6}). The nature of Eq. (4.14) has been informed by the long history of developing effective pairwise potentials as discussed in Chapter 3.

Several workers (Bobetic and Barker, 1970; Barker et al., 1971; Maitland and Smith, 1971) have assessed experimental data to obtain accurate potentials for argon. The potentials of Barker and Pompe (1968) and Bobetic and Barker (1970) have the following form:

$$u_2(r) = \varepsilon \left[\sum_{i=0}^{5} A_i (x-1)^i \exp[\alpha(1-x)] - \sum_{j=0}^{2} \frac{C_{2j+6}}{\delta + x^{2j+6}} \right]. \tag{4.15}$$

In Eq. (4.15), ε is the depth of the potential and the minimum separation (r_m), $x = r/r_m$ and the other parameters are obtained by fitting the potential to experimental data for molecular beam scattering, second virial coefficients, and long-range interaction coefficients. The contribution from repulsion has an exponential dependence on intermolecular separation and the contribution to dispersion of the C_6, C_8, and C_{10} coefficients are included. Comparing Eq. (4.15) to Eq. (4.14) yields:

$$\left.\begin{array}{c} f_i(r) = \varepsilon A_i (x-1)^i \\ g(r) = \alpha(1-x) \\ h(r) = \varepsilon \\ k = \delta \end{array}\right\}. \tag{4.16}$$

The only difference between the Barker–Pompe and Bobetic–Barker potentials is that a different set of parameters is used in each case. Later, Barker et al. (1971) proposed a linear combination of the two potentials, which is known commonly as the Barker–Fisher–Watts (BFW) potential. The BFW potential provides an accurate representation of the two-body interactions of argon.

The molecule-specific nature of the intermolecular potential is illustrated by attempts to use Eq. (4.15) for other noble gases such as krypton and xenon. Barker et al. (1974) reported that modifications to Eq. (4.15) were required to obtain an optimal representation for these larger noble gases. They determined a potential of the form $u(r) = u_0(r) + u_1(r)$, where $u_0(r)$ is identical to Eq. (4.15) and $u_1(r)$ is given by:

$$u_1(r) = \begin{cases} \left[P(x-1)^4 + Q(x-1)^5\right] \exp[\alpha'(1-x)] & x > 1 \\ 0 & x \le 1 \end{cases} \tag{4.17}$$

and α', P, and Q are additional parameters obtained by fitting data for differential scattering cross sections.

Parson et al. (1972) developed a potential based largely on molecular beam data.

$$u_2(r) = \begin{cases} \varepsilon \exp[-2\beta(x-1)] - 2\varepsilon \exp\left[-\beta'(x-1)\right] & 0 \le x \le x_1 \\ \varepsilon b_1 + \varepsilon(x-x_1)[b_2 + (x-x_2)][b_3 + (x-x_1)b_4] & x_1 < x \le x_2 \\ \varepsilon\left(C_6 x^{-6} + C_8 x^{-8} + C_{10} x^{-10}\right) & x_2 < x \le \infty \end{cases} \tag{4.18}$$

The potential provides a good description of interatomic interactions in argon.

The potential of Aziz and Slaman (1986) gives a particularly accurate description of the thermodynamic properties of argon. The Aziz−Salman potential is

$$u_2(r) = \varepsilon A^* \exp(-\alpha^* x + \beta^* x^2) - \varepsilon F(x) \sum_{j=0}^{2} \frac{c_{2j+6}}{x^{2j+6}}, \tag{4.19}$$

where

$$F(x) = \begin{cases} \exp\left[-\left(\frac{D}{x} - 1\right)^2\right] & x < D \\ 1 & x \geq D \end{cases} \tag{4.20}$$

$x = r/r_m$, and the other terms are adjustable parameters. In common with the Born−Mayer potential (1932) (Eq. (3.33)), the Aziz−Salman potential uses an exponential distance dependence for repulsion, and it includes all the dispersion coefficients. The contribution of the dispersion term is regulated by a damping term (Eq. (4.20)). It is evident that the Aziz-Salman potential also conforms to the general form given by Eq. (4.14). An accurate potential has been proposed for helium (Tang et al., 1995) that uses elements of this approach.

Although the empirical two-body potentials have arguably been superseded (see discussion below) by ab initio potentials, they remain useful for contemporary molecular simulation applications. For example, the BFW potential has been applied to vapor−liquid equilibria (VLE) (Marcelli and Sadus, 1999; Sadus, 2019), solid−liquid equilibria (SLE) (Wang and Sadus, 2006), and thermodynamic properties (Vlasiuk and Sadus, 2017b).

4.2.2 Ab initio potentials

Early ab initio potentials have been reviewed elsewhere (Klein, 1984; Rowlinson, 2002). Woon (1993) used fourth-order Møller−Plesset (MP4) ab initio calculations to determine the intermolecular forces between two argon atoms. McLean et al. (1988) also reported ab initio calculations for the interaction between argon atoms. Woon (1993) fitted ab initio results to the following Morse-type potential

$$u(r) = \varepsilon Q \exp\left[-\sqrt{\frac{(Q+1)k_e}{Q\varepsilon}}(r - r_e)\right] - \varepsilon(Q+1)\exp\left[-\sqrt{\frac{Qk_e}{(Q+1)\varepsilon}}(r - r_e)\right], \tag{4.21}$$

where ε the depth of the potential well, r_e is the equilibrium separation, k_e is the force constant, and Q is a weighting constant. Ermakova et al. (1995) used Eq. (4.21) in a molecular dynamics study of liquid argon. They concluded that the potential predicted many thermodynamic and transport

properties accurately but many-body interactions were required to improve the predictions of pressure, internal energy, and enthalpy.

Eggenberger et al. (1994) used ab initio calculations to obtain the following potential for the interactions between neon atoms:

$$u(r) = a_1\exp[-a_2(r/a_0)^2] + a_3\exp[-a_4(r/a_0)^2] + a_5\exp[-a_6(r/a_0)^2] + a_7(r/a_0)^{-10} + a_8(r/a_0)^{-8} + a_7(r/a_0)^{-6}, \quad (4.22)$$

where a_0 is the Bohr radius and the remaining parameters do not have any physical meaning. Eggenberger et al. (1994) demonstrated that Eq. (4.22) could be used to predict many of the properties of neon with reasonable accuracy. It is interesting to compare the functional similarity of Eq. (4.22) with accurate empirical potentials such as those proposed by Barker et al. (Eq. (4.15)) and Aziz and Salman (Eq. (4.19)). All of these potentials have an exponential term and contributions from r^{-6}, r^{-8}, and r^{-10} intermolecular separations.

Ab initio potentials are typically determined by fitting the first principles data to a suitable equation. The general form of many of these relationships is

$$u_2(r) = \gamma\exp(f(r)) - \sum_{n=3}^{8} g_{2n}(\tau r)\frac{C_{2n}}{r^{2n}}, \quad (4.23)$$

where λ and τ are the most commonly simple constants, although γ could also be a function of r. Eqs. (4.23) and (4.15) share a high degree of mathematical similarity. This is perhaps not surprising because both the experimental data fitted to Eq. (4.15) and the ab initio data fitted to Eq. (4.23) represent two-body interactions. Therefore, we would expect the optimal strategy used to accurately fit the data from either to be broadly similar (Deiters et al., 1999; Nasrabad et al., 2004).

The $g_{2n}(x)$ contribution in Eq. (4.23) is a damping function for the dispersion coefficients. Contemporary two-body potentials almost invariably use the Tang–Toennies term (Tang and Toennies, 1984) given by:

$$g_{2n}(x) = 1 - e^{-x}\sum_{k=0}^{2n}\frac{x^k}{k!} \quad (4.24)$$

The use of Eq. (4.24) in Eq. (4.23) introduces a nested summation into the evaluation of the potential energy, which is computationally expensive in a molecular simulation. This is particularly the case in MD, which also requires the derivative of the potential to evaluate the force.

Ab initio data and accurate two-body potentials for helium (Hellmann et al., 2007), neon (Hellmann et al., 2008a), argon (Jäger et al., 2009; Patkowski and Szalewicz, 2010), krypton (Jäger et al., 2016), and xenon (Hellmann et al., 2017) are available. The ab initio data for most of these cases are illustrated in Fig. 4.2. The nature of some of these potentials, which all use the Tang–Toennies term, are summarized in Table 4.1. The potentials have been applied to the molecular simulation prediction of

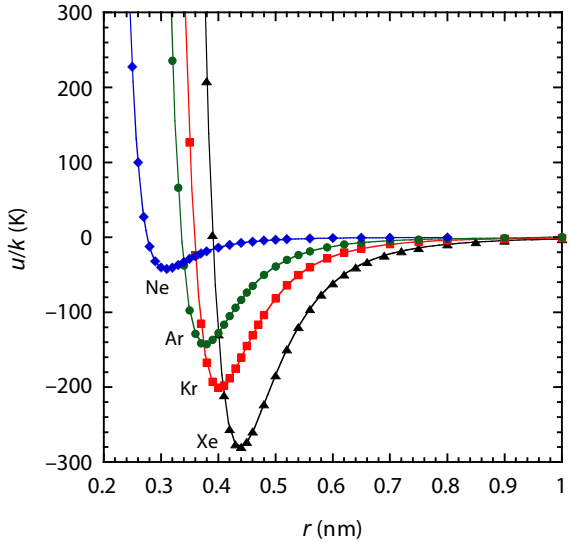


FIGURE 4.2 Comparison of potential energies for neon (◆), argon (●), krypton (■), and xenon (▲) obtained from ab initio calculations (see text for sources). The lines through the data points are only for guidance.

TABLE 4.1 Comparison of the contribution of different terms to the two-body ab initio noble gas potentials obtained from Eq. (4.23) and using Eq. (4.24).

Atom	γ	τ	$f(r)$
Helium[a]	A	b	$a_1 r + a_2 r^2 + a_{-1} r^{-1} + a_{-2} r^{-2} + d_1 \sin(d_2 r + d_3)$
Neon[b]	A	b	$a_1 r + a_2 r^2 + a_{-1} r^{-1} + a_{-2} r^{-2}$
Argon[c]	A	b	$a_1 r + a_2 r^2 + a_{-1} r^{-1} + a_{-2} r^{-2}$
Argon[d]	$a + b' r + c r^{-1} + d r^2 + e r^3$	b	$-\alpha$
Krypton[e]	A	b	$a_1 r + a_2 r^2 + a_{-1} r^{-1}$
Xenon[f]	A	b	$a_1 r + a_2 r^2 + a_{-1} r^{-1} + a_{-2} r^{-2}$

[a] *Hellmann et al., 2007.*
[b] *Hellmann et al., 2008a.*
[c] *Jäger et al., 2009.*
[d] *Patkowski and Szalewicz, 2010.*
[e] *Jäger et al., 2016.*
[f] *Hellmann et al., 2017.*

thermodynamic properties (Vlasiuk and Sadus, 2017b) and VLE (Vlasiuk and Sadus, 2017a; Deiters and Sadus, 2019a).

In addition to the various parameters summarized in Table 4.1, Eq. (4.23) also requires five values of C_{2n}. This means that the accurate determination of a two-body potential from ab initio data is a major exercise in parameter fitting. This is also the case for empirical two-body potentials. It is apparent from Fig. 4.2 that the ab initio data are not uniformly distributed over the full range of separations. In particular, there are relatively few data in the vicinity of the energy minimum and the data points, and there are very large differences in energy at small separations. There is arguably a danger of overfitting to such limited data, resulting in a potential that reflects unrecognized errors in the data. This issue is not widely appreciated in the literature.

It is apparent from Fig. 4.2 that the magnitude of the minimum in the potential well increases with the size of the noble gas atom. The magnitudes (ε/k) for neon (41.19 K), argon (142.223 K), krypton (200.875 K), and xenon (279.98 K) are considerably larger than values of 34.91 K, 119.8 K, 164.0 K, and 222.2 K, respectively, typically assigned to the LJ potential (Maitland et al., 1981). As discussed in Chapter 3, the discrepancy can be at least partly attributed to the fact that the LJ potential does not exclusively reflect the influence of two-body interaction. Multi-body effects generally make a positive contribution to interaction energy resulting in a smaller potential well than two-body interactions alone.

The complexity of ab initio two-body potentials is a serious computational impediment to their usefulness in molecular simulation, particularly in comparison with the effective intermolecular potentials examined in Chapter 3 that can be easily evaluated. This impediment has been addressed by Deiters and Sadus (2019a), via the introduction of the simplified ab initio atomic potential (SAAP). The SAAP is given by:

$$u_2(r) = \frac{\left(\frac{a_0}{r}\right)\exp\left(a_1 r + a_6 r^2\right) + a_2\exp(a_3 r) + a_4}{1 + a_5 r^6}. \tag{4.25}$$

In contrast to other ab initio potential for the noble gas, the SAAP has at most six parameters and more commonly five parameters because $a_6 = 0$ for most cases (Deiters and Sadus, 2019a). Values of the SAAP parameters are summarized in Table 4.2. The values for helium in Table 4.2 reflect updated ab initio data (Deiters and Sadus, 2020) that was not available when the potential was initially parameterized (Deiters and Sadus, 2019a).

Apart from a reduced number of parameter terms, a computational advantage of the SAAP is that the ab initio data can be accurately reproduced without using the Tang−Toennies term (Eq. (4.24)). The ability of the SAAP and a multiparameter (Hellmann et al., 2017) potential using Eq. (4.23) to reproduce the ab initio data of xenon is compared in Fig. 4.3. In general, the SAAP reproduces the ab initio data to an accuracy of $\pm 0.5\%$ with a noticeable divergence in accuracy for $r \leq 0.4$ nm corresponding to repulsive behavior.

TABLE 4.2 Summary of the two-body SAAP parameters (Deiters and Sadus, 2019a, 2020).

	He	Ne	Ar	Kr	Xe
ε/k (K)	11.06271073	42.36165080	143.4899372	201.0821392	280.1837503
σ (nm)	0.2638439772	0.2759124561	0.3355134529	0.357999364	0.3901195551
$a_0/\varepsilon\sigma$	73625.983757	211781.8544	65214.64725	60249.13228	44977.3164
$a_1\sigma$	−9.672069031	−10.89769496	−9.452343340	−9.456080572	−9.121814449
a_2/ε	−11.1831994	−20.94225988	−19.42488828	−24.40996013	−29.63636182
$a_3\sigma$	−1.785403435	−2.317079421	−1.958381959	−2.182279261	−2.278991444
a_4/ε	−2.764109154	−1.854049559	−2.379111084	−1.959180470	−1.876430370
a_5/σ^6	1.003480097	0.7454617542	1.051490962	0.874092399	0.8701531593
a_6/σ^2	0	0	0	0	0

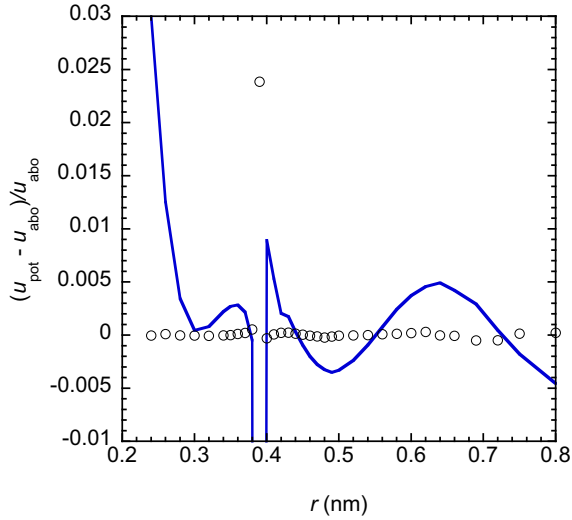


FIGURE 4.3 The relative differences between the energies calculated from the multiparameter potentials (circles) (Hellmann et al., 2017; Table 4.1) and the SAAP (solid line) and ab initio data of xenon. No actual discontinuity should be inferred from the apparent break in the line.

The SAAP has proved useful in the accurate evaluation of both VLE (Deiters and Sadus, 2019b) and SLE (Deiters and Sadus, 2021, 2022a,b; Singh et al., 2021) properties. It can also be parameterized (Deiters and Sadus, 2023) to yield accurate predictions of some properties of molecular hydrogen.

4.3 Three-body atomic potentials

The two-body potentials considered earlier are rarely useful by themselves for the accurate prediction of the properties of real substances. In principle, to fully consider many-body interactions we must consider all possible n-body interactions. However, in practice the contribution from interactions other than molecular pairs and triplets is likely to be extremely small. Further, the contributions from four- or more-body interactions are of alternating sign and, therefore, there is likely to be a high degree of cancellation of terms. Consequently, including only two-body and three-body interactions is probably an excellent approximation for many-body interactions. It should be noted, however, that including three-body effects greatly increases the required computational cost. For a three-body calculation involving N atoms there are $N(N-1)(N-2)/6$ distinct interactions compared with only $N(N-2)/2$ interactions for a pairwise calculation.

Adopting a potential-based approach for many-body interactions requires the formulation of a suitable intermolecular potential for three-body interactions. In contrast to the large literature on two-body potentials, the subject of three-body or more-body potentials has received relatively little attention.

Most work reported in the literature has focused on three-body dispersion interactions as calculated by Axilrod and Teller (1943) and Mutō (1943). However, three-body repulsion is also likely to be significant in some cases. In multipolar fluids, we must also consider the influence of three-body effects on interactions involving permanent dipoles. Very little work has been reported, which addresses either three-body repulsion or three-body effects for polar molecules. The effect of three-body interactions has been reviewed elsewhere (Elrod and Saykally, 1994; Gray et al., 2011).

4.3.1 Three-body dispersion

Various contributions to three-body dispersion interactions can be envisaged arising from instantaneous dipole (D), quadrupole (Q), octupole (O), and hexadecapole (H) moments of a triplet of atoms (see Fig. 4.4).

Different types of interaction are possible depending on the distribution of multipole moments between the atoms. In principle, the dispersion or long-range, nonadditive three-body interaction is the sum of these various combinations of multipole moments (Bell, 1970).

$$u_{disp}^{3-body} = u_{DDD} + u_{DDQ} + u_{DQQ} + u_{DDO} + u_{DDD4} + u_{QQQ} + u_{DQO} + u_{DDH} + .. \tag{4.26}$$

These terms are all third-order with the exception of the contribution of the fourth-order triple-dipole term (u_{DDD4}). The main contribution to

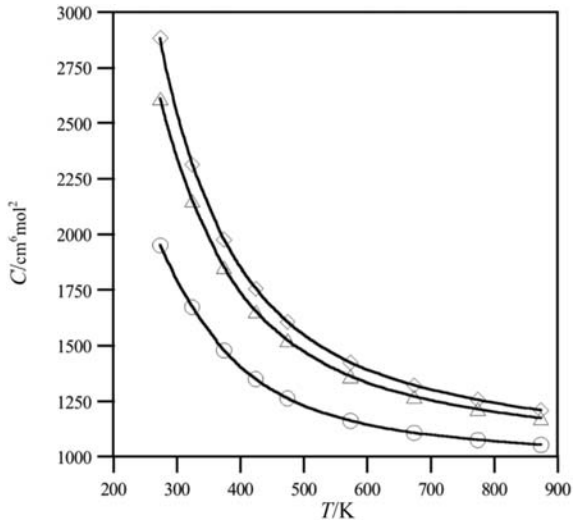


FIGURE 4.4 Calculation of the third virial coefficient (C) of krypton using additive (○); additive + triple-dipole (△); and additive + triple-dipole + other third-order multipolar (◊) terms (Barker et al., 1972b).

attractive three-body interaction is the third-order triple-dipole term (u_{DDD}). The other terms collectively ($u_{DDQ} + u_{DQQ}, + u_{DDO}, + u_{DDD4} + \ldots$) are the higher multipole contributions.

4.3.2 The triple-dipole term and the Axilrod–Teller–Mutō potential

The triple-dipole potential can be evaluated from the formula proposed by Axilrod and Teller (1943) and Mutō (1943):

$$u_{DDD}(ijk) = \frac{v_{DDD}(ijk)\left(1 + 3\cos\theta_i \cos\theta_j \cos\theta_k\right)}{\left(r_{ij} r_{ik} r_{jk}\right)^3}, \tag{4.27}$$

where $v_{DDD}(ijk)$ is the nonadditive coefficient, and the angles and intermolecular separations refer to the triangular configuration of atoms as illustrated in Fig. 4.5. We will refer to Eq. (4.27) as the Axilrod–Teller–Mutō (ATM) potential. A detailed derivation of the ATM potential from third-order perturbation theory has been given by Axilrod (1951). Mutō's original 1943 paper in Japanese also provides a derivation that appears to have a sign error in equation 15. The ATM potential can be conveniently expressed without the cosine terms:

$$u_{DDD}(ijk) = \frac{v_{DDD}(ijk)\left(\left(r_{ij} r_{jk} r_{ik}\right)^2 + 3\left(\mathbf{r}_{ik} \cdot \mathbf{r}_{jk}\right)\left(\mathbf{r}_{ik} \cdot \mathbf{r}_{ij}\right)\left(\mathbf{r}_{ij} \cdot \mathbf{r}_{jk}\right)\right)}{\left(r_{ij} r_{jk} r_{ik}\right)^5}. \tag{4.28}$$

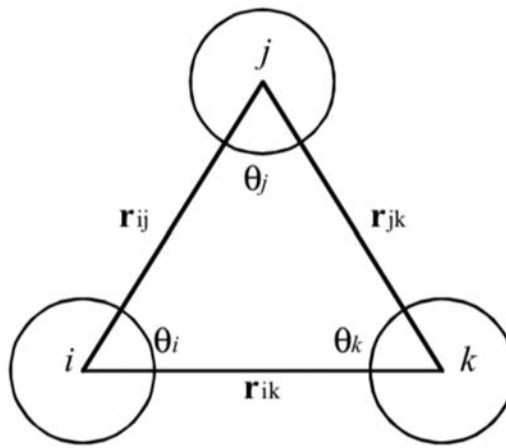


FIGURE 4.5 Triplet configuration of atoms I, j, and k in Eq. (4.25).

The contribution of the ATM potential can be either negative or positive depending on the orientation adopted by the three atoms. The potential is positive for an acute triangular arrangement of atoms, whereas it is negative for near linear geometries. The potential can be expected to make an overall repulsive contribution in a close-packed solid and in the liquid phase. The r^{-3} terms indicate that the magnitude of the potential is very dependent on intermolecular separation. The major contribution to the potential will occur for configurations in which at least one pair of atoms is in close proximity to each other.

The nonadditive coefficient ($v_{DDD}(ijk)$) can be calculated approximately from the C_6 dispersion coefficient and polarizability data (α) via the following relationship:

$$v_{DDD}(ijk) \approx \frac{3C_6\alpha}{4}. \tag{4.29}$$

Alternatively, the relationship proposed by Tang (1969) can be used

$$v_{DDD}(ijk) \approx \frac{2S_iS_jS_k(S_i + S_j + S_k)}{(S_i + S_j)(S_j + S_k)(S_k + S_i)}, \tag{4.30}$$

where

$$S_i = \frac{C_{6,i}\alpha_j\alpha_k}{\alpha_i} \tag{4.31}$$

However, the preferred option is to obtain v_{DDD} directly from experimental data. This can be achieved by analyzing dipole oscillator strength distributions. Leonard and Barker (1975) have reported experimental values of the nonadditive coefficient for the noble gases and mixtures of noble gases. Experimental values of v have also been reported by Kumar and Meath (1984, 1985). An analysis using updated theoretical procedures has also been reported (Kumar and Thakkar, 2010). Values of v_{DDD} for all of the atomic triplets containing a stable noble gas atom are given in Table 4.3, which compares historical experimental values to contemporary theoretical analysis. It is noteworthy that the contemporary theoretical values are in very good agreement with the experimental values of early decades.

The ATM potential was initially studied by several workers (Bobetic and Barker, 1970; Barker et al., 1971; Haile, 1978; Hoheisel, 1981; Monson et al., 1983; Rittger, 1990a,b,c; Attard, 1992; Smit et al., 1992; Miyano, 1994). It appeared that the ATM interactions typically contribute 5% to the energy of the liquid. This conclusion was also supported by additional Monte Carlo (MC) results (Sadus and Prausnitz, 1996). Most of the calculations relied on some type of approximation rather than a rigorous evaluation. The question that remained unresolved is: What effect does this relatively small contribution have on the properties of the fluids?

The advent of greater computational capability has enabled the full evaluation of the ATM potential (Anta et al., 1997; Sadus, 1998a,b; Marcelli and

TABLE 4.3 Comparison of $v_{DDD}(ijk)$ values[a] obtained from the historical analysis of experimental data[b] and contemporary theory.[c]

Atom *i*	Atom *j*	Atom *k*	$v_{DDD}(ijk)$ (a.u.) (experiment)	$v_{DDD}(ijk)$ (a.u.) (theory)
He	He	He	1.481	
He	He	Ne	2.296	
He	He	Ar	10.25	
He	He	Kr	14.55	
He	He	Xe	21.89	
He	Ar	Ar	5.95	
He	Ne	Ar	20.33	
He	Ne	Kr	28.80	
He	Ne	Xe	43.27	
He	Ar	Ar	72.24	
He	Ar	Kr	103.3	
He	Ar	Xe	156.4	
He	Kr	Kr	148.2	
He	Kr	Xe	224.9	
He	Xe	Xe	342.4	
Ne	Ne	Ne	12.02	11.92
Ne	Ne	Ar	40.49	40.41
Ne	Ne	Kr	57.20	57.48
Ne	Ne	Xe	85.79	85.11
Ne	Ar	Ar	142.5	142.7
Ne	Ar	Kr	203.4	204.8
Ne	Ar	Xe	307.5	306.6
Ne	Kr	Kr	291.2	294.6
Ne	Kr	Xe	441.5	442.6
Ne	Xe	Xe	671.4	667.9
Ar	Ar	Ar	517.4	519.0
Ar	Ar	Kr	744.6	750.2
Ar	Ar	Xe	1134	1134

(*Continued*)

TABLE 4.3 (Continued)

Atom *i*	Atom *j*	Atom *k*	$\nu_{DDD}(ijk)$ (a.u.) (experiment)	$\nu_{DDD}(ijk)$ (a.u.) (theory)
Ar	Kr	Kr	1074	1087
Ar	Kr	Xe	1640	1647
Ar	Xe	Xe	2509	2505
Kr	Kr	Kr	1554	1577
Kr	Kr	Xe	2377	2397
Kr	Xe	Xe	3646	3656
Xe	Xe	Xe	5605	5595

[a] *1 a.u. = 1.41825 × 10^{-92} J cm^9.*
[b] *Leonard and Barker, 1975.*
[c] *Kumar and Thakkar, 2010.*

Sadus, 1999; Wang and Sadus, 2006; Vlasiuk and Sadus, 2017a). Molecular simulation studies of VLE (Marcelli and Sadus, 1999; Vlasiuk and Sadus, 2017a) have reported that three-body interactions can alter significantly the coexisting density of the liquid phase, whereas the vapor-phase branch is unaffected. The ATM interactions also significantly influence SLE (Wang and Sadus, 2006; Deiters and Sadus, 2021). These influences are much larger than the 5% of total potential energy would intuitively suggest. The clear lesson is that the contribution to potential energy of three-body interactions per se is not necessarily a good indicator of the importance of the effect of three-body interactions on shaping the macroscopic properties of the fluid.

4.3.3 Density-dependent effective three-body potential

Although the ATM potential is an accurate representation of three-body interactions, the computational expense of calculating the interactions of all possible triplets remains far from a routine undertaking. As discussed in Chapter 3, most simulation codes are exclusively confined to the evaluation of pair interactions. Therefore, it is highly desirable to have a means of incorporating three-body interactions via pairwise interactions only. By comparing the energy obtained from the ATM potential (U_{ATM}) to the two-body contribution (U_{BFW}) of BFW potential of argon, krypton, and xenon, Marcelli and Sadus (2000) observed the following density ($\rho = N/V$; N is the number of atoms; V is the volume)-dependent empirical relationship:

$$U_{ATM} = -\frac{\lambda \nu_{DDD} \rho U_{BFW}}{\varepsilon \sigma^6}, \tag{4.32}$$

where ε is the depth of the two-body potential well; σ is the distance at which the two-body energy is zero; and λ is a constant required to optimally fit the simulation data. Originally its value was assigned as 2/3 but a later study (Wang and Sadus, 2006), using a larger number of atoms, indicated that $\lambda = 0.85$ was a better choice.

A theoretical rationale (Ustinov, 2010) for this relationship has been given, and it appears to be independent (Vlasiuk and Sadus, 2017a,b; Deiters and Sadus, 2019b) of the choice of the two-body potential. Therefore, in general, we have

$$U_3 \approx U_{ATM} = -\frac{\lambda \nu_{DDD} \rho}{\varepsilon \sigma^6} \sum_i \sum_{j>i} u_2(r_i, r_j), \tag{4.33}$$

which can be utilized for any two-body potential for which ε and σ have been determined. Eq. (4.33) means that the overall intermolecular potential for the fluid is simply:

$$u = u_2 \left(1 - \frac{\lambda \nu_{DDD} \rho}{\varepsilon \sigma^6} \right). \tag{4.34}$$

The usefulness of Eq. (4.34), which is denoted as the Marcelli−Wang−Sadus (MWS) potential, is discussed in greater detail below as part of the case study.

4.3.4 Multipolar contributions beyond the triple-dipole term

Bell (1970) has derived the other multipolar nonadditive third-order potentials that contribute to Eq. (4.26),

$$u_{DDQ}(ijk) = \frac{3\nu_{DDQ}(ijk)}{16 r_{ij}^3 (r_{jk} r_{ik})^4} \times \left[9\cos\theta_k - 25\cos 3\theta_k + 6\cos(\theta_i - \theta_j)(3 + 5\cos 2\theta_k) \right] \tag{4.35}$$

$$u_{DQQ}(ijk) = \frac{15\nu_{DQQ}(ijk)}{64 r_{jk}^5 (r_{ij} r_{ik})^4} \times \left[\begin{array}{l} 3(\cos\theta_i + 5\cos 3\theta_i) + 20\cos(\theta_j - \theta_k)(1 - 3\cos 2\theta_i) \\ + 70\cos 2(\theta_j - \theta_k)\cos\theta_i \end{array} \right] \tag{4.36}$$

$$u_{QQQ}(ijk) = \frac{15\nu_{QQQ}(ijk)}{128 (r_{ij} r_{ik} r_{jk})^5} \times \left[\begin{array}{l} -27 + 220\cos\theta_i \cos\theta_j \cos\theta_k + 490\cos 2\theta_i \cos 2\theta_j \cos 2\theta_k \\ + 175[\cos 2(\theta_i - \theta_j) + \cos 2(\theta_j - \theta_k) + \cos 2(\theta_k - \theta_i)] \end{array} \right], \tag{4.37}$$

where Eqs. (4.35)−(4.37) represent the effect of dipole−dipole−quadrupole, dipole−quadrupole−quadrupole, and quadrupole−quadrupole−quadrupole

interactions, respectively. The derivation of these formulas uses third-order perturbation theory (Bell and Zucker, 1976). Formulas for the different ordering of the multipole moments on the three atoms (i.e., *QDD*, *DQD*, *QDQ*, and *QQD*) can be generated from Eq. (4.36) by cyclic permutation of θ_i, θ_j, and θ_k. It should be noted that Eq. (4.34) disagrees with an earlier derivation for dipole−dipole−quadrupole interactions reported by Ayres and Tredgold (1956). In addition to these terms, the dipole−dipole−octupole term has also been evaluated (Doran and Zucker, 1971).

$$u_{DDO}(ijk) = \frac{45 v_{DDO}(ijk)}{64} \times \left[\frac{1+\cos^2\theta_i}{(r_{ij}r_{ik})^6} + \frac{1+\cos^2\theta_j}{(r_{ij}r_{jk})^6} + \frac{1+\cos^2\theta_k}{(r_{ik}r_{jk})^6} \right] \quad (4.38)$$

Bell (1970) proposed that the multipolar nonadditive coefficients could be related to the triple-dipole nonadditive coefficient via the relationships,

$$\left.\begin{aligned} v_{DDQ}(ijk) &\approx \gamma v_{DDD}(ijk) \\ v_{DQQ}(ijk) &\approx \gamma^2 v_{DDD}(ijk) \\ v_{QQQ}(ijk) &\approx \gamma^3 v_{DDD}(ijk) \end{aligned}\right\}, \quad (4.39)$$

where γ is the ratio of the dipole polarizability and the quadrupole polarizability.

$$\gamma = \frac{\alpha_{QQQ}}{\alpha_{DDD}} \quad (4.40)$$

The fourth-order triple-dipole term can be evaluated from the relationship (Johnson and Spurling, 1974),

$$u_{DDD4}(ijk) = v_{DDD4}(ijk) \left[\frac{1+\cos^2\theta_i}{(r_{ik}r_{jk})^6} + \frac{1+\cos^2\theta_j}{(r_{ij}r_{ik})^6} + \frac{1+\cos^2\theta_k}{(r_{ij}r_{jk})^6} \right], \quad (4.41)$$

where

$$v_{DDD4}(ijk) = -\frac{5\alpha^2 C_6}{24} \quad (4.42)$$

A discussion of dipole−dipole contributions to dispersion is available (Bade, 1957a,b; Bade, 1958).

How significant is the contribution of the higher multipolar terms in comparison to the triple-dipole potential? The other, higher multipole terms are generally believed to be small by comparison. Fig. 4.6 illustrates the contribution of additive and nonadditive terms to the calculated third virial coefficient for krypton reported by Barker et al. (1972b).

It is apparent from Fig. 4.5 that the triple-dipole term has a significant effect, whereas the contributions from other third-order multipolar terms are small. There is also evidence that the higher multipole terms compensate for each other (Barker et al., 1974). For example, Etters and Danilowicz (1979) reported that the third-order (u_{DDQ}) and fourth-order (u_{DDD4}) terms for rare

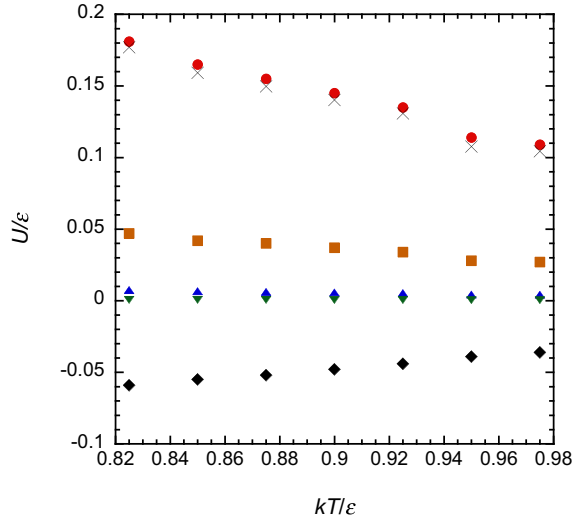


FIGURE 4.6 Contribution of *DDD* (●), *DDQ* (■), *DQQ* (▲), *QQQ* (▼), and *DDD*4 (◆) interactions (Marcelli and Sadus, 1999) to the total three-body energy along the liquid saturation curve of xenon. The sum (*DDD* + *DDQ* + *DQQ* + *QQQ* + *DDD*4) of the contributions is also illustrated (×), which coincides closely with the DDD values.

TABLE 4.4 Contribution of three-body interactions to the crystalline energy of noble gases.[a]

U(J/mol)

Atom	*DDD*	*DDQ*	*DQQ*	*QQQ*	*DDD*4
Ne	62.4	15.5	2.7	0.2	−4.8
Ar	579.6	173.8	37.0	3.5	−126.7
Kr	1004.0	220.1	34.3	2.4	−281.4
Xe	1597.9	373.6	62.0	4.6	−636.6

[a] *Doran and Zucker, 1971; Barker et al., 1972a.*

gases at equilibrium separations are typically 20% of the triple-dipole term. Furthermore, the sign of the fourth-order triple-dipole term is negative, which effectively cancels the positive contribution of the third-order u_{DDQ}, u_{DQQ}, and u_{QQQ} terms. Barker et al. (1972a) calculated the contribution of the fourth-order triple-dipole term to the crystalline energy of noble gases and compared their results with the contribution of other higher multipole moments reported by Doran and Zucker (1971). These data are summarized in Table 4.4, which shows that the fourth-order triple-dipole contribution

cancels to a large extent the contribution of the other higher multipole terms. For xenon, the magnitude of the fourth-order term is significantly larger than the combined contributions of the other higher multipole terms. The reported contributions from octupoles and fourth- and higher-order energy perturbations are small (Meath and Aziz, 1984).

In addition to the theoretical evaluations detailed previously for the crystalline energy of the noble gases, the contributions of the multipolar nonadditive potentials have also been evaluated (Marcelli and Sadus, 1999) via molecular simulation for the vapor and liquid phases of the noble gases. The various contributions to the total three-body energy of xenon along the liquid saturation curve are illustrated in Fig. 4.6. It is apparent that *DDD* makes the largest overall contribution followed by *DDQ*. Both *DQQ* and *QQQ* are close to zero, whereas *DDD*4 is negative. In contrast, the *DQQ* contribution in the crystalline phase (Table 4.4) is considerably much more substantial. However, irrespective of the nature of the phase (crystalline, vapor, or liquid), the negative value of *DDD*4 cancels the contributions of *DDQ*, *DQQ*, and *QQQ*. Therefore, the leading, triple-dipole *DDD* term alone is a good approximation for three-body dispersion interaction. This an important conclusion because it means that the calculation of three-body interactions can be limited to the ATM potential.

The substantial negative contribution to the energy of *DDD*4 interactions is very important in establishing the primary role of ATM potential. Commentaries in the simulation literature, which sometimes suggest that the ATM underpredicts three-body interactions, typically neglect this countervailing contribution. This is perhaps not surprising as comprehensive simulations involving *DDD*4 are possibly confined to a single study (Marcelli and Sadus, 1999). Adding either the higher multipolar terms or other contributions the ATM potential without the counteracting *DDD*4 term would lead to the erroneous conclusion that three-body contribution to the energy is larger than it should be. However, legitimate sources of uncertainty are the calculations of the nonadditive coefficients. Accurate values of ν_{DDD} are available for the noble gases (Table 4.2), but this is not the case for both other terms and other atoms.

A molecular dynamics technique has been reported (van der Hoef and Madden, 1998) that accounts for the contribution of both two- and three-body dispersion interactions up to the triple-quadrupole (*QQQ*) term.

4.3.5 Three-body repulsion

The effect of three-body repulsion has not been studied as extensively as three-body dispersion interactions. Consequently, a rigorous potential for three-body repulsion potential has not been developed. Sherwood et al. (1966) investigated the effect of three-body interactions on the virial coefficients. In particular, they developed models of repulsion between three

atoms. Their electrostatic distortion model is particularly convenient for molecular simulation:

$$u^{rep}(ijk) = \frac{4\varepsilon\sigma^{21}}{9}\left[\frac{\cos 2\theta_i}{(r_{ij}r_{ik})^9 r_{jk}^3} + \frac{\cos 2\theta_j}{(r_{ij}r_{jk})^9 r_{ik}^3} + \frac{\cos 2\theta_k}{(r_{ik}r_{jk})^9 r_{ij}^3}\right], \tag{4.43}$$

where ε and σ are the parameters for the LJ potential, and the angles and intermolecular separation refer to Fig. 4.5. The use of the LJ potential means that Eq. (4.41) should only be considered as an approximation. The sign of the potential depends on the geometry of the three atoms. If the atoms form an equilateral or right-angle triangle, the potential is negative, whereas it is positive for a linear configuration. Thus, the sign of the repulsive potential is opposite to the ATM potential. Consequently, the effect of repulsion will be to partially cancel the effect of the triple-dipole term.

Three-body repulsion can also be calculated from the following relationship:

$$u^{rep}(ijk) = v\exp(-\gamma_{ij}r_{ij} - \gamma_{ik}r_{ik} - \gamma_{jk}r_{jk}), \tag{4.44}$$

where the γ constants characterize the interactions between the different pairs of atoms. Lybrand and Kollman (1985) have used this potential to calculate the overlap of ion−water−water trimers. It should be noted that the theoretical basis of the three-body repulsion is considerably less well developed than three-body dispersion. The derivation of both the above repulsion potentials is less rigorous than the three-body dispersion potentials. Nonetheless, these potentials can be used to provide a useful approximation of three-body repulsion.

4.3.6 Three-body dispersion versus three-body repulsion

As outlined previously in the discussion of three-body dispersion interactions, for practical purposes, three-body dispersion interactions can be modeled exclusively using the triple-dipole term. What is the role of three-body repulsion? The answer to this question is complicated by uncertainties in the two-body potential and various approximations used in the calculation of three-body interaction.

Early work (Rosen, 1953; Jansen, 1962) indicated that three-body repulsion made a negative contribution. Later, Williams et al. (1967) observed that three-body overlap was positive and contributed 10% of the Axilrod−Teller term. Similarly, O'Shea and Meath (1974, 1976) reported that the effect of charge overlap reduced the impact of the Axilrod−Teller term by between 15% and 40%. Calculations of the third virial coefficient (Graben and Present, 1962; Dymond et al., 1965; Barker et al., 1972b) indicate that including the Axilrod−Teller term improved substantially the agreement of theory with experiment. Sherwood and Prausnitz (1964)

reported that the virial coefficient of helium was influenced by three-body repulsion. In general, calculations using a pair potential + ATM term alone have predicted successfully many experimental properties. For example, the crystal-binding energies of noble gases are predicted satisfactorily when the pairwise potential is supplemented by the ATM term. Ab initio calculations for the nonadditive forces for helium (Bulski, 1975; Wells and Wilson, 1985), neon (Bulski and Chałasiński, 1980; Wells and Wilson, 1986) and argon (Bulski and Chałasiński, 1982; Wells, 1987) indicate that the ATM term was offset largely by three-body repulsion.

The relative importance of repulsion for noble gases using the ATM potential and Eq. (4.40) is illustrated in Fig. 4.7. The energies of the intermolecular potentials were calculated for equidistant atoms arranged in either a linear or a triangular configuration. The ν_{DDD} values were taken from Leonard and Barker (1975) (Table 4.3), and the LJ parameters required for Eq. (4.42) were taken from Hirschfelder et al. (1954). Fig. 4.8 shows that repulsion completely dominates the ATM term at small interatomic separations. For argon, krypton, and xenon, the dominance of repulsion commences at a similar interatomic separation, whereas for both helium and neon, repulsion is important at considerably larger distances. A comparison of Fig. 4.7 (a) with Fig. 4.7 (b) indicates that for a given interatomic separation, repulsion is more important in an equilateral configuration compared with a linear geometry. These results need to be considered with the caveat that Eq. (4.42) is a relatively crude approximation and as such may require future reassessment using a more accurate model of repulsion.

MC estimates are available (Sadus and Prausnitz, 1996) for the contribution of three-body repulsion as calculated from Eq. (4.43) and ATM interactions to the configurational energy of liquid argon for several densities. The results are compared in Fig. 4.8. The magnitudes of both ATM and repulsive interactions increase synchronously with increasing density. Three-body repulsion makes a negative contribution to the energy, which is approximately 45% of the contribution of the ATM term. Therefore, the effect of three-body repulsion is to offset substantially the effect of three-body dispersion interactions. However, a confounding aspect of the analysis given in Fig. 4.9 is the use of the LJ potential for pairwise interactions instead of a genuine two-body potential. This means that the relative contribution of ATM interactions may not be accurately represented. This, in turn, may be counterbalanced by the fact that Eq. (4.43) was derived specifically for the LJ potential.

Schwerdtfeger and Hermann (2009) have proposed an empirical modification of the ATM potential to combine dispersion and repulsion:

$$u_3(ijk) = \left(1 + 3\cos\theta_i \cos\theta_j \cos\theta_k\right)\left[\frac{\nu_{EATM}}{\left(r_{ij}r_{ik}r_{jk}\right)^3} + e^{-\alpha\left(r_{ij}+r_{ik}+r_{jk}\right)}\sum_{i=0}^{n} A_{2i}\left(r_{ij}r_{ik}r_{jk}\right)^{2i/3}\right], \tag{4.45}$$

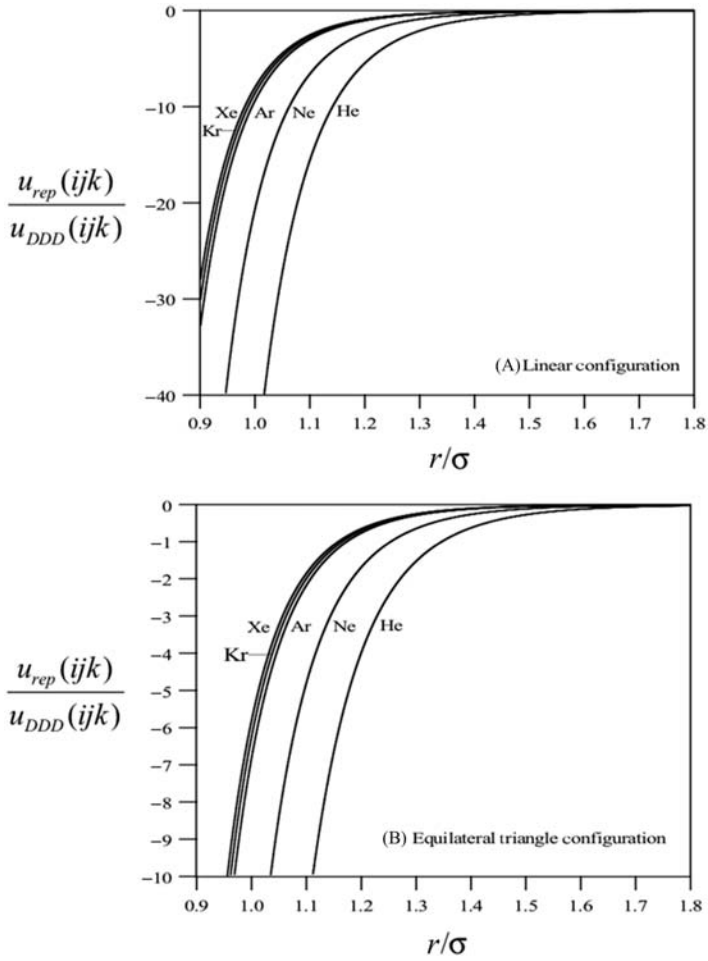


FIGURE 4.7 The ratio of repulsion to triple-dipole interactions as a function of distance for equidistant noble gas atom triplets arranged in (a) linear and (b) triangular configurations.

where n is typically equal to 5; the terms in Eq. (4.45) are usually treated as adjustable parameters that are fitted to data. We would normally expect that $\nu_{\mathrm{EATM}} = \nu_{\mathrm{DDD}}$; however, using fitting procedures for the parameters means that this is not necessarily the case. For example, although there is good agreement (Smits et al., 2020) for krypton ($\nu_{\mathrm{EATM}} = 1506$ a.u; $\nu_{\mathrm{DDD}} = 1554$ a.u.), the value obtained for xenon ($\nu_{\mathrm{EATM}} = 3471$ a.u; $\nu_{\mathrm{DDD}} = 5605$ a.u.) is much smaller than expected (Smits et al., 2020).

This extended ATM (EATM) potential has the advantage that it provides a continuous transition between repulsion and dispersion at separations corresponding to atomic overlap. This transition also occurs at considerably

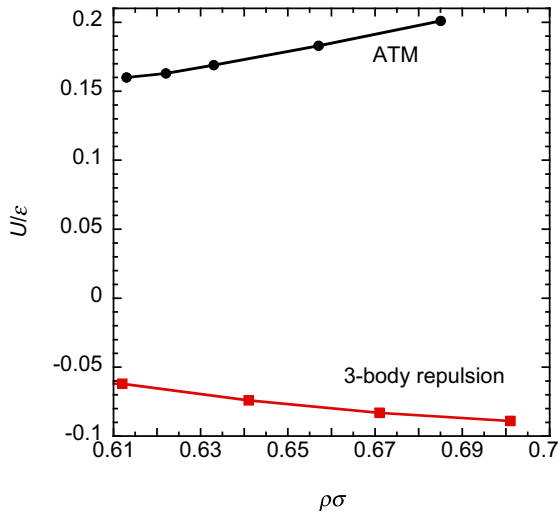


FIGURE 4.8 Comparison of the contributions of ATM (● Sadus, 1998c) and three-body repulsion (Eq. (4.40)) interactions (■, Sadus and Prausnitz, 1996) to the energy of argon at saturated liquid densities.

smaller separation than alternative models (Ermakova et al., 1998; Freiman and Tretyak, 2007). The potential has proved useful for third virial coefficients (Jäger et al., 2016) and the SLE of the noble gases (Smits et al., 2020). However, there appears to be no difference (Vlasiuk and Sadus, 2017b) between the ATM and EATM calculations for the isochoric heat capacities of krypton along the liquid saturation curve.

In summary, the relative importance of three-body repulsion and three-body dispersion remains unresolved. There is some evidence (Figs. 4.7 and 4.8) that the ATM term could be canceled by three-body repulsion. However, as will be discussed subsequently, comparison with experimental properties indicates that three-body repulsion is not required to gain a good representation of macroscopic properties such as VLE. Our knowledge of three-body dispersion is superior to three-body repulsion and better theoretical techniques, and experimental data are required to resolve the issue. These issues have been discussed extensively by Elrod and Saykally (1994).

4.4 Four- and higher-body atomic interactions

As noted previously, the literature on many-body interactions is confined almost exclusively to three-body interactions. The work of Johnson and Spurling (1974) is a rare example of the theoretical investigation of four-body interaction. Johnson and Spurling (1974) investigated the effect of

four-body interactions on the fourth virial coefficient using the following intermolecular potential (Bade, 1958),

$$u(ijkl) = v(ijkl)[f(ijkl) + f(ijkl) + f(ikjl)], \tag{4.46}$$

where:

$$f(ijkl) = \frac{1}{(r_{ij}r_{jk}r_{kl}r_{li})^3} \times \left[\begin{array}{l} -1 + (\mathbf{u}_{ij}\cdot\mathbf{u}_{jk})^2 + (\mathbf{u}_{ij}\cdot\mathbf{u}_{kl})^2 + (\mathbf{u}_{ij}\cdot\mathbf{u}_{li})^2 + (\mathbf{u}_{jk}\cdot\mathbf{u}_{kl})^2 + (\mathbf{u}_{kl}\cdot\mathbf{u}_{li})^2 \\ -3(\mathbf{u}_{ij}\cdot\mathbf{u}_{jk})(\mathbf{u}_{jk}\cdot\mathbf{u}_{kl})(\mathbf{u}_{kl}\cdot\mathbf{u}_{li}) - 3(\mathbf{u}_{ij}\cdot\mathbf{u}_{jk})(\mathbf{u}_{jk}\cdot\mathbf{u}_{kl})(\mathbf{u}_{li}\cdot\mathbf{u}_{ij}) \\ -3(\mathbf{u}_{ij}\cdot\mathbf{u}_{kl})(\mathbf{u}_{kl}\cdot\mathbf{u}_{li})(\mathbf{u}_{li}\cdot\mathbf{u}_{ij}) - 3(\mathbf{u}_{jk}\cdot\mathbf{u}_{kl}\cdot\mathbf{u}_{li})(\mathbf{u}_{li}\cdot\mathbf{u}_{jk}) \\ +9(\mathbf{u}_{ij}\cdot\mathbf{u}_{jk})(\mathbf{u}_{jk}\cdot\mathbf{u}_{kl})(\mathbf{u}_{kl}\cdot\mathbf{u}_{li})(\mathbf{u}_{li}\cdot\mathbf{u}_{ij}) \end{array}\right] \tag{4.47}$$

and where $\mathbf{u}_{ij}$ is the unit vector in the direction of particle i to particle j. Johnson and Spurling (1974) demonstrated that four-body interactions made an important contribution to the fourth virial coefficients at low to moderate temperatures.

4.5 Potentials for molecules

In common with the development of molecular force fields discussed in Chapter 3, the starting point for either two- or three-body potentials for molecules is a suitable model that captures the required molecular features such as the distribution of the constituent arrangement of atoms, bond lengths, bond angles, and the like. This modeling procedure is common to the development of a two- or three-body molecular potential, irrespective of whether the interatomic potentials are empirical, theoretical, or ab initio. The discussion given subsequently will focus primarily on two-body ab initio potentials.

It is likely that many-body interactions play a greater role in molecular fluids than in atomic fluids because induction interactions, which are not pairwise additive, are possible in nonspherically symmetric molecules. In view of the results obtained for atoms, we can reasonably expect three-body interactions to make the most important contribution to many-body interactions. Similarly, it is likely that the third-order triple-dipole potential will make the most important contribution to three-body dispersion interaction between molecules.

4.5.1 Two-body ab initio molecular potentials

The construction of a two-body potential for molecules follows the same general procedure as for pairwise force fields detailed in Chapter 3: (a) propose a molecular framework and identify interaction sites (N); (b) apply the two-body atomic potential to the interaction sites between molecular dimers;

and (c) add an angle-dependent (θ, Φ, see Figure 3.8) contribution. In general, this means:

$$u_2(r,\ \theta_n, \phi_m) = \sum_{i=1}^{N}\sum_{j=1}^{N} u_{ij}\left[r_{ij}(r,\ \theta_n, \phi_m)\right] \tag{4.48}$$

4.5.1.1 Diatomic molecules

Apart from molecular hydrogen, which includes quantum considerations (Deiters and Sadus, 2023), the simplest molecular systems are diatomic molecules such as N_2, F_2, Cl_2, and the like, which have the same constituent atoms. These cases only involve quadrupole moments (Harrison, 2005) in addition to dispersion and repulsion. The contribution of two-body interactions in ab initio potentials for diatomic molecules can often be generalized by relationships of the type,

$$u_{ij} = u_{ij}^{rep} - \sum_{i=0}^{M} f_{6+2i}(x)\frac{C_{6+2i}}{r_{ij}^{6+2i}} + g(x)\frac{q_i q_j}{r_{ij}}, \tag{4.49}$$

which combines contributions from repulsion (rep), dispersion coefficients, and charges (q); and $f(x)$ and $g(x)$ are damping functions in terms of an appropriate distance contribution (x). In Eq. (4.47), the charges are chosen to represent the effect of the quadrupole moment of the dimer via the Coulomb potential (Eq. (3.51)). For example, the Leonhard and Deiters (LD) (2002) proposed a five-site ($N = 5$) potential for nitrogen, which conforms to the previously discussed relationship when $M = 0$, and the repulsion is obtained from a modified Morse-type potential. The five-site potential for nitrogen reported by Hellmann (2013) also uses $M = 0$, with $f(x)$ obtained from Eq. (4.22), $g(x) = 1$, and $u_{ij}^{rep} = A_{ij}e^{-\alpha_{ij}r_{ij}}$, where A and α are adjustable parameters. Both the LD and Hellmann potentials involved CCSD(T) calculations, yielding good agreement with experiment for a range of thermodynamic properties.

The previously discussed principles can be easily adapted for heterogeneous diatomic molecules such as CO, NO, and so forth. In these cases, the influence of molecular diploes (Zakharov et al., 2005) is reflected in the choice of charges. The alternative to assigning charges is to specify values for the dipole and other multiple moments as exemplified by Eqs. (3.52)−(3.56). However, this alternative (Sadus, 1996) is not widely used. It is of interest to note that the contribution of repulsion used in Eq. (4.48) is mathematically simpler than the contribution used for atoms (Eq. 4.23, Table 4.1).

4.5.1.2 Triatomic molecules

Triatomic molecules are by definition heterogeneous and can be either linear or nonlinear with carbon dioxide and water being notable examples

of the former and latter, respectively. Ab initio two-body potentials for carbon dioxide have been developed via the supermolecular/MP2 method (Bock et al., 2000; Oakley and Wheatley, 2009) and symmetry-adapted perturbation theory (SAPT) (Bukowski et al., 1999; Wang et al., 2012). Good agreement with experiment for second virial coefficient viscosities and thermal conductivities of carbon dioxide in the dilute gas limit has been reported (Hellmann, 2014) using a potential based on CCSD(T) level of theory. Despite involving three atoms, the functional form of the potential is identical to the diatomic case Eq. (4.49) with the number of interaction sites increased to seven.

The attempt to determine an accurate ab initio intermolecular potential for water is a good example of the use of ab initio calculations for a nonlinear triatomic molecules. The intermolecular potential of water is complicated by the significant influence exerted by polarizability, many-body effects, molecular flexibility, and quantum corrections. Generally, ab initio two-body potentials for water do not predict accurately thermodynamic properties for a wide range of conditions. A review of ab initio potentials for water has been reported by Shvab and Sadus (2016), and our discussion will be restricted to a few examples that highlight key attributes of the modeling approach.

The potential reported by Niesar et al. (1990) (NCC) builds on the four-site model (see Figure 3.11) proposed by Matsuoka, Clementi, and Yoshimine (MCY) (1976), which is the common backbone of many water potentials. The NCC potential uses partial charges on the hydrogen atoms and a compensating negative charge on a site located along the bisector of the HOH angle.

$$u(r) = q^2\left(\frac{1}{r_{13}} + \frac{1}{r_{14}} + \frac{1}{r_{23}} + \frac{1}{r_{24}}\right) + \frac{4q^2}{r_{78}} - 2q^2\left(\frac{1}{r_{81}} + \frac{1}{r_{82}} + \frac{1}{r_{73}} + \frac{1}{r_{74}}\right)$$
$$+ A_{\mathrm{OO}}\exp(-B_{\mathrm{OO}}r_{56}) + A_{\mathrm{HH}}\exp(-B^4_{\mathrm{HH}}r_{13}r_{14}r_{23}r_{24})$$
$$+ A_{\mathrm{OH}}\exp(-B^4_{\mathrm{OH}}r_{53}r_{54}r_{61}r_{62}) + A'_{\mathrm{OH}}\exp(-B'_{\mathrm{HH}}4r_{53}r_{54}r_{61}r_{62})$$
$$+ A_{PH}\exp(-B^4_{PH}r_{73}r_{74}r_{81}r_{82}) + A_{PO}\exp(B^2_{PO}r_{76}r_{85}) \tag{4.50}$$

The NCC potential considers interaction between two water molecules; the oxygens are labeled 5 and 6; labels 1, 2, 4, and 8 denote the hydrogens associated with oxygen 5 and 6, respectively. The positions 7 and 8 are for the negative charge associated with each oxygen and q is the charge on each hydrogen (Fig. 4.9). The NCC potential predicts correctly the equilibrium water dimer in the gas phase, and the predicted binding energy is in good agreement with experiment. It also provides accurate predictions for the second virial coefficient, heat capacity, and diffusion coefficient at room temperature. However, the predicted pressure is too low and the predicted configurational energy is too high by 7%.

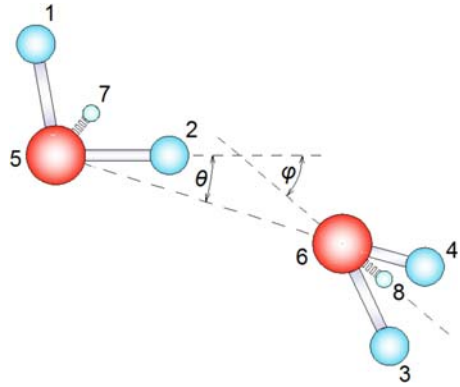


FIGURE 4.9 Illustration of the dimer geometry showing the interatomic separations common to many four-site models of water.

Bukowski et al. (2007) performed first principles calculations for water and obtained a potential that utilized a five-site monomer model (Mas et al., 2000) for water plus a polarization contribution. The functional form of the potential used by Mas et al. (2000) is identical to Eq. (4.49). The quality of predictions obtained from ab initio potentials for the thermodynamic properties of water are of moderate accuracy. This is in contrast to empirical potential that often yield very good agreement with experiment (Chapter 3). However, the contribution from ab initio terms can be augmented to improve the overall accuracy (Li et al., 2007, 2008).

4.5.1.3 *Polyatomic molecules*

Polyatomic molecules represent a challenge to ab initio calculations because of the increase in both the number of atoms and the complexity of the molecular geometry. For example, the tetrahedral geometry of methane means that there are seventeen different angular orientations between the molecular pairs (Hellmann et al., 2008b). Nonetheless, a potential for methane can be obtained (Hellmann et al., 2008b) from the ab initio data via Eqs. (4.48) and (4.49) with $M = 1$ and $N = 9$. The same approach has been used successfully for ethylene oxide ($M = 1\ N = 19$) (Crusius et al., 2014) and propane ($M = 1\ N = 14$) (Hellmann, 2017b). In common with other ab initio potentials, these potentials have only been evaluated for the second virial coefficients and/or dilute gas thermodynamic properties.

4.5.2 Three-body molecular potentials

The triple-dipole potential for nonspherical molecules depends on both position and molecular orientation. Stogryn (1970) generalized the

Axilrod–Teller potential for nonspherical molecules. The result is

$$u_{DDD}(\alpha\beta\gamma) = \frac{v_{DDD}}{3\overline{\alpha}_\alpha\overline{\alpha}_\beta\overline{\alpha}_\gamma}\left[(T_{\alpha\beta})_{ij}(T_{\alpha\gamma})_{kl}(T_{\beta\gamma})_{mn}(\alpha_\alpha)_{ik}(\alpha_\beta)_{jm}(\alpha_\gamma)_{\ln}\right], \quad (4.51)$$

where $\overline{\alpha}_\alpha$ is the mean polarizability of molecule α, $(\alpha_\alpha)_{ij}$ is the component of the polarizability tensor of molecule α, and $(T_{\alpha\beta})_{ij}$ is the ij component of the interaction tensor between molecules α and β. This tensor is defined by,

$$(T_{\alpha\beta})_{ij} = -\frac{3(\mathbf{u}_{\alpha\beta})_i(\mathbf{u}_{\alpha\beta})_j - \delta_{ij}}{r_{\alpha\beta}^3}, \quad (4.52)$$

where $(\mathbf{u}_{\alpha\beta})_i$ is the ith component of a unit vector pointing from molecule α toward β.

Stogryn (1970) examined the relative contribution of three-body interaction for nonspherical molecules compared with the ATM potential for spherical symmetry. If the centers of three linear molecules are positioned on a common axis, two different orientations can be envisaged as illustrated in Fig. 4.10. In Case 1, the axes of the two end molecules are aligned with the common axis, whereas in Case 2, the molecular axes are perpendicular to it. In both cases, the middle molecule can rotate in the plane of the figure making an angle ω with the common axis. For Case 1,

$$\frac{u_{non-shperical}}{u_{spherical}} = 1 + \frac{9}{2}\kappa - \frac{11}{2}\kappa^3 + \frac{3}{2}\kappa(3 + 10\kappa + 11\kappa^2)\cos^2\omega \quad (4.53)$$

and for Case 2,

$$\frac{u_{non-shperical}}{u_{spherical}} = 1 + \frac{3}{2}\kappa^2 - \frac{5}{2}\kappa^3 - \frac{9}{2}\kappa\sin^2\omega - 6\kappa^2(1-\kappa)\cos^2\omega \quad (4.54)$$

where κ is the anisotropy.

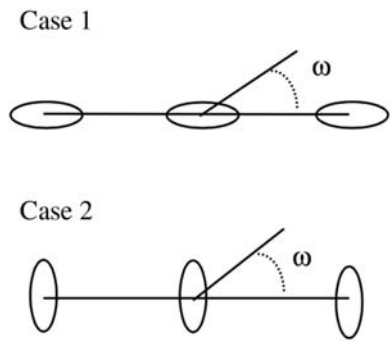


FIGURE 4.10 Two possible orientations of three molecules sharing a common axis.

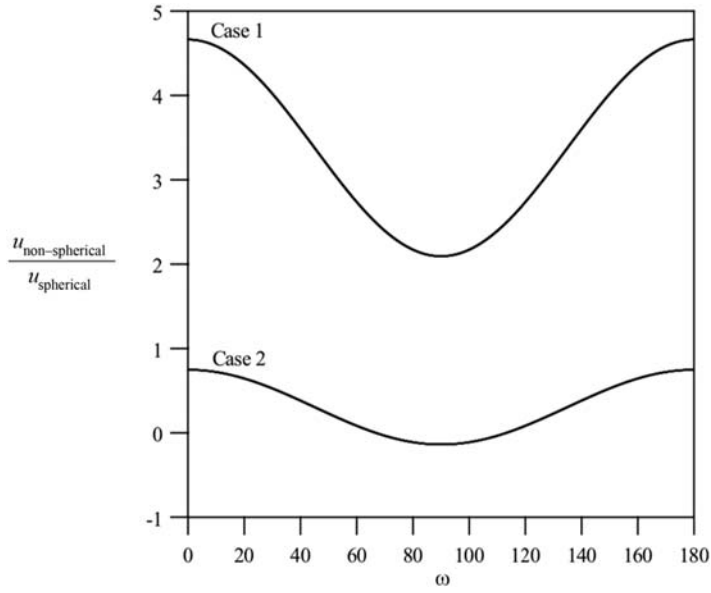


FIGURE 4.11 Relative three-body dispersion energy of linear molecules compared with spherically symmetric molecules. Results are shown for $\kappa = 0.266$, which is the anisotropy associated with nitrogen (Stogryn, 1970).

The effect of nonsphericity on the three-body interactions of nitrogen ($\kappa = 0.266$) is illustrated in Fig. 4.11. It is apparent from Fig. 4.11 that the contribution of three-body interactions between nonspherical molecules can be considerably greater than for either spherical molecules or atoms. The ATM potential is negative for atoms arranged in a linear configuration, whereas Fig. 4.11 indicates that three-body potential for nonspherical molecules is positive in some circumstances.

The analysis of three-body interactions in molecules is complicated by the absence of accurate molecular pair potentials. Instead of isolating the effect of two-body interactions in a true two-body potential, "effective" potentials are used commonly in which many-body effects are incorporated into the pairwise parameters. Monson et al. (1983) used the ATM potential in conjunction with the LJ potential to determine configurational internal energy of chlorine. They concluded that the contribution of triple-dipole interactions was between 8% and 12% of the pairwise term. This compares with a value of 5% reported typically for atomic fluids. Latajka and Scheiner (1988) observed that the three-body contribution for liquid hydrogen fluoride was 17.2% of pairwise interactions. Ab initio calculations (Kofranek et al., 1987) for hydrogen cyanide indicate a 12% contribution from three-body interactions.

McDowell (1996) used third-order perturbation theory to derive a triple-dipole potential for linear Σ-state molecules. The result is:

$$u_{DDD}(abc) = \frac{v_{DDD}(abc)(1 + 3\cos\theta_A \cos\theta_B \cos\theta_C)}{(r_{ab} r_{bc} r_{ac})^3} \times$$
$$\mathrm{Re}\left[\begin{array}{l} 1 - \Gamma(a)W(\theta_a, \phi_a, \theta_A, \theta_B) - \Gamma(b)W(\theta_b, \phi_b, \theta_A, \theta_B) \\ - \Gamma(c)W(\theta_c, \phi_c, \theta_A, \theta_B) \\ + \Gamma(a,b)W(\theta_a, \phi_a, \theta_b, \phi_b, \theta_A, \theta_B) + \Gamma(a,c)W(\theta_a, \phi_a, \theta_c, \phi_b, \theta_A, \theta_B) \\ + \Gamma(b,c)W(\theta_b, \phi_b, \theta_c, \phi_c, \theta_A, \theta_B) \\ - \Gamma(a,b,c)W(\theta_a, \phi_a, \theta_b, \phi_b, \theta_c, \phi_c, \theta_A, \theta_B) \end{array}\right] \tag{4.55}$$

The first term on the right-hand side of Eq. (4.55) is the ATM potential for three linear molecules a, b, c forming internal angles θ_A, θ_B, and θ_C. This is formally identical to the situation depicted in Fig. 4.5 for atoms. The Re[...] term indicates the real part of [....], the W terms are angular coefficients, and θ_a and ϕ_a are spherical polar angles describing the orientation of molecule a. The Γ's are coefficients that depend on the oscillator strengths/excitation energies of the allowed dipolar transitions in the constituent molecules.

In addition to the third-order triple term, for nonpolar linear molecules there is also a second-order nonadditive induction contribution to the long-range energy (McDowell, 1996; McDowell et al., 1996). Linear molecules can have a quadrupole moment that can induce dipole moments in neighboring molecules resulting in a second-order nonadditive induction energy in molecular trimers that varies as r^{-8}. McDowell (1996) investigated the effect of this second-order nonadditive correction for trimers of hydrogen molecules. Unlike the ATM potential, which typically makes a negative contribution, the second-order nonadditive term makes a positive contribution to the energy of interaction. Furthermore, for most orientations, the second-order nonadditive induction energy is comparable in magnitude or larger than the triple-dipole dispersion energy.

The effect of many-body interactions has been investigated extensively for water (Elrod and Saykally, 1994). The influence on the structure and properties of water of three-body interactions is considerably greater than is observed in atomic systems. Hartree−Fock calculations (Kistenmacher et al., 1974) show that three-body interactions contribute 10% of pairwise interactions for cyclic water trimers and tetramers. The main contribution to many-body interactions can be attributed to orbital deformation as modeled by classical polarization (Clementi and Corongiu, 1983). Kim et al. (1986) reported MC simulations using an intermolecular potential containing up to four-body interactions for the minimum energy structures of $(H_2O)_n$ clusters. They concluded that three- and four-body terms made an important contribution when $n > 5$. Ab initio calculations of the cyclic water tetramer (Koehler et al., 1987) indicate that three-body interactions contribute 18% of pair interactions to the stabilization of the structure.

An ab initio potential for carbon dioxide has been proposed (Hellmann, 2017a), which involves using the ATM potential coupled with contributions from induction and exchange effects. Although the ATM potential is strictly only valid for atomic interactions, it can be applied to molecules by using it as a site−site potential for the constituent atoms of a molecules. The site−site approach has proved useful in incorporating three-body interactions into a model for water (Li et al., 2007, 2008).

4.5.3 Ab initio potentials for combinations of molecules

In Chapter 3, it was observed that both intermolecular pair potentials and force fields have been almost exclusively developed for pure component and that combining rules (Sadus, 1992; Hasalm et al., 2008) need to be used to apply the potentials to systems containing molecules of dissimilar components, that is, a mixture. Some efforts (Klein, 1984) have been made to obtain mixed two-body potentials from experimental data. However, the limitation of this approach is that experimental data invariably contain both like and dissimilar interactions, which are difficult to separate from each other. In contrast, it is possible to determine the interactions between nonidentical atoms and molecules from first principles and use the data to obtain an ab initio potential for the dissimilar interactions. This would be particularly useful in molecular simulation of mixtures by enabling completely a prior prediction without the need for combining rules or unlike adjustable parameters. Despite this, the development of ab initio potentials has been overwhelmingly focused on pure component properties.

Combined potentials for dissimilar noble gas atoms have been reported (Haley and Cybulski, 2003; Cacheiro et al., 2004; Jäger and Bich, 2017). Ab initio data for dissimilar molecules are relatively scarce and largely confined to either CH_4, CO_2, or H_2O as one of the components. Examples include H_2S + H_2O (Hellmann, 2023), SO_2 + H_2O (Hellmann, 2023), CO_2 + H_2O (Hellmann, 2019), CO_2 + CH_4 (Hellmann et al., 2016), CO_2 + H_2S (Hellmann et al., 2016), CH_4 + H_2S (Hellmann et al., 2016), and CH_4 + N_2 (Hellmann et al., 2014). Obtaining the intermolecular potential for the mixed pair follows the same procedure that is used for identical pairs via a simple modification to Eq. (4.48).

$$u_2(r,\ \theta_n, \phi_m) = \sum_{i=1}^{N_\alpha} \sum_{j=1}^{N_\beta} u_{ij}\left[r_{ij}(r,\ \theta_n, \phi_m)\right] \tag{4.56}$$

N_α and N_β are the number of interaction sites on molecule α and β, respectively, with the pair interaction energy obtained from Eq. (4.49). Performing this analysis means that any parameters required by Eq. (4.49) reflect the mixed interaction. The combined potential allows the direct evaluation of the mixed virial coefficient.

As noted previously, the number of ab initio three-body potentials is limited to a few examples and no three-body mixture potential has been

reported. It this case, there is usually no alternative to combining rules. In a binary mixture, it has been observed (Sadus, 1996) that the nonadditive coefficient required for the ATM potential can be obtained by taking a geometric average of the like interactions.

$$\left.\begin{aligned} \nu_{\alpha\alpha\beta} &= \sqrt[3]{\nu_{\alpha\alpha\alpha}\nu_{\alpha\alpha\alpha}\nu_{\beta\beta\beta}} \\ \nu_{\beta\beta\alpha} &= \sqrt[3]{\nu_{\beta\beta\beta}\nu_{\beta\beta\beta}\nu_{\alpha\alpha\alpha}} \end{aligned}\right\} \tag{4.57}$$

4.6 Case study: Augmenting ab initio potentials for fluid properties

Although some three-body ab initio potentials have been developed for a limited set of circumstances, the current state-of-the-art is limited to two-body potentials. As documented above, advances in computational chemistry have been utilized to determine high accurate two-body interaction, which undoubtedly genuinely reflect the contribution of pair interaction. However, the scope of useful application of these two-body potentials is largely limited to the properties of dilute gases. It is appropriate to utilize two-body potentials more broadly because two-body interactions dominate the interactions of nonpolar molecules. This can be achieved by augmenting the two-body ab initio potentials (u_2^{ab}) with contribution that are not available from first principles,

$$u = u_2^{ab} + \sum_i^N u_i^{non-ab}, \tag{4.58}$$

where $u^{\text{non-ab}}$ represents any contributions that involves neither two-body interactions nor contributions from ab initio data.

4.6.1 Systematic improvement of water potentials

The large number of empirical potentials for water was discussed in Chapter 3. Although some of the empirical potentials for water (Shvab and Sadus, 2016) have resulted in considerable improvement in accuracy, this has not been achieved via a systematic approach. It is often the case that the high degree of parameterization required to achieve optimal results at a given state point means reduced accuracy at other conditions. In the spirit of Eq. (4.58), Li et al. (2007) identified contributions from three-body interactions (u_3) and polarization (u_p).

$$u = u_2^{ab} + u_3 + u_{pol} \tag{4.59}$$

In Eq. (4.59), u_2^{ab} was obtained from the four-site MCY potential (similar to Eq. (4.50)), which alone cannot be used to accurately represent the properties of real water. This deficiency is rectified by the other nonadditive (na) terms, and the new potential is designated as MCYna. The contribution of

three-body interactions was modeled via the ATM potential, interacting via only oxygen triplets.

The contribution from polarization (Gray and Gubbins, 1984) interactions was obtained from,

$$u_{pol} = -\frac{1}{2}\sum_{i=1}^{N} \boldsymbol{\mu}_i^{\text{ind}} \cdot \mathbf{E}_i^o, \tag{4.60}$$

where $\mathbf{E}^0$ is the electrostatic field of surrounding charges and $\boldsymbol{\mu}^{ind}$ is the induced dipole at site i given by:

$$\boldsymbol{\mu}_i^{ind} = \alpha\beta \cdot \mathbf{E}_i = \alpha\beta \cdot \left[\mathbf{E}_i^o + \sum_{j=1, j\neq i}^{N} T_{ij}\boldsymbol{\mu}_j^{ind}\right]. \tag{4.61}$$

In Eq. (4.61), $\alpha\beta$ is the polarizability and T_{ij} element of the dipole tensor given by:

$$T_{ij} = \frac{1}{4\pi\varepsilon_o r_{ij}^5}\left[3r_{ij}r_{ij}^{'} - r_{ij}^2\right]. \tag{4.62}$$

For the MCYna potential, a value of $\beta = 0.557503$ is used to reproduce the dipole moment of liquid water.

The MCYna potential yields a good representation of the dielectric constant of both pure water (Shvab and Sadus, 2013) and its mixtures (Shvab and Sadus, 2014); and thermodynamic properties (Yigzawe and Sadus, 2013a,b) such as heat capacities (C_v, C_p), compressibilities (β_T, β_S), thermal pressure coefficients (γ_V), expansion coefficient (α_p), the speed of sound (w_0), and the Joule−Thomson coefficient (μ_{JT}) over a wide range of temperatures and pressures. A partial comparison with experiment and empirical intermolecular potentials discussed in Chapter 3 is given in Table 4.5. The comparison indicates that the MCYna gives superior agreement with experiment than the other potentials. This can be attributed to the addition of nonadditive effects, most notably polarization. Perhaps more importantly, it highlights the effectiveness of the augmentation process, which has expanded the scope of the MCY potential form dilute gas to liquid phase properties.

4.6.2 Efficient three-body calculations

The MWS potential (Eq. 4.34) is a straightforward example of augmentation of a two-body potential. It was originally developed (Marcelli and Sadus, 2000) in conjunction with the empirical BFW potential, yielding calculations that were in very good agreement with the ATM potential. Subsequently (Vlasiuk et al., 2016; Vlasiuk and Sadus, 2017a,b), it has been proved useful when used in conjunction with ab initio potentials.

TABLE 4.5 Comparison of experiment data with predictions from different intermolecular potentials.[a]

Property	Expt	MCYna	SPC/E	TIP4P	TIP4P/2005
C_V (J/molK)	74.44	74.4	79.3	82.1	81.8
C_p (J/molK)	75.312	74.6	80.4	89.0	82.6
β_T (GPa^{-1})	0.458	0.448	0.476	0.590	0.497
β_S (GPa^{-1})	0.425	0.446	0.470	0.544	0.492
γ_V (MPa/K)	0.436	0.383	0.731	0.746	0.618
α_p (10^{-4}/K)	2.00	2.63	3.48	4.4	3.08
μ_{JT} (K/MPa)	−0.222	−0.230	−0.201	−0.187	−0.194
w_0 (m/s)	1496.7	1498.4	1460.1	1357.4	1427.2

[a] *Shvab and Sadus, 2016*.

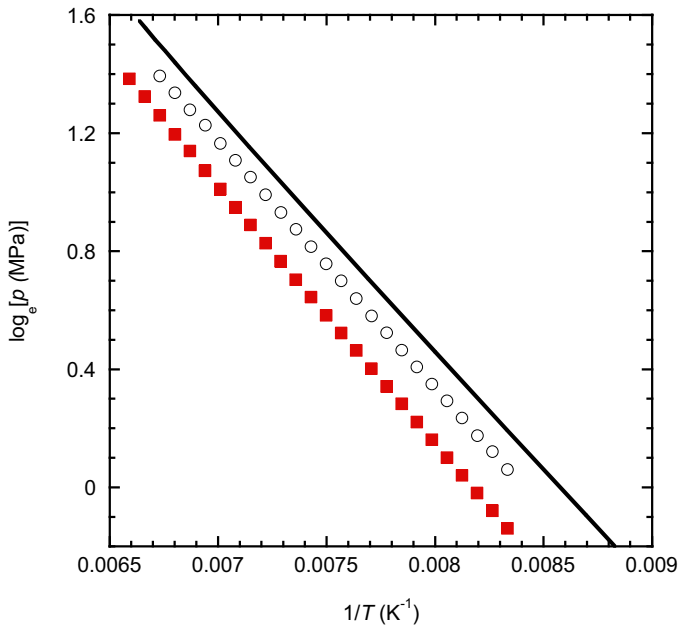


FIGURE 4.12 Comparison of experimental vapor pressures of argon (solid line) with predictions of the SAAP (■) and the three-body augmented SAAP (○) (Deiters and Sadus, 2019b).

The SAAP provides a computationally efficient way of calculating two-body interactions in a molecular simulation. However, in common with other two-body potentials its usefulness is largely restricted to the dilute properties of gases. This is illustrated in Fig. 4.12, which compares the vapor pressures of argon predicted by the SAAP alone and the MWS three-body augmentation of the SAAP (Deiters and Sadus, 2019b). It is apparent from this comparison that the SAAP considerably underestimates the vapor pressures over the entire pressure range. The three-body augmentation results in much closer agreement with experiment. Although the agreement with experiment remains imperfect, the calculation is a genuine prediction that has expanded the usefulness of the SAAP. A similar conclusion can be reached for other thermodynamic properties (Deiters and Sadus, 2020, 2021).

4.7 Summary

Historically, intermolecular potentials have been empirically determined, whereas the advent of recent advances in computational chemistry means that potentials can be determined from first principles or ab initio data. This has resulted in some highly accurate two-body potentials for both atomic systems and some small molecules. Although two-body potentials accurately represent the properties of dilute gases, the prediction of the properties of liquids require the contributions from multi-body interactions. However, the combination of two-body plus three-body interactions is often sufficient to obtain accurate results for the thermodynamic properties of real molecules.

The computational expense of three-body calculation can often be avoided by a simple density-dependent representation of the ATM that can be added to the two-body potential. This straightforward augmentation means that the use of accurate ab initio two-body potentials is not confined to the properties of dilute gases. That is, they can be usefully employed to determine the properties of real substances in the dense fluid region.

References

Anta, J.A., Lomba, E., Lombardero, M., 1997. Influence of three-body forces on the gas-liquid coexistence of simple fluids: the phase equilibrium of argon. Phys. Rev. E 55, 2707–2712.

Attard, P., 1992. Simulation for a fluid with the Axilrod-Teller triple dipole potential. Phys. Rev. A 45, 5649–5653.

Axilrod, B.M., 1951. Triple-dipole interaction. I. Theory. J. Chem. Phys. 19, 719–729.

Axilrod, B.M., Teller, E., 1943. Interaction of the van der Waals' type between three atoms. J. Chem. Phys. 11, 299–300.

Ayres, R.U., Tredgold, R.H., 1956. On the crystal structure of the rare gases. Proc. Phys. Soc. 69, 840–842.

Aziz, R.A., Slaman, M.J., 1986. The argon and krypton interatomic potentials revisited. Mol. Phys. 58, 679–697.

Bade, W.L., 1957a. Drude-model calculation of dispersion forces. I. General theory. J. Chem. Phys. 27, 1280–1284.
Bade, W.L., 1957b. Drude-model calculation of dispersion forces. II. The linear lattice. J. Chem. Phys. 27, 1284–1288.
Bade, W.L., 1958. Drude-model calculation of dispersion forces. III. The fourth-order contribution. J. Chem. Phys. 28, 282–284.
Barker, J.A., Pompe, A., 1968. Atomic interactions in argon. Aust. J. Chem. 21, 1683–1694.
Barker, J.A., Fisher, R.A., Watts, R.O., 1971. Liquid argon: Monte Carlo and molecular dynamics calculations. Mol. Phys. 21, 657–673.
Barker, J.A., Johnson, C.H.J., Spurling, T.H., 1972a. On the cancellation of certain three- and four-body interactions in inert gases. Aust. J. Chem. 25, 1811–1812.
Barker, J.A., Johnson, C.H.J., Spurling, T.H., 1972b. Third virial coefficients for krypton. Aust. J. Chem. 25, 1813–1814.
Barker, J.A., Watts, R.O., Lee, J.K., Schafer, T.P., Lee, Y.T., 1974. Interatomic potentials for krypton and xenon. J. Chem. Phys. 61, 3081–3088.
Bartlett, R.J., 1981. Many-body perturbation theory and coupled cluster theory for electron correlation in molecules. Ann. Rev. Phys. Chem. 32, 359–401.
Bell, R.J., 1970. Multipolar expansion for the non-additive third-order interaction energy of three atoms. J. Phys. B 3, 751–762.
Bell, R.J., Zucker, I.J., 1976. In: Klein, M.L., Venables, J.A. (Eds.), Rare Gas Solids, Vol. 1. Academic Press, London.
Bobetic, M.V., Barker, J.A., 1970. Lattice dynamics with three-body forces: argon. Phys. Rev. B. 2, 4169–4175.
Bock, S., Bich, E., Vogel, E., 2000. A new intermolecular potential energy surface for carbon dioxide from ab initio calculations. Chem. Phys. 257, 147–156.
Born, M., Mayer, J.E., 1932. Zur Gittertheorie der Ionenkristalle. Z. Phys. 75, 1–18.
Bretonnet, J.-L., 2017. Basics of the density functional theory. AIMS Mat. Sci. 4, 1372–1405.
Bukowski, R., Sadlej, J., Jeziorski, B., Jankowski, P., Szalewicz, K., Kucharski, S.A., Williams, H.L., Rice, B.M., 1999. Intermolecular potential of carbon dioxide dimer from symmetry-adapted perturbation theory. J. Chem. Phys. 110, 3785–3803.
Bukowski, R., Szalewicz, K., Groenenboom, G.C., van der Avoird, A., 2007. Predictions of the properties of water from first principles. Science 315, 1249–1252.
Bulski, M., 1975. On the exchange repulsion between beryllium atoms. Mol. Phys. 29, 1171–1179.
Bulski, M., Chałasiński, G., 1980. Many-orbital cluster expansion for the exchange-repulsion nonadditivity in the interaction of rare gas atoms. The neon trimer. Theor. Chim. Acta 56, 199–210.
Bulski, M., Chałasiński, G., 1982. On exchange-repulsion non-additivity in the interaction of three argon atoms. Chem. Phys. Lett. 89, 450–453.
Cacheiro, J.L., Fernández, B., Marchesan, D., Coriani, S., Hättig, C., Rizzo, A., 2004. Coupled cluster calculations of the ground state potential and interaction induced electric properties of the mixed dimers of helium, neon and argon. Mol. Phys. 102, 101–110.
Car, R., Parrinello, M., 1985. Unified approach for molecular dynamics and density-functional theory. Phys. Rev. Lett. 55, 2471–2474.
Clementi, E., Corongiu, G., 1983. Monte Carlo study of liquid water with two- and three-body ab initio potentials. Int. J. Quant. Chem. Quant. Bio. Symp. 10, 31–41.
Cohen, A.J., Mori-Sanchez, P., Yang, W.T., 2012. Challenges for density functional theory. Chem. Rev. 112, 289–320.

Crusius, J.P., Hellmann, R., Hassel, E., Bich, E., 2014. Intermolecular potential energy surface and thermophysical properties of ethylene oxide. J. Chem. Phys. 141, 164322. Publisher's note: *J. Chem. Phys.* **141**, 189902 (2014).

Deiters, U.K., Sadus, R.J., 2019a. Two-body interatomic potentials for He, Ne, Ar, Kr, and Xe from ab initio data, J. Chem. Phys., 150. p. 134504.

Deiters, U.K., Sadus, R.J., 2019b. Fully a priori prediction of the vapor-liquid equilibria of Ar, Kr, and Xe from ab initio two-body plus three-body interatomic potentials. J. Chem. Phys. 151, 034509.

Deiters, U.K., Sadus, R.J., 2020. Ab Initio interatomic potentials and the classical molecular simulation prediction of the thermophysical properties of helium. J. Phys. Chem. B 124, 2268−2276.

Deiters, U.K., Sadus, R.J., 2021. Interatomic interactions responsible for the solid-liquid and vapor-liquid phase equilibria of neon. J. Phys. Chem. B 125, 8522−8531.

Deiters, U.K., Sadus, R.J., 2022a. First-principles determination of the solid-liquid-vapor triple point: the noble gases. Phys. Rev. E. 105, 054128 (2022).

Deiters, U.K., Sadus, R.J., 2022b. Accurate determination of solid-liquid equilibria by molecular simulation: behavior of Ne, Ar, Kr, and Xe from low to high pressures. J. Chem. Phys. 157, 204504 (2022).

Deiters, U.K, Sadus, R.J., 2023. An intermolecular potential for hydrogen: classical molecular simulation of pressure-denisty-temperature behavior, vapor-liquid equilibria, critical and triple point properties. J. Chem. Phys. 158, 194502.

Deiters, U.K., Hloucha, M., Leonhard, K., 1999. Experiments? − No, thank you!. In: Letcher, T. M. (Ed.), Chemistry for the 21st Century: Chemical Thermodynamics. Blackwell Science, Oxford, pp. 187−195.

del Berne, J.E., Person, W.B., Szczepaniak, K., 1995. Properties of hydrogen-bonded complexes obtained from the B3LYP functional with 6−31 G(d,p) and 6−31 + G(d,p) basis sets. J. Phys. Chem. 99, 10705−10707.

Doran, M.B., Zucker, I.J., 1971. Higher order multipole three-body van der Waals interactions and stability of rare gas solids. J. Phys. C. 4, 307−312.

Dymond, J.H., Rigby, M., Smith, E.B., 1965. Intermolecular potential-energy function for simple molecules. J. Chem. Phys. 42, 2801−2806.

Eggenberger, R., Gerber, S., Huber, H., Welker, M., 1994. A new ab initio potential for the neon dimer and its application in molecular dynamics simulations of the condensed phase. Mol. Phys. 82, 689−699.

Elrod, M.J., Saykally, R.J., 1994. Many-body effects in intermolecular forces. Chem. Rev. 94, 1975−1997.

Ermakova, E., Solca, J., Huber, H., Welker, M., 1995. Argon in condensed phase: quantitative calculations of structural, thermodynamic, and transport properties from pure theory. J. Chem. Phys. 102, 4942−4951.

Ermakova, E., Solca, J., Steinebrunner, G., Huber, H., 1998. Ab initio calculation of a three-body potential to be applied in simulations of fluid neon. Chem. Eur. J. 4, 377−382.

Etters, R.D., Danilowicz, R., 1979. Three-body interactions in small rare-gas clusters. J. Chem. Phys. 71, 4767−4768.

Frank, I., 2020. Some simple facts about water: CPMD simulation. Mol. Phys. 118 (21), 1−6.

Freiman, Y.A., Tretyak, S.M., 2007. Many-body interactions and high-pressure equations of state in rare-gas solids. Low. Temp. Phys. 33, 545−552.

Friesner, R.A., 1991. New methods for electronic structure calculations on large molecules. Ann. Rev. Phys. Chem. 42, 341−367.

Goddard III, W.A., Harding, L.B., 1978. The description of chemical bonding from ab initio calculations. Ann. Rev. Phys. Chem. 29, 363–396.

Graben, H.W., Present, R.D., 1962. Evidence for three-body forces from third virial coefficients. Phys. Rev. Lett. 9, 247–248.

Gray, C.G., Gubbins, K.E., 1984. Theory of Molecular Fluids, Vol. 1: Fundamentals. Clarendon Press, Oxford.

Gray, C.G., Gubbins, K.E. Joslin, C.G., 2011. Theory of Molecular Fluids. Vol. 2: Applications, Oxford University Press, Oxford.

Haile, J.M., 1978. Molecular dynamics simulation of simple fluids with three-body interactions included. ACS Symp. Ser. 86, 172–191.

Haley, T.P., Cybulski, S.M., 2003. Ground state potential energy curves for He-Kr, Ne-Kr, Ar-Kr and Kr_2: coupled-cluster calculations and comparison with experiment. J. Chem. Phys. 119, 5487–5496.

Harrison, J.F., 2005. On the representation of molecular quadrupole moments in terms of atomic moments. J. Phys. Chem. A 109, 5492–5497.

Hasalm, A.J., Galindo, A., Jackson, G., 2008. Prediction of binary intermolecular potential parameters for use in modeling fluid mixtures. Fluid Phase Equilib 266, 105–128.

Hellmann, R., 2013. *Ab initio* potential energy surface for the nitrogen molecule pair and thermophysical properties of nitrogen gas. Mol. Phys. 111, 387–401.

Hellmann, R., 2014. *Ab initio* potential energy surface for the carbon dioxide molecule pair and thermophysical properties of dilute carbon dioxide gas. Chem. Phys. Lett. 613, 133–138.

Hellmann, R., 2017a. Nonadditive three-body potentials and third to eighth virial coefficients of carbon dioxide. J. Chem. Phys. 146, 054302.

Hellmann, R., 2017b. Intermolecular potential energy surface and thermophysical properties of propane. J. Chem. Phys. 146, 114304.

Hellmann, R., 2019. Cross second virial coefficient and dilute gas transport properties of the ($H_2O + CO_2$) system from first-principles calculations. Fluid Phase Equilib. 485, 252–263. Erratum: *Fluid Phase Equilib.* **518**, 112624 (2020).

Hellmann, R., 2023. Cross second virial coefficients of the H_2O-H_2S and H_2O-SO_2 systems from first principles. J. Chem. Eng. Data 68, 108–117.

Hellmann, R., Bich, E., Vogel, E., 2007. *Ab initio* potential energy curve for the helium atom pair and thermophysical properties of dilute helium gas. I. Helium–helium interatomic potential. Mol. Phys. 105, 3013–3023.

Hellmann, R., Bich, E., Vogel, E., 2008a. *Ab initio* potential energy curve for the neon atom pair and thermophysical properties of the dilute neon gas. I. Neon–neon interatomic potential and rovibrational spectra. Mol. Phys. 106, 133–140.

Hellmann, R., Bich, E., Vogel, E., 2008b. *Ab initio* intermolecular potential energy surface and second pressure virial coefficients of methane. J. Chem. Phys. 128, 214303.

Hellmann, R., Bich, E., Vogel, E., Vesovic, V., 2014. Intermolecular potential energy surface and thermophysical properties of the CH_4-N_2 system. J. Chem. Phys. 141, 224301.

Hellmann, R., Bich, E., Vesovic, V., 2016. Cross second virial coefficients and dilute gas transport properties of the ($CH_4 + CO_2$), ($CH_4 + H_2S$) and ($H_2S + CO_2$) systems from accurate intermolecular potential energy surfaces. J. Chem. Thermodyn. 102, 429–441.

Hellmann, R., Jäjer, B., Bich, E., 2017. State-of-the-art *ab initio* potential energy curve for the xenon atom pair and related spectroscopic and thermophysical properties. J. Chem. Phys. 147, 034304.

Hellmann, R., Gaiser, C., Fellmuth, B., Vasyltsova, T., Bich, S., 2021. Thermophysical properties of low-density neon gas from highly accurate first-principles calculations and dielectric-constant gas thermometry measurements. J. Chem. Phys. 154, 164304.

Hirschfelder, J.O., Curtiss, C.F., Bird, R.B., 1954. Molecular Theory of Gases and Liquids. John Wiley & Sons, New York.

Hofbauer, F., Frank, I., 2012. CPMD simulation of a bimolecular chemical reaction: Nucleophilic attack of a disulfide bond under mechanical stress. Chem. Eur. J. 18, 16322–16338.

Hoheisel, C., 1981. Effect of three-body forces on the static and dynamic correlation functions of a model liquid. Phys. Rev. A 23, 1998–2005.

Hohenberg, P., Kohn, W., 1964. Inhomogeneous electron gas. Phys. Rev. B 136, 864–871.

Jäger, B., Bich, E., 2017. Thermophysical properties of krypton-helium gas mixtures from ab initio pair potentials. J. Chem. Phys. 146, 214302.

Jäger, B., Hellmann, R., Bich, E., Vogel, E., 2009. *Ab initio* pair potential energy curve for the argon atom pair and thermophysical properties of the dilute argon gas. I. Argon–argon interatomic potential and rovibrational spectra. Mol. Phys. 107, 2181–2188.

Jäger, B., Hellmann, R., Bich, E., Vogel, E., 2016. State-of-the-art *ab initio* potential energy curve for the krypton atom pair and thermophysical properties of dilute krypton gas. J. Chem. Phys. 144, 11304.

Jansen, L., 1962. Systematic analysis of many-body interactions in molecular solids. Phys. Rev. 125, 1798–1804.

Johnson, C.H.J., Spurling, T.H., 1974. Non-additivity of intermolecular forces: Effects on the fourth virial coefficient. Aust. J. Chem. 27, 241–247.

Johnson, R.R., Mackie, I.D., DiLabio, G.A., 2009. Dispersion interactions in density-functional theory. J. Phys. Org. Chem. 22, 1127–1135.

Kim, K.S., Dupuis, M., Lie, G.C., Clementi, E., 1986. Revisiting small clusters of water molecules. Chem. Phys. Lett. 131, 451–456.

Kistenmacher, H., Lie, G.C., Popkie, H., Clementi, E., 1974. Study of the structure of molecular complexes. VI. Dimers and small clusters of water molecules in the Hartree-Fock Approximation. J. Chem. Phys. 61, 546–561.

Klein, M.L. (Ed.), 1984. Inert Gases: Potentials, Dynamics and Energy Transfer in Doped Crystals. Springer-Verlag, Berlin.

Klein, M.L., Venables, J.A. (Eds.), 1976. Rare Gas Solids, Vol. 1. Academic Press, London.

Koehler, J.E.H., Sanger, W., Lesyng, B., 1987. Cooperative effects in extended hydrogen bonded systems involving O-H groups. Ab initio studies of the cyclic S_4 water tetramer. J. Comput. Chem. 8, 1090–1098.

Kofranek, M., Karpfen, A., Lischka, H., 1987. Ab initio studies on hydrogen-bonded clusters. I. Linear and cyclic oligomers of hydrogen cyanide. Chem. Phys. 113, 53–64.

Kohn, W., Sham, L.J., 1965. Self-consistent equations including exchange and correlation effects. Phys. Rev. A 140, 1133–1138.

Kohn, W., Vashista, P., 1983. In: Lundquist, S., March, N.H. (Eds.), Theory of Inhomogeneous Electron Gas. Plenum, New York.

Kryanko, E.S., Ludeña, E.V., 2014. Density functional theory: foundations reviewed. Phys. Rep 544, 123–239.

Kumar, A., Meath, W.J., 1984. Pseudo-spectral dipole oscillator-strength distributions for SO_2, CS_2 and OCS and values of some related dipole-dipole and triple-dipole dispersion energy constants. Chem. Phys. 91, 411–418.

Kumar, A., Meath, W.J., 1985. Pseudo-spectral dipole oscillator strengths and dipole-dipole and triple-dipole dispersion energy coefficients for HF, HCl, HBr, He, Ne, Ar, Kr and Xe. Mol. Phys. 54, 823–833.

Kumar, A., Thakkar, A.J., 2010. Dipole oscillator strength distributions with improved high-energy behavior: Dipole sum rules and dispersion coefficients for Ne, Ar, Kr, and Xe revisited. J. Chem. Phys. 132, 074301.

Latajka, Z., Scheiner, S., 1988. Structure, energetics and vibrational spectra of H-bonded systems. Dimers and trimers of HF and HCl. Chem. Phys. 122, 413–430.

Leonard, P.J., Barker, J.A., 1975. In: Eyring, H., Henderson, D. (Eds.), Theoretical Chemistry: Advances and Perspectives, Vol. 1. Academic Press, London.

Leonhard, K., Deiters, U.K., 2002. Monte Carlo simulations of nitrogen using an *ab initio* potential. Mol. Phys. 100, 2571–2585.

Li, J., Zhou, Z., Sadus, R.J., 2007. Role of nonadditive forces on the structure and properties of liquid water. J. Chem. Phys. 127, 154509.

Li, J., Zhou, Z., Sadus, R.J., 2008. Parallel algorithms for molecular dynamics with induction forces. Comput. Phys. Comm. 178, 384–392 (2008).

Lybrand, T.P., Kollman, P.A., 1985. Water-water and water-ion potential functions including terms for many body effects. J. Chem. Phys. 83, 2923–2933.

Maitland, G.C., Smith, E.B., 1971. The intermolecular potential of argon. Mol. Phys. 22, 861–868.

Maitland, G.C., Rigby, M., Smith, E.B., Wakeham, W.A., 1981. Intermolecular Forces: Their origin and Determination. Clarendon Press, Oxford.

Marcelli, G., Sadus, R.J., 1999. Molecular simulation of the phase behavior of noble gases using accurate two-body and three-body intermolecular potentials. J. Chem. Phys. 111, 1533–1540.

Marcelli, G., Sadus, R.J., 2000. A link between the two-body and three-body interaction energies of fluids from molecular simulation. J. Chem. Phys. 112, 6382–6385.

March, N.H., 1992. Electron Density Theory of Atoms and Molecules. Academic Press, London.

Marx, D., Hutter, J., 2009. Ab Initio Molecular Dynamics: Basic Theory and Advanced Methods. Cambridge University Press, Cambridge.

Mas, E.M., Bukowski, R., Szalewicz, K., Groenenboom, G.C.S., Wormer, P.E., van der Avoird, A., 2000. Water pair potential of near spectroscopic accuracy. I. Analysis of potential surface and virial coefficients. J. Chem. Phys. 113, 6687–6701.

Matsuoka, O., Clementi, E., Yoshimine, M., 1976. CI study of the water dimer potential surface. J. Chem. Phys. 64, 1351–1361.

McDowell, S.A.C., 1996. Nonadditive interactions in the H_2 trimer. J. Chem. Phys. 105, 4180–4184.

McDowell, S.A.C., Kumar, A., Meath, W.J., 1996. On the anisotropy of the triple-dipole dispersion energy for interactions involving linear molecules. Mol. Phys. 87, 845–858.

McLean, A.D., Liu, B., Barker, J.A., 1988. Ab initio calculation of argon-argon potential. J. Chem. Phys. 89, 6339–6347.

Meath, W.J., Aziz, R.A., 1984. On the importance and problems in the construction of many-body potentials. Mol. Phys. 52, 225–243.

Miyano, Y., 1994. An effective triplet potential for argon. Fluid Phase Equilib 95, 31–43.

Møller, C., Plesset, M.S., 1934. Note on an approximation treatment for many-electron systems. Phys. Rev. 46, 618–621.

Monson, A.P., Rigby, M., Steele, W.A., 1983. Non-additive energy effects in molecular liquids. Mol. Phys. 49, 893–898.

Mutō, Y., 1943. On the forces acting between nonpolar molecules. J. Phys.-Math. Soc. Jpn. 17, 629–631. Available (in Japanese) from: https://doi.org/10.11429/subutsukaishi1927.17.10-11-12_629 [This paper is most often incorrectly cited in the literature as Muto, Y. (1943). *Proc. Phys. Math. Soc. Japan* **17**, 629].

Nasrabad, A.E., Laghaei, R., Deiters, U.K., 2004. Prediction of the thermophysical properties of pure neon, pure argon, and the binary mixtures neon−argon and argon−krypton by Monte Carlo simulation using ab initio potentials. J. Chem. Phys. 121, 6423−6434.

Niesar, U., Corongiu, G., Clementi, E., Keller, G.R., Bhattacharya, D.K., 1990. Molecular dynamics simulations of liquid water using the NCC ab initio potential. J. Phys. Chem. 94, 7949−7956.

O'Shea, S.F., Meath, W.J., 1974. Charge-overlap effects in the non-additive triple-dipole interaction. Mol. Phys. 28, 1431−1439.

O'Shea, S.F., Meath, W.J., 1976. On the validity of the triple-dipole interaction as a representation of non-additive intermolecular forces. Mol. Phys. 31, 515−528.

Oakley, M.T., Wheatley, R.J., 2009. Additive and nonadditive models of vapor-liquid equilibrium in CO_2 from first principles. J. Chem. Phys. 130, 034110.

Orio, M., Pantazis, D.A., Neese, F., 2009. Density functional theory. Photosyn. Res. 102, 443−453.

Parr, R.G., Yang, W., 1990. Density Functional Theory. Springer-Verlag, Berlin.

Parson, J.M., Siska, P.E., Lee, Y.T., 1972. Intermolecular potentials from crossed-beam differential elastic scattering measurements. IV. Ar + Ar. J. Chem. Phys. 56, 1511−1516.

Patkowski, K., Szalewicz, K., 2010. Argon pair potential at basis set and excitation limits. J. Chem. Phys. 133, 094304.

Payne, M.C., Teter, M.P., Allan, D.C., 1990. Car-Parrinello methods. J. Chem. Soc. Faraday Trans. 86, 1221−1226.

Payne, M.C., Teter, M.P., Allan, D.C., Arias, T.A., Joannopoulos, J.D., 1992. Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients. Rev. Mod. Phys. 64, 1045−1097.

Pérez-Jordá, J.M., Becke, A.D., 1995. A density-functional study of van der Waals forces: rare gas diatomics. Chem. Phys. Lett. 233, 134−137.

Raghavachari, K., Trucks, G.W., Pople, J.A., Head-Gordon, M., 1989. A fifth-order perturbation comparison of electron correlations theories. Chem. Phys. Lett. 157, 479.

Rittger, E., 1990a. The chemical potential of liquid xenon by computer simulation. Mol. Phys. 69, 853−865.

Rittger, E., 1990b. Can three-atom potentials be determined from thermodynamic data? Mol. Phys. 69, 867−894.

Rittger, E., 1990c. An empirical three-atom potential for xenon. Mol. Phys. 71, 79−96.

Rosen, P., 1953. The nonadditivity of the repulsive potential of helium. J. Chem. Phys. 21, 1007−1012.

Rowlinson, J.S., 2002. Cohesion: A Scientific History of Intermolecular Forces. Cambridge University Press, Cambridge.

Sadus, R.J., 1992. High Pressure Phase Behaviour of Multicomponent Fluid Mixtures. Elsevier, Amsterdam.

Sadus, R.J., 1996. Monte Carlo simulation of vapour-liquid equilibria in "Lennard-Jones + three-body potential" binary fluid mixtures. Fluid Phase Equilib 116, 289−295.

Sadus, R.J., 1998a. Exact calculation of the effect of three-body Axilrod-Teller interactions on vapour-liquid phase coexistence. Fluid Phase Equilib 144, 351−359.

Sadus, R.J., 1998b. The effect of three-body interactions on the liquid-liquid equilibria of binary fluid mixtures. Fluid Phase Equlib 150, 63−72.

Sadus, R.J., 1998c. The effect of three-body interactions between dissimilar molecules on the phase behaviour of binary mixtures: The transition from vapour-liquid equilibria to type III behaviour. Ind. Eng. Chem. Res. 37, 2977−2982.

Sadus, R.J., 2019. Molecular simulation of orthobaric heat capacities near the critical point. Phys. Rev. E 99, 012139.
Sadus, R.J., Prausnitz, J.M., 1996. Three-body interactions in fluids from molecular simulation: vapor-liquid phase coexistence of argon. J. Chem. Phys. 104, 4784–4787.
Sahoo, S.K., Nair, N., 2016. CPMD/GULP QM/MM interface for modeling periodic solids: implementation and its application in the study of Y-zeolite supported Rh_n clusters. J. Comp. Chem. 37, 1657–1667.
Sandré, E., Pasturel, A., 1997. An introduction to ab-initio molecular dynamics schemes. Mol. Sim. 20, 63–77.
Scheiner, S. (Ed.), 1997. Molecular Interactions: From van der Waals to Strongly Bound Complexes. John Wiley & Sons, Chichester.
Schwerdtfeger, P., Hermann, A., 2009. Equation of state for solid neon from quantum theory. Phys. Rev. B 80, 064106.
Shepard, R., 1987. The multiconfiguration self-consistent field method. Adv. Chem. Phys. 69, 63–200.
Sherwood, A.E., Prausnitz, J.M., 1964. Third virial coefficient for the Kihara, exp-6, and square-well potentials. J. Chem. Phys. 41, 413–428.
Sherwood, A.E., de Rocco, A.G., Mason, E.A., 1966. Nonadditivity of intermolecular forces: effects on the third virial coefficient. J. Chem. Phys. 44, 2984–2994.
Shvab, I., Sadus, R.J., 2013. Intermolecular potentials and the accurate predication of the thermodynamic properties of water. J. Chem. Phys. 139, 194505.
Shvab, I., Sadus, R.J., 2014. Thermodynamic properties and diffusion of water + methane binary mixtures. J. Chem. Phys. 140, 104505.
Shvab, I., Sadus, R.J., 2016. Atomistic water models: aqueous thermodynamic properties from ambient to supercritical conditions. Fluid Phase Equilib. 407, 7–30.
Singh, A.N., Dyre, J.C., Pedersen, U.R., 2021. Solid–liquid coexistence of neon, argon, krypton, and xenon studied by simulations. J. Chem. Phys. 154, 134501.
Smit, B., Hauschild, T., Prausnitz, J.M., 1992. Effect of a density-dependent potential on the phase behaviour of fluids. Mol. Phys. 77, 1021–1031.
Smits, O.R., Jerabek, P., Phal, E., Schwerdtfeger, P., 2020. First-principles melting of krypton and xenon based on many-body relativistic coupled-cluster interaction potentials. Phys. Rev. B. 101, 104103.
Stogryn, D.E., 1970. Pairwise nonadditive dispersion potential for asymmetric molecules. Phys. Rev. Lett. 24, 971–973.
Stone, A.J., 2013. The Theory of Intermolecular Forces, second ed. Oxford University Press, Oxford.
Sun, L., Deng, W.-Q., 2017. Recent developments of first-principles force fields. WIREs Comput. Mol. Sci. 7, 1–15.
Tang, K.T., 1969. Dynamic polarizabilities and van der Waals coefficients. Phys. Rev. 177, 108–114.
Tang, K.T., Toennies, J.P., 1984. An improved simple model for the van der Waals potential based on universal damping functions for the dispersion coefficients. J. Chem. Phys. 80, 3726–3741.
Tang, K.T., Toennies, J.P., Yiu, C.L., 1995. Accurate analytical He-He van der Waals potential based on perturbation theory. Phys. Rev. Lett. 74, 1546–1549.
Trickey, S.B. (Ed.), 1990. Density Functional theory of Many-Fermion Systems. Adv. Quantum Chem., Vol. 21. Academic Press, San Diego.
Tuckerman, M.E., Parrinello, M., 1994a. Integrating the Car-Parrinello equations. I. Basic integration techniques. J. Chem. Phys. 101, 1302–1315.

Tuckerman, M.E., Parrinello, M., 1994b. Integrating the Car-Parrinello equations. II. Multiple time scale techniques. J. Chem. Phys. 101, 1316–1329.

Tuckerman, M.E., Marx, D., Klein, M.L., Parrinello, M., 1996. Efficient and general algorithms for path integral Car-Parrinello molecular dynamics. J. Chem. Phys. 104, 5579–5588.

Ustinov, E.A., 2010. Effect of three-body interactions on Ar adsorption on graphitized carbon black. J. Chem. Phys. 132, 194703.

van der Hoef, M.A., Madden, P.A., 1998. Novel simulation model for many-body multipole dispersion interactions. Mol. Phys. 94, 417–433.

Verma, P., Truhlar, D.G., 2020. Status and challenges of density functional theory. Trends Chem 4, 302–318.

Vlasiuk, M., Sadus, R.J., 2017a. Predicting vapour-liquid phase equilibria with augmented *ab initio* interatomic potentials. J. Chem. Phys. 146, 244504.

Vlasiuk, M., Sadus, R.J., 2017b. *Ab Initio* interatomic potentials and the thermodynamic properties of fluids. J. Chem. Phys. 147, 024505.

Vlasiuk, M., Frascoli, F., Sadus, R.J., 2016. Molecular simulation of the thermodynamic, structural, and vapor-liquid equilibrium properties of neon. J. Chem. Phys. 145, 104501.

Wang, L., Sadus, R.J., 2006. Three-body interactions and solid-liquid Phase equilibria: application of a molecular dynamics algorithm. Phys. Rev. E 74, 031203.

Wang, F.-F., Kumar, R., Jordan, K.D., 2012. A distributed point polarizable force field for carbon dioxide. Theor. Chem. Acc. 131, 1132.

Wells, B.H., 1987. Van der Waals interaction potentials. Non-additivity in neon and argon lattices. Mol. Phys. 61, 1283–1293.

Wells, B.H., Wilson, S., 1985. Van der Waals interaction potentials. Many-body effects. Mol. Phys. 55, 199–210.

Wells, B.H., Wilson, S., 1986. Van der Waals interaction potentials. Many-body effects in Ne_3. Mol. Phys. 57, 21–32.

Williams, D.R., Schaad, L.J., Murrell, J.N., 1967. Deviations from pairwise additivity in intermolecular potentials. J. Chem. Phys. 47, 4916–4922.

Wilson, M., Madden, P.A., 1994. Anion polarization and the stability of layered structures in MX_2 systems. J. Phys.: Condens. Matter 6, 159–170.

Woo, T.K., Margl, P.M., Blöchl, P.E., Ziegler, T., 1997. A combined Car-Parrinello QM/MM implementation for ab initio molecular dynamics simulations of extended systems: application to transition metal catalysis. J. Phys. Chem. B 101, 7877–7880.

Woon, D.E., 1993. Accurate modeling of intermolecular forces: a systematic Møller-Plesset study of the argon dimer using correlation consistent basis sets. Chem. Phys. Lett. 204, 29–35.

Xantheas, S.S., 1995. Ab initio studies of cyclic water clusters $(H_2O)_n$, n = 1-6. III. Comparison of density functional with MP2 results. J. Chem. Phys. 102, 4505–4517.

Yigzawe, T.M., Sadus, R.J., 2013a. Thermodynamic properties of liquid water from a polarizable intermolecular potential. J. Chem. Phys. 138, 044503.

Yigzawe, T.M., Sadus, R.J., 2013b. Intermolecular interactions and the thermodynamic properties of supercritical fluids. J. Chem. Phys. 138, 194502.

Zakharov, I.I., Anufrienko, V.F., Zakharova, O.I., Yashnik, S.A., Ismaguilov, Z.R., 2005. *Ab Initio* calculation of nitogen oxide dimer structure and its anion-radical. J. Struct. Chem. 46, 213–219.

This page intentionally left blank

Chapter 5

Calculating molecular interactions

Calculating the interaction between molecules is at the heart of either a Monte Carlo (MC) or a molecular dynamics (MD) simulation program. In a MC simulation, the configurational energy resulting from the interaction of each molecule with all other molecules is evaluated. This information is used to either accept or reject the current configuration. In contrast, a MD simulation determines the individual forces experienced by each molecule. The force experienced by the molecule is used to determine new molecular coordinates in accordance with the equations of motion. In either case, evaluating the effect of molecular interaction is the most computational timeconsuming step and as such, it governs the order of the overall algorithm.

In principle, for a system of N molecules, the order of the algorithm scales as N^m, where m is the number of interactions that are included. Typically, only interactions between pairs of molecules are considered (Chapter 3) resulting in an algorithm of order N^2. If only the distinguishable interactions are considered, $N(N-1)/2$ calculations are required for each configuration. This is a large computational cost because simulations involve routinely several hundred molecules. However, in practice, the use of computation-saving strategies (detailed below) can sometimes result in an order N algorithm. In many circumstances there is no need to re-evaluate all pair interactions to generate new configurations.

The requirement to evaluate forces means that MD algorithms for pairwise interactions have an intrisinc N^2 dependence. In a MC algorithm not involving either polar or Coulombic interaction, the N^2 calculation is typically performed only once at the start of the simulation to obtain the initial energy of the system. Subsequent MC moves involve updating the energy via an order N calculation. However, these MC moves are typically repeated for N cycles, which means the overall order remains N^2. Many of the computation-saving procedures that are applied to MD algorithms may also potentially benefit MC simulations, particularly if either nonpolar or Coulombic interactions are involved.

Molecular forces can be characterized (Chapters 3 and 4) as either short-range or long-range, and different techniques are required to simulate both types of properties. A long-range force is defined as one that falls off no faster than r^{-d}, where d is the dimensionality of the system. Typically, ion−ion

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00014-9

and dipole−dipole potentials are proportional to r^{-1} and r^{-3}, respectively. Dispersion and repulsion are examples of short-range forces.

Many computation-saving devices can be employed to calculate short-range interactions such as periodic boundary conditions and neighbor lists. However, calculating long-range interaction requires special methods such as the Ewald sum, reaction field, and particle−mesh methods, because the effect of long-range interaction extends well past half the length of the simulation box.

The purpose of this chapter is to provide an overview of algorithms for molecular interactions. Calculating molecular interactions is an example of the many-body problem that is common to many branches of science. Considerable progress has been achieved in developing algorithms for many-body interactions particularly for long-range interactions. Various tree and fast multipole (FMM) algorithms are available, which reduce the computational cost of pairwise interactions to either $O(N\ ln\ N)$ or $O(N)$. Some of these algorithms are not used widely in molecular simulation programs because several thousand molecules are required to make the FMM algorithms competitive computationally with conventional simulation strategies. This is in contrast to their wider use for other applications, such as cosmology, that also tackle N-body calculations. Nonetheless, on going improvements to FMM algorithms, particularly parallel implementations, are likely to lower this efficiency threshold. These developments, when coupled with an increasing desire to study large number of molecules, is likely to result in a greater use of such algorithms for molecular simulation. This is particularly the case for implementations using graphics processing units (Chapter 12).

5.1 Calculation of short-range interactions

5.1.1 Naive energy and force calculations

Let us consider a simple short-range potential, such as the Lennard-Jones (LJ) potential, which was discussed extensively in Chapter 3.

$$u(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{5.1}$$

If intermolecular interaction is restricted to pairs of molecules, the total potential energy (U) is obtained by summing up the contribution of all of the individual pairs.

$$U = \sum_{i>j}^{N-1}\sum_{j=1}^{N} u(r_{ij}) \tag{5.2}$$

The summation in Eq. (5.2) is confined only to distinct pairs, which is usually the most computationally efficient option. Instead, if all pairs were evaluated, the resulting summation would have to be halved to yield the same value as Eq. (5.2).

The force can be obtained by differentiating the potential with respect to intermolecular separation. To calculate the force $\mathbf{f}_i$ on each atom i, we must account for the contribution from all other atoms,

$$\mathbf{f}_i = -\sum_{j\neq i}^{N} \frac{1}{r_{ij}} \left(\frac{\partial u(r_{ij})}{\partial r_{ij}} \right) \mathbf{r}_{ij} = -\sum_{j\neq i}^{N} \frac{w_{ij}}{r_{ij}^2} \mathbf{r}_{ij}, \tag{5.3}$$

where w_{ij} is the virial introduced in Eq. (2.17) and r_{ij} is required in the denominator because force is directed along the interatomic vector

($\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$). A computationally convenient feature of Eq. (5.3) is that, if $u(r_{ij})$ is an even function of r_{ij}, then it is only necessary to determine r_{ij}^2 and not the absolute magnitude of $\mathbf{r}_{ij}$.

It is apparent that the calculation of either energy or force requires a pair of nested loops. The outer loop in the force calculation is required to obtain values of f_i for all $i = 1$ to N molecules. The general procedure is summarized by Algorithm 5.1.

ALGORITHM 5.1 Calculation of either intermolecular forces or energy.

```
loop i ← 1 ... N - 1
   loop j ← i + 1 ... N
      Evaluate rij.
      Evaluate u(rij) or fi.
      Accumulate energy or force.
   end j loop
end i loop
```

The pseudo code in Algorithm 5.1 follows directly from the problem description and illustrates the nesting of loops required for the evaluation of either forces or energy. Care is taken to avoid identical molecules, and a total of $N(N-1)/2$ iterations will be performed. However, this algorithm is normally used with a very important modification. Simulations are performed typically with a few hundred molecules arranged commonly on a cubic lattice. The small sample size means that a large fraction of the molecules can be expected at the surface of the lattice rather than in the bulk. This is undesirable because the forces experienced by the surface molecules are different from the bulk. Periodic boundary conditions are incorporated into Algorithm 5.1 to avoid this problem.

5.1.2 Periodic boundary conditions and minimum image convention

Periodic boundary conditions can be applied by constructing an infinite lattice that replicates the cubic simulation box throughout space. Molecules in any box on this lattice have a mirror image counterpart in all the other boxes. Changes in one box are matched exactly in the other boxes. If a molecule

leaves a box, then its counterpart in a neighboring box enters through the opposite face. Consequently, the central box is devoid of any boundaries, and surface effects are eliminated.

The use of a periodic boundary condition solves the problem of surface molecules, but it poses another potential difficulty for the simulation. The heart of a simulation program is the calculation of either molecular forces or energies. If there are N molecules in the simulation, then to calculate the pairwise force or energy experienced by each other molecule involves a summation of $N-1$ terms. In principle, we are also faced with the impossible task of including the interactions of the infinite array of periodic images. In the case of a short-ranged intermolecular potential, this problem is avoided by invoking the minimum image convention. To calculate the intermolecular interaction of any molecule, we position it at the center of a box with dimensions identical to the simulation box. The central molecule interacts with all molecules whose centers fall with this region, that is, the closest periodic images of the other $N-1$ molecules.

If the origin of the coordinate system is taken as the center of the simulation box of length L, then the algorithms for applying the minimum image convention and periodic boundaries are very simple. All the coordinates lie within the range of ½ L and -½L. Consequently, when the molecule leaves the box, we can focus attention on its mirror image by either adding or subtracting L to its coordinates. This can be easily coded in FORTRAN using the intrinsic anint(x) to determine the nearest integer of x. For example, to add the correct number of box lengths to the molecular coordinates r_{ij} or the pair separation vector, we write:

```
rxij(i) = rxij(i) - boxl  *  anint( rxij(i) / boxl )
```

The original C/C++ programming language did not have a nearest integer function as part of its standard maths library, which meant that a user-defined function was required. An example of such a function (nearestInt) is given in Chapter 11. The corresponding C/C++ statement is:

```
rxij[i] = rxij[i] - boxl * nearestInt(rx[ij], boxl);
```

Alternatively, this can now be achieved using the C/C++ round() function, i.e., `rxij[i] = rxij[i] - boxl * (int) round(rx[ij] / boxl);` The above statements are usually evaluated in a loop, for example, during the evaluation of energies.

Although using a cubic box is overwhelmingly the most common choice for implementing a molecular simulation, other geometries are possible (Allen and Tildesley, 2017) that require the application of different periodic boundary conditions. Examples include the rhombic dodecahedron (Wang and Krumhansl, 1972) and the truncated octahedron (Adams, 1979). Reasons for using different geometries include avoiding simulation-induced artefacts

in specific cases (Reed and Flurchick, 1996; White et al., 2008) or more closely reproducing the behavior that is being investigated. The shape of the simulation box may affect the properties of some protein systems (Wassenaar and Mark, 2006), and helical periodic boundary conditions (Kessler and Bour, 2014) may more closely match the conditions of some biopolymers. The efficient study of elongational flow also requires special boundary conditions (Hunt and Todd, 2003; Todd and Daivis, 1998).

The general procedure for calculating the energy or force using periodic boundary conditions is given in Algorithm 5.2.

ALGORITHM 5.2 Calculation of either intermolecular forces or energy using periodic boundary conditions.

```
loop  i ←1 ... N - 1
   loop j ← i + 1 ... N
      Evaluate rij.
         Convert rij to its  periodic image (rij').
         if (ri' < cutOffDistance)
            Evaluate u(rij') or fi.
            Accumulate energy or force.
         end if
   end j loop
end i loop
```

Algorithm 5.2 differs from Algorithm 5.1 in two important respects. First, the accumulated energy and forces are calculated for the periodic separation distances. Second, only molecules separated by a distance less than the cut-off distance contribute to the calculated energy or forces. The cut-off distance is a consequence of the periodic boundary conditions. It can be any distance up to half the length of the simulation box. A higher cut-off is not permitted because it would violate the minimum image convention. Consequently, Algorithm 5.2 evaluates the properties of a truncated intermolecular potential rather than the full potential. The full contributions of the intermolecular potential can be obtained by using long-range correction terms.

5.1.3 Long-range corrections to periodic boundary calculations

The contribution to a property X of the fluid from the missing long-range part of the potential can be obtained by adding a long-range correction.

$$X_{full} = X_c + X_{lrc} \tag{5.4}$$

The long-range corrections are calculated by assuming that the pair-distribution function is unity beyond the cut-off distance r_c. Using this simplification, the long-range corrections for energy (U), the virial term (W),

and the chemical potential (μ) are obtained (Allen and Tildesley, 2017) from,

$$U_{lrc} = 2\pi N\rho \int_{r_c}^{\infty} r^2 u(r) dr \tag{5.5}$$

$$W_{lrc} = -\frac{2\pi N\rho}{3} \int_{r_c}^{\infty} r^2 w(r) dr \tag{5.6}$$

$$\mu_{lrc} = 4\pi\rho \int_{r_c}^{\infty} r^2 u(r) dr, \tag{5.7}$$

where:

$$w(r) = r\frac{du(r)}{dr} \tag{5.8}$$

If these formulas are applied to the LJ potential, we obtain:

$$U_{lrc} = \frac{8\varepsilon\pi N\rho\sigma^3}{9}\left[\left(\frac{\sigma}{r}\right)^9 - 3\left(\frac{\sigma}{r}\right)^3\right] \tag{5.9}$$

$$W_{lrc} = \frac{32\varepsilon\pi N\rho\sigma^3}{9}\left[\left(\frac{\sigma}{r}\right)^9 - \frac{3}{2}\left(\frac{\sigma}{r}\right)^3\right] \tag{5.10}$$

$$\mu_{lrc} = \frac{16\varepsilon\pi\rho\sigma^3}{9}\left[\left(\frac{\sigma}{r}\right)^9 - 3\left(\frac{\sigma}{r}\right)^3\right] \tag{5.11}$$

Most short-range potentials decay rapidly with increasing intermolecular separations. For example, the LJ potential asymptotes rapidly to zero for distances greater than 2.5 σ. Therefore, $r_c \geq 2.5\ \sigma$ is an appropriate choice for the LJ potential in many situations. The cut-off value of 2.5 σ should be considered as the absolute lower limit for the LJ potential and, in many circumstances, it is prudent to use a larger value. This is particularly the case for the investigation of phase coexistence, which requires much larger cut-off values to obtain consistently reproducible results (Baidakov et al., 2000; Ahmed and Sadus, 2010). Cut-off-induced discrepancies have also been observed for the pressure (Huang et al., 2010) in various ensembles. The widespread ongoing use of a cut-off value of 2.5 σ is arguably an outdated legacy of early simulation studies, which were severely restricted by meager computational resources. This should no longer be a significant limitation for most contemporary simulations.

5.1.4 Neighbor list

The implementation of periodic boundary conditions reduces the computation cost significantly because calculations involving intermolecular separation

greater than the cut-off distance are not included. Verlet (1967) proposed that keeping a periodically updated list of the neighbors of each molecule could reduce computation time further. Instead of searching repeatedly for neighboring molecules, the neighbors of each molecule are found from the list and the interactions are calculated. This avoids searching for molecules that do not contribute to the short-range interaction because they are beyond the cut-off separation. Algorithm 5.3 illustrates the neighbor list method.

Starting with an empty list (Algorithm 5.3, Part 1), each molecule is selected in turn (Algorithm 5.3, Part 1.1), and a search is made for neighbors (Algorithm 5.3, Part 2), which involves evaluating the intermolecular separation (Algorithm 5.3, Part 2.1). If a neighbor is found, it is entered on the list (Algorithm 5.3, Part 2.2). The creation of a neighbor list involves two arrays. The array *list* contains the neighbors of molecule *i*. The array *listEntry* stores the position of the last neighbor of molecule *i* in *list*. For each molecule, the *listEntry* value allows us to index the neighboring molecules. The variable *d* is used to encompass molecules slightly outside the cut-off distance. This provides a buffer for those molecules that are likely to come within the cut-off distance in the next few MC cycles or MD time steps, thereby reducing the update frequency of the neighbor list.

ALGORITHM 5.3 Creation of a Verlet neighbor list.

```
Part 1    topOfList ← 0                                  //start with empty list
Part 1.1  loop i ←1 ... N – 1                            //select molecule i
            listEntry_i ← 0
Part 2      loop j ← i + 1 ... N                         //look for neighbors of i
Part 2.1      Evaluate rx_ij, ry_ij and rz_ij.
              Evaluate periodic images ( rx_ij', ry_ij' and rz_ij')
              r^2 ← rx_ij'^2 + ry_ij'^2 + rz_ij'^2
Part 2.2      if (r^2 < (rCut^2 + d))                    //neighbor found
                topOfList ← topOfList + 1
                listEntry_i ← topOfList                  //position of j on list
                list_topOfList ← j                       //enter j on list
              end if
            end j loop
          end i loop
```

The use of the neighbor list in conjunction with energy/force evaluation is illustrated in Algorithm 5.4. At the start of the simulation, the neighbor list is created. The neighbor listed is subsequently updated periodically (Algorithm 5.4, Part 1) during the course of the simulation (*update* ← *true*). In the force/energy evaluation (Algorithm 5.4, Part 2), the inner loop (Algorithm 5.4, Part 2.1) over molecules of type *j* is replaced by a loop over the neighbors of *I* found in the array *list*, where the value of $listEntry_i$ is the position of the last neighbor of *i* in the *list*. After all the neighbors of *i*

are considered, the value of *last* is updated in preparation for the next *i* molecule.

The use of neighbor lists reduces the computing substantially, particularly for systems with 500 or more molecules (Allen and Tildesley, 2017). An update interval of 10−20 steps is usually sufficient; the optimal number depends on the size of the cut-off distance. Alternatively, the algorithm can be extended to update the neighbor list automatically (Fincham and Ralston, 1981) when the sum of the magnitudes of the two largest displacements exceeds the buffer distance (*d*). Algorithm 5.5 illustrates this latter approach.

ALGORITHM 5.4 Calculation of energies/forces using a neighbor list.

```
Part 1    If (update)
             Create neighbor list (Algorithm 5.3).
             update ← false
          end if
          last ← 0
Part 2    loop i ← 1 ... N - 1
             if (listEntry_i > 0)
Part 2.1        loop m ← last + 1 ... listEntry_i
                   j ← list_m
                   Evaluate rx_ij, ry_ij and rz_ij.
                   Evaluate periodic images ( rx_ij', ry_ij' and rz_ij').
                   r^2 ← rx_ij'^2 + ry_ij'^2 + rz_ij'^2
                   if (r^2 < rCut^2)
                      Evaluate u(r_ij) or f_i.
                      Accumulate energy or force.
                   end if
                end m loop
                last ←listEntry_i
             end if
          end i loop
```

Before each force/energy evaluation (Algorithm 5.5, Part 1), the highest *newMax* and second highest displacements *oldMax* in all directions are evaluated (Algorithm 5.5, Part 1.1) and compared with *d* (Algorithm 5.5, Part 1.2). At the commencement of the force/energy evaluation (Algorithm 5.5, Part 2) if *update* is true, the current coordinates are stored (Algorithm 5.5, Part 2.1), a new list is created (Algorithm 5.5, Part 2.2), and the forces/energies (Algorithm 5.5, Part 2.3) are calculated. Otherwise, the forces/energies are evaluated from the existing list (Algorithm 5.5, Part 2.4). The advantage of this method is that it avoids the overhead of re-evaluating the neighbor list too often.

The importance of the neighbor list is reflected by the continuing efforts made to improve its capabilities (Awile et al., 2012; Gonnet, 2013;

Sutmann and Stegailov, 2006). The common feature of all neighbor list implementations is the need to periodically update the list. This is a source of imprecision that can be expected to increase during the course of the simulation. Therefore when practical, it is particularly useful to compare the results obtained from a neighbor list implementation with a brute-force calculation.

ALGORITHM 5.5 Use of an automatically updated neighbor list.

```
Part 1    Check if displacement has been exceeded:
          newMax ← 0
          oldMax ← 0
Part 1.1  loop i ← 1 ... N
             xDisp ← |rx_i - rxOld_i|
             yDisp ← |ry_i - ryOld_i|
             zDisp ← |rz_i - rzOld_i|
             if (xDisp > newMax)
                oldMax ← newMax
                newMax ← xDisp
             end if
             if (yDisp > newMax)
                oldMax ← newMax
                newMax ← yDisp
             end if
             if (zDisp > newMax)
                oldMax ← newMax
                newMax ← zDisp
             end if
          end i loop
Part 1.2  if (newMax + oldMax > d)
             update ← true
          else
             update ←false
          end if
Part 2    Calculate forces/energies:
          if (update)
Part 2.1     Save current configuration:
             loop i ←1 ... N
                rxOld_i ←rx_i
                ryOld_i ←ry_i
                rzOld_i ←rz_i
             end i loop
Part 2.2     Create new neighbor list (Algorithm 5.3).
Part 2.3     Evaluate forces/energies (Algorithm 5.4).
          else
Part 2.4     Evaluate forces/energies (Algorithm 5.4) from the existing list.
          end if
```

5.1.5 Linked cells

For simulations involving a large number of molecules ($N \geq 1000$), the logical testing of every pair in the system required by the neighbor list becomes increasingly inefficient computationally. Quentrec and Brot (1973) proposed an alternative bookkeeping method that is particularly advantageous for large systems. The method involves creating a linked list of cells. The simulation box is divided into a lattice of $M \times M \times M$ cells. The value of M is chosen such that the length of the cells (L/M) is greater than the cut-off distance of the forces.

A two-dimensional cell has eight neighbors, whereas a three-dimensional cell has twenty-six neighbors. However, if Newton's third law is used to evaluate forces in a MD simulation, only half the neighbors need to be considered. At the start of the simulation, the identities of the cells and their neighbors must be specified. The indexing required in three dimensions is illustrated by Algorithm 5.6. Algorithm 5.6 uses a three-dimensional array to locate the various cells in the simulation box. This involves a triple loop (Algorithm 5.6., Part 1). An index is assigned to each cell in the inner loop (Algorithm 5.6, Part 2). When locating nearest neighbors, care must be taken to account for indices that are outside the scope of the array (Algorithm 5.6, Parts 2.1−2.4). In these cases, a value of -1 can be assigned to *map,* which indicates that there are no neighbors. Fig. 5.1 illustrates the allocation of cell identities to the different cells for the two-dimensional case. According to Algorithm 5.6, the cell numbered 5 is located at $cell_{2,2}$ and the numbers 6, 9, 8, and 7, which correspond to map_{13}, map_{14}, map_{15}, and map_{16}, respectively, identify their four nearest neighbors.

7	8	9
4	5	6
1	2	3

FIGURE 5.1 Illustration of cell numbering for a two-dimensional simulation cell with $M = 3$.

ALGORITHM 5.6 Three-dimensional map of cells and nearest neighbors.

```
Part 1    Triple loop for 3-D cases:
          loop z ←1 ... M
             loop y ←1 ... M
                loop x ← 1 ... M
Part 2          Assign an index to each cell:
                cell(x)(y)(z) ←1 + mod(x - 1 + M, M)
                              + mod(y - 1 + M, M) × M
                              + mod(z - 1 + M, M) × M^2
Part 2.1        Identify half the nearest neighbors of each cell:
                index ← 13 × (cell(x)(y)(z) - 1)
                map(index + 1) ← cell(x+1)(y)(z)
                map(index + 2) ← cell(x+1)(y+1)(z)
                map(index + 3) ← cell(x)(y+1)(z)
Part 2.2        if (x = 1)
                   map(index + 4) ← -1
                else
                   map(index + 4) ← cell(x-1)(y+1)(z)
                end if
Part 2.3        if (x = 1 or z = 1)
                   map(index + 5) ← -1
                   map(index + 6) ← -1
                   map(index + 7) ← -1
                   map(index + 8) ← -1
                else
                   map(index + 5) ← cell(x+1)(y)(z-1)
                   map(index + 6) ← cell(x+1)(y+1)(z-1)
                   map(index + 7) ← cell(x)(y+1)(z-1)
                   map(index + 8) ← cell(x-1)(y+1)(z-1)
                end if
                map(index + 9) ← cell(x+1)(y)(z+1)
                map(index + 10) ← cell(x+1)(y+1)(z+1)
                map(index + 11) ← cell(x)(y+1)(z+1)
Part 2.4        if (x = 1)
                   map(index + 12) ← -1
                else
                   map(index + 12) ← cell(x-1)(y+1)(z+1)
                end if
                map(index + 13) ← cell(x)(y)(z+1)
             end x loop
          end y loop
       end z loop
```

The procedure for creating a linked-cell list is illustrated by Algorithm 5.7. At the outset of the simulation (Algorithm 5.7, Part 1) the length and number of cells are determined. At each time step of the simulation, sorting all the molecules into their appropriate cells creates a linked list of cells. The process involves initializing $head_i$ to zero (Algorithm 5.7, Part 2), allocating particles to cells Algorithm 5.7, Part 3.1), and identifying "head" particles (Algorithm 5.7, Part 3.1).

ALGORITHM 5.7 Creation of a linked-cell list.

```
Part 1    Determine length and number of cells:
          lCell ←int(lBox/rCutOff)
          nCell ← lCell^3

Part 2    Initialise head array to zero:
          loop i ← 1 .... nCell
             head_i ← 0
          end i loop

Part 3    Allocate particles to cells and identify 'head' particles:
          loop i ← 1 ..... nParticles
Part 3.1     cell ← 1 + int( (rx_i + lBox/2) × lCell)
                     + int( (ry_i + lBox/2) × lCell) × lCell
                     + int( (rz_i + lBox/2) × lCell) × lCell × lCell
             list_i ←head_cell
Part 3.2     head_cell ← i
          end i loop
```

Algorithm 5.8 gives an example of using a linked-cell list in a force evaluation. At the outset (Algorithm 5.8, Part 1) the parameters are initialized as in a conventional simulation. The length and number of cells are determined (Algorithm 5.8, Part 2), followed by the creation of a cell map (Algorithm 5.8, Part 3) and a linked-cell list (Algorithm 5.8, Part 4). The evaluation of forces (Algorithm 5.8, Part 5) involves loops over all cells (Algorithm 5.8, Part 5.1), over all molecules in the cell (Algorithm 5.8, Part 5.2), and over neighboring cells (Algorithm 5.8, Part 5.3). Comparing the execution times of simulations, the linked-cell procedure with the neighbor list indicates that the former may be advantageous in some circumstances (Morales, 1996).

ALGORITHM 5.8 Force calculation using a linked-cell list.

```
Part 1    Initialise parameters as in conventional simulation.

Part 2    Determine length and number of cells:
          lCell ←int(lBox/rCutOff)
          nCell ← lCell^3

Part 3    Create cell map (Algorithm 5.6).

Part 4    Create linked-cell list (Algorithm 5.7).

Part 5    Evaluate forces:
Part 5.1  loop cell ← 1 ... nCell                          //loop over all cells
            i ← head_cell
Part 5.2    loop while (i > 0)          // loop over all molecules in the cell
              j ← list_i
              loop while (j > 0)
                Evaluate forces.
                j ← list_j
              end j loop
              jCell0 ←13 × (cell - 1)
Part 5.3      loop neighbor ←1 ... 13          //loop over neighboring cells
                jCell ←map_jCell0 + neighbor
                if (jCell > 0)
                  j ← head_jCell
                else
                  j ← 0
                end if
                loop while (j > 0)
                  Evaluate forces.
                  j ←list_j
                end j loop
                i ← list_i
              end neighbor loop
            end i loop
          end cell loop
```

The basic approach illustrated in Algorithms 5.6–5.8 has been modified to improve its efficiency (Everaers and Kremer, 1994; Fanourgakis et al., 2007; Fomin, 2011; Wang et al., 2007; Welling and Germano, 2011). Watanabe et al. (2011) have provided details of a parallel implementation and it has been combined (Fomin, 2011; Luo et al., 2015) with a look-up table (see Section 5.1.6). The refinements sometime result in a many-fold improvement (Heinz and Hünenberger, 2004; Yao et al., 2004) in computational speed. Traditionally, the

linked-cell list has been implemented for a cubic simulation box, but there are examples of its use in other geometries (Cui et al., 2009). The approach has been modified (Matin et al., 2003) to deal with the special case of elongational flow, which requires nonconventional boundary conditions.

5.1.6 Look-up table

A look-up table is a computation-saving alternative to the direct evaluation of either the intermolecular potential or force. At the start of the simulation, the potential $u(s)$ and force ($\boldsymbol{f} = -2\boldsymbol{r}_{ij}\,(du/ds)$) are calculated at intervals of δs, where $s = r_{ij}^2$. Typically, $\delta s = 0.01 r_m^2$, where r_m is the separation that corresponds to the minimum of the potential. The values of the calculated potentials and/or forces are stored in look-up tables as is illustrated by Algorithm 5.9.

ALGORITHM 5.9 Creation of look-up tables.

```
δs ←0.01 × rPotMin
max ←int(rCutOff/δs)
loop i ←1... max
    s ← i × δs
    rij ←√s
    Evaluate potential (potTable_i).
    Evaluate force (forceTable_i).
end i loop
```

During the simulation, values of $s = r_{ij}^2$ are calculated for each pair of molecules, and the potential and/or force is interpolated for the table. The most widely used interpolation formula is the Newton–Gregory forward difference method (Booth, 1972),

$$u(s) = u_k + \xi(u_{k+1} - u_k) + \frac{1}{2}\xi(\xi - 1)(u_{k+2} - 2u_{k+1} + u_k), \tag{5.12}$$

where:

$$\xi = \frac{s - s_k}{\delta_k} \tag{5.13}$$

Algorithm 5.10 illustrates the use of a look-up table in conjunction with a neighbor list. A look-up table is particularly useful for complicated intermolecular potentials. It has been used (Barker et al., 1971) for the Barker–Fisher–Watts (BFW) potential, which contains 11 adjustable parameters and an exponential term. It is also likely to be particularly useful in the implementation of the molecular mechanics force fields described in Chapter 3. In this context, several tables could be used to store values of the different contributions from both inter- and intramolecular interactions. Improvements to the basic algorithm have been proposed (Andrea et al., 1983), which reduce the storage requirements of the

tables. The use of look-up tables in conjunction with an Ewald sum has been described (Danese et al., 1998).

ALGORITHM 5.10 Calculation of energies using a neighbor list and a look-up table.

```
if (update)
    Create neighbor list.
    update ← false
end if
last ← 0
loop i ← 1 ... N - 1
   if (listEntry_i > 0)
      loop m ← last + 1 ... listEntry_i
         j ← list_m
         Evaluate rx_ij, ry_ij and rz_ij.
         Evaluate periodic images (rx_ij', ry_ij' and rz_ij').
         s ← rx_ij'^2 + ry_ij'^2 + rz_ij'^2
         if (s < rCut^2)
            k ←int(s/δs)                          // Look-up table index
            ξ ← s/δs - k
            pot ← potTable_k + ξ × (potTable_k+1 − potTable_k)
                  + 0.5 × (ξ^2 - ξ) ×
                  (potTabke_k+2 - 2 × potTable_k+1 + potTable_k)
            Accumulate energy.
         end if
      end m loop
      last ←listEntry_i
   end if
end i loop
```

The linked-cell method is used frequently as part of the particle−particle particle−mesh (PPPM) algorithm (see subsequent discussion) and improvements are being made continually. Fincham (1994) has reported a small-cell version of the linked-cell method that can be used to improve the efficiency of calculating the real space contribution of the Ewald sum (see subsequent discussion). Milchev et al. (1993) have applied the linked-cell algorithm in a MC simulation of the properties of polymers. Grest et al. (1989) reported MD code using a vectorized linked-cell algorithm. Pinches et al. (1996) have implemented a parallel version of the linked-cell algorithm.

5.1.7 Comparison of computational efficiency

The computation time t of the brute-force evaluation of all distinct pairs is,

$$t = \alpha \frac{N(N-1)}{2}, \tag{5.14}$$

where α is a constant. In contrast, when a Verlet list is used, the calculation of energy is proportional to the number of molecules, and the N^2 term only arises when the list is updated. Consequently,

$$t = \gamma N + \lambda \frac{N(N-1)}{2}, \tag{5.15}$$

where γ and λ are constants associated with the energy/force evaluation and with the frequency of updating the Verlet list, respectively. It is evident that if λ is small, the Verlet list can reduce the computational complexity of the algorithm. Typically, algorithms, which use a Verlet, list scale as $N^{3/2}$. In contrast, the two main processes involved in the linked-cell method, namely, energy evaluation and creation of the cell list, both have a linear dependence on the number of molecules,

$$t = \kappa N + \tau N \tag{5.16}$$

where κ and τ are constants associated with energy/force evaluation and cell list creation, respectively.

It is apparent that using either the Verlet list or the linked-cell list can potentially have considerable computational advantages. However, it should be noted that the benefits of the list methods for a system with few molecules ($N = 1000$) may be outweighed by the computational overhead of creating and maintaining a list. It is also clear that automatic updates of the Verlet list are essential to maximize computational efficiency. The list methods are likely to be more beneficial for MD simulations rather than MC. For example, Frenkel and Smit (2002) have compared the computation times for MC simulations using various methods. They concluded that the simple brute-force algorithm was superior to the list methods for simulating the properties of a LJ fluid of 200−500 molecules. However, the cell method is vastly superior when a large number of molecules are involved, particularly at low densities. For 10,000 particles, Frenkel and Smit (2023) reported an 18-fold improvement using the cell method compared with a brute-force evaluation of all distinct pairs.

5.2 Calculation of long-range interactions

The use of periodic boundary conditions is a standard feature of the molecular simulation of bulk properties using short-range intermolecular potentials. It avoids inaccuracies caused by surface effects and it also provides the additional benefit that computation time is reduced significantly by using cut-off corrections instead of calculating the interactions of all periodic images. However, long-range interactions such as Coulombic or dipolar interactions typically exert an influence that is much greater than half the length of the simulation box. The decay of the Coulombic [Eq. (3.51)], Stockmayer [Eq. (3.80)], and LJ [Eq. (3.25)] potentials as a function of intermolecular separation are compared in Fig. 5.2. It is apparent from Fig. 5.2 that the Coulombic and, to a much lesser extent, the Stockmayer

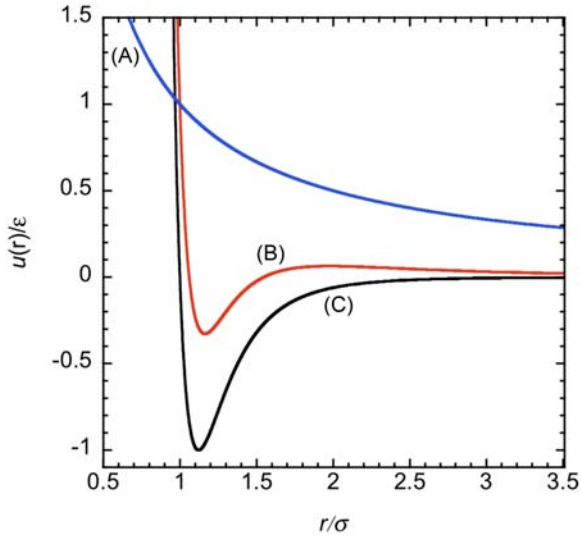


FIGURE 5.2 Comparison of the decay of (A) Coulombic ($q_a q_b/\varepsilon\sigma = 1$), (B) Stockmayer ($\mu_a\mu_b/\varepsilon\sigma^3 = 1$, $\theta_a = 90$ degrees, $\theta_b = 90$ degrees and $\phi_a = \phi_b$), and (C) LJ (u/ε) reduced potentials (non-SI versions) as a function of reduced intermolecular separation.

potentials, are nonzero for quite large intermolecular separations. The obvious solution is to use a very large simulation box; however, this approach is prohibitive computationally. Instead, various computational procedures have been devised and some of the alternatives are discussed below. It should be noted that calculations of electrostatic interactions are prone to a greater degree of uncertainty and error (Coelho and Cheary, 1997; Bogusz et al., 1998; Resat and McCammon, 1998) than simulations involving nonpolar molecules.

5.2.1 The Ewald sum

The Ewald sum has arguably become the default method for the evaluation of electrostatic interactions. It deals specifically with the interactions between charges, which makes it particularly suitable for ionic interactions. However, it can also be applied more generally to nonpolar systems because nonpolar interactions between molecules (Chapter 3) are routinely modeled by assigning partial charges to the constituent atoms. The Ewald sum is considered to be the most rigorous method available, as it has been described in considerable detail elsewhere (Allen and Tildesley, 2017; Rapaport, 2004; Tuckerman, 2010). Many efforts have been made to both make improvements and apply it to an increasing variety of situations (Wells and Chaffee, 2015; Tornberg, 2016; Lindbo and Tornberg, 2012; Villarreal and Montich, 2005; Skeel, 2016).

First, consider a central simulation box of length L. Using the non-SI version of Eq. (3.51), the energy from electrostatic interactions between N molecules confined in the central box can be obtained from:

$$U = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{q_i q_j}{r_{ij}} \tag{5.17}$$

Six periodic images of the central box can be constructed at a distance L from the central box with coordinates ($\boldsymbol{r}_{\text{box}}$) specified by (0, 0, L), (0, 0, $-L$), (0, L, 0), (0, $-L$, 0), (L, 0, 0), and ($-L$, 0, 0). Eq. (5.17) must be modified to include the contribution of Coulombic interactions between charges in the central box and all images of all particles in the six surrounding boxes.

$$U = \frac{1}{2}\sum_{nbox=1}^{6}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{q_i q_j}{\left|\mathbf{r}_{ij}+\mathbf{r}_{box}\right|} \tag{5.18}$$

We can construct a further five periodic images for each of these new boxes and they in turn, also have periodic images. A sphere of simulation boxes is constructed by continual repetition of this process. In general, the energy of interaction of an ion and all of its periodic images can be obtained from,

$$U = \frac{1}{2}\sum_{\mathrm{n}=0}^{\infty}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{q_i q_j}{\left|\mathbf{r}_{ij}+\mathbf{n}\right|}, \tag{5.19}$$

where the summation over $\mathbf{n}$ is taken over all lattice points, $\mathbf{n} = (n_x L, n_y L, n_z L)$ with n_x, n_y, n_z integers. The prime indicates the omission of $i = j$ for $\mathbf{n} = 0$. Eq. (5.19) is the Ewald sum (Ewald, 1921).

In practice, Eq. (5.19) is not evaluated directly because it converges very slowly. Instead, Eq. (5.19) is converted (de Leeuw et al., 1980; Heyes, 1981) into two series each of which converges more rapidly. The assumption is made that each charge is surrounded by a neutralizing charge distribution of equal magnitude but of opposite sign. Typically, a Gaussian charge distribution is used.

$$\rho_i(\mathrm{r}) = \frac{q_i\kappa^3}{\pi^{\frac{3}{2}}}\exp(-\kappa^2 r^2) \tag{5.20}$$

The summation over point charges becomes a summation of the interaction between charges plus the neutralizing distributions. The new summation is often referred to as the "real space" summation. The real space energy is,

$$U_{real-s} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\mathrm{n}=0}^{\infty}\frac{q_i q_j erfc\left(\kappa\left|\mathbf{r}_{ij}+\mathbf{n}\right|\right)}{\left|\mathbf{r}_{ij}+\mathbf{n}\right|}, \tag{5.21}$$

where *erfc* is the complementary error function (Abramowitz and Stegun, 1972).

$$erfc(x) = \frac{2}{\sqrt{\pi}}\int_{x}^{\infty}\exp(-t^2)dt \tag{5.22}$$

Most programming languages implement the error function as part of their mathematical library, which means that it is not a programming impediment. The value of κ is chosen such that only terms corresponding to $\mathbf{n} = 0$ (interactions involving charges in the central box only) make a contribution. Typically, $\kappa = 5/L$ (Woodcock and Singer, 1971).

A second charge distribution is required, which counteracts exactly the neutralizing distribution. The summation is performed in "reciprocal space,"

$$U_{recip-s} = \frac{1}{2}\sum_{k\neq 0}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{4\pi q_i q_j}{k^2 L^3}\exp\left(\frac{k^2}{4\kappa^2}\right)\cos(\mathbf{k}\cdot\mathbf{r}_{ij}), \tag{5.23}$$

where $\mathbf{k} = 2\pi\mathbf{n}/L^2$ are the reciprocal vectors. Typically, 100−200 $\mathbf{k}$ vectors are used (Woodcock and Singer, 1971). Evaluating a triple summation is expensive computationally. Instead, complex algebra (Heyes, 1981; Anastasiou and Fincham, 1982) is used to replace the sum involving i and j with a single summation. The result is:

$$U_{recip-s} = \frac{1}{2}\sum_{k\neq 0}\frac{4\pi}{k^2 L^3}\exp\left(\frac{k^2}{4\kappa^2}\right)\left|\sum_{i=1}^{N} q_i \exp(i\mathbf{k}\cdot\mathbf{r}_i)\right|^2 \tag{5.24}$$

The summation of Gaussian functions in real space also includes the interaction of each Gaussian with itself. The effect of this interaction on the energy is:

$$U_{Gauss} = -\frac{\kappa}{\sqrt{\pi}}\sum_{k=1}^{N} q_i^2 \tag{5.25}$$

The medium surrounding the sphere of simulation boxes must also be considered in the calculation because the sphere can interact with its surroundings. No correction is required if the surrounding medium is a good conductor characterized by infinite relative permittivity ($\varepsilon_0 = \infty$). However, if the surrounding medium is a vacuum ($\varepsilon_0 = 1$), the following correction applies.

$$U_{corr} = \frac{2\pi}{3L^3}\left|\sum_{i=1}^{N} q_i \mathbf{r}_i\right|^2 \tag{5.26}$$

Consequently, the final expression for Coulombic interaction is:

$$U = U_{real-s} + U_{recip-s} + U_{Gauss} + U_{corr} \tag{5.27}$$

The Ewald sum can be applied to dipole−dipole interactions (Adams and McDonald, 1976; de Leeuw et al., 1980). The contributions to Eq. (5.27) for dipole−dipole interactions are,

$$U_{real-s} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{|\mathrm{n}|=0}^{\infty}(\boldsymbol{\mu}_i\cdot\boldsymbol{\mu}_j)B(\mathbf{r}_{ij}+\mathbf{n}) - (\boldsymbol{\mu}_i\cdot\mathbf{r}_{ij})C(\mathbf{r}_{ij}+\mathbf{n}) \tag{5.28}$$

$$U_{recip-s} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\mathrm{k}\neq 0}\frac{4\pi(\boldsymbol{\mu}_i\cdot\mathbf{k})(\boldsymbol{\mu}_j\cdot\mathbf{k})}{\kappa^2 L^3}\exp\left(\frac{k^2}{4\kappa^2}\right)\cos(\mathbf{k}\cdot\mathbf{r}_{ij}) \tag{5.29}$$

$$U_{Gauss} = -\frac{2\kappa}{3\sqrt{\pi}}\sum_{i=1}^{N}\boldsymbol{\mu}_i^2 \tag{5.30}$$

$$U_{corr} = \frac{2\pi}{3L^3}\sum_{i=1}^{N}\sum_{j=1}^{N}\boldsymbol{\mu}_i\cdot\boldsymbol{\mu}_j \tag{5.31}$$

where the summations over i and j are for dipoles in the central box and:

$$B(r) = \frac{erfc(\kappa r)}{r^3} + \frac{2\kappa}{\sqrt{\pi}r^2}\exp(-\kappa^2 r^2) \tag{5.32}$$

$$C(r) = \frac{3erfc(\kappa r)}{r^5} + \frac{2\kappa}{\sqrt{\pi}r^2}\left(2\kappa^2 + \frac{3}{r^2}\right)\exp(-\kappa^2 r^2) \tag{5.33}$$

The triple summation in Eq. (5.29) can be avoided by using complex algebra resulting in:

$$U_{recip-s} = \frac{1}{2}\sum_{k\neq 0}\frac{4\pi}{k^2 L^3}\exp\left(\frac{k^2}{4\kappa^2}\right)\left|\sum_{i=1}^{N}(\boldsymbol{\mu}_j\cdot\mathbf{k})\exp(i\mathbf{k}\cdot\mathbf{r}_i)\right|^2 \tag{5.34}$$

The implementation of the Ewald sum for ions is illustrated by Algorithms 5.11 and 5.12 (Smith, 1992). Before calculating the reciprocal space sum (Algorithm 5.11), the forces of the molecules (Algorithm 5.11, Part 1) and exponential **k** vector terms (Algorithm 5.11, Part 2) are initialized. A feature of the practical implementation of algorithms to calculate the reciprocal space contribution is inner loops (Algorithm 5.11, Parts 3.1 and 3.2) with variable terms ($\mathrm{m}_{\min}$ and $\mathrm{n}_{\min}$). This exploits the inversion symmetry of the reciprocal lattice, halving the required computation time.

The implementation of the real space terms is straightforward (Algorithm 5.12). The Gaussian (Algorithm 5.12, Part 1) and correction terms (Algorithm 5.12, Part 2) are determined prior to the evaluation of the real space contribution. The calculation of the real space contribution (Algorithm 5.12, Part 3) involves a relatively minor change to the normal force evaluation algorithms. An improvement in computational efficiency can be obtained by incorporating a neighbor list (Algorithm 5.4). Smith (1992) has discussed an improved replicated data algorithm for the Ewald sum that exploits parallel computers. Sperb (1998) has reported an alternative to the Ewald sum based on redefining particle identities. Details are also available (Lekner, 1998) of the calculation of Coulombic interactions in an orthorhombic unit cell.

ALGORITHM 5.11 Calculation of the reciprocal space contribution in the Ewald sum.

Part 1 Initialise forces on molecules:
loop $j \leftarrow 1 \ldots N$
$fx_j \leftarrow 0$
$fy_j \leftarrow 0$
$fz_j \leftarrow 0$
end *j* **loop**

Part 2 Initialise exponential k vector terms:
loop $k \leftarrow -kMax \ldots kMax$
loop $j \leftarrow 1 \ldots N$
$ex_{kj} \leftarrow \exp(i2\pi k/L)$
$ey_{kj} \leftarrow \exp(i2\pi k/L)$
$ez_{kj} \leftarrow \exp(i2\pi k/L)$
end *j* **loop**
end *k* **loop**

Part 3 Calculate reciprocal space terms:
$mMin \leftarrow 0$
$nMin \leftarrow 1$
loop $l \leftarrow 0 \ldots k$
$r_l \leftarrow 2\pi l/L$
Part 3.1 **loop** $m \leftarrow mMin \ldots kMax$
$r_m \leftarrow 2\pi m/L$
loop $j \leftarrow 1 \ldots N$
$exy_j \leftarrow ex_{lj} \times ey_{mj}$
end *j* **loop**
Part 3.2 **loop** $n \leftarrow nMin \ldots kMax$
$r_n \leftarrow 2\pi n/L$
$k^2 \leftarrow l^2 + m^2 + n^2$
if $k^2 \leq k_{max}^2$
$r_k^2 \leftarrow r_l^2 + r_m^2 + r_n^2$
$A_k \leftarrow \exp(r_k^2/4\kappa^2)/r_k^2$
loop $j \leftarrow 1 \ldots N$
$exyz_j \leftarrow C_j \times exy_j \times ez_{nj}$
end *j* **loop**
$QSum \leftarrow 0$
loop $j \leftarrow 1 \ldots N$
$QSum \leftarrow QSum + exyz_j$
end *j* **loop**
$E \leftarrow A_k |QSum|^2/L^3$
loop $j \leftarrow 1 \ldots N$
$fx_j \leftarrow fx_j + \text{real}(2i \times A_k \times QSum \times exyz_j \times r_l)/L^3$
$fy_j \leftarrow fy_j + \text{real}(2i \times A_k \times QSum \times exyz_j \times r_m)/L^3$
$fz_j \leftarrow fz_j + \text{real}(2i \times A_k \times QSum \times exyz_j \times r_n)/L^3$
end *j* **loop**
end if
end *n* **loop**
$nMin \leftarrow -kMax$
end *m* **loop**
$mMin \leftarrow -kMax$
end *l* **loop**

ALGORITHM 5.12 Calculation of the real space contribution in the Ewald sum.

Part 1 Calculate Gaussian term using eq. (5.25).

Part 2 Calculate correction term:
if ($\varepsilon_0 = \infty$)
  Obtain E_{corr} form eq. (5.26).
else
  $E_{corr} \leftarrow 0$
end if

Part 3 Calculate Real Space Term:
loop $i \leftarrow 1$ *to* $N - 1$
  loop $j \leftarrow j + 1$ *to* N
    Evaluate rx_{ij}, ry_{ij} and rz_{ij}.
    Evaluate periodic images (rx_{ij}', ry_{ij}' *and* rz_{ij}').
    $r^2 \leftarrow rx_{ij}'^2 + ry_{ij}'^2$ t $rz_{ij}'^2$
    if ($r^2 < rCut^2$)
      $r \leftarrow \sqrt{r^2}$
      $E \leftarrow E + E_{corr} + E_{Gauss} + C_iC_j\text{erfc}(\kappa r)/r$
      $\gamma \leftarrow C_iC_j[\text{erfc}(\kappa r)/r + exp(-(\kappa r)^2)]/r^2$
      $fx_i \leftarrow fx_i - \gamma rx_{ij}$
      $fy_i \leftarrow fy_i - \gamma ry_{ij}$
      $fz_i \leftarrow fz_i - \gamma rz_{ij}$
      $fx_j \leftarrow fx_j + \gamma rx_{ij}$
      $fy_j \leftarrow fy_j + \gamma ry_{ij}$
      $fz_j \leftarrow fz_j + \gamma rz_{ij}$
    end if
  end *j* **loop**
end *i* **loop**

5.2.2 The Wolf approximation of the Ewald sum

The calculation of the reciprocal part of the contribution to energy is the most computational expense aspect of the Ewald sum. An alternative, proposed by Wolf (Wolf, 1992; Wolf et al., 1999) is to truncate the calculation beyond a certain distance (R_c). For ionic systems, this means, the calculation of energy becomes (Avendaño and Gil-Villegas, 2006),

$$U(R_c) \approx \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j\neq i \\ (r_{ij}<R_c)}} q_iq_j\left(\frac{erfc(\alpha r_{ij})}{r_{ij}} - \frac{erfc(\alpha R_c)}{R_c}\right) - \left(\frac{erfc(\alpha R_c)}{2R_c} + \frac{\alpha}{\sqrt{\pi}}\right)\sum_{i=1}^{N} q_i^2 \quad , \tag{5.35}$$

Where α is a tuning parameter that is used to restrict the contribution of the real part of the potential to the first cell of the Ewald system. In contrast to the Ewald

sum, using Eq. (5.35) means a relatively straightforward adaptation of Algorithm 5.2. The key advantage of the Wolf method is that it is computationally much less expensive (Sadeghifar et al., 2012) than the Ewald sum. The disadvantage of the Wolf method is that the values of both α and R_c are not known beforehand. Instead, they are typically obtained by matching the results of Ewald sum calculations. Modifications have been proposed (Ma and Garofalini, 2005; Waibel et al., 2019), and it has been applied successfully to a broad range of situations (Gdoutos et al., 2010; Mendoza et al., 2008).

5.2.3 The reaction field method

The reaction field method is a relatively simple procedure for accounting for dipole interactions. Every molecule is at the center of an imaginary sphere the size of which is determined by the cut-off radius r_c. The sphere is inside a homogeneous medium with a characteristic dielectric constant ε_s. The energy resulting from interaction of the central molecule with all other molecules within the sphere are calculated fully.

$$U(short)_i = \sum_{j;r \le r_c} u(\boldsymbol{r}_{ij}) \tag{5.36}$$

This corresponds to the short-range contribution of dipolar interaction. The long-range contribution to the energy results from interaction with the medium beyond the sphere. The contribution to energy caused by the surrounding medium is,

$$U(long)_i = -\frac{\mu_i(\varepsilon_s - 1)}{2\varepsilon_s + 1}\left(\frac{1}{r_c^3}\right)\sum_{j;r_{ij} \le r_c} \boldsymbol{\mu}_j \tag{5.37}$$

where μ_i is the dipole moment of the central molecule and μ_j is the dipole moment of neighboring molecules within the sphere. Consequently, the total energy experienced by each molecule is evaluated from:

$$U_i = U(short)_i + U(long)_i \tag{5.38}$$

It should be noted that a discontinuity in energy is generated when molecules enter or leave the sphere. Therefore, it is usual practice to taper the interactions at the sphere's surface (Adams et al., 1979) by applying a weighting factor $f(r_{ij})$ that approaches zero continuously at $r_{ij} = r_c$. Several different tapering functions have been proposed (Adams et al., 1979; Andrea et al., 1983). In many cases, a simple linear function is appropriate,

$$f(r_{ij}) = \begin{cases} 1 & r_{ij} < r_t \\ \dfrac{r_c - r_{ij}}{r_c - r_t} & r_t \le r_{ij} \le r_c \\ 0 & r_c < r_{ij} \end{cases}, \tag{5.39}$$

where $r_t = 0.95 r_c$.

The advantage of the reaction field method is that it can be incorporated easily into a conventional MD or MC simulation with a minimal increase in computation time. The necessary changes are illustrated in Algorithm 5.13, which represents a relatively small modification to Algorithm 5.2. The main disadvantages of the method are the potential discontinuity in energy arising from molecules entering and leaving the sphere, and the need to know the external dielectric constant in advance. The external dielectric constant can be estimated from the following relationship (Allen and Tildesley, 2017),

$$\frac{1}{\varepsilon - 1} = \frac{3kT}{4\pi\rho\mu^2 g(\varepsilon_s)} - \frac{1}{2\varepsilon_s + 1}, \tag{5.40}$$

where ε is the relative permittivity and $g(\varepsilon_s)$ is related to the fluctuation of the total dipole moment of the central box.

$$g(\varepsilon_s) = \frac{\left\langle \left| \sum_{i=1}^{N} \boldsymbol{\mu}_i \right|^2 \right\rangle - \left\langle \left| \sum_{i=1}^{N} \boldsymbol{\mu}_i \right| \right\rangle^2}{N\mu^2} \tag{5.41}$$

Alternatively, the reaction field algorithm itself can be used to calculate the dielectric constant (Essex, 1998).

ALGORITHM 5.13 Calculation of dipole interactions using the reaction field method.

```
EShort ← 0
ELong ← 0
loop i ←1 to N - 1
   loop j ← i + 1 to N
      Evaluate rxij, ryij and rzij.
      Evaluate periodic images (rxij', ryij' and rzij').
      r2 ← rxij'2 + ryij'2 + rzij'2
      if (r < cutOffDistance)
         Calculate u(rij) to determine Ei.
         Determine weighting factor f.
         EShort ← EShort + f × Ei
         ELong, ← ELong + f × μi × μj
      end if
   end j loop
end i loop
E ←Eshort + ELong × (εs - 1)/(rc3(2εs + 1))
```

The reaction field method has been applied widely to dipolar fluids (Alder and Pollock, 1981; de Leeuw et al., 1986). However, it cannot be used for ionic fluids without modification because the assumption that the average charge density is zero beyond the cut-off radius excludes the interactions of the central particle with ions. Barker (1994) proposed a method that can be used for fluids

containing both ions and dipoles. According to Barker (1994), the energy resulting from interaction between dipoles (identified by i and j) and ions (identified by α and β) can be calculated as:

$$U = \sum_i v_i + \sum_{i<j}{}' v_{ij} + \sum_\alpha w_\alpha + \sum_{\alpha<\beta}{}' w_{\alpha\beta} + \sum_{i\alpha}{}' x_{i\alpha} \tag{5.42}$$

Eq. (5.42) includes contribution from self-dipole, self-ion, unlike-dipole, unlike-ion, and ion−diploe interactions. The primed summations are performed over all pairs with separation less than r_c. The terms v, w, and x are defined by the equations,

$$v_i = -\frac{d_1}{2r_c^3}\mu_i^2 \tag{5.43}$$

$$v_{ij} = -\frac{d_1}{r_c^3}\mu_i\mu_j \tag{5.44}$$

$$w_\alpha = -\frac{d_0}{2r_c^3}q_\alpha^2 \tag{5.45}$$

$$w_{\alpha\beta} = -\frac{d_0}{r_c}q_\alpha q_\beta - \frac{d_1}{r_c^3}q_i q_j r_{\alpha\beta}^2 \tag{5.46}$$

$$x_{ij} = -\frac{d_1}{r_c^3}q_\alpha r_{i\alpha}\mu_j, \tag{5.47}$$

where:

$$d_0 = \frac{\varepsilon_s(1+\kappa r_c)-1}{\varepsilon_s(1+\kappa r_c)}, \tag{5.48}$$

$$d_1 = \frac{\varepsilon_s(2+2\kappa r_c+(\kappa r_c)^2)-2(1+\kappa r_c)}{\varepsilon_s(2+2\kappa r_c+(\kappa r_c)^2)+1+\kappa r_c}, \tag{5.49}$$

$$\kappa^2 = \frac{4\pi e^2}{\varepsilon_s kT}\sum_i n_i z_i^2, \tag{5.50}$$

e is the charge of an electron, n is the bulk concentration, and z is the sign of the charge.

The simplicity of the reaction field method has made it a popular choice for molecular simulations involving dipoles, although application (Baumketner, 2009; Lee, 2020) to ions is also possible via modified schemes as noted previously. Improvements to the basic method are being made continually. Garzon et al. (1994) applied the reaction field to the simulation of vapor−liquid equilibria of dipolar fluids. They concluded that the reaction field dielectric constant does not affect either the coexistence properties or the structure of the coexisting phases.

Ruocco and Sampoli (1994) have extended the reaction field method to include polarization effects. The extended procedure provides a convenient method for incorporating both long-range dipolar forces and many-body polarization effects for large numbers of molecules. It has been applied successfully (Ruocco and Sampoli, 1995) to determine the induced dipole of a polarizable model for water. The reaction field has also been applied (Hloucha and Deiters, 1997) to a polarizable model for acetonitrile. Alper and Levy (1993) have proposed a generalized reaction field method for the simulation of liquid water. It is based on a solution of the Poisson equation for the reaction field acting on charges in a spherical cavity that is surrounded by a dielectric continuum. Smith and van Gunsteren (1995) and Omelyan (1997) have also investigated water via a reaction field method. Zhou et al. (1996) have reported a procedure that eliminates the need for calculating the self-interaction energy. Millot et al. (1996) have compared calculations of the static dielectric constant of polarizable Stockmayer fluids using both lattice summation and reaction field methods. Houssa et al. (1998) reported results of equal quality using either the Ewald sum or the reaction field for simulations involving the Gay−Berne potential. Omelyan (1996) has proposed modifications to the reaction field.

Tironi et al. (1995) have reported a generalized reaction field method that can be incorporated very easily into a traditional MD simulations. Two regions are defined, which are separated by a spherical boundary at a given radius R from the origin. The inner region is characterized by a dielectric permittivity of ε_1 and N point charges q_i at positions $\mathbf{r}_i$. The outer region surrounding the sphere represents a continuum characterized by a dielectric permittivity of ε_2 and constant ionic strength. The potential energy for the total system is given by:

$$U = \frac{1}{2}\sum_{i=1}^{N} \frac{q_i}{4\pi\varepsilon_0\varepsilon_1}\phi(r_i) \tag{5.51}$$

In the limit as $\mathbf{r} \to 0$, the electrostatic potential is,

$$\phi_1(0) = \frac{1}{4\pi\varepsilon_0\varepsilon_1}\sum_i q_i\left[\frac{1}{r_i} + \frac{1+B_0}{R}\right] \tag{5.52}$$

and the electric field is,

$$-\nabla_r\phi_1(\mathbf{r})\Big|_{r=0} = \frac{-1}{4\pi\varepsilon_0\varepsilon_1}\sum_i q_i\left[\frac{\mathbf{r}_i}{r_i^3} + \frac{(1+B_1)\mathbf{r}_i}{R^3}\right], \tag{5.53}$$

where:

$$B_0 = \frac{\varepsilon_1 - 2\varepsilon_2(1+\kappa R)}{\varepsilon_2(1+\kappa R)} \tag{5.54}$$

$$B_1 = \frac{(\varepsilon_1 - 4\varepsilon_2)(1+\kappa R) - 2\varepsilon_2(\kappa R)^2}{(\varepsilon_1 + 2\varepsilon_2)(1+\kappa R) + \varepsilon_2(\kappa R)^2} \tag{5.55}$$

In Eqs. (5.54) and (5.55), κ is the inverse Debye screening length, which can be obtained related to the ionic strength (I), Faraday's constant (F), temperature (T), the ideal gas constant (R), and the permittivity (ε_0):

$$\kappa^2 = \frac{2IF^2}{\varepsilon_0 \varepsilon_2 RT} \tag{5.56}$$

According to Eq. (5.53), the Coulomb force acting on a charge q_i is:

$$\begin{aligned}\mathbf{F}(\mathbf{r}_{ij}) &= \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_1}\left[\frac{1}{r_{ij}^3} + \frac{1+B_1}{R_c^3}\right]\mathbf{r}_{ij} \\ &= \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_1}\left[\frac{1}{r_{ij}^3} - \frac{1}{R_c^3}\frac{2(\varepsilon_2-\varepsilon_1)(1+\kappa R_c)+\varepsilon_2(\kappa R_c)^2}{(\varepsilon_1+2\varepsilon_2)(1+\kappa R_c)+\varepsilon_2(\kappa R_c)^2}\right]\mathbf{r}_{ij}\end{aligned} \tag{5.57}$$

Eq. (5.57) can be used directly in a conventional MD algorithm. The energy can be calculated from Eq. (5.52), or alternatively, it can be obtained by integrating Eq. (5.57). Comparison of the results of the generalized reaction field method (Tironi et al., 1995) with results obtained from the Ewald sum indicate that the values obtained for energy, temperature, and volume are similar but that the root mean fluctuations are greater for the reaction field approach. The main advantages of the method are that it is very simple to implement, and there is no significant increase in computational cost compared with nonionic simulations. Furthermore, the method avoids the singularity at the dielectric boundary, which is common to other reaction filed methods. Tironi et al. (1997) reported a space−time-correlated reaction field method that does not use the traditional static response of the dielectric continuum.

Many examples of the application of the reaction field have involved small polyatomic dipolar molecules, most notably water. However, the technique can also be applied to systems containing large biomolecules. For example, a reaction field method has been used (Sakurai et al., 1997) to investigate intermolecular interaction at the lipid−water interface. A multicavity reaction field method for flexible molecules has also been reported (Karelson et al., 1993).

The reliability of the method is sometimes affected by the nature of the application. Good agreement with the Ewald sum has been reported (Stenhammar et al., 2009) for the solvation of polar liquids, whereas inferior outcomes have been obtained (English, 2005) for water models. It has been applied successfully (Nozawa et al., 2015) to liquid crystal systems. A generalized reaction field method has been developed (Wallace and Shen, 2012) for constant pH MD and it has been useful for biomolecules (Ni and Baumketner, 2011) and proteins (Gargallo et al., 2003). Improvements to account for charges and polarizability have been reported (Nymand and Linse, 2000; Sidler et al., 2018).

5.2.4 Particle-particle and particle-mesh methods

In common with the Ewald sum and reaction field methods, the PPPM algorithm (Eastwood et al., 1980) handles short-range and long-range interactions separately.

The short-range interactions are obtained by a conventional particle−particle (PP) calculation as illustrated by Algorithm 5.1. Long-range interactions are handled by a particle−mesh (PM) technique. In principle, the particle−mesh method can be used to calculate both short- and long-range interactions. However, in practice, the particle−mesh method alone is generally inaccurate. In contrast, the combination of particle−particle and particle−mesh methods optimizes computational speed and accuracy. The PPPM calculation (Algorithm 5.14) is of order $N_g \log N_g$ (N_g is the number of grid points) compared with an order of N^2 for the PP algorithm.

The PM force calculations involve three steps. First, the charge density of the fluid is modeled by assigning charges to a finely spaced mesh in the simulation box. Second, the potential at each mesh point is calculated solving Poisson's equation for the electrostatic potential caused by the charge distribution on the mesh. In three dimensions, Poisson's equation is,

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = -4\pi\rho(x, y, z), \tag{5.58}$$

where ϕ and ρ are the electrostatic potential and source distribution, respectively. This equation is usually solved by using a fast Fourier transformation. Third, numerical differentiation is used to calculate the field at each mesh point. The force on a particle is calculated from the mesh field by interpolation.

The PM is constructed by dividing the simulation box of length L into an integral number of cells of width H. The mesh or grid point that stores the charge density, potential, and electric fields is at the center of each cell. If the origin is taken at mesh point 0, the position of the mesh point p is $x_p = pH$. When periodic boundary conditions are used, the number of cells is equal to the number of grid points N_g.

ALGORITHM 5.14 Calculation of intermolecular forces for a single time step using the PPPM method.

Part 1	Initialisation: Initialise accumulators, positions, momenta, etc.
Part 2	Calculate long-range particle-mesh interactions: Assign charge to mesh. Solve for potential. Update momenta.
Part 3	Short-range force calculation: Create linked-cell. Calculate short-ranged forces. Update momenta.
Part 4	Time integration calculation: Update particle positions, momenta, and energy accumulators.

After the PM contribution is evaluated, the contribution from short-range interactions is added. The steps required for the PPPM method are summarized by Algorithm 5.15. The principles behind the PPPM calculations can be most easily illustrated by first considering the simple case of a one-dimensional system.

In one dimension, the charge density ρ at any mesh point p for a system of N particles can be calculated from the density at the origin by,

$$\rho_p = \frac{CN}{H}\sum_{i=1}^{N_p} W(x_i - x_p) + \rho_0, \tag{5.59}$$

where W is a charge assignment function.

$$W(x) = \begin{cases} 1 & |x| \le H/2 \\ 0 & |x| > H/2 \end{cases} \tag{5.60}$$

In practice, charges can be assigned efficiently by using Algorithm 5.15.

ALGORITHM 5.15 Assignment of charges to the particle–mesh in one-dimension.

```
Part 1   Calculate average number of particles per cell:
         Nc ← ρ0H/(N × |C|)                    //assuming charge neutrality

Part 2   Initialise charge density accumulators:
         loop p ←1 ... Ng
            ρp ← -Nc
         end p loop

Part 3   Accumulate charge density:
         loop j ← 1 ... Np
            p ← nint(xj)                       // locate nearest mesh point
            ρp ← (ρ + 1)/Nc                    // increment charge density
         end j loop
```

The field equations in one dimension are,

$$\phi_{p-1} - 2\phi_p + \phi_{p+1} = -\frac{\rho_p H^2}{\varepsilon_0} \tag{5.61}$$

$$\phi_{p-1} - \phi_{p+1} = 2E_p H, \tag{5.62}$$

where ϕ and E are the potential and electric field, respectively. These equations can be expressed conveniently in reduced form,

$$\phi^*_{p-1} - 2\phi^*_p + \phi^*_{p+1} = \rho^*_p \tag{5.63}$$

$$\phi^*_{p-1} - \phi^*_{p+1} = E^*_p, \tag{5.64}$$

where

$$\phi_p^* = -\frac{q(\Delta T)^2}{2m_e H^2}\phi_p, \tag{5.65}$$

$$E_p^* = -\frac{q(\Delta T)^2}{2m_e H}E_p, \tag{5.66}$$

$$\rho_p^* = -\frac{q(\Delta T)^2}{2m_e \varepsilon_0}\rho_p, \tag{5.67}$$

m_e is the mass of an electron, and ΔT is the simulation time step. We must solve N_g equations of type (5.63) subject to the requirements of charge neutrality and the choice of a reference potential. It can be shown (Hockney and Eastwood, 1988) that the first solution can be obtained from:

$$\phi_1 = \frac{1}{N_g}\sum_{p=1}^{N_g} p\rho_p \tag{5.68}$$

Substituting the left-hand side of Eq. (5.68) into the first equation of type (5.62) yields ϕ_2. If both ϕ_1 and ϕ_2 are known, ϕ_3 can be determined from the second equation of type (5.63). This process can be repeated to find ϕ for all N_g equations of type (5.63). This is a special example of the Thomas tridiagonal algorithm (Hockney and Eastwood, 1988) that is valid when the tridiagonal coefficients are 1, -2, and 1, respectively. The Thomas tridiagonal algorithm can be generalized for arbitrary coefficients. Other algorithms for the solution of the field equations including mesh-relaxation techniques, matrix methods, and rapid elliptic solvers are discussed extensively elsewhere (Hockney and Eastwood, 1988). Algorithm 5.16 illustrates how the field equations are solved in one dimension.

ALGORITHM 5.16 Solution of the field equations in one dimension using the Thomas tridiagonal algorithm.

Part 1 Determine potential and electric field at mesh point 1:

$\phi^*_1 \leftarrow 0$
loop $p \leftarrow 1 \ldots N_g$
  $\phi^*_1 \leftarrow \phi^*_1 + p\rho^*_p$
end p loop
$\phi^*_1 \leftarrow \phi^*_1 / N_g$
$E^*_1 \leftarrow \phi^*_1$

Part 2 Determine potential at mesh point 2:

$\phi^*_2 \leftarrow \rho^*_1 + 2\,\phi^*_1$

Part 3 Determine remaining potentials and electric fields:

loop $p \leftarrow 3 \ldots N_g$
  $\phi^*_p \leftarrow \rho^*_1 + 2\,\phi^*_{p-1} - \phi^*_{p-2}$
  $E_{p-1}{}^* \leftarrow \phi^*_p - \phi^*_{p-2}$
end p **loop**
$E_p \leftarrow \phi^*_p$

The force on any molecule can be obtained by using the interpolation formula:

$$f_i = Nq \sum_{p=0}^{N_g-1} W(x_i - x_p)E_p \tag{5.69}$$

In Eq. (5.56), periodic boundary conditions are handled by treating mesh point N_g as mesh point 0. In terms of reduced properties, Eq. (5.69) becomes (Algorithm 5.17),

$$f_i^* = \sum_{p=0}^{N_g-1} W(x_i^* - p)E_p^* \tag{5.70}$$

where $x^* = x/H$.

ALGORITHM 5.17 Force interpolation in one dimension of the particle−mesh.

```
E*0 ← E*p
f*i ← 0
loop p ←0 ... Ng - 1
   if (|xi* - p| ≤ H/2)
      W ← 1
   else
      W ← 0
   end if
   f*i ← f*i + W × E*p
end p loop
```

Eastwood et al. (1980) have reported a three-dimensional PPPM program. In three dimensions, the volume L^3 of the cubic simulation box is covered by N^3 mesh points separated by a distance $H = L/N$. Each mesh point is associated with a cubic cell of side H and an integer label $\boldsymbol{p} = (p_1, p_2, p_3)$, where p_i has values from 0 to $N-1$. The position of mesh point $\boldsymbol{p}$ is given by $\boldsymbol{x_p} = (p_1H, p_2H, p_3H)$. Points $\boldsymbol{x} = (x_1, x_2, x_3)$ falling within the cell belonging to mesh point $\boldsymbol{p}$ satisfy the inequality that,

$$p_iH \le x_i < (p_i + 1)H \tag{5.71}$$

where $i = 1, 2, 3$. Any labels p_i' that are outside the range of 0 to $N-1$ are transformed by the periodic boundary condition,

$$p = p_i' + mN \tag{5.72}$$

where m is an integer.

In three dimensions, the mesh field charge density [cf. the one-dimensional case Eq. (5.59)] is given by,

$$\rho(\mathrm{p}) = \sum_s \frac{q_s}{H^3} \sum_i W(\mathbf{x}_i - \mathbf{x}_\mathrm{p}), \tag{5.73}$$

where the summation is over all species s and all particles i of each species. The interpolation function W is given by the triangular shaped cloud (TSC) charge function (Eastwood and Hockney, 1974),

$$qW(x, y, z) = qT_{p1}(x)T_{p2}(x)T_{p3}(x), \tag{5.74}$$

where:

$$T_s(t) = \begin{cases} (1-t)^2/2 & s = -1 \\ 0.5 + t - t^2 & s = 0 \\ t^2/2 & s = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{5.75}$$

In contrast to the simple one-dimensional case, the calculation of the mesh potential in three dimensions is quite complicated and requires the use of a Fourier transformation. The mesh potential is defined by the sum over all mesh points $\mathbf{p}'$ in the simulation box,

$$\phi(\mathrm{p}) = \frac{H^3}{\varepsilon_0} \sum_{p'} G(\mathbf{p} - \mathbf{p}')\rho(\mathbf{p}'), \tag{5.76}$$

where G is the mesh defined by Green's function and ε_0 is the permittivity of free space. A discrete Fourier transformation (DFT) is used to evaluate this summation. If the following triple DFT pair $A \supset \widehat{A}$ is defined,

$$A(\mathbf{p}) = \frac{1}{L^3} \sum_\mathrm{l} \hat{A}(\mathbf{I}) \exp\left(i \frac{2\pi}{N} \mathbf{I} \cdot \mathbf{p} \right) \tag{5.77}$$

$$\hat{A}(\mathbf{I}) = \frac{1}{H^3} \sum_\mathrm{p} A(\mathbf{p}) \exp\left(-i \frac{2\pi}{N} \mathbf{I} \cdot \mathbf{p} \right) \tag{5.78}$$

Eq. (5.76) is transformed to:

$$\hat{\phi} = \left(\frac{\hat{G}}{\varepsilon_0} \right) \hat{\rho} \tag{5.79}$$

For a given value of $(\hat{G}/\varepsilon_0)$, the solution to Eq. (5.79) is obtained by transforming $\rho \supset \hat{\rho}$, multiplying by $(\hat{G}/\varepsilon_0)$ to obtain $\hat{\phi}$, which is transformed to yield ϕ.

The mesh force on a particle at any position is calculated from:

$$F_m = q_s \sum_\mathrm{p} W(\mathbf{x}_i - \mathbf{x}_\mathrm{p}) \mathbf{E}(\mathbf{p}) \tag{5.80}$$

The mesh-defined field values $\mathbf{E}(\mathbf{p}) = (E_1(\mathbf{p}), E_2(\mathbf{p}), E_3(\mathbf{p}))$ are obtained by differencing the potential values. The i-component of $\mathbf{E}$ is given by,

$$\mathbf{E}_i(\mathrm{p}) = -\left\{\gamma\frac{\phi(\mathbf{p}+\mathrm{e}_i)-\phi(\mathbf{p}-\mathrm{e}_i)}{2H}+(1-\gamma)\frac{\phi(\mathbf{p}+2\mathrm{e}_i)-\phi(\mathbf{p}-2\mathrm{e}_i)}{4H}\right\}, \tag{5.81}$$

where $\mathbf{e}_i$ is a unit vector in the i direction and the value of γ is chosen to minimize errors in the interparticle force.

Some improvements to the original PPPM method have been described elsewhere (Beckers et al., 1998). It has been adapted for parallel implementation for use on both conventional central processing units (Brieu and Evrad, 2000) and graphics processing units (Brown et al., 2012; Zhang and Bridson, 2014). Optimizations have been reported for periodic (Ballenegger et al., 2008) and nonperiodic (Yao and Capecelatro, 2021) systems. Dipolar systems have been further investigated (Cerdà et al., 2011), and it has been applied (Sirk et al., 2013) in nonequilibrium MD (Chapter 8).

5.2.5 Tree-based methods

The traditional particle−particle, particle−mesh, or a combination of these methods benefit commonly from potential truncation and neighbor list strategies. The philosophy behind these computational strategies is to identify and distinguish between neighboring molecules that make large contributions to molecular interactions, and distant molecules that make only small contributions. This distinction between near and far interactions can be handled efficiently by ordering the molecules in a hierarchical tree structure. Tree-based algorithms are used commonly in computer science sorting and searching applications. The use of tree-based algorithms to many-body problems has been discussed by Pfalzner and Gibbon (1996). They offer the possibility of large gains in computational efficiency, particularly for the calculation of long-range interactions. For example, the fast multipole method is of order N compared with N^2 for a traditional particle−particle calculation. In this section, the Barnes−Hut (Barnes and Hut, 1986) and fast multipole (Greengard and Rokhlin, 1987) algorithms are discussed. Both of these algorithms use a hierarchical tree to distinguish between near and far interactions to improve the efficiency of calculating long-range interactions.

5.2.5.1 The Barnes−Hut algorithm

The Barnes−Hut algorithm utilizes a hierarchical tree. The basic philosophy is to organize particles into a nested hierarchy of cells and compute the multipole moments of each cell up to a fixed order. The acceleration is obtained

by allowing each particle to interact with various elements of the hierarchy governed by a prescribed accuracy criterion. The force from nearby particles is calculated by direct summation, whereas the influence of remote particles is included by evaluating the multipole expansion of the cells that satisfy the accuracy criterion at the location of each particle.

In computer science, a tree is a well-known and often used data structure. The so-called binary tree is often employed to assist in the positioning and retrieval of data in sorting applications. Binary trees can be applied directly to store particle positions (Benz, 1988; Jernigham and Porter, 1989). The advantage of the binary tree is that it usually closely reflects the structure of the system. However, the octagonal tree structure proposed by Barnes and Hut (1986) is conceptually simpler and easier to construct.

Barnes and Hut (1986) constructed their octagonal tree data structure by successively placing particles initially into an empty cubic box. The empty box is the "root" cell, and it is large enough to contain all the particles. As particles are added to the root cell, it is further subdivided into smaller cells such that every particle is contained within its own cell. For example, after the addition of a second particle into the root cell, the cell is divided into child cells having exactly half the length and width of their parent. In three dimensions, this corresponds to splitting the cell into eight pieces. If following this division, the two particles are found in the same child cell, the child cell is subdivided recursively until the particles are in different boxes. The next particle is then added to the root cell and the process is repeated. When all the particles have been added, the space within the root cell has been partitioned into a number of cubic cells of different sizes each containing a maximum of one particle.

In practice (Hernquist, 1988), the Barnes−Hut algorithm is usually implemented starting from a fully populated root cell rather than a completely empty box. The root cell is first divided into eight child cells. The number of particles in each child cell is examined successively. If the cell is empty, it is ignored. If there is only one particle in the cell, it is stored as a "leaf" node of the tree structure. If there is more than one particle in a cell, the cell is stored as a "twig" node, and the cell is further subdivided. The process of subdivision continues until every cell contains a maximum of one particle. The result of this subdivision and the interconnection between levels in the simple two-dimensional case is illustrated in Fig. 5.3.

The preceding description has introduced the components of the tree analogy, namely "root," "twig," and "leaf" that we will now examine more formally. The "root" is the initial simulation cell containing every particle. The tree grows from the "root" cell by successive division. Each division generates a further level in the tree with a number of nodes associated with a cubic volume of space containing a given number of particles. It should be noted that empty cells are not stored. If the division generates a cell with only one particle, a "leaf" is created, otherwise if there are two or more

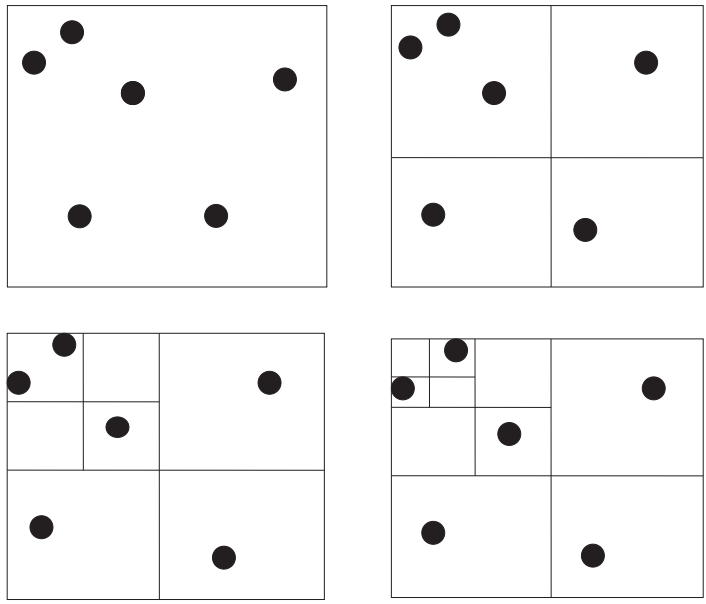


FIGURE 5.3 Division of space for a two-dimensional particle distribution.

particles, a "twig" or "branch" is created. During the tree-building procedure, the total mass, center of mass coordinates, and quadrupole moments of each cell are computed recursively.

It is apparent from the above description that some bookkeeping is necessary to keep track of the relationship between the root, twigs, and leaves. The root cell is also the biggest "twig." For the root, we must store an identifier as a twig plus the number of nonempty child cells. Every other twig requires the storage of a twig identifier; the number of nonempty child cells; and a pointer to its parent cell. Each leaf requires a leaf identifier; a pointer to the parent cell; and a particle label. The particle label associated with each leaf is the link to the physical quantities of the particle required for force evaluation.

How is the force evaluated? The force on a particle is obtained by walking through the tree starting from the root cell. At each step, the ratio of the size of the current cell (s) to its distance from the particle (d) is compared with a predefined accuracy parameter ($\theta \approx 1$). If $s/d \leq \theta$, the influence of all particles in the cell is calculated as a single particle−cell interaction. Otherwise, the cell is subdivided by continuing the descent through the tree until either the accuracy criterion is satisfied or an elementary cell is reached.

The advantage of this algorithm is that the force evaluation can be performed in $O(N\ ln\ N)$ time compared with $O(N^2)$ for a brute-force particle−particle evaluation. Hernquist (1990) has reported that the tree traversals

can be vectorized fully by using a level-by-level approach rather than traversing the tree node by node. This results in a two- to threefold increase in speed compared with the original Barnes–Hut algorithm.

The Barnes–Hut algorithm is most often implemented in the context of astronomical calculations (Belleman et al., 2008; Rein and Liu, 2012) that, in common with molecular simulation, involve the evaluation of N-body interactions. Considerable efforts have been made to develop parallel versions on both conventional multiprocessors (Munier et al., 2020; Winkel et al., 2012) and hybrid systems (Iwasawa et al., 2020; Polyakov et al., 2013) involving graphics processing units. Although it is typically used for force evaluation, it can also be applied (Gan and Xu, 2014; Stransky, 2016) in MC simulations (Chapter 6).

5.2.5.2 The fast multipole method

The fast multipole method (FMM), developed originally by Greengard and Rokhlin (Greengard, 1987; Greengard and Rokhlin, 1987; Carrier et al., 1988), also uses the hierarchical tree concept. The FMM algorithm involves multipole expansions for boxes on the lowest level of the tree. These expansions are combined and shifted as they are passed up and down the tree. Particles are assigned to cells at the finest level of the tree. In contrast to the Barnes–Hut algorithm, some cells may be empty, whereas some cells may have several particles. The multipole expansion of the particle configuration on the finest level is formed about the center of the box. Each "child" box communicates this information to its "parent" box on the next level. Aggregate information about distant particles comes back down the low-level boxes.

The original exposition of the FMM algorithm by Greengard and Rokhlin (1987) concentrated primarily on one and two dimensions. Schmidt and Lee (1991) have discussed the implementation in three dimension. In common with the Barnes–Hut algorithm, the FFM subdivides the simulation box (Fig. 5.4). The simulation box of length d is subdivided into a box of length $d/2^r$, where r is an integer representing the level of refinement. This division

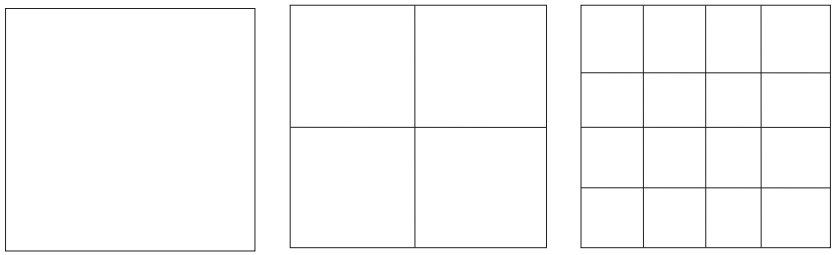


FIGURE 5.4 Two-dimensional example of the division of the simulation box in the fast multipole method algorithm.

is equivalent to forming 8^r equal-sized subvolumes. This is done for every single box to a maximum level of refinement R irrespective of the number of particles that they contain. The maximum level of refinement R is approximately equal to the number of particles (N), that is, $R = log_8 N$. At the maximum level of refinement, there is on average one particle per box. The interrelationship between the center of mass multiples at different levels of refinement is illustrated for the two-dimensional case in Fig. 5.5.

The work of Schmidt and Lee (1991) provides a very clear description of the FMM algorithm in three dimensions, and the following discussion is based largely on their interpretation. The implementation of the FMM uses three transformations of the potential. The transformations begin with a multipole expansion of the potential,

$$u(\mathrm{r}) = 4\pi \sum_{l,m} \frac{M_{lm} Y_{lm}(\theta, \phi)}{(2l+1) r^{l+1}} \tag{5.82}$$

or a local expansion,

$$u(\mathrm{r}) = 4\pi \sum_{l,m} L_{lm} r^l Y_{lm}(\theta, \phi), \tag{5.83}$$

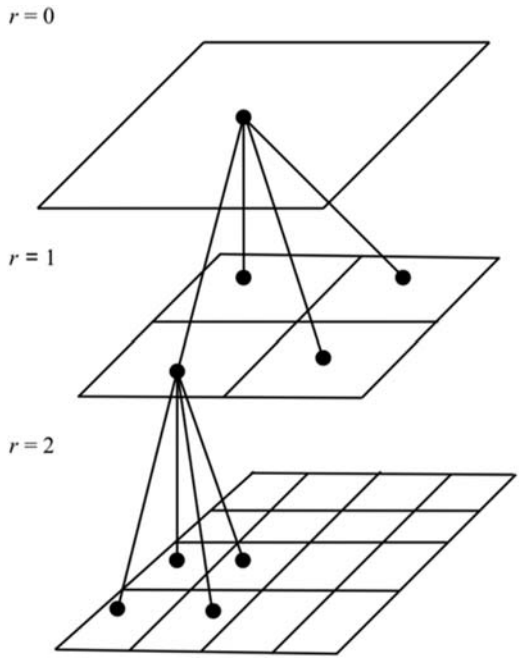


FIGURE 5.5 Two-dimensional representation of the interrelationship between the center of mass multipoles at different levels in the FMM algorithm.

where Y_{lm} is a modified spherical harmonic that does not include the normalization factor $\sqrt{(2l+1)/4\pi}$; spherical coordinates (r, θ, r) are used relative to an origin; and M_{lm} and L_{lm} are the multipole and local expansion coefficients, respectively. The multipole moments are defined by Jackson (1983):

$$M_{lm} = \sum_i q_i s_i^l Y_{lm}^*(\theta_i, \phi_i) \tag{5.84}$$

A feature of the FMM is the need to make transformations between different coordinate systems. If the properties of the different coordinate systems are distinguished by the use of primes, it can be shown (Greengard, 1987) that:

$$M'_{l'm'} = \sum_{l,m} T^{MM}_{l'm',lm} M_{lm} \tag{5.85}$$

$$L'_{l'm'} = \sum_{l,m} T^{LM}_{l'm',lm} M_{lm} \tag{5.86}$$

$$L'_{l'm'} = \sum_{l,m} T^{LL}_{l'm',lm} L_{lm} \tag{5.87}$$

In Eqs. (5.85) to (5.87), the Ts are transformation matrices of the following form:

$$T^{MM}_{l'm',lm} = 4\pi \frac{(-r_t)^{l'-l} Y^*_{l'-l,m'-m}(\theta_t, \phi_t) a_{l'-1,m'-m} a_{lm} (2l'+1)}{(2l+1)[2(l'-l)+1] a_{l'm'}}, \tag{5.88}$$

$$T^{LM}_{l'm',lm} = 4\pi \frac{(-1)^{l+m} Y^*_{l'+l,m'-m}(\theta_t, \phi_t) a_{lm} a_{l'm'}}{r_t^{l'+l+1} (2l+1)(2l'+1) a_{l'+l,m'-m}}, \tag{5.89}$$

$$T^{LL}_{l'm',lm} = 4\pi \frac{r_t^{l-l'} Y_{l-l',m-m'}(\theta_t, \phi_t) a_{l'm'} a_{l-l',m-m'}}{a_{lm} (2l'+1)[2(l-l')+1]}, \tag{5.90}$$

where the subscript t denotes a translation property between the different coordinate systems and:

$$a_{lm} = (-1)^{l+m} \sqrt{\frac{2l+1}{4n(l+m)!(l-m)!}} \tag{5.91}$$

We will first consider an implementation of the three-dimensional FMM algorithm for an open system. The issues raised by periodic boundary conditions will be addressed separately. The system that we are interested consists of N charges confined to a cube of sides d centered at the origin. The initialization phase of the FMM algorithm involves dividing the cube into

successively smaller cubic subvolumes. We define a maximum refinement level (R) such that N is approximately equal to 8^R. The maximum multipole expansion (L) is chosen in accordance with the predetermined relative precision (ε), that is, $2^{-L} \leq \varepsilon$. Each level of refinement r contains an increasing number of subvolumes. At $r = 0$ we have the original cube, whereas at $r = 1$, 2, 3 ..., there are 8, 16, 32, ... subvolumes. The refinement levels are the "parent" boxes, which contain "child" boxes. Each box has 26 touching boxes, which have a common face, edge, or corner. The algorithm proceeds in four distinct stages.

Stage 1. First, starting from refinement level R, the 8^R truncated multipole expansions about the center of each box are calculated for particles contained in each box in level R.

Stage 2. For refinement levels $R-1$ to 0, the truncated multipole moment expansions for all charges in all boxes about their centers are formed by using Eq. (5.84) to transform the multipole expansions of the eight child boxes to the center of their parent.

Stage 3. For each level of refinement, the multipole expansions are converted to local expansions about the center of all the boxes. The multipole expansions that contribute to the local expansions are those that do not touch the one box. This calculation can be performed efficiently by combining local-to-local and multipole-to-local transformations. At refinement levels 0 and 1, the local expansion is zero because the boxes are too close to allow a valid multipole-to-local transformation. For the remaining levels, the local expansion of the parent box is transformed to the center of the current box. To this local expansion is added the transformation of the multipole potentials for all boxes of the current level, which satisfy the conditions that (1) the box does not touch the current box and (2) the charges in the box do not contribute to the local potential. This results in a local potential in each box from interactions between all particles in all boxes at the same level, which do not touch the current box.

Stage 4. The final potential and forces of all particles in the boxes at the finest level of refinement are calculated. This involves (1) evaluating the potential and force from the local expansion at each particle location for each box in level R and (2) evaluating the remaining interactions of each particle with the other particles in the current box plus the particles i touching boxes.

Schmidt and Lee (1991) showed how the previously discussed algorithm could be modified to include periodic boundary conditions. The FMM algorithm requires the local expansion of the potential from all periodic images except the 26 nearest neighbors of the original simulation cell. To accommodate periodic boundary conditions, the values of $r_t^{-(l+1)} Y_{l,m}(\theta_t, \phi_t)$ in Eq. (5.89) are replaced by sums over values corresponding to all the images of the simulation cell except its 26 surrounding

cells. The sum over all the cells except the simulation cell can be calculated using an Ewald sum.

$$\sum_{n,o,p\neq 0,0,0} r_{nop}^{-(1+l)} Y_{lm}(\theta_{nop}, \phi_{nop})$$

$$= \sum_{n,o,p}{}' \frac{Y_{lm}(\theta,\phi)}{(2l+1)!} \left[\begin{array}{l} \dfrac{2\alpha(\alpha r_{nop})'}{\pi^{3/2}} \exp(-\alpha^2 r_{nop}^2) \\ \\ + \dfrac{4\times 2^l}{\sqrt{\pi} r^{l+1}} I_l + (2\pi i)^l r^{l-2} \exp\left(\dfrac{-\pi^2 r_{nop}^2}{\alpha^2}\right) \end{array} \right], \quad (5.92)$$

where r_{nop}, θ_{nop}, and ϕ_{nop} terms are the spherical coordinates of the vector $r_{nop} = d(n\hat{x} + o\hat{y} + p\hat{z})$, α is the Ewald convergence parameter, and I_l is given by the recursion relation,

$$I_l = \frac{2l+1}{2} I_{l-1} + \frac{1}{2}\exp(-\alpha^2 r^2)(\alpha r)^{2l+1} \quad (5.93)$$

with:

$$I_0 = \frac{\sqrt{\pi}}{4} erfc(\alpha r) + \frac{\alpha r}{2}\exp(-\alpha^2 r^2) \quad (5.94)$$

The prime on the sum indicates omission of the $n = o = p = 0$ term. This enables the contribution of the 26 surrounding cells to be subtracted.

The incorporation of periodic boundary conditions requires minor modifications to the conventional algorithm. During the initialization phase, the image mutipole-to-local transformations are created. Stage 3 is modified by adding the nonzero local expansions for levels $r = 0$ and $r = 1$, and full multipole transformations are now required for each local expansion.

How does the efficiency of the FMM algorithm compare with particle−particle algorithms? The answer to this question depends on the number of particles in the simulation, the maximum level of refinement, and the number of mutipoles. Schmidt and Lee (1991) analyzed several different scenarios. Typically, the FMM algorithm is advantageous for simulations involving tens of thousands of molecules.

It should be noted that the FMM algorithm is an active area of research and improvements in efficiency are being made continually. White and Head-Gordon (1994) have proposed changes to the transformations that improves the efficiency of the algorithm. White et al. (1994) have developed a continuous FMM (CFMM), which uses continuous functions to calculate charge interactions between particles. The CFMM approach is more efficient than direct calculations for less than 10,000 or more particles. Kutteh et al. (1995) have reported a generalized FMM algorithm for Hartree−Fock and density functional computations. Efficiency gains have also been reported by

McKenney et al. (1996). Petersen et al. (1994) have proposed a very FMM (VFMM). In three dimensions, the VFMM algorithm is two to three times faster than FMM without any loss of accuracy. Niedermeier and Tavan (1996) have applied the principles of FMM to the simulation of electrostatic interaction in proteins. A large-scale simulation of macromolecules have been reported (Figueirido et al., 1997), which combines the FMM approach with multiple time step integrators. Wang and LeSar (1996) obtained the multipole expansions in terms of solid harmonics instead of spherical methods. They concluded that using solid harmonics greatly increases the computational efficiency of the FMM algorithm.

The FMM approach is increasingly being applied to a wide variety of large-scale systems (Andoh et al., 2013; Kurzak and Petititt, 2006), including the use of polarizable force fields (Coles and Masella, 2015) and application to dipolar systems (Shamshirgar et al., 2019). Its usefulness is further enhanced by parallel implementations (Board et al., 1992; Ibeid et al., 2016), which also include the use of graphics processing units (Kohnke et al., 2020; Wilson et al., 2021; Yokota et al., 2013). Parallel versions of the FMM algorithm specifically for MD have been reported (Andoh et al., 2013, 2021), which could facilitate long-time simulations.

The effect of these improvements to the FMM algorithm is that the number of particles for which FMM codes are more efficient than direct particle−particle evaluation is being progressively lowered. Simple particle−particle algorithms assisted by time-saving devices such as a neighbor list are adequate for atoms or simple polyatomic molecules involving relatively few particles ($N \approx 1000$). However, the rigorous evaluation of the properties of real macromolecules (Cisneros, et al., 2014) by molecular simulation requires thousands of interaction sites per molecules. Alternative computational strategies are required to deal with this increased level of complexity. It is in this context that hierarchical tree algorithms such as FMM or Barnes−Hut are likely to play an increasingly important role in molecular simulation.

5.2.6 Comparison of computational efficiency

There are no definitive studies comparing the computational efficiency and accuracy of the reaction field, Ewald sum, PPPM, and hierarchical tree methods. The reaction field method is undoubtedly the simplest to implement and, when used in conjunction with computation-saving devices such as a neighbor list, it is an $O(N)$ algorithm. The main limitations of the reaction field method are that the dielectric constant of the medium must be known, and there is some evidence to suggest that accuracy is compromised when highly charged species are involved. The Ewald sum is possibly the most widely used method for the simulation of charged species. Luty et al. (1994, 1995) have reported comparisons of the Ewald sum with PPPM calculations. They concluded that the PPPM algorithm was considerably more efficient

than the Ewald sum, particularly if a large number of charges are involved. This can be partly attributed to the use of a discrete Fourier transform in the PPPM method, whereas the Ewald method sums the Fourier series of the long-range function analytically. Either the PPPM or FMM algorithms can also be used to calculate efficiently the Coulombic interactions in large biomolecules (Shimada et al., 1993, 1994). For nonperiodic systems, the PPPM method may introduce an artificial periodicity into the system. Techniques are available (Luty and van Gunsteren, 1996) to overcome this possible limitation. Pollock and Glosli (1996) have reported a comparison of the PPPM, FMM, and Ewald techniques. They concluded that the PPPM technique was more efficient computationally than either the FMM or Ewald algorithms. The $O(N)$ scaling of the FMM algorithms means that it is potentially the most efficient method but, because of computational overheads, it is currently viable only for large systems. However, improvements to FMM are resulting in substantial efficiency gains for increasingly smaller systems, which are further magnified by parallel implementations.

5.3 Summary

Evaluating either the energy or force experienced by a molecule from intermolecular interactions is at the heart of a molecular simulation. Periodic boundary conditions coupled with long-range corrections are used commonly to evaluate short-range interactions experienced by nonpolar molecules and atoms. Computational procedures such as neighbor lists and linked-cell lists have been developed to reduce the computational overhead. They are particularly advantageous in MD simulation. Special techniques such as Ewald sum, reaction field, particle−mesh, or tree-based methods are required to calculate the long-range interactions experienced by ions of polar molecules. Tree-based methods such as the FMM are used less commonly in molecular simulations because a large number of molecules is required to make them competitive with traditional approaches. However, the computational limitation is being eroded continually. The process is being further accelerated by parallel implementations on both traditional multicore and hybrid computing architectures (Chapter 12).

References

Abramowitz, M., Stegun, I.A. (Eds.), 1972. Handbook of Mathematical Functions with Formula, Graphs and Mathematical Tables. Dover, New York, p. 295.

Adams, D.J., 1979. Computer simulation of ionic systems: the distorting effects of the boundary conditions. Chem. Phys. Lett. 62, 329−332.

Adams, D.J., McDonald, I.R., 1976. Thermodynamic and dielectric properties of polar lattices. Mol. Phys. 32, 931−947.

Adams, D.J., Adams, E.H., Hills, G.J., 1979. The computer simulation of polar liquids. Mol. Phys. 38, 387−400.

Ahmed, A., Sadus, R.J., 2010. Effect of potential truncations and shifts on the solid-liquid phase coexistence of Lennard-Jones Fluids. J. Chem. Phys. 133, 124515.

Alder, B.J., Pollock, E.L., 1981. Simulation of polar and polarizable fluids. Ann. Rev. Phys. Chem. 32, 311–329.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Alper, H., Levy, R.M., 1993. Dielectric and thermodynamic response of a generalized reaction field model for liquid state simulations. J. Chem. Phys. 99, 9847–9852.

Anastasiou, N., Fincham, D., 1982. Programs for the dynamic simulation of liquids and solids II. MDIONS: rigid ions using the Ewald sum. Comp. Phys. Commun. 25, 159–176.

Andoh, Y., Yoshii, N., Fujimoto, K., Mizuntani, K., Kojima, H., Yamada, A., Okazaki, S., Kawaguchi, K., Nago, H., Iwahashi, K., Mizutani, F., Minami, K., Ichikawa, S.-I., Komatsu, H., Ishizuki, S., Takeda, Y., Fukushima, M., 2013. MODYLAS: a highly parallelized general-purpose molecular dynamics simulation program for large-scale systems with long-range forces calculated by fast multipole method (FMM) and highly scalable fine-grained new parallel processing algorithms. J. Chem. Theory Comput. 9, 3201–3209.

Andoh, Y., Ichikawa, S.-I., Sakashita, T., Yoshii, N., Okazaki, S., 2021. Algorithm to minimize MPI communications in the parallelized fast multipole method combined with molecular dynamics. J. Comput. Chem. 42, 1073–1087.

Andrea, T.A., Swope, W.C., Andersen, H.C., 1983. The role of long ranged forces in determining the structure and properties of liquid water. J. Chem. Phys. 79, 4576–4584.

Avendaño, C., Gil-Villegas, A., 2006. Monte Carlo simulations of primitive models for ionic systems using the Wolf method. Mol. Phys. 104, 1475–1486.

Awile, O., Büyükkeçeci, F., Reboux, S., Sbalzarini, I.F., 2012. Fast neighbor lists for adaptive-resolution particle simulations. Comput. Phys. Commun. 183, 1073–1081.

Baidakov, V.G., Chernykh, G.G., Protsenko, S.P., 2000. Effect of the cut-off radius of the intermolecular potential on phase equilibrium and surface tension in Lennard–Jones systems. Chem. Phys. Lett. 321, 315–320.

Ballenegger, V., Cerda, J.J., Lenz, O., Holm, C., 2008. The optimal P3M algorithm for computing electrostatic energies in periodic systems. J. Chem. Phys. 128, 034109.

Barker, J.A., 1994. Reaction field, screening, and long-range interactions in simulations of ionic and dipolar systems. Mol. Phys. 83, 1057–1064.

Barker, J.A., Fisher, R.A., Watts, R.O., 1971. Liquid argon: Monte Carlo and molecular dynamics calculations. Mol. Phys. 21, 657–673.

Barnes, J., Hut, P., 1986. A hierarchical O(N log N) force-calculation algorithm. Nature 324, 446–449.

Baumketner, A., 2009. Removing systematic errors in interionic potentials of mean force computed in molecular simulations using reaction-field-based electrostatics. J. Chem. Phys. 130, 104106.

Beckers, J.V.L., Lowe, C.P., de Leeuw, S.W., 1998. An iterative PPPM method for simulating Coulombic systems on distributed memory parallel computers. Mol. Sim. 20, 369–383.

Belleman, R.G., Bédorg, J., Portegies Zwart, S.F., 2008. High performance direct gravitational *N*-body simulations on graphics processing units II: an implementation in CUDA. N. Astron 13, 103–112.

Benz, W., 1988. Applications of smooth particle hydrodynamics (SPH) to astrophysical problems. Comp. Phys. Commun. 48, 97–105.

Board Jr, J.A., Causey, J.W., Leathrum Jr, J.F., Windemuth, A., Schulten, K., 1992. Accelerated molecular dynamics simulation with the parallel fast multipole algorithm. Chem. Phys. Lett. 198, 89–94.

Bogusz, S., Cheatham III, T.E., Brooks, B.R., 1998. Removal of pressure and free energy artifacts in charged periodic systems via net charge corrections to the Ewald potential. J. Chem. Phys. 108, 7070–7084.

Booth, A.D., 1972. Numerical Methods, third ed. Butterworth, London.

Brieu, P.P., Evrad, A.E., 2000. P4M: a parallel version of P3M. New Astron. 5, 163–180.

Brown, W.M., Kohlmeyer, A., Plimpton, S.J., Tharrington, A.N., 2012. Implementing molecular dynamics on hybrid high performance computers – Particle–particle particle-mesh. Comp. Phys. Commun. 183, 449–459.

Carrier, J., Greengard, L., Rokhlin, V., 1988. A fast adaptive multipole algorithm for particle simulations. SIAM J. Sci. Stat. Comput. 9, 669–686.

Cerdà, J.J., Ballenegger, V., Holm, C., 2011. Particle-particle-mesh methods for dipolar interactions: on error estimates and efficiency of schemes with analytical differentiation and mesh interlacing. J. Chem. Phys. 135, 184110.

Cisneros, G.A., Karttunen, M., Ren, P., Sagui, C., 2014. Classical electrostatics for biomolecular simulations. Chem. Rev. 114, 779–814.

Coelho, A.A., Cheary, R.W., 1997. A fast and simple method for calculating electrostatic potentials. Comp. Phys. Commun. 104, 15–22.

Coles, J.P., Masella, M., 2015. The fast multipole method and point dipole moment polarizable force fields. J. Chem. Phys. 142, 024109.

Cui, Z.W., Sun, Y., Qu, J.M., 2009. The neighbor list algorithm for parallelepiped box in molecular dynamics simulations. Chin. Sci. Bull. 54, 1463–1469.

Danese, G., de Lotto, I., Dotti, D., Leporati, F., 1998. Ewald potentials evaluated through look-up tables. Comp. Phys. Commun. 108, 211–217.

de Leeuw, S.W., Perram, J.W., Smith, E.R., 1980. Simulation of electrostatic systems in periodic boundary conditions. I. Lattice sums and dielectric constant. Proc. R. Soc. Lond. A373, 27–56.

de Leeuw, S.W., Perram, J.W., Smith, E.R., 1986. Computer simulation of the static dielectric constant of systems with permanent electric dipoles. Ann. Rev. Phys. Chem. 37, 245–270.

Eastwood, J.W., Hockney, R.W., 1974. Shaping the force law in two-dimensional particle-mesh models. J. Comput. Phys. 16, 342–359.

Eastwood, J.W., Hockney, R.W., Lawerence, D., 1980. P3M3DP - the three dimensional periodic particle-particle/particle-mesh program. Comp. Phys. Commun. 19, 215–261.

English, N.J., 2005. Molecular dynamics simulations of liquid water using various long-range electrostatic techniques. Mol. Phys. 103, 1945–1960.

Essex, J.W., 1998. The application of the reaction-field method to the calculation of dielectric constants. Mol. Sim. 20, 159–178.

Everaers, R., Kremer, K., 1994. A fast grid search algorithm for molecular dynamics simulations with short-range interactions. Comp. Phys. Commun. 81, 19–55.

Ewald, P., 1921. Die Berechnung optischer und elektrostatischer Gitterpotentiale. Ann. Phys. 64, 253–287.

Fanourgakis, G.S., Tipparaju, V., Nieplocha, J., Xantheas, S.S., 2007. An efficient parallelization scheme for molecular dynamics simulations with many-body, flexible, polarizable empirical potentials: application to water. Theor. Chem. Acc. 117, 73–84.

Figueirido, F., Levy, R.M., Zhou, R., Berne, B.J., 1997. Large scale simulation of macromolecules in solution: combing the periodic fast multipole method with multiple time step integrators. J. Chem. Phys. 106, 9835–9849.

Fincham, D., 1994. Optimisation of the Ewald sum for large systems. Mol. Sim. 13, 1–9.

Fincham, D., Ralston, B.J., 1981. Molecular dynamics simulation using the CRAY-1 vector processing computer. Comp. Phys. Commun. 23, 127–134.

Fomin, E.S., 2011. Consideration of data load time on modern processors for the Verlet table and linked-cell algorithms. J. Comput. Chem. 32, 1386–1599.

Frenkel, D., Smit, B., 2023. Understanding Molecular Simulation. From Algorithms to Applications, second ed. Academic Press, San Diego, p. 635.

Gan, Z.C., Xu, Z.L., 2014. Efficient implementation of the Barnes-Hut octree algorithm for Monte Carlo simulations of charged systems. Sci. China Math. 57, 1331–1340.

Gargallo, R., Hünenberger, P.H., Avilés, F.X., Oliva, B., 2003. Molecular dynamics simulation of highly charged proteins: comparison of the particle-particle particle-mesh and reaction field methods for the calculation of electrostatic interactions. Protein Sci. 12, 2161–2172.

Garzon, B., Lago, S., Vega, C., 1994. Reaction field simulations of the vapor-liquid equilibria of dipolar fluids. Does the reaction field dielectric constant affect the coexistence properties? Chem. Phys. Lett. 231, 366–372.

Gdoutos, E.E., Agrawal, R., Espinosa, H.D., 2010. Comparison of the Ewald and Wolf methods for modeling electrostatic interactions in nanowires. Int. J. Numer. Meth. Engng 84, 1541–1551.

Gonnet, P., 2013. Pseudo-Verlet lists: a new, compact neighbour list representation. Mol. Sim. 39, 721–727.

Greengard, L., 1987. The Rapid Evaluation of Potential Fields in Particle Systems. MIT Press, Cambridge, MA.

Greengard, L., Rokhlin, V., 1987. A fast algorithm for particle simulations. J. Comput. Phys. 73, 325–348.

Grest, G.S., Dünweg, B., Kremer, K., 1989. Vectorised link cell FORTRAN code for molecular dynamics simulations of large number of particles. Comp. Phys. Commun. 55, 269–285.

Heinz, T.M., Hünenberger, P.H., 2004. A fast pairlist-construction algorithm for molecular simulations under periodic boundary conditions. J. Comput. Chem. 25, 1474–1486.

Hernquist, L., 1988. Hierarchical N-body methods. Comp. Phys. Commun. 48, 107–115.

Hernquist, L., 1990. Vectorization of tree traversals. J. Comput. Phys. 87, 137–147.

Heyes, D.M., 1981. Electrostatic potentials and fields in infinite point charge lattices. J. Chem. Phys. 74, 1924–1929.

Hloucha, M., Deiters, U.K., 1997. Monte Carlo simulations of acetonitrile with an anisotropic polarizable molecular model. Mol. Phys. 90, 593–597.

Hockney, R.W., Eastwood, J.W., 1988. Computer Simulation Using Particles. Adam Hilger, Bristol.

Houssa, M., Oualid, A., Rull, L.F., 1998. Reaction field and Ewald summation study of mesophase formation in dipolar Gay-Berne model. Mol. Phys. 94, 439–446.

Huang, C., Li, C., Choi, P.Y.K., Nandakumarc, K., Kostiuk, L.W., 2010. Effect of cut-off distance used in molecular dynamics simulations on fluid properties. Mol. Sim. 36, 856–864.

Hunt, T.A., Todd, B.D., 2003. On the Arnold cat map and periodic boundary conditions for planar elongational flow. Mol. Phys. 101, 3445–3454.

Ibeid, H., Yokota, R., Keyes, D., 2016. A performance model for the communication in fast multipole methods on high-performance computing platforms. Int. J. High. Perform. Comput. Appl. 30, 423–437.

Iwasawa, M., Namekata, D., Sakamoto, R., Makamura, T., Kimura, Y., Nitadori, K., Wang, L., Tsubouchi, M., Makino, J., Liu, Z., Fu, H., Yang, G., 2020. Int. J. High. Perform. Compt. Appl. 34, 615–628.

Jackson, J.D., 1983. Classical Electrodynamics. Wiley, New York.

Jernigham, J.G., Porter, D.H., 1989. A tree code with logarithmic reduction of force terms, hierarchical regularization of all variables, and explicit accuracy controls. Astrophys. J. Suppl. Ser. 71, 871–893.

Karelson, M., Tamm, T., Zerner, M., 1993. Multicavity reaction field method for the solvent effect description in flexible molecular systems. J. Phys. Chem. 97, 11901–11907.

Kessler, J., Bour, P., 2014. Molecular dynamics with helical periodic boundary conditions. J. Comput. Chem. 35, 1552–1559.

Kohnke, B., Kutzner, C., Beckmann, A., Lube, G., Kabadshow, I., Dachsel, H., Grubmüller, H., 2020. High. Perform. Comput. Appl. 35, 97–117.

Kurzak, J., Petititt, B.M., 2006. Fast multipole methods for particle dynamics. Mol. Sim. 32, 775–790.

Kutteh, R., Aprà, E., Nichols, J., 1995. A generalized fast multipole approach for Hartree-Fock and density functional computations. Chem. Phys. Lett. 238, 173–179.

Lee, S.H., 2020. Ionic mobilities of Na^+ and Cl^- at 25°C as a function of Ewald sum parameter: A comparative molecular dynamics simulation study. Mol. Sim. 46, 262–270.

Lekner, J., 1998. Coulomb forces and potentials in systems with an orthorhombic unit cell. Mol. Sim. 20, 357–368.

Lindbo, D., Tornberg, A.-K., 2012. Fast and spectrally accurate Ewald summation for 2-periodic electrostatic systems. J. Chem. Phys. 136, 164111.

Luo, J., Liu, L., Su, P., Duan, P., Lu, D., 2015. A piecewise lookup table for calculating nonbonded pairwise atomic interactions. J. Mol. Model. 21, 288–290.

Luty, B.A., van Gunsteren, W.F., 1996. Calculating electrostatic interactions using the particle-particle particle-mesh method with nonperiodic long-range interactions. J. Phys. Chem. 100, 2581–2587.

Luty, B.A., Davis, M.E., Tironi, I.G., van Gunsteren, W.F., 1994. A comparison of particle-particle, particle-mesh (PPPM) and Ewald methods for calculating electrostatic interactions in periodic molecular systems. Mol. Sim. 14, 11–20.

Luty, B.A., Tironi, I.G., van Gunsteren, W.F., 1995. Lattice-sum methods for calculating electrostatic interactions in molecular systems. J. Chem. Phys. 103, 3014–3021.

Ma, Y., Garofalini, S.H., 2005. Modified Wolf electrostatic summation: Incorporating an empirical charge overlap. Mol. Sim. 31, 739–748.

Matin, M.L., Daivis, P.J., Todd, B.D., 2003. Cell neighbor list method for planar elongational flow: Rheology of a diatomic fluid. Comp. Phys. Commun. 151, 35–46.

McKenney, A., Pachter, R., Patnaik, S., Adams, W., 1996. Molecular dynamics simulations of a siloxane-based liquid crystal using an improved fast multipole algorithm implementation. Mat. Res. Soc. Symp. Proc. 408, 99–105.

Mendoza, F.N., López-Lemus, J., Chapela, G.A., Alejandre, J., 2008. The Wolf method applied to the liquid-vapor interface of water. J. Chem. Phys. 129, 024706.

Milchev, A., Paul, W., Binder, K., 1993. Off-lattice Monte Carlo simulation of dilute and concentrated polymer solutions under theta conditions. J. Chem. Phys. 99, 4786–4798.

Millot, C., Soetens, J.C., Costa, M.T.C.M., 1996. Static dielectric constant of the polarizable Stockmayer fluid. Comparison of lattice summation and reaction field methods. Mol. Sim. 18, 367–383.

Morales, J.J., 1996. Computer dependence of an improvement of the cell neighbour-table method for short-range potentials. Mol. Sim. 18, 325–337.

Munier, B., Aleem, M., Khan, M., Islam, M.A., Iqbal, M.I., Khattak, K., 2020. On the parallelization and performance analysis of Barnes-Hut algorithm using Java parallel platforms. SN Appl. Sci. 2, 601.

Ni, B., Baumketner, A., 2011. Effect of atom- and group-based truncations on biomolecules simulated with reaction-field electrostatics. J. Mol. Model. 17, 2883–2893.

Niedermeier, C., Tavan, P., 1996. Fast version of the structure adapted multipole method - efficient calculation of electrostatic forces in protein dynamics. Mol. Sim. 17, 57–66.

Nozawa, T., Takahashi, K.Z., Narumi, T., Yasuoka, K., 2015. Comparison of the accuracy of periodic reaction field methods in molecular dynamics simulations of a model liquid crystal system. J. Comput. Chem. 36, 2406–2411.

Nymand, T.M., Linse, P., 2000. Ewald summation and reaction field methods for potentials with atomic charges, dipoles, and polarizabilities. J. Chem. Phys. 112, 6152–6160.

Omelyan, I.P., 1996. On the reaction field for interaction site models of polar systems. Phys. Lett. A 223, 295–302.

Omelyan, I.P., 1997. Ewald summation technique for interaction site models of polar fluids. Comp. Phys. Commun. 107, 113–122.

Petersen, H.G., Soelvason, D., Perram, J.W., Smith, E.R., 1994. The very fast multipole method. J. Chem. Phys. 101, 8870–8876.

Pfalzner, S., Gibbon, P., 1996. Many-Body Tree Methods in Physics. Cambridge University Press, Cambridge.

Pinches, M.R.S., Tildesley, D.J., Smith, W., 1996. In: Gubbins, K.E., Quirke, N. (Eds.), Molecular Simulation and Industrial Applications. Methods, Examples and Properties. Gordon and Breach, Amsterdam.

Pollock, E.L., Glosli, J., 1996. Comments on P^3M, FMM, and the Ewald method for large periodic Coulombic systems. Comp. Phys. Commun. 95, 93–110.

Polyakov, A.Y., Lyutyy, T.V., Denisov, S., Reva, V.V., Hänggi, P., 2013. Large-scale ferrofluid simulations on graphics processing units. Comp. Phys. Commun. 184, 1483–1489.

Quentrec, B., Brot, C., 1973. New method for searching for neighbors in molecular dynamics computations. J. Comput. Phys. 13, 430–432.

Rapaport, D.C., 2004. The Art of Molecular Dynamics Simulation, second ed. Cambridge University Press, Cambridge.

Reed, M.S.C., Flurchick, K.M., 1996. Investigation of artifacts due to periodic boundary conditions. Comp. Phys. Commun. 95, 39–46.

Rein, H., Liu, S.-F., 2012. REBOUND: an open-source multi-purpose N-body code for collisional dynamics. Astron. Astrophys. 537, A128.

Resat, H., McCammon, J.A., 1998. Correcting for electrostatic cutoffs in free energy simulations: toward consistency between simulations with different cutoffs. J. Chem. Phys. 108, 9617–9623.

Ruocco, G., Sampoli, M., 1994. Computer simulation of polarizable fluids: a consistent and fast way for dealing with polarizability and hyperpolarizability. Mol. Phys. 82, 875–886.

Ruocco, G., Sampoli, M., 1995. Computer simulation of polarizable fluids: on the determination of the induced dipoles. Mol. Sim. 15, 281–300.

Sadeghifar, A., Dadvar, M., Karimi, S., Ghobadi, A.F., 2012. The Wolf method applied to the type I methane and carnon dioxide hydrates. J. Mol. Graph. Model. 38, 455–464.

Sakurai, M., Tamagawa, H., Inoue, Y., Ariga, K., Kunitake, T., 1997. Theoretical study of intermolecular interaction at the lipid-water interface. 1. Quantum chemical analysis using a reaction field theory. J. Phys. Chem. B 101, 4810–4816.

Schmidt, K.E., Lee, M.A., 1991. Implementing the fast multipole method in three dimensions. J. Stat. Phys. 63, 1223–1235.

Shamshirgar, D.S., Yokota, R., Tornberg, A.-K., Hess, B., 2019. Regularizing the fast multipole method for use in molecular simulation. J. Chem. Phys. 151, 234113.

Shimada, J., Kaneko, H., Takada, T., 1993. Efficient calculations of Coulombic interactions in biomolecular simulations with periodic boundary conditions. J. Comput. Chem. 14, 867–878.

Shimada, J., Kaneko, H., Takada, T., 1994. Performance of fast multipole methods for calculating electrostatic interactions in biomacromolecular simulations. J. Comput. Chem. 15, 28–43.

Sidler, D., Frasch, S., Cristòfol-Clough, M., Riniker, S., 2018. Anisotropic reaction field correction for long-range electrostatic interactions in molecular dynamics simulations. J. Chem. Phys. 148, 234105.

Sirk, T.W., Moore, S., Brown, E.F., 2013. Characteristics of thermal conductivity in classical water models. J. Chem. Phys. 138, 064505.

Skeel, R.D., 2016. An alternative construction of the Ewald sum. Mol. Phys. 114, 3166–3170.

Smith, W., 1992. A replicated data molecular dynamics strategy for parallel Ewald sum. Comp. Phys. Commun. 67, 392–406.

Smith, P.E., van Gunsteren, W.F., 1995. Reaction field effects on the simulated properties of liquid water. Mol. Sim. 15, 233–245.

Sperb, R., 1998. An alternative to Ewald sums. Part 1. Identities for sums. Mol. Sim. 20, 179–200.

Stenhammar, J., Linse, P., Karlström, G., 2009. Nondielectric long-range solvation of polar liquids in cubic symmetry. J. Chem. Phys. 131, 164507.

Stransky, M., 2016. Monte Carlo simulations of ionization potential depression in dense plasmas. Phys. Plasmas 23, 012708.

Sutmann, G., Stegailov, V., 2006. Optimization of neighbor list techniques in liquid matter simulations. J. Mol. Liq. 125, 197–203.

Tironi, I.G., Sperb, R., Smith, P.E., van Gunsteren, W.F., 1995. A generalized reaction field method for molecular dynamics simulations. J. Chem. Phys. 102, 5451–5459.

Tironi, I.G., Luty, B.A., van Gunsteren, W.F., 1997. Space-time correlated reaction field. A stochastic dynamical approach to the dielectric continuum. J. Chem. Phys. 106, 6068–6075.

Todd, B.D., Daivis, P.J., 1998. Nonequilibrium molecular dynamics simulations of planar elongational flow with spatially and temporally periodic boundary conditions. Phys. Rev. Lett. 81, 1118–1121.

Tornberg, A.-K., 2016. The Ewald sums for singly, doubly and triply periodic electrostatic systems. Adv. Comput. Math. 42, 227–248.

Tuckerman, M.E., 2010. Statistical Mechanics: Theory and Molecular Simulation. Oxford University Press, Oxford.

Verlet, L., 1967. Computer "experiments" on classical fluids. I. thermodynamical properties of Lennard-Jones molecules. Phys. Rev. 159, 98–103.

Villarreal, M.A., Montich, G.G., 2005. On the Ewald artifacts in computer simulations. The testcase of the octaalanine peptide with charged termini. J. Biomol. Struct. Dyn. 23, 135–142.

Waibel, C., Feinler, M.S., Gross, J., 2019. A modified shifted force approach to the Wolf summation. J. Chem. Theory Comput. 15, 572–583.

Wallace, J.A., Shen, J.K., 2012. Charge-leveling and proper treatment of long-range electrostatics in all-atom molecular dynamics at constant pH. J. Chem. Phys. 137, 184105.
Wang, S.S., Krumhansl, J.A., 1972. Superposition approximation. II. High density fluid argon. J. Chem. Phys. 56, 4287–4290.
Wang, H.Y., LeSar, R., 1996. An efficient fast-multipole algorithm based on expansion in the solid harmonics. J. Chem. Phys. 104, 4173–4179.
Wang, D.-B., Hsiao, F.-B., Chuang, C.-H., Lee, Y.-C., 2007. Algorithm optimization in molecular dynamics simulation. Comp. Phys. Commun. 177, 551–559.
Wassenaar, T.A., Mark, A.E., 2006. The effect of box shape on the dynamic properties of proteins simulated under periodic boundary conditions. J. Comput. Chem. 27, 316–325.
Watanabe, H., Suzuki, M., Ito, N., 2011. Efficient implementations of molecular dynamics simulations for Lennard-Jones systems. Prog, Theor. Phys. 126, 203–235.
Welling, U., Germano, G., 2011. Efficiency of linked Verlet algorithms. Comp. Phys. Commun. 182, 611–615.
Wells, B.A., Chaffee, A.L., 2015. Ewald summation for molecular simulations. J. Chem. Theory Comput. 11, 3684–3695.
White, C.A., Head-Gordon, M., 1994. Derivation and efficient implementation of the fast multipole method. J. Chem. Phys. 101, 6593–6605.
White, C.A., Johnson, B.G., Gill, P.M.W., Head-Gordon, M., 1994. The continuous fast multipole method. Chem. Phys. Lett. 230, 8–16.
White, J.A., Román, F.L., González, A., Velasco, S., 2008. Periodic boundary conditions and the correct molecular-dynamics ensemble. Phys. A 387, 6705–6711.
Wilson, K., Vaughn, N., Krasny, R., 2021. A GPU-accelerated fast multipole method based on barycentric Lagrange interpolation and dual tree traversal. Comp. Phys. Commun. 265, 108017.
Winkel, M., Speck, R., Hübner, H., Arnold, L., Krause, R., Gibbon, P., 2012. A massively parallel, multi-disciplinary Barnes-Hut tree code for extreme-scale N-body simulations. Comp. Phys. Commun. 183, 880–889.
Wolf, D., 1992. Reconstruction of NaCl surfaces from a dipolar solution to the Madelung problem. Phys. Rev. Lett. 68, 3315–3318.
Wolf, D., Keblinski, P., Phillpot, S.R., Eggebrecht, J., 1999. Exact method for the simulation of Coulombic systems by spherically truncated, pairwise r^{-1} summation. J. Chem. Phys. 110, 8254–8282.
Woodcock, L.V., Singer, K., 1971. Thermodynamic and structural properties of liquid ionic salts obtained by Monte Carlo computation. Trans. Faraday Soc. 67, 12–30.
Yao, Y., Capecelatro, J., 2021. An accurate particle-mesh method for simulating charged particles in wall-bounded flows. Powder Tech. 387, 239–250.
Yao, Z., Wang, J.-S., Liu, G.-R., Cheng, M., 2004. Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method. Comp. Phys. Commun. 161, 27–35.
Yokota, R., Barba, L.A., Narumi, T., Yasuoka, K., 2013. Comp. Phys. Commun. 184, 445–455.
Zhang, X., Bridson, R., 2014. A PPPM fast summation method for fluids and beyond. ACM Trans. Graph. 33, 206.
Zhou, Z., Payne, P., Vasquez, M., Kuhn, N., Levitt, M., 1996. Finite-difference solution of the Poisson-Boltzmann equation. Complete elimination of self-energy. J. Comput. Chem. 17, 1344–1351.

This page intentionally left blank

Chapter 6

Monte Carlo simulation

The Monte Carlo (MC) method is a stochastic strategy that relies on probabilities. To simulate fluids, transitions between different states or configurations are achieved by (1) generating a random trial configuration; (2) evaluating an "acceptance criterion" by calculating the change in energy and other properties in the trial configuration; and (3) comparing the acceptance criterion to a random number and either accepting or rejecting the trial configuration.

It is important to realize that not all states will make a significant contribution to the configurational properties of the system. To accurately determine the properties of the system in the finite time available for the simulation, it is important to sample those states that make the most significant contributions. This is achieved by generating a Markov chain, which is a sequence of trials in which the outcome of successive trials depends only on the immediate predecessor. In a Markov chain, a new state will only be accepted if it is more "favorable" than the existing state. In the context of a simulation using an ensemble, this usually means that the new trial state is lower in energy.

A MC simulation samples from a $3N$-dimensional space represented by the positions of particles. Unlike molecular dynamics (MD) methods (Chapter 7), particle momenta are not involved. It is not necessary to know particle momenta to calculate thermodynamic properties because the momenta contribute exclusively to the ideal gas term. Deviations from ideal gas behavior are caused by interactions between particles, which can be calculated from a potential energy function (Chapters 3 and 4). The potential energy depends only on the positions of atoms and not on their momenta. In effect, a MC simulation calculates the residual thermodynamic properties (Chapter 2) that result in deviations from ideal gas behavior. The appropriate ideal gas term can be simply added at the conclusion of the simulation to obtain the total thermodynamic property.

The general underlying principles of the MC method that underpin its use in molecular simulation have been examined by Hammersley and Handscomb (1964), Soboľ (1994), Sabelfeld (1991), and Moony (1997). Gilks et al. (1996) have detailed the importance of the Markov chain in MC processes. A very broad range of MC methods have been documented by Kroese and Rubinstein (2011) and Kroese et al. (2011). Binder (1995a),

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00004-6

Binder and Heermann (1997), Newman and Barkema (1999), and Landau and Binder (2000) have discussed MC in the context of the statistical physics relevant to molecular simulation. Binder (1997) has reviewed the application of the MC methods to problems in statistical physics in general. Many molecular simulation applications of MC have been detailed by Ferguson et al. (1999), and MC algorithms for polymers have been addressed by Binder (1995b). Descriptions of MC algorithms also form part of other texts (Allen and Tildesley, 2017; Berendsen, 2007; Hansen and McDonald, 2013; Heermann, 1990; Frenkel and Smit, 2023; Tuckerman, 2010).

In this chapter, the theoretical basis of MC simulations is discussed with emphasis on contemporary techniques. Algorithms for phase equilibria are detailed separately in Chapter 10. It should be noted that the application of MC algorithms is ensemble dependent. Sampling in the various ensembles is discussed in detail in Chapter 9.

6.1 Basic concepts

6.1.1 Background

The average of any thermodynamic property $\langle A(\mathbf{r}^N)\rangle$ can be obtained by evaluating the following multidimensional integral over the $3N$ degrees of freedom on the N particles in the system,

$$\langle A(\mathbf{r}^N)\rangle = \int A(\mathbf{r}^N)\rho(\mathbf{r}^N)d\mathbf{r}^N, \tag{6.1}$$

where $\rho(\mathbf{r}^N)$ is the probability of obtaining configuration $\mathbf{r}^N$, which depends on the potential energy (U) of the configuration,

$$\rho(\mathbf{r}^N) = \frac{\exp\left[-\beta U(\mathbf{r}^N)\right]}{\int \exp[-\beta U(\mathbf{r}^N)]d\mathbf{r}^N} \tag{6.2}$$

and $\beta = 1/kT$, where T denotes temperature and k is Boltzmann's constant. These integrals cannot be evaluated analytically, and conventional methods of integration are also not feasible. For example, to apply either Simpson's rule or the trapezium rule to evaluate a $3N$-dimensional integral would require m^{3N} function evaluations, where m is the number of points required to determine the integral in each dimension. The MC solution is to generate a large number of trial configurations $\mathbf{r}^N$ and replace the integrations by summations over a finite number of configurations. If the configurations are chosen randomly, Eq. (6.1) becomes:

$$\langle A(\mathbf{r}^N)\rangle = \frac{\sum_{i=1}^{N_{trial}} A_i(\mathbf{r}^N)\exp\left[-\beta U_i(\mathbf{r}^N)\right]}{\sum_{i=1}^{N_{trial}} \exp[-\beta U_i(\mathbf{r}^N)]} \tag{6.3}$$

In practice, this simple approach is not feasible because random sampling yields many configurations, which have a very small Boltzmann factor. Such configurations make very little contribution to the average. Therefore, a prohibitively large number of configurations are required to obtain the correct answer.

6.1.2 Metropolis sampling and Markov chains

The limitations of random sampling can be avoided by generating configurations that make a large contribution to the right-hand side of Eq. (6.3). This is the philosophy behind Metropolis sampling (Metropolis et al., 1953). Metropolis sampling biases the generation of configurations toward those that make the most significant contribution to the integral. It generates states with a probability of $\exp[-\beta U(\mathbf{r}^N)]$ and counts each of them equally. In contrast, simple MC integration generates states with equal probability assigning them a weight of $\exp[-\beta U(\mathbf{r}^N)]$.

Metropolis sampling generates a Markov chain, which satisfies the conditions, that (1) the outcome of each trial depends only on the proceeding trail and (2) each trail belongs to a finite set of possible outcomes. The former condition highlights the distinction between MC and MD simulation (Chapter 7) methods. In a MD simulation, all the states are connected in time. Metropolis et al. (1953) showed that a transition probability matrix that ensures Eq. (6.2) is obeyed,

$$\left.\begin{array}{ll} \pi_{mn} = C_{mn} & \dfrac{\rho_n}{\rho_m} \geq 1, \quad m \neq n \\ \pi_{mn} = C_{mn}\left(\dfrac{\rho_n}{\rho_m}\right) & \dfrac{\rho_n}{\rho_m} < 1, \quad m \neq n \end{array}\right\}, \tag{6.4}$$

where ρ_m and ρ_n are the probability densities for states m and n given by Eq. (6.2) and C_{mn} is the conditional probability of choosing n as the trial state. The calculation of the integral in the denominator of Eq. (6.2) is not required because Eq. (6.4) only involves the ratio of the probability densities. Furthermore,

$$\sum_n \pi_{mn} = 1 \tag{6.5}$$

$$\rho_m \pi_{mn} = \rho_n \pi_{nm} \tag{6.6}$$

The Metropolis scheme is implemented by attempting a trial displacement and calculating the change in energy $\Delta U = U_n - U_m$. If $\Delta U < 0$, then $(\rho_n/\rho_m) > 1$, and the move is accepted. Otherwise, if $\Delta U > 0$, $(\rho_n/\rho_m) < 1$ and the move is accepted with a probability of $(\rho_n/\rho_m) = \exp(-\beta\Delta U)$. A random number R is generated and the move is accepted if $\exp(-\beta\Delta U) \geq R$.

Typically, this procedure is summarized by defining the probability of acceptance as:

$$p = \min[1, \exp(-\beta \Delta U)] \tag{6.7}$$

It should be noted that Eq. (6.7) only represents the acceptance criteria resulting from molecular displacement. A different criterion is required as a result of either a volume change or a change in a number of particles. The criteria for accepting different types of moves are discussed in Chapter 9 in the context of MC algorithms for different ensembles. Efficiency issues relating to the Metropolis algorithm are discussed elsewhere (Stedman et al., 1998).

ALGORITHM 6.1 Basic MC procedure for accepting changes in atomic displacements.

```
Part 1   loop cycle ← 1 ... maxCycle
Part 2      Attempt trial displacements:
            loop atom ← 1 ... maxAtom
Part 2.1       Generate a trial displacement:
               rxTrial ← pbc(rx_atom + (2 × rand() - 1) × dMax)
               ryTrial ← pbc(ry_atom + (2 × rand() - 1) × dMax)
               rzTrial ← pbc(rz_atom + (2 × rand() - 1) × dMax)
Part 2.2       Calculate change in energy:
               Calculate energy of the atom (UTrial) at trial position.
               ΔU ← UTrial - U_atom
Part 2.3       Apply Metropolis acceptance criterion:
               if (ΔU < 0 or exp(-β × ΔU) ≥ rand())
                  rx_atom ← rxTrial                    //accept move
                  ry_atom ← ryTrial
                  rz_atom ← rzTrial
                  U_atom ← UTrial
                  U ← U + ΔU
                  numAccept ← numAccept + 1
               end if
            end atom loop
Part 2.4    Update periodically the maximum displacement:
            if (mod(cycle, updateCycle) = 0)
               acceptRatio ← numAccept/(cycle × atomMax)
               if (acceptRatio > 0.5)
                  dMax ← 1.05 × dMax
               else
                  dMax ← 0.95 × dMax
               end if
            end if
         end cycle loop
```

Application of the Metropolis method is illustrated by Algorithm 6.1. Typically, a MC simulation is performed in cycles (Algorithm 6.1, Part 1). During each cycle, a displacement is attempted for every atom (Algorithm 6.1, Part 2). The atom to be

displaced can be chosen randomly or alternatively, each atom can be considered in turn as illustrated in Algorithm 6.1. It is also possible to displace every atom and apply the acceptance criterion to the combined move. New trial coordinates (Algorithm 6.1, Part 2.1) are obtained randomly up to a maximum value of *dMax*. This involves generating random numbers on the interval from 0 to 1 and applying periodic boundary conditions (pbc). The change in energy resulting from the trial displacement (Algorithm 6.1, Part 2.2) is simply the energy experienced by the atom at the trial position (*Utrial*) minus the energy of the atom at its existing position (U_{atom}). This change in energy is evaluated and the Metropolis acceptance (Algorithm 6.1, Part 2.3) rule is applied. If the move is accepted, the atom's coordinates, the atom's energy, and the total energy (*U*) are updated. The maximum allowed displacement (*dMax*) affects the acceptance rate. Experience indicates that an acceptance rate of approximately 50% is often desirable for a MC simulation. A small value of *dMax* will generally improve the acceptance rate but the phase space of the liquid may be sampled too slowly. Conversely, increasing the value of *dMax* will reduce the acceptance rate. It is common to adjust the value of *dMax* during the course of the simulation to maintain a 50% acceptance rate. In Algorithm 6.1 (Part 2.4), the value of *dMax* is adjusted at intervals specified by *updateCycle*. If *acceptRatio* > 0.5, the value of *dMax* is increased, whereas it is decreased if *acceptRatio* < 0.5.

It should be noted that there is no theoretical basis for using an acceptance rate of 50% and in some cases it may be actually detrimental to efficient sampling. Mountain and Thirumalai (1994) proposed an algorithm for determining the efficiency of MC simulations. They reported that an acceptance ratio of 20% was twice as efficient in generating a satisfactory sample as compared with the traditional acceptance rate of 50%. Therefore, it may be advantageous to first establish the optimal acceptance rate from short exploratory MC simulations.

6.2 Application to molecules

As illustrated by Algorithm 6.2 and discussed previously, application of the MC method to atoms is straightforward. However, the treatment of polyatomic molecules requires us to consider bond rotation in addition to translational motion.

6.2.1 Rigid molecules

The usual approach adopted for the MC simulation of rigid molecules is to combine translation and rotation of one molecule. The translational move is obtained by displacing randomly the center of mass of a molecule along each of its axes. The rotational move involves attempting a random change of the Euler angles φ, θ, and ψ that define molecular orientation (Chapter 6).

For a rigid molecule of arbitrary geometry, trial changes to the Euler angles can be obtained from:

$$\phi_{trial} = \phi + (2R_1 - 1)\Delta\phi_{\max} \tag{6.8}$$

$$\cos\theta_{trial} = \cos\theta + (2R_2 - 1)\Delta(\cos\theta)_{\max} \tag{6.9}$$

$$\psi_{trial} = \psi + (2R_3 - 1)\Delta\psi_{\max} \tag{6.10}$$

$\Delta\phi_{\max}$, $(\cos\theta)_{\max}$, and $\Delta\psi_{\max}$ are the maximum allowed changes in the Euler angles and R_1, R_2, and R_3 are random numbers between 0 and 1. Notice that in Eq. (6.9), a random change is made to cosθ rather than θ. Sampling in cosθ is used to achieve a uniform distribution of points. Following the attempted translation−rotational move, the energy experienced by the molecule in its trial position is calculated and the combined move is accepted with a probability given by Eq. (6.7). It should be noted that in the case of a linear rigid molecule, only Eq. (6.9) is required because the other Euler angles are unaffected.

There are alternatives to the above procedure. For example, Barker and Watts (1969) proposed a method for rotating the whole molecule. Furthermore, the use of Euler angles can be avoided by using the quaternion approach (Chapter 7). If quaternions are used, a new orientation is generated by rotating the quaternion vector to a new random orientation (Veseley, 1982). The application of these techniques is described elsewhere (Allen and Tildesley, 2017; Leach, 2001; Tildesley, 1993).

6.2.2 Small flexible molecules

The technique used for rigid molecules can be extended with minor modification (Tildesley, 1993) to flexible molecules of moderate size such as *n*-butane. A trial configuration is generated by (1) displacing the head atom; (2) changing the molecular orientation by making random changes to the Euler angles; and (3) changing the conformation by making a random change to the torsional angle χ in the range 0 to 2π. The trial configuration is accepted with a probability given by Eq. (6.7), where the change in energy now includes contributions from both intermolecular and intramolecular potentials.

6.2.3 Polymers

There is considerable interest in the application of MC algorithms (Binder 1995a,b) to determine the properties of large molecules such as polymers. In principle, the approach used for small flexible molecules could be extended to polymers. However, the practice of changing randomly the torsional angle is likely to lead to a high rejection rate. Even a relatively small change in the torsional angle in the middle of a large flexible molecule is likely to result in a large translational displacement of the terminal atoms. Consequently, there is a high probability of molecular overlap resulting in the rejection of the move.

Another limitation of the small flexible molecule approach is that determining Euler angles or using quaternion ions for each atom of a large molecule requires considerable computational effort. Instead, specific MC algorithms have been developed to model molecular chains. Some commonly used algorithms are described subsequently. Comprehensive reviews of alternative algorithms are available elsewhere (Binder, 1995a,b).

6.2.3.1 The reptation or slithering snake algorithm

Wall and Mandel (1975) proposed an algorithm that propagates a chain in the fashion of a slithering snake. The motion is alternatively described as "reptation" (de Gennes, 1971). The reptation algorithm was originally devised for a lattice polymer but it can be used also for an off-lattice polymer. A freely jointed linear polymer consists of $p + 1$ monomers, and p links. The reptation algorithm propagates the polymer via a four-stage process:

Stage 1: One end of the chain is chosen randomly to be the head of the chain, for example, atom 1.

Stage 2: The tail link and the corresponding tail atom $p + 1$ is removed. The tail link is attached to the head of the chain at a randomly chosen orientation.

Stage 3: The change in energy is calculated as $\Delta U = U_1(\text{trial}) - U_{p+1}(\text{old})$, which is the energy difference for swapping the tail atom to a new head position.

Stage 4: The move is accepted by applying Eq. (6.7).

The effect of this algorithm is to generate a slithering motion as illustrated by Fig. 6.1. The random choice of the head and tail atoms is important to avoid the chain being locked with the head unable to move because of potential overlap with neighboring molecules. This algorithm has been applied to polymers on a three-dimensional lattice (Wall et al., 1977) and to continuum fluids (Bishop et al., 1980; Brender and Lax, 1983).

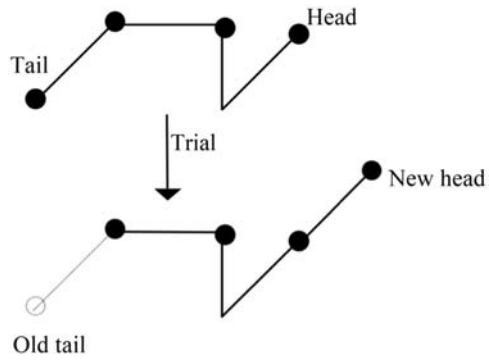


FIGURE 6.1 Illustration of a trial move using the reptation algorithm.

The application of the reptation algorithm is restricted generally to open linear chains characterized by free end units. Various modifications and improvements to the basic reptation algorithm have been proposed. Murat and Witten (1990) generalized the reptation algorithm by allowing the removal of an arbitrary monomer and replacing it randomly elsewhere in the chain. The two neighbors of the removed monomer are connected by a new bond. If the new bond is longer than the maximum bond length, the move is rejected. If the removal is allowed, an attempt is made to reconnect the removed monomer elsewhere. A collective reptation method has been reported (Pakula, 1987; Pakula and Geyler, 1987; Geyler et al., 1990; Reiter et al., 1990), which involves a conventional reptation of a kink-jump algorithm (see subsequent discussion) to one chain followed by specific rearrangements of other chains. The rearrangements of other chains potentially remove excluded volume violations caused by the first move chain.

Variations on the general reputation algorithm have proved particularly useful in the MC investigation of the properties of polymers (Foo and Pandey, 2000; Kreer et al., 2001; Terranova et al., 2007), particularly when lattice dynamics (Hua and Ke, 2005) is involved. Improvements include adaptations (Santos et al., 2001; Uhlherr et al., 2002) for parallel execution. The general principles have been applied to specific polymer mixtures (Gharibi et al., 2006), polydispersity (Rorrer and Dorgan, 2014), and melts (Lin et al., 2007; Kampmann et al., 2015; Karantrantos et al., 2019).

6.2.3.2 *The kink-jump algorithm*

The kink-jump algorithm was used initially (Verdier and Stockmayer, 1962) for lattice polymers. In this context, it effectively involves "flipping" a randomly chosen monomer from its position $\mathbf{r}_n$ to a trial position $\mathbf{r}'_n = \mathbf{r}_{n+1} + \mathbf{r}_{n-1} + \mathbf{r}_n$. At small intermolecular separations, strong intermolecular repulsion between monomer units results in an excluded, or empty volume. To account for the effect of excluded volume, a local "crankshaft" motion (Lax and Brender, 1977) must be added.

Baumgärtner and Binder (1979) formulated a kink-jump algorithm to study freely jointed linear off-lattice chains with excluded volume interactions. If the chain has p links and $p + 1$ monomers, the kink-jump algorithm proceeds in the following fashion:

Stage 1: A monomer i is chosen randomly.

Stage 2: If i is either the head ($i = 1$) or tail ($i = p + 1$) of the chain, a trial state is obtained by choosing a random orientation for the first or pth link, respectively. Otherwise, i is rotated in a circle around the vector joining $i-1$ to $i + 1$ by choosing an angle $\Delta\phi$ uniformly in the range $-\Delta\phi_{\max}$ to $\Delta\phi_{\max}$.

Stage 3: The trial configuration is accepted according to Eq. (6.7).

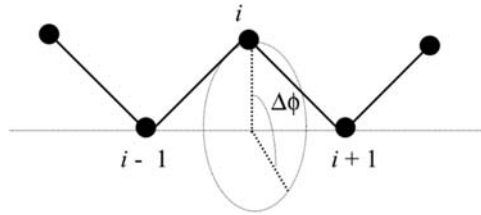


FIGURE 6.2 Illustration of a trial move using the kink-jump algorithm for off-lattice chains.

The move is illustrated in Fig. 6.2. The kink-jump algorithm has been particularly useful (Stampe and Sokolov, 2001) in describing Rouse behavior (Doi and Edwards, 1995) of polymer solutions.

It should be noted that the various configurations changing MC moves are not mutually exclusive and it is common to combine different moves in a single simulation. Mańka et al. (2013) combined reputation and kink-jump moves to study polymer chains on a cubic lattice. A similar combined approach has been used for polymer membranes (Pandey and Seyfarth, 2003) and other properties (Bentrem et al., 2002a,b; Maycock and Bhattacharya, 2006). A comparison of the kink-jump algorithm and configuration bias MC methods has been reported (Tsonchev et al., 2004).

6.2.3.3 The pivot algorithm

The pivot algorithm (Lal, 1969) is used in conjunction with a self avoiding walk (SAW) (Binder and Heermann, 1997) to generate new configurations on a lattice. A pivot point is chosen randomly at some point on the SAW. A new chain is obtained by rotating part of the chain around the pivot point. If the position of the rotated chain overlaps with the unmoved part of the chain, the move is rejected. Otherwise, acceptance of the move is governed by Eq. (6.7). Although there is a high probability of rejection, relatively few successful moves are required to obtain a radically different configuration. For a chain of p links, Madras and Sokal (1988) have demonstrated that the pivot algorithm is p times more efficient than standard SAW algorithms. The pivot algorithm can also be used for off-lattice simulations (Freire and Horta, 1976). It is an efficient means of studying single polymer chains but it is unlikely to be useful for polymer melts because of the very high probability of overlap with other chains. Other SAW-based algorithms have been reviewed (Sokal, 1995). The pivot algorithm can be applied to ring polymers (Zifferer and Preusser, 2001; Kindt, 2002), star polymers (von Ferber et al., 2009), and dendrimers (Gergidis et al., 2009). Very long SAW simulations with the pivot algorithm have been attempted (Clisby, 2018).

6.3 Advanced techniques

Many improvements to the basic MC techniques have been proposed. Allen and Tildesley (2017) have discussed “smarter MC” sampling, and

Binder (1995a,b) has detailed the application of MC techniques to polymers. In this section, we will focus on a few important methods. One of the most significant advances has been the application of the Gibbs ensemble grand canonical MC method (Panagiotopoulos, 1987) to predict phase equilibria. The histogram reweighting technique (Ferrenberg and Swendsen, 1989) is another important MC for phase equilibria. The Gibbs ensemble and other techniques for phase equilibria are discussed separately in Chapter 10.

6.3.1 Configurational bias MC

An important application of the MC method is the simulation of a grand canonical ensemble (Chapter 9). The simulation of a grand canonical ensemble requires both particle transfer and insertion moves. The transfer or insertion of either atoms or small molecules is relatively straightforward and it can be handled via a conventional Widom (1963) method. However, it is difficult to move large molecules such as polymers because of the very high probability of steric overlap. This problem has been addressed by configurational bias MC (CBMC) methods (Siepmann, 1990; Frenkel et al., 1991; Siepmann and Frenkel, 1992; de Pablo et al., 1992a,b; Laso et al., 1992, Mooij and Frenkel, 1996). CBMC combines elements of sampling proposed by Rosenbluth and Rosenbluth (1955) with segment-by-segment insertion. The general approach has been reviewed (Frenkel, 1995; Frenkel and Smit, 2023). It has proved (de Pablo et al., 1999: Panagiotopoulos, 2000; Singh et al., 2009) particularly advantageous in the simulation of phase equilibria (Chapter 10). The use of a continuous fractional MC method (Shi and Maginn, 2008) is also advantageous (Rahbari et al., 2021; Mayr et al., 2023).

6.3.1.1 Lattice chains

Let us consider the conventional insertion of a three-unit molecule beginning at point s on a two-dimensional lattice illustrated by Fig. 6.3. From point s there are four possible lattice sites, however only sites a and b are free. Therefore, the probability of a successful insertion is only 50%. If site b is chosen, further growth of the molecule is inhibited, whereas if site c was chosen, there would be three further sites to grow the molecule. On average, only one trial in twelve will successfully grow a molecule from site s.

CBMC (Siepmann, 1990; Siepmann and Frenkel, 1992) tackles this problem in a different fashion. Starting from s, the vacant sites (a and b) are identified and one of these vacant sites is selected randomly. Unlike the conventional MC procedure, the occupied sites are not considered in the selection process. A Rosenbluth weight is calculated for the move,

$$W_i = \frac{n_{avail}}{n} W_{i-1} \tag{6.11}$$

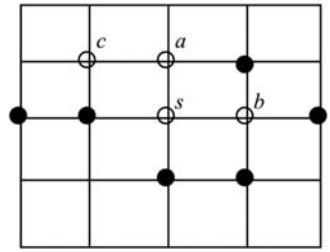


FIGURE 6.3 Insertion possibilities for a three-unit molecule on a two-dimensional lattice from a starting point *s*. Closed and open circles represent occupied and vacant sites, respectively.

where $W_{i\text{-}1}$ is the weight for the previous step ($W_0 = 1$) and n_{avail} is the number of available sites where n is the total number of neighboring sites excluding the site occupied by the previous unit. For the present example, $W_1 = 2/4 = 1/2$. If site b is chosen, the attempt is abandoned because there is no site available for the third unit. If site a is chosen, the adjacent sites are examined and the only vacant site is filled with the third unit. The Rosenbluth weight for this step is $1/3 \times 1/2 = 1/6$. The overall statistical weight for the move is obtained by multiplying the number of successful trials by the Rosenbluth weight, that is, $1/2 \times 1/6 = 1/12$. This is the same result as conventional sampling. However, in the conventional scheme only one trial in twelve is successful whereas the proportion of successful trials for the method is one in two.

The algorithm can be extended to account for the interactions between the growing chain and its lattice neighbors. If $U_\Gamma(i)$ denotes the energy of segment i when occupying a particular site Γ, that site is chosen with a probability of,

$$p_\Gamma(i) = \frac{\exp[-\beta U_\Gamma(i)]}{Z_i} \tag{6.12}$$

where Z_i is defined as,

$$Z_i = \sum_{\Gamma=1}^{b} \exp[-U_\Gamma(i)] \tag{6.13}$$

that is, the sum of the Boltzmann factors for the b possible orientations of the chain segment. The sum of Boltzmann factors plays the role of n_{avail} in Eq. (6.11). Therefore the Rosenbluth weight for an entire chain of length l can be obtained from,

$$W_l = \exp[-\beta U_{total}(l)] \prod_{i=2}^{l} \frac{Z_i}{b} \tag{6.14}$$

where $U_{total}(l)$ is the total energy of the chain, which is equal to the sum of the individual segment energies, $U_\Gamma(i)$. The residual chemical potential is related to the average Rosenbluth weight.

$$\mu_r = kT\ln\langle W_l\rangle \tag{6.15}$$

There are many examples for which the properties of both polymers (Abreu and Escobedo, 2005; Cootes et al., 2000) and proteins (Zou and Saven, 2003) can be obtained using lattice-based CBMC.

6.3.1.2 Flexible chains

de Pablo et al. (1992a, 1993) and Siepmann and Frenkel (1992) demonstrated that the algorithm could be generalized for a fully flexible chain. de Pablo et al. (1992a) introduced the term "continuum configurational bias (CCB)" to describe the application of method to flexible chains. When each segment of a flexible chain is grown, a subset k random directions is chosen. The energy $U_\Gamma(i)$ for each of these orientations is calculated and an orientation is chosen with a probability of,

$$p_\Gamma(i) = \frac{\exp[-\beta U_\Gamma(i)]}{\sum_{\Gamma=1}^{k}\exp[-\beta U_\Gamma(i)]} \tag{6.16}$$

and the Rosenbluth factor is accumulated from:

$$W_i = W_{i-1}\frac{1}{k}\sum_{\Gamma=1}^{k}\exp[-\beta U_\Gamma(i)] \tag{6.17}$$

The trial conformation is accepted if the following criterion is satisfied,

$$R \le \frac{W_{l,trial}}{W_{l,old}} \tag{6.18}$$

where R is a random number between 0 and 1.

The general procedure for implementing a CBMC simulation is outlined in Algorithm 6.2. It involves three distinct stages. The first stage (Algorithm 6.2, Part 1) generates a trial configuration and determines the Rosenbluth weight. This involves selecting a chain randomly (Algorithm 6.2, Part 1.1); selecting a starting position randomly (Algorithm 6.2, Part 1.2); growing the chain and calculating the Rosenbluth weight (*wTrial*) for the trial chain (Algorithm 6.2, Part 1.3). The starting point can occur anywhere on the chain. If the value of *start* equals 1, the whole chain is regrown, whereas any other value means that the chain is only regrown partially. The number of attempted insertions *k* is specified and the chain is grown using Algorithm 6.3. The second stage (Algorithm 6.2, Part 2) calculates the Rosenbluth weight for the old configuration (*wOld*) by using Algorithm 6.3 to retrace the chain.

To retrace the old chain, Algorithm 6.3 is used with k set to unity. In stage 3 (Algorithm 6.3, Part 3) the old and trial Rosenbluth weights are used to either accept or reject the trial configuration (Algorithm 6.3, Part 3.1) in accordance with criterion (6.15).

ALGORITHM 6.2 CBMC algorithm for a chain.

```
Part 1      New trial configuration:
            trialConfiguration ← true
Part 1.1    Select a chain randomly:
            chain ← int(rand() × numberOfChains) + 1
Part 1.2    Select starting position:
            start ←Int(rand() × Length_chain)
            k ← number of attempted insertions
Part 1.3    Grow chain from start using Algorithm 6.3
            and determine wTrial.

Part 2      Old configuration:
            trialConfiguration ← false
            k ← 1
Part 2.1    Retrace chain from start using Algorithm 6.3
            and determine wOld.

Part 3      Acceptance of trial:
Part 3.1    test ← wTrial/wOld
            if (test > rand())
               Accept move.
            end if
```

Algorithm 6.3 is responsible for fully or partially regrowing (*trialConfiguration* = true) a chain or retracing part or all of an existing chain (*trialConfiguration* = false). At the outset (Algorithm 6.3, Part 1), the coordinates of that part of the chain, which are not involved in the regrowth, are stored. The chain is either regrown or retraced by considering each atom in turn (Algorithm 6.3, Part 2). If the first atom of the chain is encountered it is treated specially (Algorithm 6.3, Part 2.1). For the remaining atoms, k trial positions are located (Algorithm 6.3, Part 2.2). The location of the trial positions depends on the nature of the chain (Algorithm 6.3, Part 2.3). If the chain is fully flexible, the trial positions can be located randomly, whereas bond and angular constraints must be imposed for semiflexible or rigid molecules. The external energy or energy from nonbonded interactions is calculated and the Rosenbluth weight is calculated. After the k orientations are found, the Rosenbluth weight is updated and one of the orientations is selected (Algorithm 6.3, Part 2.4). The selection is made with the probability given by Eq. (6.16). The algorithm has been discussed in detail by Frenkel and Smit (2023).

ALGORITHM 6.3 Growing/retracing either a fully flexible or semi-flexible chain.

```
Part 1   Store positions that are not regrown or retraced:
         loop atom ← 1 .... start - 1
            rxStore_atom ← rx_atom
            ryStore_atom ← ry_atom
            rzStore_atom ← rz_atom
         end atom loop

Part 2   Grow/retrace chain:
         W ← 1                                        //initial Boltzmann weight
         loop atom ← start ... chainLength
Part 2.1    if (atom = 1)
               if(trialConfiguration)                 //generate random positions
                  rxTrial_1 ← rand() × boxLength
                  ryTrial_1 ← rand() × boxLength
                  rzTrial_1 ← rand() × boxLength
               else
                  rxTrial_1 ← rand() × rxStore_1
                  ryTrial_1 ← rand() × ryStore_1
                  rzTrial_1 ← rand() × rzStore_1
               end if
               Calculate change in energy:
               Calculate energy of the atom (UTrial) at trial position.
               ΔU_1 ← UTrial - U_1
               W ← k × exp(-β × ΔU_1)                 //Rosenbluth factor
            else                                      //consider other atoms
               sumW ← 0
Part 2.2       loop attempt ← 1 ... k
                  if (trialConfiguartion = false and attempt = 1)
                     rxTrial ← rx_atom
                     ryTrial ← ry_atom
                     rzTrial ← rz_atom
                  else
Part 2.3             Generate a trial position (rxTrial, ryTrial, rzTrial) in
                     accordance with the chain's constraint.
                  end if
                  Calculate change in energy:
                  Calculate energy (UTrial) at trial position.
                  ΔU_atom ← UTrial - U_atom
                  WTrial_attempt ← exp(-β × ΔU_atom)
                  sumW ← sumW + WTrial_attempt
               end loop
               W ←W × sumW                            //update Rosenbluth factor
```

```
Part 2.4        if (trialConfiguration)
                   Select one of the trial attempts (position):
                   selectW ← rand() × sumW
                   cummulativeW ← Wtrial_1
                   position ← 1
                   loop while (cummulativeW < selectW )
                      position ← position + 1
                      cummulativeW ← cummulativeW + WTrial_position
                   end while loop
                   rxStore_atom ← rxTrial_position
                   ryStore_atom ← ryTrial_position
                   rzStore_atom ← rzTrial_position
                else
                   rxStore_atom ← rx_atom
                   ryStore_atom ← ry_atom
                   rzStore_atom ← rz_atom
                end if
             end if
          end atom loop
```

The computation time and efficiency of Algorithm 6.3 depends largely on the value of k. If $k = 1$, the algorithm is equivalent to Widom particle insertion, which is inefficient for chain molecules. For chains, $k > 1$ is required to obtain improvements in efficiency. However, if k is too large, too much time is used in calculating Rosenbluth weights for trial positions that are very close to each other. The effect of the value k has been investigated by Frenkel et al. (1991). For small chains, an excessive value of k leads to poor computational performance, whereas larger k values are required to sample long chains. Typically, a value of $k = 4$ is often a good compromise between efficient sampling and excessive computation (Frenkel, 1995).

An important application of methods is the simulation of phase equilibria involving either polymers or chain molecules. The method has been combined (Laso et al., 1992; Mooij et al., 1992) with the Gibbs ensemble (Chapter 10) to predict the phase equilibria of chain molecules. In particular, good results have been obtained (Siepmann et al., 1993a,b; Smit et al., 1995, 1998) for the vapor−liquid equilibria of alkanes from pentane to octatetracontane and branched alkanes (Cui et al., 1997). A hybrid Gibbs ensemble/MD method has been used (de Pablo, 1995) to improve the efficiency of particle interchange. Vapor−liquid coexistence of square-well chains consisting of up to one hundred segments has been reported (Escobedo and de Pablo, 1996).

The method is clearly superior to the simple Widom test particle approach. However, it has some limitations. In common with the Widom method, it relies on finding vacancies in the fluid, therefore, insertion problems will still arise for very long chains. A reasonably large system size (> 200 chains) is also required to avoid finite size effects.

Improving methods is an active area of research. Shelley and Patey (1994, 1995) developed a configurational bias method for the insertion and deletion of ions and water molecules in grand canonical MC simulations. Escobedo and de Pablo (1995a) proposed an extended continuum configurational bias (ECCB) method, which is particularly useful for very large molecular chains. Suen et al. (1997) used the ECCB method to examine the collapse of a polymer chain in an athermal solvent. Configurational bias sampling has also been used in an expanded ensemble (Escobedo and de Pablo, 1995b). The expanded ensemble formalism has been used successfully (Escobedo and de Pablo, 1997) to simulate both pure athermal mesogenic chains and mixtures.

Deem and Bader (1996) have described a configurational bias method that can be applied to either linear or cyclic peptides. Haas and Hilfer (1996) have reported continuum MC calculations for stiff chain molecules at surfaces. Methods of optimization have also been addressed (Mooij and Frenkel, 1996). Sadanobu and Goddard (1997) combined continuum configurational bias sampling with a Boltzmann factor-biased enrichment method. The resulting algorithm requires the simulation of dramatically fewer chains to obtain an accurate value of free energy. Vendruscolo (1997) has proposed modifications to the configurational bias method involving the regrowth of the internal segments of polymer chains. The computational efficiency of CBMC is improved considerably via parallel implementations (Vlugt, 1999; Cezar et al., 2020; Glaser et al., 2015).

6.3.2 Cluster moves

Systems of strongly interacting particles such as ions in a medium of low dielectric constant or surfactants exhibit significant clustering. One consequence of clustering is that independent particle moves may be inhibited. In such cases, it may be preferable to attempt to move clusters of atoms rather than individual atoms. Swendsen and Wang (1987) proposed a cluster algorithm to facilitate MC sampling of near-critical systems. Later, Wu et al. (1992) modified the procedure for self-assembling systems. The key feature of the Swedsen-Wang algorithm is the creation of virtual bonds between neighboring particles. Wu et al. (1992) proposed that the probability of bond formation is only a function of distance. Once clusters have formed, they are moved as a whole.

The probability of accepting a cluster move depends on the probabilities of bond formation at the old and new cluster positions. Instead of applying Eq. (6.7), cluster moves are accepted with a probability given by,

$$p = \min\left[1, \exp(-\beta\Delta U)\prod_{kl}\frac{1-p^f(k,l)}{1-p^r(k,l)}\right] \tag{6.19}$$

where k denotes a particle on the cluster, l denotes a particle outside the cluster, p^r is the probability of the reverse move, and p^f is the probability of the

forward move. Wu et al. (1992) have used the simplifying assumption that $p(i,j) = 1$ for r_{ij} less than a specified cluster distance r_c, and $p(i,j) = 0$ beyond that distance. Orkoulas and Panagiotopoulos (1993, 1994) have demonstrated the usefulness of cluster moves in the simulation of vapor−liquid coexistence of ionic fluids. Cluster moves have been examined in detail by Frenkel (1995) and Frenkel and Smit (2023). The general algorithm for implementing a cluster move is:

1. Identify clusters;
2. Choose a cluster at random;
3. Change the position of the cluster by applying the same random displacement to all atoms of the cluster;
4. Calculate the energy and probability factor experienced by all atoms of the cluster in the new position; and
5. Apply Eq. (6.19).

A possible implementation of this procedure is given by Algorithm 6.4. Before the clusters are identified (Algorithm 6.4, Part 1.1), every atom is available $taken_{atom} \leftarrow$ false for cluster formation. The clusters are identified by choosing a "seed" atom *clusterSeed* (Algorithm 6.4, Part 1.2) and finding all atoms within the *separation* distance from *clusterSeed* (Algorithm 6.4, Parts 1.3 and 1.4). These atoms are now not available $taken_{atom} \leftarrow$ false for any other cluster. The coordinates are assigned to a two-dimensional array $rx_{cluster,\ cAtomIndex} \leftarrow rx_{atom}$. This process is continued until all atoms are assigned to a cluster $totalTaken = maxAtoms$. Algorithm 6.4 also allows for the possibility that some atoms may be outside any cluster (Algorithm 6.4, Part 1.5). The MC procedure (Algorithm 6.4, Part 2) is broadly similar to Algorithm 6.1. For each cycle, a *cluster* is selected randomly (Algorithm 6.4, Part 2.1). All the atoms ($numOfAtoms_{cluster}$) of the chosen cluster are displaced randomly by the same amount (Algorithm 6.4, Part 2.2). For each atom of the cluster, the change in energy experienced at the trial position is calculated (Algorithm 6.4, Part 2.3). The probability factor is also determined. The change in energy ($\Delta Ucluster$) for the move is simply the sum of the change in energy experienced by atoms of the cluster at their trial positions. The combined probability factor (*prob*) is the product of one minus the probability factor for all the atoms in the cluster. If *prob* becomes equal to zero, the move is rejected immediately because this signifies that an atom of the trial cluster has become part of another cluster and microscopic reversibility has been violated. If microscopic reversibility is not violated, the decision to accept the move depends on the change in energy of the move (Algorithm 6.4, Part 2.4). The maximum allowed displacement is updated periodically (Algorithm 6.4, Part 2.5) to obtain the desired acceptance rate. It should be noted that cluster moves should be combined with single particle moves, to allow for a change in both the configuration of the cluster and the number of particles in the cluster.

ALGORITHM 6.4 MC procedure for accepting cluster moves.

```
Part 1    Identify Clusters:
Part 1.1  loop atom ← 1 ... maxAtom
              taken_atom ← false
          end atom loop
          totalTaken ← 0
          clusterSeed ← 0
          cluster ← 0
Part 1.2  loop while (totalTaken < maxAtom)
              cAtomIndex ← 0
              cluster ← cluster + 1
              clusterSeed ← clusterSeed + 1
              loop atom ← 2 .. maxAtom
Part 1.3          if (atom ≠ clusterSeed and taken_atom = false)
                      Calculate the separation between atom
                      and clusterSeed.
Part 1.4              If (separation < rCluster)       //member of cluster found
                          cAtomIndex ← cAtomIndex + 1
                          rx_ctuster, cAtomIndex ← rx_atom
                          ry_cluster, cAtomIndex ← ry_atom
                          rz_cluster, cAtomIndex ← rz_atom
                          taken_atom ← true
                          totalTaken ← totalTaken + 1
                      end if
                  end if
              end atom loop
Part 1.5      if (taken_clusterSeed = false)              //take care of lone atoms
                  cAtomIndex ← cAtomIndex + 1
                  rx_ctuster, cAtomIndex ← rx_clusterSeed
                  ry_cluster, cAtomIndex ← ry_atomSeed
                  rz_cluster, cAtomIndex ← rz_atomSeed
                  taken_clusterSeed ← true
                  totalTaken ← totalTaken + 1
              end if
              numOfAtoms_cluster ← cAtomIndex
          end while loop
          totalCluster ← cluster
Part 2    MC Simulation:
          loop cycle ←1 ... maxCycle
Part 2.1  Select cluster at random:
          cluster ←[1, totalCluster]
Part 2.2  Attempt trial displacements of a cluster:
          loop cAtomIndex ← 1 ... numOfAtoms_cluster
              rxTrial_cluster,cAtomIndex
                    ← pbc(rx_cluster, cAtomIndex + (2 × rand() - 1) × dMax)
              ryTrial_cluster,cAtomIndex
                    ← pbc(ry_cluster,cAtomIndex + (2 × rand() - 1) × dMax)
              rzTrial_cluster,cAtomIndex
                    ← pbc(rz_cluster,cAtomIndex + (2 × rand() - 1) × dMax)
          end cAtomIndex loop
```

```
Part 2.3 Calculate Energy/Probability factors of trial position:
        prob ← 1
        cAtomIndex ← 1
        ΔUCluster ← 0
        loop
            Calculate ΔU_cAtomIndex from interaction with outside atoms.
            Calculate prob_cAtomIndex
            ΔUCluster ← ΔUCluster + ΔU_cAtomIndex
            prob ←prob*(1 - prob_cAtomIndex)
            cAtomIndex ← cAtomIndex + 1
        while (cAtomIndex ≤ numOfAtoms_cluster and prob = 1)
Part 2.4 Apply acceptance criterion:
        If (prob = 1)
            if (ΔUCluster < 0 or exp(-β × UCluster) ≥ rand())
                loop cAtomIndex ← 1 ... numOfAtoms_cluster
                    rx_cluster,cAtomIndex ← rxTrial_cluster, cAtomIndex
                    ry_cluster,cAtomIndex ← ryTrial_cluster,cAtomIndex
                    rz_cluster,cAtomIndex ← rzTrial_cluster,cAtomIndex
                    U_cluster,cAtomIndex ← U_cluster,cAtomIndex + ΔU_cAtomIndex
                end cAtomIndex loop
                U ← U+ ΔUCluster
                numAccept ← numAccept + 1
        end if
Part 2.5 Update the maximum displacement periodically:
        if (mod(cycle, updateCycle) = 0)
            acceptRatio ← numAccept/cycle × atomMax)
            if (acceptRatio > 0.5)
                dMax ← 1.05 × dMax
            else
                dMax ← 0.95 × dMax
            end if
        end if
    end cycle loop
```

6.4 Path integral Monte Carlo

The above MC methods are appropriate for classical molecular interactions. In contrast, path integral MC (PIMC) methods allow the investigation of systems involving quantum interactions (Feynman and Hibbs, 2010). Instead of discrete particle trajectories, a particle is treated as a ring polymer consisting of P beads. Assuming a sufficiently large value of P (Chandler and Wolynes, 1981), the canonical (NVT) partition function (Chapter 2) of a three-dimensional system of N rings of P beads is given by,

$$Z_P = \left(\frac{mP}{2\pi\beta\hbar^2}\right)^{3NP/2} \int \prod_{i=1}^{N} \mathrm{d}\mathbf{r}_i^1 \ldots \mathrm{d}\mathbf{r}_i^P \, e^{-\beta\,\Phi(\mathbf{r}_i^1 \ldots \mathbf{r}_N^P)} \Big|_{r_i^1 = r_i^{P+1}} \tag{6.20}$$

where m is the mass of a single particle, h is Planck's constant divided by 2π, and $\mathbf{r}_i^k$ is the radius vector of bead k of ring i. The propagator potential is defined as,

$$\Phi(\mathbf{r}_1^1,\ldots,\mathbf{r}_N^P) = \frac{mP}{2\beta^2\hbar^2}\sum_{k=1}^{P}\sum_{i=1}^{N}\left(\mathbf{r}_i^k-\mathbf{r}_i^{k-1}\right)^2 + \frac{1}{P}\sum_{k=1}^{P}\sum_{j>i}^{N}u\left(\left|\mathbf{r}_i^k-\mathbf{r}_j^k\right|\right) \quad (6.21)$$

and $\mathbf{r}_i^1 = \mathbf{r}_i^{P+1}$ to satisfy the ring condition. In Eq. (6.21), the first term is the quantum kinetic contribution that describes the harmonic bead−bead interaction within the ring. The second term of Eq. (6.21) accounts for the potential energy contribution from the interaction of beads with the same index.

A property of interest can be obtained (Barker, 1979) by taking the derivative of the partition function with respect to a corresponding variable. For example, the isochoric heat capacity (C_V) can be obtained from,

$$C_V = \beta^2\left[\left\langle {U_{TD}}^2\right\rangle - \langle U_{TD}\rangle^2 + \frac{3NP}{2\beta^2} - \frac{mP}{\beta^3\hbar^2}\sum_{\tau=1}^{P}\sum_{j=1}^{N}\left(\mathbf{r}_j^{(\tau)}-\mathbf{r}_j^{(\tau+1)}\right)^2\right] \quad (6.22)$$

where U_{TD} is the energy calculated using the thermodynamic method.

$$U_{TD} = \frac{3NP}{2\beta} - \frac{mP}{2\beta^2\hbar^2}\sum_{\tau=1}^{P}\sum_{j=1}^{N}\left(\mathbf{r}_j^{(\tau)}-\mathbf{r}_j^{(\tau+1)}\right)^2 + \frac{1}{P}\sum_{\tau=1}^{P}\sum_{i>j}^{N}u\left(\left|\mathbf{r}_i^{(\tau)}-\mathbf{r}_j^{(\tau)}\right|\right) \quad (6.23)$$

The so-called thermodynamic or primitive estimators (Barker, 1979) are associated with increasing standard error with P because the kinetic energy exhibits large fluctuations. This has been partly addressed by Herman et al. (1982), who formulated a path integral version of the virial theorem. Both virial and virial centroid estimator expressions are available for energy and pressure. Glaesemann and Fried (2002a,b) have also introduced the double virial and double centroid virial estimators for specific heat capacity at constant volume, which is more efficient than primitive or virial estimators. These estimators, which require either the calculation of the second-order derivatives of the potential function or the evaluation of the Hessian matrix at each accepted step, are very computationally expensive. As such, they are only computationally feasible using powerful parallel computing platforms.

A PIMC simulation can be simply implemented by conducting a conventional MC Markov chain process with Metropolis importance sampling (Algorithm 6.1). There are alternative ways of generating the trial displacements. Thirumalai et al. (1984) proposed first moving a ring as a whole, and then displacing all beads of the same chain by a small amount. Singer and Smith (1988) implemented a trial displacement of one bead at a time. Both these methods are relatively slow and result in low acceptance rates. A more computationally efficient approach is to use a staging algorithm (Ceperley and Pollock, 1984), where a ring is divided into segments and an attempt is made to

displace a moderate number of beads in one move. This approach increases the sampling efficiency. The staging variables are sampled randomly from their Gaussian distribution. A ring is moved as a whole after internal displacement is completed. The Li-Broughton (1987) correction is commonly applied to reduce error caused by the discreet values of P.

Thirumalai et al. (1984) applied PIMC to the properties of neon, showing the importance of quantum efforts at both 35 K and 40 K. Singer and Smith (1988) have reported a PIMC study of LJ systems in liquid phase, including several state points of neon and helium. Vlasiuk et al. (2016) have determined the heat capacity of neon using PIMC and Eq. (6.22). Although considerable uncertainty was observed, the errors were not as large as reported by Sesé (1995). A hybrid PIMC method has been proposed (Miura, 2007) to more easily handle rotation.

6.5 Summary

The MC strategy provides a relatively simple mechanism for determining the nondynamical properties of fluids without requiring specific details of the underlying physics. The basis of the procedure involves generating a Markov chain with an appropriate stochastic sampling scheme. The change in the energy of intermolecular interaction at different trail states is used to determine whether or not the trial configurations are accepted. MC algorithms can be applied relatively to a wide variety of situations. Atomic systems represent the simplest case with the acceptance criterion depending on the change in energy of a trial displacement. In contrast, for molecules, additional acceptance criteria and MC move a required account for changes in molecular configuration. Nonetheless, the MC approach can be used to determine efficiently the properties of large molecules such as polymers and proteins.

References

Abreu, C.R.A., Escobedo, F.A., 2005. A novel configurational-bias Monte Carlo method for lattice polymers: application to molecules with multicyclic architectures. Macromolecules 38, 8532–8545.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Barker, J.A., 1979. A quantum-statistical Monte Carlo method: path integrals with boundary conditions. J. Chem. Phys. 70, 2914–2918.

Barker, J.A., Watts, R.O., 1969. Structure of water: a Monte Carlo calculation. Chem. Phys. Lett. 3, 144–145.

Baumgärtner, A., Binder, K., 1979. Monte Carlo studies on the freely jointed polymer chain with excluded volume interaction. J. Chem. Phys. 71, 2541–2544.

Bentrem, F.W., Xie, J., Pandey, R.B., 2002a. Density and conformation with relaxed substrate, bulk, and interface in electrophoretic deposition of polymer chains. J. Mol. Struct. (Theochem) 592, 95–103.

Bentrem, F.W., Xie, J., Pandey, R.B., 2002b. Interface relaxation in electrophoretic deposition of polymer chains: effects of segmental dynamics, molecular weight, and field. Phys. Rev. E 65, 041606.

Berendsen, H.J., 2007. Simulating the Physical World: Hierarchical Modeling from Quantum Mechanics to Fluid Dynamics. Cambridge University Press, Cambridge.

Binder, K. (Ed.), 1995a. The Monte Carlo Method in Condensed Matter Physics. second ed. Springer-Verlag, Berlin.

Binder, K. (Ed.), 1995b. Monte Carlo and Molecular Dynamics Simulations in Polymer Science. Oxford University Press, New York.

Binder, K., 1997. Applications of Monte Carlo methods to statistical physics. Rep. Prog. Phys. 60, 487–599.

Binder, K., Heermann, D.W., 1997. Monte Carlo Simulation in Statistical Physics: An Introduction, third ed. Springer-Verlag, Berlin.

Bishop, M., Ceperley, D., Frisch, H.L., Kalos, M.H., 1980. Investigations of static properties of model bulk polymer fluids. J. Chem. Phys. 72, 3228–3235.

Brender, C., Lax, M., 1983. A Monte Carlo off-lattice method: the slithering snake in a continuum. J. Chem. Phys. 79, 2423–2425.

Ceperley, D.M., Pollock, E.L., 1984. Simulation of quantum many-body systems by path-integral methods. Phys. Rev. B 30, 2555–2568.

Cezar, H.M., Canuto, S., Coutinho, K., 2020. DICE: a Monte Carlo code for molecular simulation including the configurational bias Monte Carlo method. J. Chem. Info. Model. 60, 3472–3488.

Chandler, D., Wolynes, P.G., 1981. Exploiting the isomorphism between quantum theory and classical statistical mechanics of polyatomic fluids. J. Chem. Phys. 74, 4078–4095.

Clisby, N., 2018. Monte Carlo study of four-dimensional self-avoiding walks of up to one billion steps. J. Stat. Phys. 172, 477–492.

Cootes, A.P., Gurmi, P.M.G., Torda, A.E., 2000. Biased Monte Carlo optimization of protein sequences. J. Chem. Phys. 113, 2489–2496.

Cui, S.T., Cummings, P.T., Cochran, H.D., 1997. Configurational bias Gibbs ensemble Monte Carlo simulation of vapor-liquid equilibria of linear and short-branched alkanes. Fluid Phase Equilib. 141, 45–61.

de Gennes, P., 1971. Reptation of a polymer chain in the presence of fixed obstacles. J. Chem. Phys. 55, 572–579.

de Pablo, J.J., 1995. Simulation of phase equilibria for chain molecules. Fluid Phase Equilib. 104, 195–206.

de Pablo, J.J., Laso, M., Suter, U.W., 1992a. Estimation of the chemical potential of chain molecules by simulation. J. Chem. Phys. 96, 6157–6162.

de Pablo, J.J., Laso, M., Suter, U.W., 1992b. Simulation of polyethylene above and below the melting point. J. Chem. Phys. 96, 2395–2403.

de Pablo, J.J., Laso, M., Siepmann, J.I., Suter, U.W., 1993. Continuum-configurational-bias Monte Carlo simulations of long-chain alkanes. Mol. Phys. 80, 55–63.

de Pablo, J.J., Yan, Q., Escobedo, F.A., 1999. Simulation of phase transitions in fluids. Annu. Rev. Phys. Chem. 50, 377–411.

Deem, M.W., Bader, J.S., 1996. A configurational bias Monte Carlo method for linear and cyclic peptides. Mol. Phys. 87, 1245–1260.

Doi, M., Edwards, S.E., 1995. The Theory of Polymer Dynamics. Clarendon Press, Oxford.

Escobedo, F.A., de Pablo, J.J., 1995a. Extended continuum configurational bias Monte Carlo methods for simulation of flexible molecules. J. Chem. Phys. 102, 2636–2652.

Escobedo, F.A., de Pablo, J.J., 1995b. Monte Carlo simulation of the chemical potential of polymers in an expanded ensemble. J. Chem. Phys. 103, 2703–2710.
Escobedo, F.A., de Pablo, J.J., 1996. Simulation and prediction of vapour-liquid equilibria for chain molecules. Mol. Phys. 87, 347–366.
Escobedo, F.A., de Pablo, J.J., 1997. Monte Carlo simulation of athermal mesogenic chains: pure systems, mixtures, and constrained environments. J. Chem. Phys. 106, 9858–9868.
Ferguson, D.M., Siepmann, J.I., Truhlar, D.G. (Eds.), 1999. Monte Carlo Methods in Chemical Physics, Advances in Chemical Physics, Vol. 105. John Wiley & Sons, New York.
Ferrenberg, A.M., Swendsen, R.H., 1989. New Monte Carlo technique for studying phase transitions. Phys. Rev. Lett. 61, 2635–2638.
Feynman, R.P., Hibbs, A.R., 2010. Quantum Mechanics and Path Integrals (emended by Styer, D. F.). Dover Publications, Mineola.
Foo, G.M., Pandey, R.B., 2000. Characteristics of driven polymer surfaces: growth and roughness. Phys. Rev. E 61, 1793–1799.
Freire, J.J., Horta, A., 1976. Mean reciprocal distances of short polymethylene chains. Calculation of the translational diffusion coefficient of n-alkanes. J. Chem. Phys. 65, 4049–4054.
Frenkel, D., 1995. In: Baus, M., Rull, L.F., Ryckaert, J.-P. (Eds.), Observation, Prediction and Simulation of Phase Transitions in Complex Fluids. Kluwer, Dordecht.
Frenkel, D., Smit, B., 2023. Understanding Molecular Simulation. From Algorithms to Applications, third ed. Academic Press, San Diego.
Frenkel, D., Mooij, G.C.A.M., Smit, B., 1991. Novel scheme to study structural and thermal properties of continuously deformable molecules. J. Phys.: Condens. Matter 3, 3053–3076.
Gergidis, L.N., Moultos, O., Georgiadis, C., Kosmas, M., Vlahos, C., 2009. Off lattice Monte Carlo simulations of AB hybrid dendritic star copolymers. Polymers 50, 328–355.
Geyler, S., Pakula, T., Reiter, J., 1990. Monte Carlo simulation of dense polymer systems on a lattice. J. Chem. Phys. 92, 2676–2680.
Gharibi, H., Behjatmanesh-Ardakani, R., Hashemianzadeh, M., Mousavi-Khoshdel, M., 2006. Complexation between a macromolecule and an amphiphile by Monte Carlo technique. J. Phys. Chem. B 110, 13547–13553.
Gilks, W.R., Richardson, S., Spiegelhalter, D.J. (Eds.), 1996. Markov Chain Monte Carlo in Practice. Chapman & Hall, Boca Raton.
Glaesemann, K.R., Fried, L.E., 2002a. An improved thermodynamic energy estimator for path integral simulations. J. Chem. Phys. 116, 5951–5955.
Glaesemann, K.R., Fried, L.E., 2002b. Improved heat capacity estimator for path integral simulations. J. Chem. Phys. 117, 3020–3026.
Glaser, J., Karas, A.S., Glotzer, S.C., 2015. A parallel algorithm for implicit depletant simulations. J. Chem. Phys. 143, 184110.
Haas, F.M., Hilfer, R., 1996. Continuum Monte Carlo simulation at constant pressure of stiff chain molecules at surfaces, J. Chem. Phys., 105. pp. 3859–3867.
Hammersley, J.M., Handscomb, D.C., 1964. Monte Carlo Methods. Methuen & Co., London.
Hansen, J.P., McDonald, I.R., 2013. Theory of Simple Liquids, fourth ed. Academic Press, Amsterdam.
Heermann, D.W., 1990. Computer Simulation Methods in Theoretical Physics, second ed. Springer-Verlag, Berlin.
Herman, M.F., Bruskin, E.J., Berne, B.J., 1982. On path integral Monte Carlo simulations. J. Chem. Phys. 76, 5150–5155.
Hua, C.C., Ke, C.C., 2005. A reptation-based lattice model with tube-chain coupling for the linear dynamics of bidisperse entangled linear polymer. Int. Polym. Res. 12, 181–197.

Kampmann, T.A., Boltz, H.-H., Kierfeld, J., 2015. Monte Carlo simulation of dense polymer melts using event chain algorithms. J. Chem. Phys. 143, 0441095.
Karantrantos, A., Composto, R.J., Winey, J.I., Kröger, M., Clarke, N., 2019. Modeling of entangled polymer diffusion in melts and nanocomposites: a review. Polymers 11, 876.
Kindt, J.T., 2002. Pivot-coupled grand canonical Monte Carlo method for ring simulations. J. Chem. Phys. 116, 6817–6825.
Kreer, T., Baschnagel, J., Müller, Binder, K., 2001. Monte Carlo simulation of long chain polymers melts: crossover from Rouse to reputation dynamics. Macromolecules 34, 1105–1117.
Kroese, D.P., Rubinstein, R.Y., 2011. Monte Carlo methods. WIREs Comp. Stat. 4, 48–58.
Kroese, D.P., Taimre, T., Botev, Z.I., 2011. Handbook of Monte Carlo Methods. John Wiley & Sons, Hoboken.
Lal, M., 1969. 'Monte Carlo' computer simulation of chain molecules. I. Mol. Phys. 17, 57–64.
Landau, D.P., Binder, K., 2000. A Guide to Monte Carlo Simulations in Statistical Physics. Cambridge University Press, Cambridge.
Laso, M., de Pablo, J.J., Suter, U.W., 1992. Simulation of phase equilibria for chain molecules. J. Chem. Phys. 97, 2817–2819.
Lax, M., Brender, C., 1977. Monte Carlo study of lattice polymer dynamics. J. Chem. Phys. 67, 1785–1787.
Leach, A.R., 2001. Molecular Modelling: Principles and Applications, second ed. Addison Wesley Longman, Harlow.
Li, X.-P., Broughton, J.Q., 1987. High-order correction to the Trotter expansion for use in computer simulation. J. Chem. Phys. 86, 5094–5100.
Lin, H., Mattice, W.L., von Meerwall, E.D., 2007. Chain dynamics of bidisperse polyethylene melts: a Monte Carlo study on a high-coordination lattice. Macromolecules 40, 959–966.
Madras, N., Sokal, A.D., 1988. The pivot algorithm: a highly efficient Monte Carlo method for the self-avoiding walk. J. Stat. Phys. 50, 109–186.
Mańka, A., Nowicki, W., Nowicka, G., 2013. Monte Carlo simulations of a polymer chain conformation. The effectiveness of local moves algorithms and estimation of entropy. J. Mol. Model. 19, 3659–3670.
Maycock, V., Bhattacharya, A., 2006. Effect of head-tail ratio and the range of the head-head interaction in amphiphlic self-assembly. Eur. Phys. J. E 20, 201–207.
Mayr, N., Haring, M., Wallek, T., 2023. Continuous fraction component Gibbs ensemble Monte Carlo. Am. J. Phys. 91, 235–246.
Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., 1953. Equation of state calculations by fast computing machines. J. Chem. Phys. 21, 1087–1092.
Miura, S., 2007. Rotational fluctuation of molecules in quantum clusters. I. Path integral hybrid Monte Carlo algorithm. J. Chem. Phys. 126, 114308.
Mooij, G.C.A.M., Frenkel, D., 1996. A systematic optimization scheme for configurational bias Monte Carlo. Mol. Sim. 17, 41–55.
Mooij, G.C.A.M., Frenkel, D., Smit, B., 1992. Direct simulation of phase equilibria of chain molecules. J. Phys.: Condens. Matter 4, L255–L259.
Moony, C.Z., 1997. Monte Carlo Simulation. Sage Publications, Thousand Oaks.
Mountain, R.D., Thirumalai, D., 1994. Quantative measure of efficiency of Monte Carlo simulations. Phys. A 210, 453–460.
Murat, M., Witten, T.A., 1990. Relaxation in bead-jump polymer simulations. Macromolecules 23, 520–527.
Newman, M.E.J., Barkema, G.T., 1999. Monte Carlo Methods in Statistical Physics. Clarendon Press, Oxford.

Orkoulas, G., Panagiotopoulos, A.Z., 1993. Chemical potentials in ionic systems from Monte Carlo simulations with distance-biased test particle insertion. Fluid Phase Equilib. 93, 223–231.

Orkoulas, G., Panagiotopoulos, A.Z., 1994. Free energy and phase equilibria for the restricted primitive model of ionic fluids from Monte Carlo simulations. J. Chem. Phys. 101, 1452–1459.

Pakula, T., 1987. Cooperative relaxations in condensed macromolecular systems. 1. A model for computer simulation. Macromolecules 20, 679–682.

Pakula, T., Geyler, S., 1987. Cooperative relaxations in condensed macromolecular systems. 2. Computer simulation of self-diffusion of linear chains. Macromolecules 20, 2909–2914.

Panagiotopoulos, A.Z., 1987. Direct determination of phase coexistence properties of fluids by Monte Carlo simulation in a new ensemble. Mol. Phys. 61, 813–826.

Panagiotopoulos, A.Z., 2000. Monte Carlo methods for phase equilibria of fluids. J. Phys.: Condens. Matter 12, R25–R52.

Pandey, R.B., Seyfarth, R., 2003. Driven chain macromolecule in a heterogeneous membrane-like medium. Struct. Chem. 14, 445–449.

Rahbari, A., Hens, R., Ramdin, M., Moultos, O.A., Dubbeldam, D., Vlugt, T.J.H., 2021. Recent advances in the continuous fractional component Monte Carlo methodology. Mol. Sim. 47, 804–823.

Reiter, J., Eding, T., Pakula, T., 1990. Monte Carlo simulation of lattice models for macromolecules at high densities. J. Chem. Phys. 93, 837–844.

Rorrer, N.A., Dorgan, J.R., 2014. Effects of polydispersity on confined homopolymer melts: a Monte Carlo study. J. Chem. Phys. 141, 214905.

Rosenbluth, M.N., Rosenbluth, A.W., 1955. Monte Carlo calculation of the average extension of molecular chains. J. Chem. Phys. 23, 356–359.

Sabelfeld, K.K., 1991. Monte Carlo Methods in Boundary Value Problems. Springer-Verlag, Berlin.

Sadanobu, J., Goddard III, W.A., 1997. The continuous configurational Boltzmann biased direct Monte Carlo method for free energy properties of polymer chains. J. Chem. Phys. 106, 6722–6729.

Santos, S., Suter, U.W., Müller, M., Nievergelt, J., 2001. A novel parallel-rotation algorithm for atomistic Monte Carlo simulation of dense polymer systems. J. Chem. Phys. 114, 9772–9779.

Sesé, L.M., 1995. Feynman-Hibbs potentials and path integrals for quantum Lennard-Jones systems: theory and Monte Carlo simulations. Mol. Phys. 85, 931–947.

Shelley, J.C., Patey, G.N., 1994. A configuration bias Monte Carlo method for ionic solutions. J. Chem. Phys. 100, 8265–8270.

Shelley, J.C., Patey, G.N., 1995. A configuration bias Monte Carlo method for water. J. Chem. Phys. 102, 7656–7663.

Shi, W., Maginn, E.J., 2008. Improvement in molecular exchange efficiency in Gibbs ensemble Monte Carlo: development and implementation of continuous fractional component move. J. Comput. Chem. 29, 2520–2530.

Siepmann, J.I., 1990. A method for the direct calculation of chemical potentials for dense chain systems. Mol. Phys. 70, 1145–1158.

Siepmann, J.I., Frenkel, D., 1992. Configurational bias Monte Carlo: a new sampling scheme for flexible chains. Mol. Phys. 75, 59–70.

Siepmann, J.I., Karaborni, S., Smit, B., 1993a. Vapor-liquid equilibria of model alkanes. J. Am. Chem. Soc. 115, 6454–6455.

Siepmann, J.I., Karaborni, S., Smit, B., 1993b. Simulating the critical behaviour of complex fluids. Nature 365, 330–332.
Singer, K., Smith, W., 1988. Path integral simulations of condensed phase Lennard-Jones systems. Mol. Phys. 64, 1215–1231.
Singh, S.K., Sinha, A., Deo, G., Singh, J.K., 2009. Vapor-liquid phase coexistence, critical properties, and surface tension of confined alkanes. J. Phys. Chem. C. 113, 7170–7180.
Smit, B., Karaborni, S., Siepmann, J.I., 1995. Computer simulation of vapor-liquid phase equilibria of n-alkanes. J. Chem. Phys. 102, 2126–2140.
Smit, B., Karaborni, S., Siepmann, J.I., 1998. Erratum: "Computer simulation of vapor-liquid phase equilibria of n-alkanes." [J. Chem. Phys. 102, 2126 (1995)]. J. Chem. Phys. 109, 352.
Soboľ, I.M., 1994. A Primer for the Monte Carlo Method. CRC Press, Boca Raton.
Sokal, A.D., 1995. In: Binder, K. (Ed.), Monte Carlo and Molecular Dynamics Simulations in Polymer Science. Oxford University Press, New York.
Stampe, J., Sokolov, I.N., 2001. Cyclization of a polymer with charged reactive end groups. J. Chem. Phys. 114, 5043–5048.
Stedman, M.L., Foulkes, W.M.C., Nekovee, M., 1998. An accelerated Metropolis method. J. Chem. Phys. 109, 2630–3634.
Suen, J.K.C., Escobedo, F.A., de Pablo, J.J., 1997. Monte Carlo simulation of polymer chain collapse in an athermal solvent. J. Chem. Phys. 106, 1288–1290.
Swendsen, R.H., Wang, J.-S., 1987. Nonuniversal critical dynamics in Monte Carlo simulations. Phys. Rev. Lett. 58, 86–88.
Terranova, G., Aldao, C.M., Mártin, H.O., 2007. Diffusion in the two-dimensional necklace model for reputation. Phys. Rev. E 76, 031111.
Thirumalai, D., Hall, R.W., Berne, B.J., 1984. A path integral Monte Carlo study of liquid neon and the quantum effective pair potential. J. Chem. Phys. 81, 2523–2527.
Tildesley, D.J., 1993. In: Allen, M.P., Tildesley, D.J. (Eds.), Computer Simulation in Chemical Physics. Kluwer, Dordrecht.
Tsonchev, S., Coalson, R.D., Liu, A., Beck, T.L., 2004. Flexible polyelectrolyte simulations at the Poisson-Boltzmann level: a comparison of the kink-jump and multigrid configurational-bias Monte Carlo methods. J. Chem. Phys. 120, 9817–9821.
Tuckerman, M.E., 2010. Statistical Mechanics: Theory and Molecular Simulation. Oxford University Press, Oxford.
Uhlherr, A., Leak, S.J., Adam, N.E., Nyber, P.E., Doxastakis, M., Mavrantzas, V.G., Theodorou, D.N., 2002. Large scale atomistic polymer simulations using Monte Carlo methods for parallel vector processors. Comp. Phys. Commun. 144, 1–22.
Vendruscolo, M., 1997. Modified configurational bias Monte Carlo method for simulation of polymer systems. J. Chem. Phys. 106, 2970–2976.
Verdier, P.H., Stockmayer, W.H., 1962. Monte Carlo calculations on the dynamics of polymers in dilute solution. J. Chem. Phys. 36, 227–235.
Veseley, F.J., 1982. Angular Monte Carlo integration using quaternion parameters: a spherical reference potential for CCl_4. J. Comput. Phys. 47, 291–296.
Vlasiuk, M., Frascoli, F., Sadus, R.J., 2016. Molecular simulation of the thermodynamic, structural, and vapor-liquid equilibrium properties of neon. J. Chem. Phys. 145, 104501.
Vlugt, T.J.H., 1999. Efficiency of parallel CBMC simulations. Mol. Sim. 23, 63–78.
von Ferber, C., Monteith, J.T., Bishop, M., 2009. Shapes of two-dimensional excluded volume continuum star polymers. Macromolecules 42, 3627–3631.
Wall, F.T., Mandel, F., 1975. Macromolecular dimensions obtained by an efficient Monte Carlo method without sample attrition. J. Chem. Phys. 63, 4592–4596.

Wall, F.T., Chin, J.C., Mandel, F., 1977. Configurations of macromolecular chains confined to strips or tubes. J. Chem. Phys. 66, 3066–3069.

Widom, B., 1963. Some topics in the theory of fluids. J. Chem. Phys. 39, 2808–2812.

Wu, D., Chandler, D., Smit, B., 1992. Electrostatic analogy for surfactant assemblies. J. Phys. Chem. 96, 4077–4083.

Zifferer, G., Preusser, W., 2001. Monte Carlo simulation studies of the size and shape of ring polymers. Macromol. Theory Sim. 10, 397–407.

Zou, J., Saven, J.G., 2003. Using self-consistent fields to bias Monte Carlo methods with applications to designing and sampling protein sequences. J. Chem. Phys. 118, 3843–3854.

This page intentionally left blank

Chapter 7

Integrators for molecular dynamics

A common feature of all molecular dynamics (MD) algorithms is that the positions of molecules are evolved with time by integrating the equations of motion. Mathematically, the equations of motion represent a set of initial value differential equations, which involve the evaluation of forces. In principle, using any standard finite-difference algorithm can solve these equations. However, calculating the forces is the most time-consuming step in MD and any algorithm that requires more than one force evaluation per time step is unlikely to be viable computationally. This often excludes many common finite-difference algorithms such as the conventional Runge−Kutta (RK) method, which would require multiple force evaluations. The other important requirement is that the integration algorithm must be well behaved for the force encountered in MD. As a rule of thumb, this latter requirement means that the order of the algorithm defined as the highest order of the time step for the equation for the solution of the coordinates should be at least two. A consequence of these requirements is that the algorithms available for MD form a small subset of the finite-difference algorithms in the literature (Dahlquist and Björk, 1974). Some finite-difference algorithms have been both developed and specifically tuned for applications to MD.

Finite-difference algorithms used in molecular simulation can be classified as either predictor or predictor−corrector methods. In the predictor methods, the molecular coordinates are updated from quantities that are either calculated in the current step or that are known from previous steps. The Verlet (1967) algorithm and its subsequent modifications are widely used examples of predictor algorithms. In contrast, predictor−corrector algorithms involve predicting new molecular coordinates; using the predicted coordinates to calculate the value of some function; and subsequently using this value to correct the initial prediction. The Gear (1966, 1971) algorithm is a widely used predictor−corrector algorithm for MD. It should be noted that these algorithms are most easily applied to atoms, and special modifications are required for molecular systems.

In this chapter, the motivation for integrator algorithms is discussed briefly and some of the most commonly used integration schemes are

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00015-0

considered in detail. The problem imposed by molecular systems is also addressed. The merits of various integration schemes have been discussed elsewhere (Berendsen and van Gunsteren, 1986; Hockney and Eastwood, 1988; Heermann, 1990; Gould and Tobochnik, 1996; Gubbins and Quirke, 1996; Frenkel and Smit, 2023; Rapaport, 2004; Allen and Tildesley, 2017).

7.1 Integrating the equations of motion

At the outset, it is instructive to formulate the problem posed specifically by MD. The force experienced by a particle during displacement is given by Newton's second law of motion,

$$\mathbf{F}_i = m_i \mathbf{a}_i \tag{7.1}$$

where $\mathbf{F}$ is the force acting on the particle, and m and $\mathbf{a}$ are the particle's mass and acceleration, respectively. It is more convenient to express acceleration as the second derivative of displacement $\mathbf{r}$ with respect to time t because we are interested in the evolution of the particle with time.

$$\frac{d^2\mathbf{r}_i}{dt^2} = \frac{\mathbf{F}_i}{m_i} \tag{7.2}$$

The dynamic behavior of the whole system can be obtained by solving Eq. (7.2) for each and every particle. Integrating Eq. (7.2) with respect to time for a small time interval during which the force is constant yields:

$$\frac{d\mathbf{r}_i}{dt} = \left(\frac{\mathbf{F}_i}{m_i}\right) t + c_1 \tag{7.3}$$

At time $t = 0$, the first term vanishes and the velocity is given by the constant c_1, which represents the initial velocity $\mathbf{v}_i$. In general, at time t, we obtain:

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{a}_i t + \mathbf{v}_i \tag{7.4}$$

If we integrate Eq. (7.4) with respect to t, we obtain,

$$\mathbf{r}_i = \mathbf{r}_i t + \frac{\mathbf{a}_i t^2}{2} + c_2 \tag{7.5}$$

where the constant c_2 is the particle's current position. Eq. (7.5) enables us to calculate the particle's displacement simply from the initial velocity and acceleration.

7.2 Gear predictor–corrector methods

In this section, the concept of a finite-difference approach to integrating the equations of motion is introduced and the Gear algorithm is derived (Gear, 1971). There are many possible predictor–corrector algorithms

(Dahlquist and Björk, 1974) but only the Gear algorithm appears routinely in MD. We note that the Gear algorithm is used much less commonly than alternatives, such as the Verlet algorithm (discussed below) and it has been almost completely omitted from some revised texts (Allen and Tildesley, 2017). Nonetheless, it has proved to be a consistently reliable many contemporary applications including thermodynamic properties (Losey and Sadus, 2019), solid−liquid equilibria (Ahmed and Sadus, 2010), and nonequilibrium behavior (Ahmed et al., 2010). The Gear approach has proved to be particularly stable for the MD of polarizable molecules (Kolafa, 2005) and improvements have been reported (Janek and Kolafa, 2020). When compared (Haile, 1992) with the Verlet method, the Gear algorithm shows superior conservation of energy for small time steps. In view of such considerations, it has useful ongoing role in MD.

7.2.1 The basic Gear algorithm

Equation (7.5) shows how the particle coordinates change with time. However, any changes in particle displacement will also affect the strength of interparticle forces, velocities, and accelerations. Consequently, a general method is required to evolve all of the dynamic properties of the system. In general, the evolution of particle coordinates or any time-dependent property can be estimated from a Taylor series expansion. The Taylor series expansion for the coordinate vector is:

$$\mathbf{r}(t+\Delta t)=\mathbf{r}(t)+\frac{d\mathbf{r}}{dt}\Delta t+\frac{1}{2!}\frac{d^2\mathbf{r}}{dt^2}\Delta t^2+\frac{1}{3!}\frac{d^3\mathbf{r}}{dt^3}\Delta t^3+\frac{1}{4!}\frac{d^4\mathbf{r}}{dt^4}\Delta t^4+\cdots \quad (7.6)$$

Equation (7.6) represents the "finite-difference" approach that is commonly used to solve ordinary differential equations such as Eq. (7.2). Naively, the particle velocities and accelerations could be updated by simply using other appropriate expansions similar to Eq. (7.6).

$$\left.\begin{aligned}\mathbf{v}(t+\Delta t)&=\mathbf{v}(t)+\Delta t\frac{d^2\mathbf{r}}{dt^2}+\frac{1}{2!}\Delta t^2\frac{d^3\mathbf{r}}{dt^3}+\cdots\\ \mathbf{a}(t+\Delta t)&=\mathbf{a}(t)+\Delta t\frac{d^3\mathbf{r}}{dt^3}+\cdots\end{aligned}\right\} \quad (7.7)$$

However, this simple approach does not yield the correct trajectories because of truncation errors. Instead, the values "predicted" by Eqs. (7.6) and (7.7) must be "corrected" to yield the actual trajectory. The correction step involves actually calculating the accelerations $\mathbf{a}_c$ from the new coordinates and determining the difference $\Delta\mathbf{a}$ between these values and the predicted $\mathbf{a}_p$ values.

$$\Delta\mathbf{a}(t+\Delta t)=\mathbf{a}_c(t+\Delta t)-\mathbf{a}_p(t+\Delta t) \quad (7.8)$$

The predicted value of **r** can be corrected by,

$$\mathbf{r}_c(t+\Delta t) = \mathbf{r}_p(t+\Delta t) + k_d \Delta \mathbf{a}(t+\Delta t) \tag{7.9}$$

where $\mathbf{r}_p$ is the value of **r** originally predicted by Eq. (7.6). In Eq. (7.9), k_d is a constant that depends on the nature of the time derivative denoted by d. For example, k_0, k_1, and k_2 are the constants associated with position ($d = 0$), velocity ($d = 1$), and acceleration ($d = 2$), respectively. Suitable values of k_d have been determined by Gear (1966, 1971) and they are summarized in Table 7.1.

It should be noted that the accuracy of the values obtained from the Taylor series will depend partly on the extent of truncation. When Eq. (7.6) is truncated after the second term, it is identical to Eq. (7.5). Typically, the position vectors, velocities, and accelerations are truncated after the fourth, third, and second time derivative, respectively. The number of equations of type (7.7) depends on where Eq. (7.6) is truncated. For the Gear predictor–corrector algorithm, Eq. (7.6) cannot be truncated below the third term because the algorithm depends on the evaluation of accelerations in the corrector step.

The values of k_d (Table 7.1) are dependent on the number of properties that are being evolved using the predictor–corrector method. The minimum is three corresponding to the set of position vectors **r**, velocities **v**, and accelerations **a**. However, in the estimation of these properties, we can make use of the higher derivatives of the Taylor series (Eq. 7.6) to improve the accuracy of the predictions. Values for these higher derivatives can also be evolved using a predictor–corrector equation. The improved accuracy of these derivatives in turn impacts positively on the accuracy of the other quantities. Consequently, we can identify three-, four-, five- and six-value Gear algorithms depending on the number of predictor–corrector equations. An expanded table of coefficients for various improved Gear schemes is available (Janek and Kolafa, 2020).

The general scheme of either the Gear predictor–corrector or other predictor–corrector methods is given by Algorithm 7.1. Basically, it is a simple

TABLE 7.1 Gear corrector coefficients k_d for a second-order equation using time-scaled variables.

No. Eqs.	d					
	r	$v = dr/dt$	$a = d^2r/dt^2$	d^3r/dt^3	d^4r/dt^4	d^5r/dt^5
3	0	1	1			
4	1/6	5/6	1	1/3		
5	19/120	3/4	1	1/2	1/12	
6	3/20	251/360	1	11/18	1/6	1/60

three-stage process. Initially, the coordinates and other time derivatives are predicted from a Taylor series expansion. The new coordinates are used to determine the true accelerations. This is the most computationally expensive part of the simulation cycle because it involves force re-evaluation. Subsequently, the new accelerations are used to correct the predicted positions and other time derivatives of position.

ALGORITHM 7.1 General predictor–corrector procedure for solving equations of motion.

```
Predictor-Corrector:
loop
   Use a Taylor series expansion to predict the positions, velocities,
   accelerations, etc.
   Use the new coordinates to evaluate the true accelerations.
   Use the new accelerations to correct the initial predictions.
   t ← t + Δt
while(t < tmax)
```

It is evident from Eq. (7.6) that the various derivatives of $\mathbf{r}$ are multiplied by various constants involving time. It is common to take advantage of this by defining successive scaled time derivatives $\mathbf{r}_0$, $\mathbf{r}_1 = \Delta t\,(d\mathbf{r}_0/dt)$, $\mathbf{r}_2 = \frac{1}{2}(\Delta t)^2(d^2\mathbf{r}_0/dt^2)$, $\mathbf{r}_3 = 1/6(\Delta t)^3(d^3\mathbf{r}_0/dt^3)$. The advantage of using time-scaled coordinates is that the predictor step of the Gear algorithm can be expressed in terms of a Pascal triangle matrix, which is easy to apply. For example, the predictor step for a four-value Gear algorithm is:

$$\begin{pmatrix} \mathbf{r}_0^p(t+\Delta t) \\ \mathbf{r}_1^p(t+\Delta t) \\ \mathbf{r}_2^p(t+\Delta t) \\ \mathbf{r}_3^p(t+\Delta t) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{r}_0(t) \\ \mathbf{r}_1(t) \\ \mathbf{r}_2(t) \\ \mathbf{r}_3(t) \end{pmatrix} \tag{7.10}$$

The corresponding corrector step is simply:

$$\begin{pmatrix} \mathbf{r}_0^c(t+\Delta t) \\ \mathbf{r}_1^c(t+\Delta t) \\ \mathbf{r}_2^c(t+\Delta t) \\ \mathbf{r}_3^c(t+\Delta t) \end{pmatrix} = \begin{pmatrix} \mathbf{r}_0^p(t+\Delta t) \\ \mathbf{r}_1^p(t+\Delta t) \\ \mathbf{r}_2^p(t+\Delta t) \\ \mathbf{r}_3^p(t+\Delta t) \end{pmatrix} + \begin{pmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{pmatrix} \Delta\mathbf{a}(t+\Delta t) \tag{7.11}$$

The use of time-scaled coordinates is very common and the corrector coefficients are almost invariably quoted assuming that time-scaled variables are used.

ALGORITHM 7.1 Example of the implementation of Gear's four-value (third-order) predictor–corrector algorithm using scaled time derivatives.

Overall Algorithm

Part 1 Predictor step.
Part 2 Determine forces (fx_i, fy_i, fz_i).
Part 3 Corrector step.

Sub Algorithms

Part 1 Predictor step:
loop $i \leftarrow 1 \ldots nAtom$
Part 1.1 Predict new position vectors:
$rx_i \leftarrow rx_i + vx_i + ax_i + (d^3rx_i/dt^3) + (d^4rx_i/dt^4)$
$ry_i \leftarrow ry_i + vy_i + ay_i + (d^3ry_i/dt^3) + (d^4ry_i/dt^4)$
$rz_i \leftarrow rz_i + vz_i + az_i + (d^3rz_i/dt^3) + (d^4rz_i/dt^4)$
Part 2.2 Predict new (scaled time) velocities:
$vx_i \leftarrow vx_i + ax_i + 2(d^3rx_i/dt^3) + 3(d^4rx_i/dt^4)$
$vy_i \leftarrow vy_i + ay_i + 2(d^3ry_i/dt^3) + 3(d^4ry_i/dt^4)$
$vz_i \leftarrow vz_i + az_i + 2(d^3rz_i/dt^3) + 3(d^4rz_i/dt^4)$
Part 2.3 Predict new (scaled time) accelerations:
$ax_i \leftarrow ax_i + (d^3rx_i/dt^3) + 3(d^4rx_i/dt^4)$
$ay_i \leftarrow ay_i + (d^3ry_i/dt^3) + 3(d^4ry_i/dt^4)$
$az_i \leftarrow az_i + (d^3rz_i/dt^3) + 3(d^4rz_i/dt^4)$
Part 2.4 Predict new third (scaled time) derivatives:
$(d^3rx_i/dt^3) \leftarrow (d^3rx_i/dt^3) + (d^4rx_i/dt^4)$
$(d^3ry_i/dt^3) \leftarrow (d^3ry_i/dt^3) + (d^4ry_i/dt^4)$
$(d^3rz_i/dt^3) \leftarrow (d^3rz_i/dt^3) + (d^4rz_i/dt^4)$
end *i* **loop**

Part 3 Corrector step:
loop $i \leftarrow 1 \ldots nAtom$
Part 3.1 Determine new accelerations:
$axNew \leftarrow fx_i/m_i$
$ayNew \leftarrow fy_i/m_i$
$azNew \leftarrow fz_i/m_i$
Part 3.2 Determine corrections:
$deltax \leftarrow axNew - ax_i$
$deltay \leftarrow ayNew - ay_i$
$deltaz \leftarrow azNew - az_i$
Part 3.3 Correct position vectors:
$rx_i \leftarrow rx_i + k0 \times deltax$
$ry_i \leftarrow ry_i + k0 \times deltay$
$rz_i \leftarrow rz_i + k0 \times deltaz$

Part 3.4 Correct velocities:
$vx_i \leftarrow vx_i + k1 \times deltax$
$vy_i \leftarrow vy_i + k1 \times deltay$
$vz_i \leftarrow vz_i + k1 \times deltaz$
Part 3.5 Correct accelerations:
$ax_i \leftarrow axNew$
$ay_i \leftarrow ayNew$
$az_i \leftarrow azNew$
Part 3.6 Correct third time derivatives:
$(d^3rx_i/dt^3) \leftarrow (d^3rx_i/dt^3) + k3 \times deltax$
$(d^3ry_i/dt^3) \leftarrow (d^3ry_i/dt^3) + k3 \times deltay$
$(d^3rz_i/dt^3) \leftarrow (d^3rz_i/dt^3) + k3 \times deltaz$
end *i* **loop**

Algorithm 7.2 demonstrates the use of a four-value Gear method. The implementation is relatively straightforward. New positions, velocities, accelerations, and third time derivatives are predicted (Algorithm 7.2, Parts 1.1−1.4). The forces on the atoms at their predicted positions are evaluated (Algorithm 7.2, Part 2) as detailed in Chapter 5. This is the most time-consuming part of the algorithm. In the corrector step (Algorithm 7.2, Part 3), the forces are used to update the accelerations (Algorithm 7.2, Part 3.1) and to determine corrections. The corrections are subsequently applied (Algorithm 7.2, Parts 3.2−3.6) to update the positions, velocities, and the like. In Algorithm 7.2, the Gear constants are represented by the variables *k0*, *k1*, and *k3*.

7.2.1.1 Cautionary note on nomenclature

There is some confusion in the literature in the description of the different Gear algorithms. The order of the algorithm should refer to the order of the highest derivative. This means that a fourth-order Gear algorithm involves at most a fourth-order derivative but it has five coefficients or values (see Table 7.1). Therefore, fourth-order and five-value are interchangeable descriptions for the same Gear algorithm. However, a four-value (third-order) Gear algorithm is sometimes mistakenly referred to as fourth order, which would incorrectly imply the use of five values. The "value" description used here avoids this ambiguity.

7.2.2 Other representations of the Gear algorithm

The algorithm described earlier represents the most commonly used representation of the Gear predictor−corrector algorithm, namely, the Nordsieck

or *N*-representation (Berendsen and van Gunsteren, 1986). Other representations are also possible depending on what information is kept. In the four-value *N*-representation, three derivatives of r are kept, which can be used to define the column vector $\mathbf{r}_n$. Using dot notation for the time derivatives, we have (Berendsen and van Gunsteren, 1986),

$$\mathbf{r}_n = \left[\mathbf{r}_n, \Delta t\dot{\mathbf{r}}_n, \frac{\Delta t^2\ddot{\mathbf{r}}_n}{2}, \frac{\Delta t^3 \dddot{\mathbf{r}}_n}{6}\right]^T \cdots \tag{7.12}$$

where the elements of this vector are also vectors of length 3 *N*. The *N*-representation only uses values from step n. An alternative is to keep only the coordinates from previous steps. For example:

$$\mathbf{r}_n = [\mathbf{r}_n, \mathbf{r}_{n-1}, \mathbf{r}_{n-2}]^T \tag{7.13}$$

Equation (7.13) represents a three-value, three-step coordinate or *C*-representation of the Gear algorithm. It is also sometimes convenient to use coordinates and velocities of one step with forces from a previous step.

$$\mathbf{r}_n = \left[r_n, \Delta t\dot{\mathbf{r}}_n, \frac{\Delta t^2 t\ddot{\mathbf{r}}_n}{2}, \frac{\Delta t^2\ddot{\mathbf{r}}_{n-1}}{2}, \frac{\Delta t^2\ddot{\mathbf{r}}_{n-2}}{2}\right]^T. \tag{7.14}$$

Equation (7.14) is a five-value, three-step method in the force or *F*-representation. Berendsen and van Gunsteren (1986) have discussed the different representations in greater detail and they have reported values of the corrector coefficients for several cases. Janek and Kolafa (2020) have also proposed alternative Gear algorithms.

7.3 Verlet predictor methods

Verlet (1967) demonstrated that a simple predictor method developed by Störmer could be used successfully in MD. Many subsequent modifications and improvements have been proposed, and the "Verlet family" of algorithms collectively are the most widely used integration algorithms for MD.

The Verlet algorithm belongs to the symplectic class of integrators that preserves a canonical structure. There is evidence that symplectic algorithms are likely to be the preferred choice for the long-time integration of Hamiltonian dynamic systems (Leimkuhler and Skeel, 1994). Isbister et al. (1997) have discussed symplectic algorithms for nonequilibrium MD. The development and improvement of such integrators is an increasingly active area of research (Greer, 1994; Okunbor and Skeel, 1994; Janežič and Merzel, 1995; Jay, 1996; Okabe et al., 1996; Skeel et al., 1997). In this section, we trace the development of the cornerstone Verlet algorithms and we show how they are implemented.

7.3.1 Original Verlet algorithm

In common with the corrector–predictor algorithm, the starting point for the Verlet (1967) algorithm is a Taylor series expansion. We start by considering the following Taylor series expansions about $\mathbf{r}(t)$.

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \frac{d\mathbf{r}}{dt}\Delta t + \frac{1}{2!}\frac{d^2\mathbf{r}}{dt^2}\Delta t^2 + \cdots \tag{7.15}$$

$$\mathbf{r}(t-\Delta t) = \mathbf{r}(t) - \frac{d\mathbf{r}}{dt}\Delta t + \frac{1}{2!}\frac{d^2\mathbf{r}}{dt^2}\Delta t^2 + \cdots \tag{7.16}$$

When Eq. (7.15) and Eq. (7.16) are added, we obtain:

$$\mathbf{r}(t+\Delta t) = 2\mathbf{r}(t) - \mathbf{r}(t-\Delta t) + \frac{d^2\mathbf{r}}{dt^2}\Delta t^2 \tag{7.17}$$

Equation (7.17) is known as Verlet's algorithm. It enables us to advance the position of the molecules without calculating their velocities. However, the velocities are required to determine kinetic energy. They are obtained from:

$$\mathbf{v}(t) = \frac{\mathbf{r}(t+\Delta t) - \mathbf{r}(t-\Delta t)}{2\Delta t} \tag{7.18}$$

The implementation of the Verlet algorithm is illustrated by Algorithm 7.3. Before the Verlet algorithm can be implemented, the forces must be calculated (Algorithm 7.3, Part 1). The forces are used subsequently (Algorithm 7.3, Part 2.1) to calculate the accelerations (*ax*) required in Eq. (7.17). The first step in the Verlet procedure (Algorithm 7.3, Part 2.1) is to calculate the accelerations, which are used subsequently to advance the atomic coordinates (Algorithm 7.3, Part 2.2). Unlike the Gear algorithm, Eq. (7.17) requires both the current (rx_i) and previous values of position ($rxOld_i$) to be stored in order to advance the coordinates (*rxNew*). The velocities must also be calculated during the execution (Algorithm 7.3, Part 2.3) of the Verlet algorithm because this is the only time that both the updated positions (*rxNew*) and the previous positions (*rxOld*) required by Eq. (7.18) are available. Strictly, the velocity calculations in the Verlet procedure are optional because they do not contribute to the advancement of the atomic coordinates. However, the calculation should be included because the individual velocities and velocity sums are required to determine the total linear momentum, and kinetic energy, respectively. After the new positions and velocities are calculated, the values of the current and previous positions are updated (Algorithm 7.3, Parts 2.4 and 2.5).

ALGORITHM 7.3 Original Verlet method for integrating equations of motion.

Overall Algorithm

Part 1 Determine forces (fx_i, fy_i, ,fz_i).
Part 2 Apply Verlet algorithm.

Sub Algorithms

Part 2 Apply Verlet algorithm:
$\Sigma v^2 \leftarrow 0$, $\Sigma vx \leftarrow 0$, $\Sigma vy \leftarrow 0$, $\Sigma vz \leftarrow 0$ (optional)
loop $i \leftarrow 1 \ldots nAtom$
Part 2.1 Calculate current accelerations:
$ax \leftarrow fx_i/m_i$
$ay \leftarrow fy_i/m_i$
$az \leftarrow fz_i/m_i$
Part 2.2 Calculate new positions:
$rxNew \leftarrow$pbc$(2 \times rx_i - rxOld_i + \Delta t^2 \times ax)$
$ryNew \leftarrow$pbc$(2 \times ry_i - ryOld_i + \Delta t^2 \times ay)$
$rzNew \leftarrow$pbc$(2 \times rz_i - rzOld_i + \Delta t^2 \times az)$
Part 2.3 Calculate new velocities (optional):
$vx \leftarrow (rxNew - rxOld_i)/2\Delta t$
$vy \leftarrow (ryNew - ryOld_i)/2\Delta t$
$vz \leftarrow (rzNew - rzOld_i)/2\Delta t$
$\Sigma v^2 \leftarrow \Sigma v^2 + vx^2 + vy^2 + vz^2$
$\Sigma vx \leftarrow \Sigma vx + vx$
$\Sigma vy \leftarrow \Sigma vy + vy$
$\Sigma vz \leftarrow \Sigma vz + vz$
Part 2.4 Update old coordinates:
$rxOld_i \leftarrow rx_i$
$ryOld_i \leftarrow ry_i$
$rzOld_i \leftarrow ry_i$
Part 2.5 Update current coordinates:
$rx_i \leftarrow rxNew$
$ry_i \leftarrow ryNew$
$rz_i \leftarrow rzNew$
end *i* **loop**

The Verlet method is compact, easy to implement, time reversible, and it conserves energy even for long time steps. A minor disadvantage is that the algorithm is not self-starting. At $t = 0$, an estimate is required for the position at $t = -\Delta t$. Any reasonable estimate can be used without affecting the integrity of the algorithm. The main disadvantages of the Verlet algorithm are that it does not handle velocities well and its numerical precision is not optimal (Dahlquist and Björk, 1974). It should be noted that the velocities calculated by Eq. (7.18) are the current velocities rather than the velocities corresponding to the new time step. Consequently, the calculated velocities lag the calculated positions by one time interval. These deficiencies have been addressed by various modifications described below.

7.3.2 Leap-frog Verlet algorithm

Hockney (1970) proposed the following "leap-frog" scheme, which makes use of the velocity at half-time intervals.

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \Delta t \mathbf{v}\left(t + \frac{\Delta t}{2}\right) \tag{7.19}$$

$$\mathbf{v}\left(t + \frac{\Delta t}{2}\right) = \mathbf{v}\left(t - \frac{\Delta t}{2}\right) + \Delta t \mathbf{a}(t) \tag{7.20}$$

$$\mathbf{v}(t) = \frac{\mathbf{v}\left(t + \frac{\Delta t}{2}\right) + \mathbf{v}\left(t - \frac{\Delta t}{2}\right)}{2} \tag{7.21}$$

Unlike the original Verlet method, the next position is determined by both the current position and the velocity at the next half-time interval (Eq. 7.19). The velocity at the next half-time interval is determined by the current acceleration and the velocity at the previous half-time interval (Eq. 7.20). The current velocity is the average velocity of the velocities at the next and previous half-time intervals.

The implementation of the leap-frog scheme is illustrated by Algorithm 7.4. The algorithm follows the same general procedure as the Verlet algorithm but the calculation of velocities is now an integral part of the algorithm. The calculated accelerations (Algorithm 7.4, Part 2.1) are used to determine the half-time velocities (Algorithm 7.4, Part 2.2). In turn, the half-time velocities are used to update the atomic positions (Algorithm 7.4, Part 2.3) and the current velocities (Algorithm 7.4, Part 2.4). In the leap-frog algorithm, the values that must be stored at each time step are the current position (rx_i) and the previous half-time velocity ($vxHalfOld_i$). These values are updated following each change in coordinates (Algorithm 7.4, Parts 2.4 and 2.5).

ALGORITHM 7.4 Leap-frog Verlet method for integrating equations of motion.

Overall Algorithm

Part 1 Calculate forces (fx_i, fy_i, fz_i) at time t.
Part 2 Apply Verlet Leap-Frog algorithm at time $t + \Delta t$.

Sub Algorithms

Part 2 Apply Verlet Leap-Frog algorithm at time $t + \Delta t$:
$\Sigma v^2 \leftarrow 0$, $\Sigma vx \leftarrow 0$, $\Sigma vy \leftarrow 0$, $\Sigma vz \leftarrow 0$
loop $i \leftarrow 1 \ldots nAtom$
Part 2.1 Calculate accelerations:
$ax \leftarrow fx_i/m_i$
$ay \leftarrow fy_i/m_i$
$az \leftarrow fz_i/m_i$
Part 2.2 Calculate new half-time velocities:
$vxHalf \leftarrow vxHalfOld_i + \Delta t \times ax$
$vyHalf \leftarrow vyHalfOld_i + \Delta t \times ay$
$vzHalf \leftarrow vzHalfOld_i + \Delta t \times az$

Part 2.3 Calculate new positions:
$rxNew \leftarrow \text{pbc}(rx_i + \Delta t \times vxHalf)$
$ryNew \leftarrow \text{pbc}(ry_i + \Delta t \times vyHalf)$
$rzNew \leftarrow \text{pbc}(ry_i + \Delta t \times vzHalf)$
Part 2.4 Calculate current velocities:
$vx \leftarrow (vxHalf - vx\text{HalfOld}_i)/2$
$vy \leftarrow (vyHalf - vy\text{HalfOld}_i)/2$
$vz \leftarrow (vzHalf - vy\text{HalfOld}_i)/2$
$\Sigma v^2 \leftarrow \Sigma v^2 + vx^2 + vy^2 + vz^2$
$\Sigma vx \leftarrow \Sigma vx + vx$
$\Sigma vy \leftarrow \Sigma vy + vy$
$\Sigma vz \leftarrow \Sigma vz + vz$
Part 2.5 Update old half-time velocities:
$vxHalfOld_i \leftarrow vxHalf$
$vyHalfOld_i \leftarrow vyHalf$
$vzHalfOld_i \leftarrow vzHalf$
Part 2.6 Update current coordinates:
$rx_i \leftarrow rxNew$
$ry_i \leftarrow ryNew$
$rz_i \leftarrow rzNew$
end *i* **loop**

The main advantage of the leaf-frog algorithm is that numerical imprecision is reduced because it uses differences between smaller quantities. However, the calculation of velocities relies on velocity averaging at different time intervals and the algorithm is not self-starting because initially, we need to know the velocity at $t = -\Delta t/2$. This latter difficultly can be overcome by using reverse Euler estimates of velocities.

$$\mathbf{v}\left(-\frac{\Delta t}{2}\right) = \mathbf{v}(0) - \mathbf{a}(0)\frac{\Delta t}{2} \tag{7.22}$$

In common with the original Verlet algorithm, the calculation of velocities lags one time interval behind the calculation of positions.

7.3.3 Velocity-Verlet algorithm

Swope et al. (1982) proposed the following velocity-Verlet procedure that improves the calculation of the velocities.

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t) + \frac{\Delta t^2\mathbf{a}(t)}{2} \tag{7.23}$$

$$\mathbf{v}\left(t + \frac{\Delta t}{2}\right) = \mathbf{v}(t) + \frac{\Delta t\mathbf{a}(t)}{2} \tag{7.24}$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}\left(t + \frac{\Delta t}{2}\right) + \frac{\Delta t\mathbf{a}(t + \Delta t)}{2} \tag{7.25}$$

The velocity-Verlet algorithm is effectively a three-value predictor–corrector procedure as illustrated by Algorithm 7.5. First, the new positions ($rxNew_i$)

(Algorithm 7.5, Part 1.1) and half-time velocities ($rxHalf_i$) (Algorithm 7.5, Part 1.2) are calculated in accordance with Eqs. (7.23) and (7.24), respectively. The forces experienced by the atoms in their new positions are calculated (Algorithm 7.5, Part 2). Notice that in contrast to either the Verlet or leap-frog algorithms, the calculation of forces occurs at the midway point and not at the start of the algorithm. The remaining part of the calculation involves completing the velocity calculation (Algorithm 7.5, Part 3). The newly calculated forces are used to calculate the accelerations at $t = t + \Delta t$ (Algorithm 7.5, Part 3.1), which then enables the completion of the velocity calculation (Algorithm 7.5, Part 3.2) via Eq. (7.25). The current velocities (Algorithm 7.5, Part 3.3), accelerations (Algorithm 7.5, Part 3.3), and positions (Algorithm 7.5, Part 3.4) are updated at the completion of the algorithm.

ALGORITHM 7.5 Velocity-Verlet method for integrating equations of motion.

Overall Algorithm

Part 1 Calculate positions at time t + Δt and half-time velocities.
Part 2 Calculate forces (fx_i, fy_i, ,fz_i) at the new positions.
Part 3 Complete velocity calculation.

Sub Algorithms

Part 1 Calculate positions at time t + Δt and half-time velocities:
loop $i \leftarrow 1 .. nAtom$
Part 1.1 Calculate new positions:
$rxNew_i \leftarrow \text{pbc}(rx_i + \Delta t \times vx_i + \Delta t^2 \times ax_i/2)$
$ryNew_i \leftarrow \text{pbc}(ry_i + \Delta t \times vy_i + \Delta t^2 \times ay_i/2)$
$rzNew_i \leftarrow \text{pbc}(rz_i + \Delta t \times vz_i + \Delta t^2 \times az_i/2)$
Part 1.2 Calculate new half-time velocities:
$vxHalf_i \leftarrow vx_i + \Delta t \times ax_i/2$
$vyHalf_i \leftarrow vy_i + \Delta t \times ay_i/2$
$vzHalf_i \leftarrow vz_i + \Delta t \times az_i/2$
end *i* loop

Part 3 Complete velocity calculation:
$\Sigma v^2 \leftarrow 0, \Sigma vx \leftarrow 0, \Sigma vy \leftarrow 0, \Sigma vz \leftarrow 0$
loop $i \leftarrow 1 .. nAtom$
Part 3.1 Calculate updated accelerations:
$axNew \leftarrow fx_i/m_i$
$ayNew \leftarrow fy_i/m_i$
$azNew \leftarrow fz_i/m_i$
Part 3.2 Complete the calculation of velocities:
$vxNew \leftarrow vxHalf_i + \Delta t \times axNew$
$vyNew \leftarrow vyHalf_i + \Delta t \times ayNew$
$vzNew \leftarrow vzHalf_i + \Delta t \times azNew$
$\Sigma v^2 \leftarrow \Sigma v^2 + vxNew^2 + vyNew^2 + vzNew^2$
$\Sigma vx \leftarrow \Sigma vx + vxNew$
$\Sigma vy \leftarrow \Sigma vy + vyNew$
$\Sigma vz \leftarrow \Sigma vz + vzNew$

```
Part 3.3    Update current velocities:
            vx_i ←vxNew
            vy_i ←vyNew
            vz_i ←vzNew
Part 3.4    Update current accelerations:
            ax_i ← axNew
            ay_i ← ayNew
            az_i ← azNew
Part 3.5    Update current positions:
            rx_i ← rxNew
            ry_i ← ryNew
            rz_i ← rzNew
          end i loop
```

Unlike either the Verlet or the leap-frog algorithms, which require the force calculation to be performed upfront, the force calculation occurs midway through the velocity-Verlet procedure. The velocity-Verlet algorithm has the advantage that the calculation of the velocities is synchronized with the evolution of position. Equation (7.23) requires the storage of both positions and velocities, whereas the acceleration can be calculated from the stored values of force as required. However, because the accelerations are calculated at step 3 (*axNew*) as part of the evaluation of Eq. (7.25), it is also advantageous to store them (ax_i). Some adaptations to the velocity-Verlet approach have been proposed, but as noted by Lippert et al. (2007), care is required to avoid unintended side effects that may degrade the numerical accuracy.

7.3.4 Beeman modification

The procedure proposed by Beeman (Beeman, 1976; Sangster and Dixon, 1976) is possibly the most accurate of the Verlet family of algorithms for the calculation of velocities.

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t) + \frac{2}{3}\Delta t^2\mathbf{a}(t) - \frac{1}{6}\Delta t^2\mathbf{a}(t-\Delta t) \tag{7.26}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{1}{3}\Delta t\mathbf{a}(t+\Delta t) + \frac{5}{6}\Delta t\mathbf{a}(t) - \frac{1}{6}\Delta t\mathbf{a}(t-\Delta t) \tag{7.27}$$

Equation (7.27) ensures that the calculation of velocities is synchronized with the calculation of positions (Eq. 7.26).

The implementation of this strategy is illustrated by Algorithm 7.6. The advancement of positions via Eq. (7.26) does not involve a recalculation of forces. Instead, in the first step (Algorithm 7.6, Part 1), the positions are advanced using the current positions (rx_i), current velocities (vx_i), current accelerations (ax_i), and previous accelerations ($axOld_i$). However, after the

new positions have been calculated, the forces must be recalculated (Algorithm 7.6, Part 2) to update the velocities (Algorithm 7.6, Part 3.1) because Eq. (7.27) requires new values of the accelerations (*axNew*). Following the calculation of the new accelerations (Algorithm 7.6, Part 3.1) and velocities (Algorithm 7.6, Part 3.2), the stored values must be updated (Algorithm 7.6, Parts 3.3 and 3.4). The algorithm requires the storage of current values of position (rx_i), velocity (vx_i), and acceleration (ax_i) in addition to accelerations ($axOld_i$) from a previous time step.

In common with other modifications of the Verlet algorithm, the Beeman–Verlet algorithm is not self-starting. It also requires the storage of the old value of acceleration $\mathbf{a}(t - \Delta t)$. However, this increase in complexity is small and the algorithm is likely to yield both improved calculation of velocities and better energy conservation. In some cases, a shift in pressure has been reported (Litniewski, 2003) using the Beeman algorithm.

ALGORITHM 7.6 Beeman–Verlet method for integrating equations of motion.

Overall Algorithm

Part 1 Calculate new positions.
Part 2 Calculate forces (fx_i, fy_i, fz_i) at the new positions.
Part 3 Calculate new accelerations and velocities.

Sub Algorithms

Part 1 Calculate new positions:
loop $i \leftarrow 1$ .. *nAtom*
$rx_i \leftarrow \text{pbc}(rx_i + \Delta t \times vx_i + 2 \times \Delta t^2 \times ax_i/3 - \Delta t^2 \times axOld_i/6)$
$ry_i \leftarrow \text{pbc}(ry_i + \Delta t \times vy_i + 2 \times \Delta t^2 \times ay_i/3 - \Delta t^2 \times ayOld_i/6)$
$rz_i \leftarrow \text{pbc}(rz_i + \Delta t \times vz_i + 2 \times \Delta t^2 \times az_i/3 - \Delta t^2 \times azOld_i/6)$
end *i* **loop**

Part 3 Calculate new accelerations and velocities:
$\Sigma v^2 \leftarrow 0$, $\Sigma vx \leftarrow 0$, $\Sigma vy \leftarrow 0$, $\Sigma vz \leftarrow 0$
loop $i \leftarrow 1$ .. *nAtom*
Part 3.1 Calculate new accelerations:
$axNew \leftarrow fx_i/m_i$
$ayNew \leftarrow fy_i/m_i$
$azNew \leftarrow fz_i/m_i$
Part 3.2 Calculate new velocities:
$vxNew \leftarrow vx_i + \Delta t \times axNew/3 + 5 \times \Delta t \times ax_i/6 - \Delta t \times axOld_i/6$
$vyNew \leftarrow vy_i + \Delta t \times ayNew/3 + 5 \times \Delta t \times ay_i/6 - \Delta t \times ayOld_i/6$
$vzNew \leftarrow vx_i + \Delta t \times azNew/3 + 5 \times \Delta t \times az_i/6 - \Delta t \times azOld_i/6$
$\Sigma v^2 \leftarrow \Sigma v^2 + vxNew^2 + vyNew^2 + vzNew^2$
$\Sigma vx \leftarrow \Sigma vx + vxNew$
$\Sigma vy \leftarrow \Sigma vy + vyNew$
$\Sigma vz \leftarrow \Sigma vz + vzNew$
Part 3.3 Update velocities:
$vx_i \leftarrow vxNew$
$vy_i \leftarrow vyNew$
$vz_i \leftarrow vzNew$

```
Part 3.4    Update accelerations:
            axOld_i ← ax_i
            ayOld_i ← ay_i
            azOld_i ← az_i
            ax_i ← axNew
            ay_i ← ayNew
            az_i ← azNew
          end i loop
```

7.4 Runge–Kutta integration

The Verlet algorithms have been specifically adapted and refined for use in MD. However, as noted in the introduction, the problem can be generalized as the study of a pair of N-coupled first-order systems of differential equations.

$$\left.\begin{aligned} y' &= f(x, y, z) \\ z' &= g(x, y, z) \end{aligned}\right\} \tag{7.28}$$

Equation (7.28) is solved in parallel. The fourth-order Runge–Kutta (RK4) solution (Abramowitz and Stegun, 1972) involves the following evaluations,

$$\left.\begin{aligned} y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \\ z_{n+1} &= z_n + \frac{l_1}{6} + \frac{l_2}{3} + \frac{l_3}{3} + \frac{l_4}{6} \end{aligned}\right\} \tag{7.29}$$

where:

$$\left.\begin{aligned} k_1 &= hf(x_n, y_n, z_n) \\ l_1 &= hg(x_n, y_n, z_n) \\ k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}\right) \\ l_2 &= hg\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}\right) \\ k_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}\right) \\ l_3 &= hg\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}\right) \\ k_4 &= hf(x_n + h, y_n + k_3, z_n + l_3) \\ l_4 &= hg(x_n + h, y_n + k_3, z_n + l_3) \end{aligned}\right\} \tag{7.30}$$

In the context of MD, $y = \mathbf{r}$, $z = \mathbf{v}$, $x = t$ and $h = \Delta t$. Algorithm 7.7 illustrates a possible MD implementation of the RK4 algorithm.

ALGORITHM 7.7 RK4 integration of the equations of motion.

Overall Algorithm

Part 1 Store existing positions and velocities.
Part 2 Determine k_1 and l_1 terms and update positions and velocities.
Part 3 Calculate forces (fx_i, fy_i, ,fz_i) at the new positions.
Part 4 Determine k_2 and l_2 terms and update positions and velocities.
Part 5 Calculate forces (fx_i, fy_i, ,fz_i) at the new positions.
Part 6 Determine k_3 and l_3 terms and update positions and velocities.
Part 7 Calculate forces (fx_i, fy_i, ,fz_i) at the new positions.
Part 8 Determine k_4 terms.
Part 9 Determine the final positions and velocities.
Part 10 Update forces (fx_i, fy_i, ,fz_i) at the final positions.

Sub Algorithms

Part 1 Store existing positions and velocities:
loop $i \leftarrow 1$ .. *nAtom*
$rxOld_i \leftarrow rx_i$
$ryOld_i \leftarrow ry_i$
$rzOld_i \leftarrow rz_i$
$vxOld_i \leftarrow vx_i$
$vyOld_i \leftarrow vy_i$
$vzOld \leftarrow vz_i$
end *i* loop

Part 2 Determine k_1 and l_1 terms and update positions and velocities:
loop $i \leftarrow 1$ .. *nAtom*
$k_1x_i \leftarrow vx_i$
$k_1y_i \leftarrow vy_i$
$k_1z_i \leftarrow vz_i$
$rx_i \leftarrow rxOld_i + \Delta t \times k_1x_i /2$
$ry_i \leftarrow ryOld_i + \Delta t \times k_1y_i /2$
$rz_i \leftarrow rzOld_i + \Delta t \times k_1z_i /2$
$l_1x_i \leftarrow fx_i/m_i$
$l_1y_i \leftarrow fy_i/m_i$
$l_1z_i \leftarrow fz_i/m_i$
$vx_i \leftarrow vxOld_i + \Delta t \times l_1x_i /2$
$vy_i \leftarrow vyOld_i + \Delta t \times l_1y_i /2$
$vz_i \leftarrow vzOld_i + \Delta t \times l_1z_i /2$
end *i* loop

Part 4 Determine k_2 and l_2 terms and update positions and velocities:
loop $i \leftarrow 1$ .. *nAtom*
$k_2x_i \leftarrow vx_i$
$k_2y_i \leftarrow vy_i$
$k_2z_i \leftarrow vz_i$
$rx_i \leftarrow rxOld_i + \Delta t \times k_2x_i /2$
$ry_i \leftarrow ryOld_i + \Delta t \times k_2y_i /2$
$rz_i \leftarrow rzOld_i + \Delta t \times k_2z_i /2$
$l_2x_i \leftarrow fx_i/m_i$
$l_2y_i \leftarrow fy_i/m_i$
$l_2z_i \leftarrow fz_i/m_i$

```
    vx_i ← vxOld_i + Δt × l2x_i /2
    vy_i ← vyOld_i + Δt × l2y_i /2
    vz_i ← vzOld_i + Δt × l2z_i /2
end i loop

Part 6 Determine k3 and l3 terms and update positions and velocities:
loop i ← 1 .. nAtom
    k3x_i ← vx_i
    k3y_i ← vy_i
    k3z_i ← vz_i
    rx_i ← rxOld_i + Δt × k3x_i
    ry_i ← ryOld_i + Δt × k3y_i
    rz_i ← rzOld_i + Δt × k3z_i
    l3x_i ← fx_i/m_i
    l3y_i ← fy_i/m_i
    l3z_i ← fz_i/m_i
    vx_i ← vxOld_i + Δt × l3x_i
    vy_i ← vyOld_i + Δt × l3y_i
    vz_i ← vzOld_i + Δt × l3z_i
end i loop

Part 8 Determine k4 terms:
loop i ← 1 .. nAtom
    k4x_i ← vx_i
    k4y_i ← vy_i
    k4z_i ← vz_i
    l4x_i ← fx_i/m_i
    l4y_i ← fy_i/m_i
    l4z_i ← fz_i/m_i
end i loop

Part 9 Determine the final positions and velocities:
loop i ← 1 .. nAtom
    rx_i ← rxOld_i + Δt × (k1x_i /6 + k1x_i/ 3 + k2x_i /3 + k3x_i /3 + k3x_i /6)
    ry_i ← ryOld_i + Δt × (k1x_i /6 + k1x_i/ 3 + k2x_i /3 + k3x_i /3 + k3x_i /6)
    rz_i ← rzOld_i + Δt × (k1x_i /6 + k1x_i/ 3 + k2x_i /3 + k3x_i /3 + k3x_i /6)
    vx_i ←vxOld_i + Δt × (l1x_i /6 + l1x_i/3 + l2x_i /3 + l3x_i /3 + l3x_i /6)
    vy_i ←vyOld_i + Δt × (l1x_i /6 + l1x_i/3 + l2x_i /3 + l3x_i /3 + l3x_i /6)
    vz_i ←vzOld_i + Δt × (l1x_i /6 + l1x_i/3 + l2x_i /3 + l3x_i /3 + l3x_i /6)
end i loop
```

The algorithm starts (Algorithm 7.7, Part 1) by storing the initial positions and velocities because these will be needed repeatedly. Values of k_1 and l_1 are determined and new coordinates are determined (Algorithm 7.7, Part 2). The new coordinates are used to determine the forces (Algorithm 7.7, Part 3). This is repeated for the k_2, l_2 (Algorithm 7.7, Parts 4 and 5) and k_3, l_3 (Algorithm 7.7, Parts 6 and 7) pairs. The k_4 and l_4 terms are subsequently evaluated (Algorithm 7.7, Part 8), which allows the final coordinates and velocities to be determined (Algorithm 7.7, Part 9) via

Eq. (7.29). Although not explicitly shown, it is also advisable to apply periodic boundary conditions after Part 9. The final update to the coordinates means that the forces have to be re-evaluated (Algorithm 7.7, Part 10).

The advantages of the RK4 algorithm are that it is self-starting and highly accurate. However, the major disadvantage is that it requires four force evaluations compared with only one force evaluation of alternative algorithms. The sequential order-dependency of the force evaluations also limits opportunities for computational gains from parallel implementation strategies. Therefore, RK4 is used in relatively limited circumstances (Janežič and Orel, 1993; Trembiay and Carrington, 2004; Tupper, 2005), which require a high degree of accuracy. One such example is the transient-time correlation function (Todd, 1997). It is also not symplectic, but modifications have been reported (Yoshida, 1990) to address this issue. A simper alternative is the second-order Runge−Kutta (RK2) algorithm (Abramowitz and Stegun, 1972) but this lacks the accuracy of RK4 while still requiring multiple force devaluations. The RK4 algorithm can play a valuable supporting role for non-self-starting algorithms by providing the first few simulation data.

7.5 Comparison of integrators

The various integrator algorithms have been compared extensively (Berendsen and van Gunsteren, 1986; Allen and Tildesley, 2017). Kolafa (1996) has examined the suitability of several different integrators for polarizable fluids. The main criteria for evaluating an algorithm are: speed, ease of implementation, and accuracy. Another issue for some applications is whether the integrator is symplectic, which would exclude the Gear and RK4 algorithms. For example, it has been reported (Ratanapisit et al., 2001) that symplectic algorithms may be more accurate than non-symplectic algorithms for some transport properties. However, in most circumstances, this is not an important decision-making consideration.

It should be noted that speed of execution of the above integrator algorithms is very similar. The Gear algorithms are slower than the Verlet algorithms but the difference in speed is not significant. This is because the speed of the integration is dominated overwhelming by the evaluation of forces. Therefore, because all of the above integrators require one force evaluation, the execution times for the algorithms are broadly equivalent. Some of the algorithms require more storage of variables than others. However, all of the memory requirements are modest and they can be handled very easily by modern computers.

The Verlet algorithms are considerably easier to implement than the Gear algorithms and this is certainly an advantage. However, the crucial factor is accuracy. The trajectory predicted by the Verlet algorithms are likely to be equivalent. However, either the velocity-Verlet or Beeman algorithms calculate the velocity more accurately. If the time step Δt is small and the duration of the simulation is relatively short, either a four- or five-value Gear algorithm is more accurate

than the Verlet algorithms. However, the accuracy of the Gear algorithms deteriorates rapidly with both increasing step size and duration of the simulation. The root mean square fluctuation in energy for the Verlet algorithm is almost a linear function of Δt^2. Consequently, for medium to large values of Δt, the Verlet algorithms are considerably more accurate than the Gear algorithms.

7.6 Integrators for molecules

The previously discussed integrators can be applied directly to determine the equations of motion for atoms interacting via an appropriate intermolecular potential. For example, many MD studies of atomic systems have been reported (Haile, 1992; Rapaport, 2004) using the square-well, Lennard-Jones, and other potentials (Chapters 3 and 4). However, the goal of molecular simulation is to model real molecules as closely as possible. Unlike atoms that can be treated as spherical particles, the simulation of molecules is complicated by features such as rotation, intramolecular bonds, and the like. In particular, molecular bonds constrain the motion of the atoms that comprise the molecule. Therefore, methods to integrate the equations of motion of molecules must take account of the constraints imposed by bonds.

7.6.1 Small molecules

The treatment of dimers, trimers, tetramers, and the like introduces the complication of molecular bonds and intramolecular orientations. From the perspective of mechanics (Chapter 2), it is possible to treat these small polyatoms as rigid bodies by assuming that the bond lengths and bond angles are fixed. This is a reasonable approximation if the amplitude of molecular vibration is small compared with molecular size.

Goldstein et al. (2008) formulated the kinematics of rigid bodies (Chapter 2). The motion of a rigid body has two independent contributions arising from the translational motion of the center of mass and rotation about the center of mass. The translational motion is determined by the total force acting on the body, whereas rotation depends on the applied torque. Translation of a polyatomic molecule can be treated in the same way as translation of an atom but the dynamics of rotation must be treated separately.

To treat rotation, three Euler angles are defined, which represent the sequence of rotations of the Cartesian axes about the origin. The ϕ angle is the extent of clockwise rotation about the z-axis, which is followed by a rotation of θ about the new x-axis, and a final rotation ψ about the new z-axis. However, the solution of either the Lagrangian or Hamiltonian equations (Chapter 2) using Euler angles to represent the orientations suffers from the limitation that each of these equations possesses singularities at $\theta = 0$ or π. The problem can be avoided by redefining the laboratory coordinate frame (Barojas et al., 1973). Alternatively, Evans (1977) proposed a set of four parameters or quaternions to replace the Euler angles in the equations of motion.

The quaternion parameters are defined in terms of Euler angles.

$$\left.\begin{aligned} q_0 &= \cos\left(\frac{\theta}{2}\right)\cos\left(\frac{(\phi+\psi)}{2}\right) \\ q_1 &= \sin\left(\frac{\theta}{2}\right)\cos\left(\frac{(\phi-\psi)}{2}\right) \\ q_2 &= \sin\left(\frac{\theta}{2}\right)\sin\left(\frac{(\phi-\psi)}{2}\right) \\ q_3 &= \cos\left(\frac{\theta}{2}\right)\sin\left(\frac{(\phi+\psi)}{2}\right) \end{aligned}\right\} \tag{7.31}$$

The quaternions are not independent but satisfy the relationship:

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \tag{7.32}$$

The rotation matrix A is defined by,

$$V_{principal} = A \cdot V_{laboratory} \tag{7.33}$$

and is given by:

$$A = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix} \tag{7.34}$$

The principal angular velocity, ω_p, is related to the quaternions by the matrix:

$$\begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} 0 \\ \omega_{px} \\ \omega_{py} \\ \omega_{pz} \end{pmatrix} \tag{7.35}$$

The quarternion method and other alternatives have been discussed in detail elsewhere (Rapaport, 2004; Allen and Tildesley, 2017). One possible alternative is rotation-*shake* (Kol et al., 1997).

7.6.2 Large molecules

In general, molecules can be considered as chains of atoms linked by bonds of various strength, length, and type. In principle, the quaternions approach could be applied to a large multi-atom molecules. For example, Refson and

Pawley (1987a, b) have applied the technique to butane. However, finding the required coordinates for large flexible molecules is difficult and would invariably result in very awkward equations of motion.

Molecules are constrained by various intra-atomic separations and bond angles. The concept of constraint introduced in the formulation of Lagrangian and Hamiltonian mechanics (Chapter 2) can be applied usefully to model the dynamics of large molecules. Consider the simple case of a triatomic molecule as illustrated in Fig. 7.1.

Ignoring any angular constraints, the triatomic molecule is subject to at least two bonding constraints,

$$\left.\begin{aligned} \mathbf{r}_{ij}^2 - d_{ij}^2 = 0 \\ \mathbf{r}_{jk}^2 - d_{jk}^2 = 0 \end{aligned}\right\} \tag{7.36}$$

where, in this case, the d values are the equilibrium bond separations. If either condition in Eq. (7.36) is not satisfied, a bond is broken and the triatomic molecule is fragmented.

In general, if $\sigma(\mathbf{r})$ represents the constraint, a molecule composed of N atoms can have M constraints (de Leeuw et al., 1990; Tildesley, 1993).

$$\left.\begin{aligned} \sigma_1(\mathbf{r}) = 0 \\ \cdot \\ \cdot \\ \sigma_M(\mathbf{r}) = 0 \end{aligned}\right\} \tag{7.37}$$

The 3 N Euler−Lagrange equations (see Chapter 2) for the molecule are:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{r}}_i}\right) - \frac{\partial L}{\partial \mathbf{r}_i} = \mathbf{g}_i \tag{7.38}$$

In Eq. (7.38), L is the Lagrangian for the system of N atoms and $\mathbf{g}_i$ is the constraint force on atom i,

$$\mathbf{g}_i = \sum_{\alpha=1}^{M} \lambda_\alpha \left(\frac{\partial \sigma_\alpha}{\partial \mathbf{r}_i}\right) \tag{7.39}$$

where λ_α are the undetermined multipliers associated with the constraints.

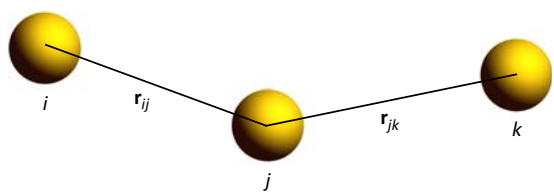


FIGURE 7.1 Representation of a triatomic molecule.

Equation (7.38) can be used (de Leeuw et al., 1990) to derive $6N$ Hamiltonian equations of motions,

$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i} - \frac{1}{m_i}\sum_{\alpha=1}^{M}\gamma_\alpha\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right) \tag{7.40}$$

$$\dot{\mathbf{p}}_i = -\frac{\partial u}{\partial\mathbf{r}_i} + \sum_{\alpha=1}^{M}\sum_{j=1}^{N}\gamma_\alpha\left(\frac{\partial^2\sigma_\alpha}{\partial\mathbf{r}_i\partial\mathbf{r}_j}\right)\dot{\mathbf{r}}_j \tag{7.41}$$

where $\dot{\gamma}_\alpha = \lambda_\alpha$. Furthermore, these first-order equations can be transformed to $3N$ second-order differential equations of the form,

$$\ddot{\mathbf{r}} = \mathbf{a}_i - \frac{1}{m_i}\sum_{\alpha}\lambda_\alpha\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right) \tag{7.42}$$

where $\mathbf{a}_i$ is the acceleration of atom i. The time derivative of the constraining forces is obtained by differentiating Eq. (7.37).

$$\frac{d\sigma_\alpha}{dt} = \sum_i \dot{\mathbf{r}}_i\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right) = 0 \tag{7.43}$$

$$\frac{d^2\sigma_\alpha}{dt^2} = \sum_i \ddot{\mathbf{r}}_i\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right) + \sum_i\sum_j \dot{\mathbf{r}}_i\left(\frac{\partial^2\sigma_\alpha}{\partial\mathbf{r}_i\partial\mathbf{r}_j}\right)\dot{\mathbf{r}}_j = 0 \tag{7.44}$$

An equation for the multipliers can be obtained by multiplying Eq. (7.42) by $(\partial\sigma_\alpha/\partial\mathbf{r}_i)$, summing over i and substituting Eqs. (7.43) and (7.44).

$$-\sum_i\sum_j \dot{\mathbf{r}}_i\left(\frac{\partial^2\sigma_\alpha}{\partial\mathbf{r}_i\partial\mathbf{r}_j}\right)\dot{\mathbf{r}}_j = \sum_i \mathbf{a}_i\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right) - \sum_i\frac{1}{m_i}\sum_\beta\lambda_\beta\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right)\left(\frac{\partial\sigma_\beta}{\partial\mathbf{r}_i}\right) \tag{7.45}$$

In matrix notation, we have

$$\Lambda = (\mathbf{M})^{-1}(\mathbf{F} + \mathbf{T}) \tag{7.46}$$

where $\mathbf{M}$ is a $M \times M$ matrix with the elements:

$$(\mathbf{M})_{\alpha\beta} = \sum_i \frac{1}{m_i}\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right)\left(\frac{\partial\sigma_\beta}{\partial\mathbf{r}_i}\right) \tag{7.47}$$

$\mathbf{\Lambda}$, $\mathbf{F}$, and $\mathbf{T}$ are vectors of length α.

$$(\mathbf{\Lambda})_\alpha = \lambda_\alpha \tag{7.48}$$

$$(\mathbf{F})_\alpha = \sum_i \mathbf{a}_i\left(\frac{\partial\sigma_\alpha}{\partial\mathbf{r}_i}\right) \tag{7.49}$$

$$(\mathbf{T})_\alpha = \sum_i\sum_j \dot{\mathbf{r}}_i\left(\frac{\partial^2\sigma_\alpha}{\partial\mathbf{r}_i\partial\mathbf{r}_j}\right)\dot{\mathbf{r}}_j \tag{7.50}$$

Equation. (7.42) can now be solved exactly because Eq. (7.48) provides the required values of the undetermined multipliers. However, in practice, an exact solution leads to considerable inaccuracy because the error in the numerical solution of the differential equation means that the constraints are only satisfied to the accuracy of the algorithm. Consequently, the error associated with the exact solution increases progressively with time.

7.6.2.1 The shake algorithm

The *shake*[1] algorithm (Ryckaert et al., 1977) is possibly the most widely used method for calculating the equations of motion for molecules that are subject to constraints.

Shake was developed specifically to be used in conjunction with the Verlet (1967) integrator, which for unconstrained atoms is:

$$\mathbf{r}_i'(t+\Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-\Delta t) + \frac{\Delta t^2}{m_i}\mathbf{f}_i \tag{7.51}$$

However, because the atoms are subject to constraints such as bond separations, the true positions are:

$$\mathbf{r}_i(t+\Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-\Delta t) + \frac{\Delta t^2}{m_i}(\mathbf{f}_i - \mathbf{g}_i) \tag{7.52}$$

Obtaining the constraining force from Eq. (7.39):

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i'(t+\Delta t) - \frac{\Delta t^2}{m_i}\sum_{\alpha=1}^{M}\lambda_\alpha\left(\frac{\partial\sigma_\alpha(\mathbf{r}(t))}{\partial\mathbf{r}_i}\right) \tag{7.53}$$

The atomic coordinates at time $t + \Delta t$ must obey the constraint equations given by (37).

$$\left.\begin{aligned}\sigma_1(\mathbf{r}(t+\Delta t)) &= 0\\ &\cdot\\ &\cdot\\ \sigma_M(\mathbf{r}(t+\Delta t)) &= 0\end{aligned}\right\} \tag{7.54}$$

Equation (7.54) represents a set of α nonlinear equations for the multipliers λ_α that can be solved using the *shake* algorithm (Ryckaert et al., 1977; Ciccotti and Ryckaert, 1986).

The first step in the *shake* algorithm is to obtain the unconstrained atomic positions $\mathbf{r}_i'(t + \Delta t)$ as represented by Eq. (7.51). These coordinates are adjusted iteratively until the constraint equations are satisfied to within a

1. We note that the *shake* algorithm is most commonly denoted as "SHAKE." This can be attributed to its origin as a FORTRAN implementation (Ryckaert et al., 1977), which only used uppercase letters. However, as the name does not represent an acronym, the use of italics is more appropriate.

specified tolerance. Each iteration cycles through all of the independent constraints. Typically, a given constraint (ω) will only involve a subset of all the atoms (N_ω). At the *I*th iterative loop, the atomic coordinates are $\mathbf{r}_i^{\text{old}}$. The forces of constraint act on these atoms at this iteration to yield new positions.

$$\left.\begin{aligned}\mathbf{r}_1^{new} &= \mathbf{r}_1^{old} - \lambda_{\varpi}^{I}\left(\frac{\partial\sigma_\omega(\mathbf{r}(t))}{\partial\mathbf{r}_1}\right)\\ &\cdot\\ &\cdot\\ \mathbf{r}_{N_\omega}^{new} &= \mathbf{r}_{N_\omega}^{old} - \lambda_{\varpi}^{I}\left(\frac{\partial\sigma_\omega(\mathbf{r}(t))}{\partial\mathbf{r}_{N_\omega}}\right)\end{aligned}\right\} \tag{7.55}$$

The Lagrange multiplier is evaluated from:

$$\lambda_\omega^I = \frac{\sigma_\omega(\mathbf{r}^{old})}{\Delta t^2 \sum_{i=1}^{N_\omega}\frac{1}{m_i}\left(\frac{\partial\sigma_\omega(\mathbf{r}^{old})}{\partial\mathbf{r}_i}\right)\left(\frac{\partial\sigma_\omega(\mathbf{r}(t))}{\partial\mathbf{r}_i}\right)} \tag{7.56}$$

The multiplier evaluated from Eq. (7.56) for the ωth constraint and *I*th iteration is used in Eq. (7.55) to obtain a new guess for the atomic positions. After tens of iterations, all the constraints are satisfied to a high degree of accuracy and the constrained atomic positions $\mathbf{r}(t + \Delta t)$ are known.

A *shake* algorithm for a chain molecule consisting of *nAtom* units with bond constraints is illustrated by Algorithm 7.8. Prior to the implementation of the *shake* algorithm (Algorithm 7.8, Part 2), values of the atomic coordinates and the forces acting on each atom must be evaluated (Algorithm 7.8, Part 1). The *shake* algorithm applies the constraints subject to a predetermined *tolerance* limit. At the outset, new atomic coordinates are evaluated for each atom of each molecule using the Verlet algorithm (Algorithm 7.8, Part 2.1). During the Verlet procedure, two bookkeeping arrays are initialized to keep track of whether the atoms are being moved. The array *moving* indicates whether the atom is being moved whereas *moved* indicates whether or not the atom was moved previously. The use of these arrays is optional, however, they enable us to minimize unnecessary computation by repeatedly correcting the position of the same atom. The iterative procedure (Algorithm 7.8, Part 2.2) for evaluating the constraints follows the Verlet step. The squared bond length *bondSq* is used to constrain the new positions. The constraint is applied between each *atom* and its *neighbor* found at position *atom + 1*. The number of bonds (*nBonds*) is equal to the number of atoms (*nAtoms*) for cyclic molecule whereas for a noncyclic molecule, *nBonds = nAtoms - 1*. The square of the distances between neighboring molecule *separationSq* in their new

positions is evaluated (Algorithm 7.8, Part 2.2.1). If *separationSq* exceeds the tolerance value, the old neighbor separations *rxOldSepartaion*, *ryOldSeparation*, and *rzOldSeparation* are calculated. A check is made to determine whether or not the vector product *rNewOld* of the old and new configurations is less than the tolerated separation. If so, an error has been encountered and the *shake* procedure is terminated. Otherwise, λ is evaluated and the atomic coordinates are adjusted accordingly. A correction is also made to the virial. The iteration terminates when either all the constraints are satisfied, an error is encountered, or a predefined iteration count is exceeded. At the end of the *shake* procedure (Algorithm 7.8, Part 2.3), the kinetic energy is calculated, and the new and old atomic coordinates are updated.

ALGORITHM 7.8 *Shake* method for incorporating bond-length constraints.

Overall Algorithm

Part 1 Calculate the positions and forces of all atoms.
Specify bond constraints.
tolerance $\leftarrow 10^{-6}$
virialCorrection $\leftarrow 0$
kineticEnergy $\leftarrow 0$
Part 2 Apply shake for all molecules:
loop *i* $\leftarrow$1 ... *nMolecule*
Part 2.1 Update atomic positions for each molecule.
Part 2.2 Iterative application of constraints.
Part 2.3 Evaluate kinetic energy and update atomic coordinates.
end *i* **loop**
if (*error* = *false* and *done* = *true)*
virialCorrection $\leftarrow$ *virialCorrection*/$(3\Delta t^2)$
kineticEnergy $\leftarrow$*kineticEnergy*/2
end if

Sub Algorithms

Part 2.1 Update atomic positions for each molecule (Verlet method):
loop *atom* $\leftarrow$ 1 ... *nAtom*
$rxOld_{atom} \leftarrow rx_{i,atom}$
$ryOld_{atom} \leftarrow ry_{i,atom}$
$rzOld_{atom} \leftarrow rz_{i,atom}$
$rxNew_{atom} \leftarrow$ pbc$(2 \times rx_{i,atom} - rxOld_{i,atom} + \Delta t^2 \times fx_{i,atom}/mass_{atom})$
$ryNew_{atom} \leftarrow$ pbc$(2 \times ry_{i,atom} - ryOld_{i,atom} + \Delta t^2 \times fy_{i,atom}/mass_{atom})$
$rzNew_{atom} \leftarrow$ pbc$(2 \times rz_{i,atom} - rzOld_{i,atom} + \Delta t^2 \times fz_{i,atom}/mass_{atom})$
$moving_{atom} \leftarrow false$
$moved_{atom} \leftarrow true$
end *atom* **loop**

```
Part 2.2 Iterative application of constraints:
count ← 1
done ← false
error ← false
loop while (done = false and count < maxCount)
   done ← true
   atom ← 1
   loop while (atom < nBonds and error = false)
      neighbor ← atom + 1
      if (neighbor > nAtom)
         neighbor ← 1
      end if
      Part 2.2.1 Evaluate separation between neighboring atoms.
   end while loop
   if(error = false)
      loop atom ←1 ... nAtom
         moved_atom ← moving_atom
         moving_atom ← false
      end atom loop
      count ← count + 1
   end if
end while loop
```

```
Part 2.2.1 Evaluate separation between neighboring atoms:
if (moved_atom or moved_neighbor)
   rxNewSeparation ← pbc(rxNew_atom - rxNew_neighbor)
   ryNewSeparation ← pbc(ryNew_atom - ryNew_neighbor)
   rzNewSeparation ← pbc(rzNew_atom - rzNew_neighbor)
end if
separationSq ← rxNewSeparation^2
             + ryNewSeparation^2 + rzNewSeparation^2
if (|bondSq_atom - separationSq| > 2 × tolerance × bondSq_atom)
   rxOldSeparation ← pbc(rxOld_atom - rxOld_neighbor)
   ryOldSeparation ← pbc(ryOld_atom - ryOld_neighbor)
   rzOldSeparation ← pbc(rzOld_atom - rzOld_neighbor)
   rNewOld → rxOldSeparation × rxNewSeparation
           + ryOldSeparation × ryNewSeparation
           + rzOldSeparation × rzNewSeparation
   if (rNewOld ≥ tolerance × bondSq_atom)
      λ ←|bondSq_atom - separationSq|/
           (2 × rNewOld × (1/mass_atom + 1/mass_neighbor))
      virialCorrection ← virialCorrection + λ × bondSq_atom
      rxNew_atom ← pbc (rxNew_atom + λ × rxOldSeparation/mass_atom)
      ryNew_atom ← pbc (ryNew_atom + λ × ryOldSeparation/mass_atom)
      rzNew_atom ← pbc (rzNew_atom + λ × rzOldSeparation/mass_atom)
      rxNew_neighbor ← pbc
                  (rxNew_neighbor + λ × rxOldSeparation/mass_neighbor)
      ryNew_neighbor ← pbc
                  (ryNew_neighbor + λ × ryOldSeparation/mass_neighbor)
      rzNew_neighbor ← pbc
                  (rzNew_neighbor + λ × rzOldSeparation/mass_neighbor)
      moving_atom ← true
      moving_neighbor ← true
```

```
      done ←false
      atom ←atom + 1
   end if
else
   error ← true
end if

Part 2.3 Evaluate kinetic energy and update atomic coordinates:
if (error = false and done = true)
   loop atom ← 1, nAtom
      vx ← (rxNew_atom - rxOld_i,atom)/2Δt
      vy ← (ryNew_atom - ryOld_i,atom)/2Δt
      vz ← (rzNeq_atom - rzOld_i,atom)/2Δt
      kineticEnergy ← kineticEnergy + mass_atom × (vx^2 + vy^2 + vz^2)
      rx_i,atom ← rxNew_atom
      ry_i,atom ← ryNew_atom
      rz_i,atom ← rzNew_atom
      rxOld_i,atom ← rx_atom
      ryOld_i,atom ← ry_atom
      rzOld_i,atom ← rz_atom
   end atom loop
end if
```

The *shake* algorithm can also be implemented (van Gunsteren and Berendsen, 1977) using the predictor−corrector integrator. Brown (1997) has clarified some aspects of the force of constraint used in the original predictor−corrector implementation of the *shake* algorithm and he reported an improved version. To describe the predictor−corrector implementation of the *shake* algorithm, we will use the following notation.

$$shake(\mathbf{r}_1,\ \mathbf{r}_2,\ \mathbf{r}_3) \tag{7.57}$$

This means that the positions $\mathbf{r}_2$ resulting from the nonconstraint step are reset to constrained positions $\mathbf{r}_3$. The direction of the displacement vectors $(\mathbf{r}_2 - \mathbf{r}_1)$ is determined by the reference positions $\mathbf{r}_1$.

The first step of the algorithm is initiated by predicting new positions, accelerations, and velocities via,

$$\mathbf{x}_p(t + \Delta t) = \mathbf{B}\mathbf{x}(t) \tag{7.58}$$

where $\mathbf{x}$ is a vector containing the positions, accelerations, and velocities, and $\mathbf{B}$ is the predictor matrix in the F-representation. At this stage, the following summation is calculated for later use.

$$Y = \sum_{\substack{i=0 \\ i \neq 1}}^{k-1} (B_{1i} - B_{11}B_{0i})r_i(t) \tag{7.59}$$

In the absence of any forces, the positions are obtained by applying the following correction to the positions $\mathbf{r}(t + \Delta t)$ predicted from Eq. (7.58),

$$\mathbf{r}_2 = \mathbf{r}_p(t + \Delta t) - k_0 \frac{\Delta t^2 \ddot{\mathbf{r}}_p(t + \Delta t)}{2} \tag{7.60}$$

where k_0 is the corrector coefficient associated with position.

The *shake* procedure is applied to positions $\mathbf{r}_2$ using positions $\mathbf{r}(t)$ as a reference.

$$shake(\mathbf{r}(t),\ \mathbf{r}_2,\ \mathbf{r}_3) \tag{7.61}$$

The positions are corrected for the effect of the unconstrained or free forces.

$$\mathbf{r}_{free}(t + \Delta t) = \mathbf{r}_3 + k_0 \frac{\Delta t^2 \mathbf{f}_{free,p}(t + \Delta t)}{2m} \tag{7.62}$$

The *shake* procedure is now applied to the above positions using the $\mathbf{r}_p(t + \Delta t)$ positions as a reference.

$$shake(\mathbf{r}_p(t + \Delta t),\ \mathbf{r}_{free,\mathrm{p}}(t + \Delta t),\ \mathbf{r}_{total}(t + \Delta t)) \tag{7.63}$$

This results in the final positions $\mathbf{r}_{\mathrm{total}}(t + \Delta t)$.

The force of constraint is calculated from:

$$\mathbf{f}_{\mathrm{constraint}} = \frac{2(\mathbf{r}_{\mathrm{total}}(t + \Delta t) - \mathbf{r}_{\mathrm{free}}(t + \Delta t))}{k_0 \Delta t^2} \tag{7.64}$$

Consequently, the total force can be calculated.

$$\mathbf{f}_{\mathrm{total}} = \mathbf{f}_{\mathrm{free}} + \mathbf{f}_{\mathrm{constraint}} \tag{7.65}$$

Finally, the accelerations and velocity can be obtained from,

$$\mathbf{a}(t + \Delta t) = \mathbf{a}_p(t + \Delta t) + \frac{k_2}{2}\left(\frac{\mathbf{f}_{\mathrm{total}}}{m} - \ddot{\mathbf{r}}_p(t + \Delta t)\right) \tag{7.66}$$

and:

$$\mathbf{v}(t + \Delta t) = \frac{B_{11}\mathbf{r}(t + \Delta t) + Y}{\Delta t} + \frac{\Delta t}{2}\left(\frac{\mathbf{f}_{\mathrm{total}}}{m} - \ddot{\mathbf{r}}_p(t + \Delta t)\right) \tag{7.67}$$

7.6.2.2 The rattle algorithm

The Verlet-based implementation of the *shake* algorithm suffers from the same limitations that characterize the Verlet algorithm. The atomic velocities are not used in the integration of the equations of motion. Instead, approximate values are calculated following the integration step. It is difficult to start the algorithm with previously chosen values of coordinates and velocities, and it is also difficult to alter the time step and continue the calculation. The

algorithm also suffers from loss of precision because it involves adding quantities that vary greatly in magnitude.

For atomic systems, the limitations of the original Verlet algorithm are addressed by the velocity-Verlet algorithm described in Section 7.3.3. However, as discussed by Andersen (1983), the *shake* algorithm cannot be expressed in terms of the velocity-Verlet method. Instead, it is possible to generalize the *shake* algorithm to include the calculation of velocities (Andersen, 1983). The resulting *rattle*[2] algorithm that is called calculates the positions and velocities at a future time step from the positions and velocities of the present time step without information from earlier time steps.

The key equations for the *rattle* algorithm (Andersen, 1983) are:

$$\dot{\mathbf{r}}_i(t+\Delta t) = \dot{\mathbf{r}}_i(t) + h\dot{\mathbf{r}}_i(t) + \frac{\Delta t^2}{2m_i}\left[\mathbf{f}_i(t) - 2\sum_j \lambda_{RRij}(t)\mathbf{r}_{ij}(t)\right] \tag{7.68}$$

$$\dot{\mathbf{r}}(t+\Delta t) = \dot{\mathbf{r}}(t) + \frac{\Delta t}{2m_i}\mathbf{f}_i(i) - \frac{\Delta t}{2m_i}\left[2\sum_j \lambda_{RRij}(t)\mathbf{r}_{ij}(t)\mathbf{f}_i(t+\Delta t) + 2\sum_j \lambda_{RVij}(t+\Delta t)\mathbf{r}_{ij}(t+\Delta t)\mathbf{f}_i\right] \tag{7.69}$$

In Eqs. (7.68) and (7.69), there are two separate Lagrange multipliers for the forces associated with the constraint. The $\lambda_{RRij}(t)$ values are chosen such that the constraint Eq. (7.37) is satisfied at time $t + \Delta t$. In contrast, the $\lambda_{RVij}(t + \Delta t)$ values are chosen to satisfy the time derivative of the constraint equations at time $t + \Delta t$.

$$\left(\dot{\mathbf{r}}_i(t) - \dot{\mathbf{r}}_j(t)\right)\left(\mathbf{r}_i(t) - \mathbf{r}_j(t)\right) = 0 \tag{7.70}$$

The Lagrange multipliers can be obtained by using the same iterative method as in the *shake* algorithm.

An example of the *rattle* algorithm is given by Algorithm 7.9. Before the *rattle* algorithm can be applied the positions, forces, and bond constraints must be known (Algorithm 7.9, Part 1). The *rattle* algorithm follows closely the *shake* algorithm, but unlike *shake*, there are two distinct parts. In the first part of *rattle* (Algorithm 7.9, Part 2), the positions are advanced and the velocities are advanced half way with applied constraints. The atomic positions are updated (Algorithm 7.9, Part 2.1) using the velocity-Verlet algorithm. The bond constraints are subsequently applied iteratively (Algorithm 7.9, Part 2.2). In common with the *shake* algorithm this involves evaluating the separation between bonded atoms (Algorithm 7.9, part 2.2.1)

2. It is often denoted in the literature as "RATTLE" but like *shake* it is not an acronym. This too is a historical legacy of previous FORTRAN programming style that is no longer necessary in modern implementations of the language.

and testing whether the atomic displacements satisfy the bond constraints. In the second part of *rattle* (Algorithm 7.9, Part 3), the velocity move is completed and the kinetic energy and the constraint contributions to the virial are calculated. This involves using the Verlet algorithm (Algorithm 7.9, Part 3.1) to update the atomic positions of each molecule and the iterative application of velocity constrains (Algorithm 7.9, Part 3.2). Applying the velocity constraints requires the calculation of the difference in both velocity and separation (Algorithm 7.9, part 3.2.1) between bonded atoms. The kinetic energy (Algorithm 7.9, Part 3.3) is evaluated at the end of *rattle*. An alternative to the *rattle* algorithm that incorporates the *shake* algorithm directly into the velocity-Verlet algorithm has been also reported (Palmer, 1993).

The utility of both the *shake* and *rattle* algorithms was witnessed an ongoing process of continuing improvements and adaptations. Versions specifically tailored to ridged water models have been reported (Miyamoto and Kollman, 1992) and angular constraints have been considered (Gonnet et al., 2009). Other improvements include improved efficiencies for small molecules (Kräutler et al., 2001); geometric generalizations (McLachlan et al., 2014); accounting for semi-rigid molecules (Forester and Smith, 1998; Gonnet, 2007), and molecules such as the alkanes (Bailey and Lowe, 2009); and dealing with internal constraints (Lee et al., 2005). Different ways of benefiting for parallelization have been investigated (Weinbach and Elber, 2005; Oh and Klein, 2006; Elber et al., 2011; Ruymgaart and Elber, 2012).

7.6.2.3 Application of Gauss's principle of least constraint

An alternative to the *shake* and *rattle* algorithms is to correct the trajectories resulting from an exact solution of Eq. (7.42). Edberg et al. (1986) observed that Gauss's principle of least constraint (Chapter 2) provides an exact prescription for deriving equations of motion involving holonomic constraint. The starting point for their procedure is to differentiate the bonding constraint (Eq. 7.37) with respect to time. For the simple case of a diatomic molecule, we find:

$$\mathbf{r}_{12}\cdot\ddot{\mathbf{r}}_{12} + (\dot{\mathbf{r}}_{12})^2 = 0 \tag{7.71}$$

The above equation defines a plane in either $\ddot{\mathbf{r}}_1$ or $\ddot{\mathbf{r}}_2$ space on which the constrained acceleration vectors must terminate. Applying Gauss's principle of least constraint, the equations of motion become:

$$\left.\begin{aligned} \dot{\mathbf{q}}_i &= \mathbf{p}_i \\ \dot{\mathbf{p}} &= \frac{\mathbf{f}_1}{m} - \lambda\mathbf{r}_{12} \\ \dot{\mathbf{p}} &= \frac{\mathbf{f}_2}{m} - \lambda\mathbf{r}_{12} \end{aligned}\right\} \tag{7.72}$$

ALGORITHM 7.9 ***Rattle*** **method for incorporating bond-length constraints.**

Overall Algorithm

```
Part 1    Calculate the positions and forces of all atoms.
          Specify bond constraints.
          tolerance ←10^-6
          virialCorrection ← 0
          kineticEnergy ←0
Part 2    Advance position and advance velocities half way:
          loop i ←1 ... nMolecule
Part 2.1     Update atomic positions for each molecule using
             the velocity-Verlet algorithm.
Part 2.2     Iterative application of constraints.
          end i loop
Part 3    Complete the advancement of velocities:
          if (error = false and done = true)
             loop i ←1 ... nMolecule
Part 3.1        Update atomic positions for each molecule.
Part 3.2        Iterative application of velocity constraints.
Part 3.3        Evaluate kinetic energy and update velocities.
             end i loop
             if (done = true)
                virialCorrection ← virialCorrection/(3Δt^2)
                kineticEnergy ←kineticEnergy/2
             end if
          end if
```

Sub Algorithms

```
Part 2.1 Update atomic positions for each molecule using
         the velocity-Verlet algorithm:
 loop atom ← ... nAtom
    rxOld_atom ← rx_i,atom
    ryOld_atom ← ry_i,atom
    rzOld_atom ← rz_i,atom
    rxNew_atom ← pbc (rxOld_i,atom + Δt × vx_i,atom + Δt^3 × fx_i,atom/(2 × mass_atom))
    ryNew_atom ← pbc (ryOld_i,atom + Δt × vy_i,atom + Δt^3 × fy_i,atom/(2 × mass_atom))
    rzNew_atom ← pbc (rzOld_i,atom + Δt × vz_i,atom + Δt^3 × fz_i,atom/(2 × mass_atom))
    vx_atom ← vx_i,atom + Δt × fx_i,atom/(2 × mass_atom)
    vy_atom ← vy_i,atom + Δt × fy_i,atom/(2 × mass_atom)
    vz_atom ← vz_i,atom + Δt × fz_i,atom/(2 × mass_atom)
    moving_atom ← false
    moved_tatom ← true
 end atom loop

Part 2.2 Iterative application of constraints:
 count ← 1
 done ← false
 error ← false
 while loop (done = false and count < maxCount)
    done ← true
    atom ← 1
    while (atom < nBonds and error = false)
       neighbor ← atom + 1
       if (neighbor > nAtom)
```

```
        neighbor ← 1
      end if
      Part 2.2.1 Evaluate separation between bonded molecules.
   end while loop
   if(error = false)
      loop atom ←1 ... nAtom
         moved_atom ← moving_atom
         moving_atom ← false
      end atom loop
      count ←count + 1
   end if
end while loop
if (error = false and done = true)
   loop atom ← 1, nAtom
      rx_i,atom ← rxNew_atom
      ry_i,atom ← ryNew_atom
      rz_i,atom ← rzNew_atom
      vx_i,atom ← vx_atom
      vy_i,atom ← vy_atom
      vz_i,atom ← vz_atom
   end atom loop
end if

Part 2.2.1 Evaluate separation between bonded molecules:
if (moved_atom or moved_neighbor)
   rxNewSeparation ← pbc(rxNew_atom - rxNew_neighbor)
   ryNewSeparation ← pbc(ryNew_atom - ryNew_neighbor)
   rzNewSeparation ← pbc(rzNew_atom - rzNew_neighbor)
end if
separationSq ←rxNewSeparation^2 + ryNewSeparation^2
                + rzNewSeparation^2
if (|bondSq_atom - separationSq| > 2 × tolerance × bondSq_atom)
   rxOldSeparation ←pbc(rxOld_atom - rxOld_neighbor)
   ryOldSeparation ←pbc(ryOld_atom - ryOld_neighbor)
   rzOldSeparation ←pbc(rzOld_atom - rzOld_neighbor)
   rNewOld → rxOldSeparation × rxNewSeparation
             + ryOldSeparation × ryNewSeparation
             + rzOldSeparation × rzNewSeparation
   if (rNewOld ≥ tolerance × BondSq_atom)
      λ ←|bondSq_atom - separationSq|/
            (2 × rNewOld × (1/mass_atom + 1/mass_neighbor))
      rxNew_atom ← pbc (rxNew_atom + λ × rxOldSeparation/mass_atom)
      ryNew_atom ← pbc (ryNew_atom + λ × ryOldSeparation/mass_atom)
      rzNew_atom ← pbc (rzNew_atom + λ × rzOldSeparation/mass_atom)
      rxNew_neighbor ← pbc
                       (rxNew_neighbor - λ × rxOldSeparation/mass_neighbor)
         ryNew_neighbor ← pbc
                       (ryNew_neighbor - λ × ryOldSeparation/mass_neighbor)
         rzNew_neighbor ← pbc
                       (rzNew_neighbor - λ × ryOldSeparation/mass_neighbor)
         vx_atom ← vx_atom +  λ × rxOldSeparation/mass_atom
         vy_atom ← vy_atom + λ × ryOldSeparation/mass_atom
         vz_atom ← vz_atom  + λ × rzOldSeparation/mass_atom
```

```
            vx_neighbor ← vx_neighbor - λ × rzOldSeparation/(Δt × mass_neighbor)
            vy_neighbor ← vy_neighbor - λ × ryOldSeparation/(Δt × mass_neighbor)
            vz_neighbor ← vz_neighbor - λ × rzOldSeparation/(Δt × mass_neighbor)
            moving_atom ← true
            moving_neighbor ← true
            done ←false
            atom ←atom + 1
        end if
    else
        error ← true
    end if
```

Part 3.1 Update atomic positions for each molecule (Verlet method):

```
loop atom ← 1 ... nAtom
    rx_atom ← rx_i,atom
    ry_atom ← ry_i,atom
    rz_atom ← rz_i,atom
    vx_atom ← vx_i,atom + Δt × fx_i,atom/(2 × mass_atom)
    vy_atom ← vy_i,atom + Δt × fy_i,atom/(2 × mass_atom)
    vz_atom ← vz_i,atom + Δt × fz_i,atom/(2 × mass_atom)
    moving_atom ← false
    moved_tatom ← true
end atom loop
```

Part 3.2 Iterative application of velocity constraints:

```
count ← 1
done ← false
error ← false
loop while (done = false and count < maxCount)
    done ← true
    atom ← 1
    loop while (atom < nBonds and error = false)
        neighbor ← atom + 1
        if (neighbor > nAtom)
            neighbor ← 1
        end if
        Part 3.2.1  Evaluate velocity difference/separation btw
         bonded atoms.
        atom ←atom + 1
    end while loop
    if (error = false)
        loop atom ←1 ... nAtom
            moved_atom ←moving_atom
            moving_atom ←false
        end atom loop
        count ←count + 1
    end if
end while loop
```

```
Part 3.2.1 Evaluate velocity difference/separation between bonded atoms:
 if (moved_atom or moved_neighbor)
    vxDiff ← vx_atom - vx_neighbor
    vyDiff ← vy_atom - vy_neighbor
    vyDiff ← vz_atom - vz_neighbor
    rxSeparation ← pbc(rx_atom - rx_neighbor)
    rySeparation ← pbc(ry_atom - ry_neighbor)
    rzSeparation ← pbc(rz_atom - rz_neighbor)
    sepVel ← rxSeparation × vxDiff + rySeparation × vyDiff
                + rzSeparation × vzDiff
    λ ← -sepVel/(bondSq_atom × (1/mass_atom + 1/mass_neighbor))
    if (|λ| > tolerance)
       virialCorrection ← virialCorrection + λ × bondSq_atom
       vx_atom ← vx_atom + λ × rxSeparation/mass_atom
       vy_atom ← vy_atom + λ × rySeparation/mass_atom
       vz_atom ← vz_atom + λ × rzSeparation/mass_atom
       vx_neighbor ← vx_neighbor + λ × rxSeparation/mass_neighbor
       vy_neighbor ← vy_neighbor + λ × rySeparation/mass_neighbor
       vz_neighbor ← vz_neighbor + λ × rzSeparation/mass_neighbor
       moving_atom ← true
       moving_neighbor ← true
       done ←false
    else
       error ← true
    end if
 end if

Part 3.3 Evaluate kinetic energy and update velocities:
 if (done = true)
    loop atom ← 1, nAtom
       vx_i,atom ← vx_atom
       vy_i,atom ← vy_atom
       vz_i,atom ← vz_atom
       kineticEnergy ←kineticEnergy + mass_atom ×(vx^2 + vy^2 + vz^2)
    end atom loop
 end if
```

If the molecule has more than two atoms, the site equations of motion are the same as Eq. (7.72) except that each site is subject to coupled constraint forces. It is apparent that a set of multipliers λ_{ij} must also be calculated to solve these equations.

Edberg et al. (1986) proposed the use of penalty functions to keep the solution on track. If Eq. (7.42) is solved exactly, at some stage the predicted positions and velocities will be a significant distance from the constraint surface. When this occurs, the trajectory can be relaxed back on the constraint surface by minimizing the following penalty functions.

$$\Phi_\alpha = (\sigma_\alpha(\mathbf{r}))^2 \tag{7.73}$$

$$\Psi_\alpha = (\dot{\sigma}_\alpha(\mathbf{r}, \dot{\mathbf{r}}))^2 \tag{7.74}$$

Overall values of Φ and Ψ are obtained for each molecule by summing the contributions from the individual constraints. When Φ and Ψ reach a predetermined value, a nonlinear minimization is performed to find the $6N$ values of $\mathbf{r}_i$ and $\dot{\mathbf{r}}_i$ at the minimum. Edberg et al. (1986) found that a simulation involving sixty-four butane molecules modeled as united atoms required a mean minimization period of eighty-six integration steps.

7.6.2.4 Matrix inversion method

de Leeuw et al. (1990) proposed a method of correcting trajectories using matrix inversion. Solving Eq. (7.42) exactly normally results in a set of coordinates $\mathbf{r}_0$, which have drifted from the true constrained values. If the constraints were satisfied at a particular time step, the drift in coordinates caused by subsequently solving Eq. (7.42) can be represented by,

$$\sigma_\alpha(\mathbf{r}_o) = 0(\Delta t^k) \tag{7.75}$$

where k is the order of the differential solver. The true coordinates can be obtained by applying the following correction.

$$\mathbf{r}_i = \mathbf{r}_{i0} + \Delta t^k \sum_{\alpha=1}^{M} \varepsilon_\alpha \left(\frac{\partial \sigma_\alpha(\mathbf{r})}{\partial \mathbf{r}_i} \right) \tag{7.76}$$

The correction parameters are obtained by substituting these coordinates into the constraint equations (Eq. 7.37). The constraints can be obtained to $0(\Delta t^{2k})$ by using a Taylor expansion of $\sigma(\mathbf{r})$ and $\partial\sigma_\alpha/\partial\mathbf{r}_i$ about $\mathbf{r}_0$.

$$\sigma_\alpha(\mathbf{r}) = \sigma_\alpha(\mathbf{r}_0) + \Delta t^k \sum_{\beta=1}^{M} \sum_{i=1}^{N} \varepsilon_\beta \left(\frac{\partial \sigma_\alpha(\mathbf{r}_0)}{\partial \mathbf{r}_i} \right) \left(\frac{\partial \sigma_\beta(\mathbf{r}_0)}{\partial \mathbf{r}_i} \right) \tag{7.77}$$

A set of M equations is obtained for the correction coefficients because the higher order terms in Eq. (7.77) are negligible and they can be neglected,

$$\varepsilon = -\Delta t^{-k} (\mathbf{M}_0)^{-1} \sigma \tag{7.78}$$

where:

$$(\mathbf{M}_0)_{\alpha\beta} = \sum_{i=1}^{N} \left(\frac{\partial \sigma_\alpha(\mathbf{r}_0)}{\partial \mathbf{r}_i} \right) \left(\frac{\partial \sigma_\beta(\mathbf{r}_0)}{\partial \mathbf{r}_i} \right) \tag{7.79}$$

$$(\sigma)_\alpha = \sigma_\alpha(\mathbf{r}_0) \tag{7.80}$$

$$(\varepsilon)_\alpha = \varepsilon_\alpha \tag{7.81}$$

The ε_α values are those used in Eq. (7.76) to convert the atomic positions to $0(\Delta t^{2k})$. The following approximation is necessary to use Eq. (7.76):

$$\left(\frac{\partial \sigma_\alpha(\mathbf{r})}{\partial \mathbf{r}_i}\right) = \left(\frac{\partial \sigma_\alpha(\mathbf{r}_0)}{\partial \mathbf{r}_i}\right) \tag{7.82}$$

7.6.2.5 Other methods

Yoneya et al. (1994) proposed a self-correcting noniterative constraint procedure. The starting point for their method is a Taylor expansion of the constraint condition.

$$\sigma_n(t+\Delta t) + O(\Delta t^3) = \sigma_n(t) + \Delta t \dot{\sigma}_n(t) + \frac{\Delta t^2 \ddot{\sigma}_n(t)}{2} \tag{7.83}$$

If the Verlet algorithm is applied to the constraint conditions, we obtain:

$$\begin{aligned}\sigma_n(t+\Delta t) = (2\mathbf{r}_n(t) - \mathbf{r}_n(t-\Delta t))^2 - d_n^2 + 2\Delta t^2(2\mathbf{r}_n(t) - \mathbf{r}_n(t-\Delta t))\ddot{\mathbf{r}}_n(t) \\ + O(\Delta t^4) = 0\end{aligned} \tag{7.84}$$

In deriving the above equation, terms nonlinear in λ were neglected. A set of M linear equations for $\lambda(t)$ is obtained by substituting the expression for $\ddot{\mathbf{r}}$ obtained from the equations of motion, into Eq. (7.84).

Slusher and Cummings (1996) proposed a velocity-Verlet implementation of the algorithm of Yoneya et al. (1994).

$$\begin{aligned}\sigma(t+\Delta t) = \mathbf{r}_n^2(t) - d_n^2 + 2\Delta t \mathbf{r}_n(t)\dot{\mathbf{r}}_n(t-\Delta t) + \Delta t^2 \mathbf{r}_n^2(t-h) \\ + \Delta t^2(\mathbf{r}_n(t) + \Delta t \dot{\mathbf{r}}_n(t-\Delta t))(2\ddot{\mathbf{r}}_n(t) + \ddot{\mathbf{r}}_n(t-\Delta t)) \\ + O(\Delta t^4) \\ = 0\end{aligned} \tag{7.85}$$

Substituting the constrained equations of motion for in the above equation yields a rank M matrix. Equation (7.85) can be used immediately after the position update and force evaluation by the velocity-Verlet algorithm. Consequently, the same force constraints are used in both the velocity and position updates.

Another alternative to the *shake* algorithm is the successive overrelaxation (SOR) method (Barth et al., 1995). The SOR method has been implemented using the CHARMM potential (Chapter 3) and a twofold improvement in execution speed over the *shake* algorithm has been reported (Barth et al., 1995).

7.7 Summary

MD simulation requires a suitable algorithm to integrate the equations of motion. The choice of integrators is constrained by the necessity of limiting the costly evaluation of forces to once per time step. For atoms, there are a variety of widely used integrator algorithms. The origin of many integrator algorithms used in MD can be traced to either the Gear predictor–corrector

or Verlet predictor algorithms. These algorithms only require one force evaluation per time step. A general fourth-order Runge−Kutta integrator is useful for accurate evaluations but is computationally expensive because it requires multiple force evaluations per time step. Solving the equations of motion for molecules is considerably more challenging than the atomic case because of the restrictions imposed by bonding constrains. The *shake* and *rattle* algorithms are possibly the most commonly used algorithms for molecules. The application of MD in the context of ensembles is discussed in Chapter 9. In Chapter 11, MD code using the Gear algorithm is provided.

References

Abramowitz, M., Stegun, A. (Eds.), 1972. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, 9th Printing. Dover Publications, p. 897.

Ahmed, A., Sadus, R.J., 2010. Effect of potential truncations and shifts on the solid-liquid phase coexistence of Lennard-Jones fluids. J. Chem. Phys. 133, 124515.

Ahmed, A., Mausbach, P., Sadus, R.J., 2010. Pressure and energy behavior of the Gaussian core model fluid under shear. Phys. Rev. E 82, 011201.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Andersen, H.C., 1983. Rattle: a "velocity" version of the shake algorithm for molecular dynamics calculations. J. Comput. Phys. 52, 24−34.

Bailey, A.G., Lowe, C.P., 2009. MILCH SHAKE: An efficient method for constraint dynamics applied to alkanes. J. Comput. Chem. 30, 2485−2493.

Barojas, J., Levesque, D., Quentrec, B., 1973. Simulation of diatomic homonuclear liquids. Phys. Rev. A7, 1092−1105.

Barth, E., Kuczera, K., Leimkuhler, B., Skeel, R.D., 1995. Algorithms for constrained molecular dynamics. J. Comput. Chem. 16, 1192−1209.

Beeman, D., 1976. Some multistep methods for use in molecular dynamics calculations. J. Comput. Phys. 20, 130−139.

Berendsen, H.J. C. and van Gunsteren, W.F. in Ciccotti, G. and Hoover, W. G. (Eds), Molecular-Dynamics Simulation of Statistical-Mechanical Systems, North-Holland, Amsterdam, 1986.

Brown, D., 1997. The force of constraint in predictor-corrector algorithms for shake constraint dynamics. Mol. Sim. 18, 339−348.

Ciccotti, G., Ryckaert, J.P., 1986. Molecular dynamics simulation of rigid molecules. Comp. Phys. Rep. 4, 345−392.

Dahlquist, G., Björk, A., 1974. Numerical Methods. Prentice-Hall, Englewood Cliffs, NJ.

de Leeuw, S.W., Perram, J.W., Petersen, H.G., 1990. Hamilton's equations for constrained dynamical systems. J. Stat. Phys. 61, 1203−1221.

Edberg, R., Evans, D.J., Morriss, G.P., 1986. Constrained molecular dynamics: simulations of liquid alkanes with a new algorithm. J. Chem. Phys. 84, 6933−6939.

Elber, R., Ruymgart, A.P., Hess, B., 2011. SHAKE parallelization. Eur. Phys. J. Spec. Top. 200, 211−223.

Evans, D.J., 1977. On the representation of orientation space. Mol. Phys. 34, 317−325.

Forester, T.R., Smith, W., 1998. SHAKE, Rattle, and roll: efficient constraint algorithms for linked rigid bodies. J. Comput. Chem. 19, 102−111.

Frenkel, D., Smit, B., 2023. Understanding Molecular Simulation. From Algorithms to Applications, third ed. Academic Press, San Diego.

Gear, C.W. (1966). The numerical integration of ordinary differential equations of various orders. Report ANL 7126, Argone National Laboratory (Available at: https://www.osti.gov/biblio/4534813-numerical-integration-ordinary-differential-equations-various-orders).

Gear, C.W., 1971. Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall, Englewood Cliffs, NJ.

Goldstein, H., Poole, C., Safko, J., 2008. Classical Mechanics, third ed. Addison Wesley, San Francisco.

Gonnet, P., 2007. P-SHAKE: A quadratically convergent SHAKE in O (n^2). J. Comput. Phys. 220, 740–750.

Gonnet, P., Walther, J.H., Koumoutsakos, P., 2009. θ-SHAKE: An extension to SHAKE for the explicit treatment of angular constraints. Comp. Phys. Commun. 180, 360–364.

Gould, H., Tobochnik, 1996. An Introduction to Computer Simulation Methods: Application to Physical Systems, second ed. Addison-Wesley, Reading.

Greer, J.C., 1994. An approximate time evolution operator to generate the Verlet algorithm. J. Comput. Phys. 115, 245–247.

Gubbins, K.E., Quirke, N. (Eds.), 1996. Molecular Simulation and Industrial Applications. Methods, Examples and Prospects. Gordon and Breach, Amsterdam.

Haile, J.M., 1992. Molecular Dynamics Simulation. Elementary Methods. John Wiley & Sons, New York.

Heermann, D.W., 1990. Computer Simulation Methods in Theoretical Physics, second ed. Springer-Verlag, Heidelberg.

Hockney, R.W., 1970. The potential calculation and some applications. Methods Comput. Phys. 9, 136–211.

Hockney, R.W., Eastwood, J.W., 1988. Computer Simulation Using Particles. Adam Hilger, Bristol.

Isbister, D.J., Searles, D.J., Evans, D.J., 1997. Symplectic properties of algorithms and simulation methods. Phys. A 240, 105–114.

Janek, J., Kolafa, J., 2020. Novel Gear-like predictor-corrector integration methods for molecular dynamics. Mol. Phys. 118, 2–10.

Janežič, D., Orel, F., 1993. Implicit Runge-Kutta method for molecular dynamics integration. J. Chem. Inf. Comput. Sci. 33, 252–257.

Janežič, D., Merzel, F., 1995. An efficient symplectic integration algorithm for molecular dynamics simulations. J. Chem. Inf. Comput. Sci. 35, 321–326.

Jay, L., 1996. Symplectic partitioned Runge-Kutta methods for constrained Hamitonian systems. SIAM J. Numer. Anal. 33, 368–387.

Kol, A., Laird, B.B., Leimkuhler, B.J., 1997. A symplectic method for rigid-body molecular simulation. J. Chem. Phys. 107, 2580–2588.

Kolafa, J., 1996. Numerical integration of equations of motion with self-consistent field given by an implicit equation. Mol. Sim. 18, 193–212.

Kolafa, J., 2005. Gear formalism of the always stable predictor-corrector method for molecular dynamics of polarizable molecules. J. Chem. Phys. 122, 164105.

Kräutler, V., van Gunsteren, W.F., Hünenberger, P.H., 2001. A fast SHAKE algorithm to solve distance constraint equations for small molecules in molecular dynamics simulations. J. Comput. Chem. 22, 501–508.

Lee, S.-H., Palmo, K., Krimm, S., 2005. WIGGLE: A new constrained molecular dynamics algorithm in Cartesian coordinates. J. Comput. Phys. 210, 171–182.

Leimkuhler, B.J., Skeel, R.D., 1994. Symplectic numerical integrators in constrained Hamiltonian systems. J. Comput. Phys. 112, 117−125.

Lippert, R.A., Bowers, K.J., Dror, R.O., Eastwood, M.P., Gregersen, B.A., Klepeis, J.L., Kolossvary, I., Shaw, D.E., 2007. A common, avoidable source of error in molecular dynamics integrators. J. Chem. Phys. 126, 046101.

Litniewski, M., 2003. On molecular dynamics algorithms. Mol. Sim. 29, 223−229.

Losey, J., Sadus, R.J., 2019. Thermodynamic properties and anomalous behavior of double-Gaussian core model potentials. Phys. Rev. E 100, 0132112.

McLachlan, R.I., Modin, K., Verdier, O., Wilkins, M., 2014. Geometric generalisations of SHAKE and RATTLE. Found. Comput. Math. 14, 339−370.

Miyamoto, S., Kollman, P.A., 1992. SETTLE − An analytical version of the SHAKE and RATTLE algorithms for rigid water models. J. Compt. Chem. 13, 952−962.

Oh, K.J., Klein, M.L., 2006. A parallel molecular dynamics simulation scheme for a molecular system with bond constraints in NPT ensemble. Comp. Phys. Commun. 174, 263−269.

Okabe, T., Yamada, H., Goda, M., 1996. How is symplectic integrator applicable to molecular dynamics. Int. J. Mod. Phys. C. 7, 613−633.

Okunbor, D.I., Skeel, R.D., 1994. Canonical numerical methods for molecular dynamics simulations. J. Comput. Chem. 15, 72−79.

Palmer, B.J., 1993. Direct application of SHAKE to the velocity Verlet algorithm. J. Comput. Phys. 104, 470−472.

Rapaport, D.C., 2004. The Art of Molecular Dynamics Simulation, second ed. Cambridge University Press, Cambridge.

Ratanapisit, J., Isbister, D.J., Ely, J.F., 2001. Transport properties of fluids: Symplectic integrators and their usefulness. Fluid Phase Equilib. 183−184, 351−361.

Refson, K., Pawley, G.S., 1987a. Molecular dynamics studies of the condensed phases of n-butane and their transitions. I. Techniques and model results. Mol. Phys. 61, 669−692.

Refson, K., Pawley, G.S., 1987b. Molecular dynamics studies of the condensed phases of n-butane and their transitions. II. The transition to the true plastic phase. Mol. Phys. 61, 693−709.

Ruymgaart, A.P., Elber, R., 2012. Revisiting molecular dynamics on a CPU/GPU system: water kernel and SHAKE parallelization. J. Chem. Theory Comput. 8, 4624−4636.

Ryckaert, J.P., Ciccotti, G., Berendsen, H.J.C., 1977. Numerical integration of the Cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. J. Comput. Phys. 23, 327−341.

Sangster, M.J.L., Dixon, M., 1976. Interionic potentials in alkali halides and their use in simulation of molten salts. Adv. Phys. 25, 247−342.

Skeel, R.D., Zhang, G., Schlick, T., 1997. A family of symplectic integrators: stability, accuracy, and molecular dynamics applications. SIAM J. Sci. Comput. 18, 203−222.

Slusher, J.T., Cummings, P.T., 1996. Non-iterative constraint dynamics using velocity-explicit Verlet methods. Mol. Sim. 18, 213−224.

Swope, W.C., Andersen, H.C., Berens, P.H., Wilson, K.R., 1982. A computer simulation method for the calculation of equilibrium constants for the formulation of physical clusters of molecules: application to small water clusters. J. Chem. Phys. 76, 637−649.

Tildesley, D.J., 1993. In: Allen, M.P., Tildesley, D.J. (Eds.), Computer Simulation in Chemical Physics. Dordecht, Kluwer.

Todd, B.D., 1997. Application of transient-time correlation functions to nonequilibrium molecular-dynamics simulations of elongational flow. Phys. Rev. E 56, 6723−6728.

Trembiay, J.C., Carrington Jr, T., 2004. Using preconditioned adaptive step size Runge-Kutta methods for solving the time-dependent Schrödinger equation. J. Chem. Phys. 121, 11535–11541.

Tupper, P.F., 2005. A test problem for molecular dynamics integrators. IMA J. Numer. Anal. 25, 286–309.

van Gunsteren, W.F., Berendsen, H.J.C., 1977. Algorithms for macromolecular dynamics and constraint dynamics. Mol. Phys. 34, 1311–1327.

Verlet, L., 1967. Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. Phys. Rev. 159, 98–103.

Weinbach, Y., Elber, R., 2005. Revisiting and parallelizing SHAKE. J. Comput. Phys. 209, 193–206.

Yoneya, M., Berendsen, H.J.C., Hirasawa, K., 1994. A non-iterative matrix method for constraint molecular dynamics simulations. Mol. Sim. 13, 395–405.

Yoshida, H., 1990. Construction of higher order symplectic integrators. Phys. Lett. A 150, 262–268.

This page intentionally left blank

Chapter 8

Nonequilibrium molecular dynamics

In addition to equilibrium properties, molecular dynamics (MD) can be applied to nonequilibrium systems. Nonequilibrium molecular dynamics (NEMD) algorithms fall into two broad categories, namely, inhomogeneous and homogeneous methods. The distinction between the two categories is based on the nature of the boundary conditions required. Inhomogeneous methods involve nonperiodic boundary conditions whereas homogeneous methods employ some sort of periodic boundaries.

Ashurst and Hoover (1973) employed an inhomogeneous NEMD approach to simulate steady flow. They used a pair of nonperiodic boundary layers consisting of a small number of particles with the same spatial correlations as the fluid in the central cell. Shear flow was induced by moving the boundary layers in opposite directions along parallel planes. A secondary role of the boundary layers was to act as thermostats to extract the heat generated by the viscous flow. Several variants of this general approach have been developed (Tenenbaum et al., 1982; Trozzi and Ciccotti, 1984).

Although inhomogeneous methods are useful in some important applications (Wold and Hafskjold, 1999; Todd and Daivis, 2017), we will focus exclusively on homogeneous methods. In general, homogeneous methods are preferred because replacing physical walls by periodic boundaries means that all atoms experience the same environment. In contrast, the properties of inhomogeneous systems are perturbed artificially by wall−fluid interactions. A disadvantage of homogeneous methods is that the equations of motions of the molecules must be altered artificially.

The earliest application of homogeneous NEMD techniques also involved shear flow. Lees and Edwards (1972) used a moving periodic boundary, whereas Gosling et al. (1973) induced shear flow by applying a sinusoidal transverse force field to the particles. A limitation of these early methods was that the strong fields used resulted in substantial heating, which required the introduction of a thermostat. This problem can be circumvented by using a subtraction of trajectories strategy (Ciccotti et al., 1979).

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00006-X

The use of a thermostat can be avoided by making only small perturbations but the nature of the perturbation must be defined for the particular problem being studied. This problem has been addressed by the use of the *dolls* tensor (Hoover et al., 1980) and the *sllod*[1] tensor (Ladd, 1984; Evans and Morriss, 1984a).

If large perturbations are used, further devices must be used to prevent the system overheating. One approach is to employ Nosé−Hoover dynamics (Hoover, 1985). Evans et al. (1983) performed NEMD by employing Gauss's principle of least constraint (Chapter 2). The general approach is to use a driving force to maintain a thermodynamic flux while using constraint forces to extract the heat.

This chapter provides a brief overview of NEMD with emphasis on synthetic NEMD and its applications. The main use for NEMD is the calculation of transport properties (Sadus, 2006). Cummings and Evans (1992) have reviewed the use of NEMD for transport properties. Evans and Morriss (2008) have discussed the statistical mechanical basis of nonequilibrium liquids. The theoretical basis and historical development of NEMD have been discussed elsewhere (Evans and Morriss, 1984b; Evans, 1986a,b; Ciccotti, 1991; Hoover 1991; Rapaport, 2004; Allen and Tildesley, 2017). A notable advance has been the development of NEMD algorithms for elongational flow (Todd and Daivis, 1998; Hunt and Todd, 2003; Hunt et al., 2010). The practical implementation of many new and promising NEMD algorithms has been very extensively detailed by Todd and Daivis (2017).

Transport properties can be also be calculated from equilibrium molecular dynamics (EMD) trajectories using either Einstein or Green−Kubo relationships. An example of the EMD to the calculation of shear viscosity is given by Schoen and Hoheisel (1985). An EMD algorithm (Heffelfinger and van Swol, 1994) has been used to investigate diffusion involving Lennard-Jones (LJ) fluids (Heffelfinger and Ford, 1998) and polymers (Ford and Heffelfinger, 1998).

8.1 An example NEMD algorithm

In Chapter 1, a general EMD algorithm was outlined (Algorithm 1.1). Before considering the theoretical aspects of NEMD, it is useful to consider the overall structure of an NEMD algorithm. This is illustrated in Algorithm 8.1 for the special case of the Gear predictor−corrector integrator at a constant strain rate ($\dot{\gamma}$). The general procedure used in Algorithm 8.1 can be applied to other integrators.

1. In the literature *dolls* and *sllod* are commonly denoted as "DOLLS" and "SLLOD," respectively. The use of capital letters is misleading as neither of the terms corresponds to any meaningful acronym. Indeed, *sllod* is simply the word *dolls* spelt backward.

In common with an EMD algorithm, NEMD Algorithm 8.1 begins with an initial configuration (Part 1) that is evolved over time (Part 2), resulting in the accumulation of time-averaged quantities (Part 2.3). At each time step, the total strain (γ) is determined by the applied strain rate ($\dot{\gamma}$). This is reset periodically following the traversal of the length (L) of the simulation box. In common with EMD, the heart of the algorithm is solving the equations of motion (Part 2.2). In the NEMD case, the Newtonian equations of motion are modified in the integrator step. The Gear predictor (Part 2.2.1) is invoked to provide new predictions of the positions and momenta. No NEMD-specific modifications are introduced into predictor part of the simulation. Following the predictor step, the coordinates in the x direction are adjusted to account for bottom/top movements out of the box (Eqs. (8.12) and (8.13)) and periodic boundary conditions are applied in all directions to account for left/right crossings (see Fig. 8.1). The use of different boundary conditions is an important departure from EMD simulations.

The position coordinates are used to determine the forces according to Newton's law (Part 2.2.3). The intermolecular contributions to the pressure tensor are calculated as part of the force evaluations. The integrator step concludes with the corrector step (Part 2.2.4), which both makes use of the newly calculated forces and applies the appropriate NEMD modifications, for example, *sllod*. That is, the NEMD-specific modifications are applied in the integrator step and not in the force evaluations, which are evaluated conventionally. This means that much of the code developed for EMD can be reused in NEMD. The kinetic contributions to the pressure tensor are

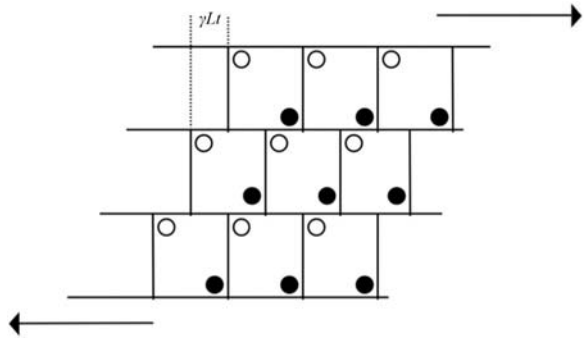


FIGURE 8.1 Illustration of the sliding brick boundary conditions for shear viscosity simulations. The central cell is replicated infinitely in the x, y, and z directions. The boxes above the central cell move to the right whereas the boxes below move to the left. The velocities of this movement are determined by the strain rate $\dot{\gamma}$.

calculated as either part of the corrector step (not shown here) or immediately afterwards.

ALGORITHM 8.1 General outline of a NEMD process at constant strain rate ($\dot{\gamma}$) using the Gear predictor–corrector integrator.

```
Part 1    Create an initial configuration (γ = 0, specify a value of γ̇).
Part 2    Simulation for a duration tMax:
          loop
Part 2.1    γ ← γ + γ̇ × Δt   //strain at each time step
            γ ← γ - integer value of γ  //reset to 0 after traversal of one L
Part 2.2    Solve modified equation of motions:
Part 2.2.1    Gear predictor.
Part 2.2.2    loop i ← 1 ... nAtom
                rxi ← rxi - γ × nearestInt(yi/L) × L
                pbc(rxi)
                pbc(ryi)
                pbc(rzi)
              end i loop
Part 2.2.3    Calculate forces according to Newton's law.
                Determine intermolecular contributions to pressure
                tensor as part of the force evaluation.
Part 2.2.4    Gear corrector.
                Determine kinetic contribution to the pressure in the
                corrector and add intermolecular contributions
Part 2.3    Calculate quantities to be time-averaged.
          while (time ≤ tMax)
```

8.2 Synthetic NEMD algorithms

The general approach (Evans and Morriss, 2008) adopted by widely used NEMD methods is to introduce an applied field (F_e) into the equations of motion of the system. The effect of F_e is to drive the conjugate thermodynamic flux (J). Examples of fluxes are the momentum flux in planar Couette flow or the heat current associated with thermal conductivity. The nature of F_e is subject to two requirements: it must be consistent with the periodic boundary condition; and any transport property (L) must be calculated from the relationship:

$$L = \lim_{F_e \to 0} \lim_{t \to \infty} \frac{J}{F_e} \tag{8.1}$$

Using an applied field generates a synthetic NEMD algorithm because the perturbation does not exist in nature. Linear response theory (Evans, 1986a,b) provides the formal proof that an algorithm satisfies these two criteria. Knowledge of the Green–Kubo (Green, 1951; Kubo, 1957) relationship

leads directly (Evans, 1986a,b) to a corresponding NEMD algorithm. The theoretical basis of many NEMD algorithms is underpinned by linear and nonlinear response theory (Evans and Morriss, 1984b, 2008; Morriss and Evans, 1985).

A Naiver−Stokes transport coefficient (L) is related to the flux (J) by the simple relationship,

$$J = LX \tag{8.2}$$

where X is a gradient in the density of the quantity that is conserved. In turn, L is related to equilibrium fluctuations by Green−Kubo relations. In general, a synthetic NEMD algorithm can be generated by using the following procedure (Evans and Morriss, 2008). A Green−Kubo relationship is identified for the transport coefficient (L_{ij}).

$$L_{ij} = \int_0^\infty \langle J_j(0) J_i(0) \rangle dt \tag{8.3}$$

The form of Eq. (8.3) identifies the flux (J). Next, a fictitious field (F_e) is invented and it is coupled to the system such that the adiabatic energy derivative ($\dot{H}_0^{ad}$) can be related directly to the dissipative flux (J_j):

$$\dot{H}_0^{ad} = -J_j F_e \tag{8.4}$$

A check is made to ensure that the adiabatic incompressibility of phase space (Evans and Morriss, 2008) is satisfied and that the equations of motion are both homogeneous and consistent with periodic boundary conditions. A thermostat is applied and F_e is coupled to the system either isothermally or isoenergetically. The steady-state average ($< J_i(t) >$) is subsequently calculated as a function of the external field (F_e) and applying Eq. (8.1), the transport coefficient is obtained from:

$$L_i = \lim_{F_e \to 0} \lim_{t \to \infty} \frac{\langle J_i(t) \rangle}{F_e} \tag{8.5}$$

The application of this procedure to the calculation of various transport coefficients is examined subsequently.

8.2.1 Shear viscosity

The Green−Kubo relationship for shear viscosity (η) is,

$$\eta = \frac{V}{kT} \int_0^\infty \langle P_{xy}(0) P_{xy}(t) \rangle dt \tag{8.6}$$

where P_{xy} is the xy component of the pressure tensor **(P)** (Evans, 1979a).

$$\mathbf{P}V = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{m_i} - \frac{1}{2}\sum_{i,j}^{N} \mathbf{r}_{ij}\mathbf{F}_{ij} \tag{8.7}$$

The procedure for calculating the pressure tensor is outlined in Algorithm 8.2, which is quite general in nature and can be used for either EMD or NEMD simulations. It is apparent from Eq. (8.7) that this involves contributions from both intermolecular and kinetic interactions. The intermolecular contributions to the pressure tensor are calculated as part of the overall evaluation of forces. In the context of predictor−corrector algorithms, this is performed immediately after the predictor step. The components are initialized (Part 1) prior to the commencement of the double loop (Part 2) that evaluates the forces between all distinct pairs of atoms. This double loop is common to all molecular simulation algorithms. Part 2.1 involves the evaluation of the x, y, and z components, which enables the calculation of the forces.

Different periodic boundaries are required for EMD and NEMD (see Algorithm 8.1, Part 2.2.2) simulations, but otherwise the force calculation for both cases is identical. These forces are used (Part 2.2.2) to evaluate the intermolecular contributions of the pressure tensor. This part of the algorithm concludes (Part 2.3) with the accumulation of the forces, which will be required in the corrector step (Part 3). The corrector step is required before the calculation of the kinetic contribution to the pressure tensors (Part 4) because it requires the updated values of velocities (Part 4). The intermolecular contributions are added to the kinetic contribution rather than being subtracted as indicated by Eq. (8.7) because the evaluation of the pair forces (Part 2.1) involved a negative sign. The tensor components are divided by the volume (Part 5) to obtain the dimensions of pressure, which can then be evaluated. It should be noted that for most commonly used intermolecular potentials, the evaluation of forces will yield $p_{xy} = p_{yx}$. When a strain rate is applied, all the off-diagonal elements of the pressure tensor will be nonzero.

Algorithm 8.2 is completely general. To use it for properties under shear requires both suitable periodic boundary conditions (e.g., Lees−Edwards) and appropriate modifications in the corrector step (e.g., *sllod*). When using Algorithm 8.2, the normal pressure, p, can be obtained as one-third of the trace of the pressure tensor (**P**), that is, $p = 1/3\mathrm{Tr}(\mathbf{P}) = (p_{xx} + p_{yy} + p_{zz})/3$.

ALGORITHM 8.2 Evaluation of the tensor components to pressure.

Part 1 Initialize pressure tensor components in all dimensions:
$p_{xx} = p_{yy} = p_{zz} = p_{xy} = p_{xz} \ \ldots = 0$
Initialize forces for all atoms: $fx_i = fy_i = fz_i = 0.$

Part 2 Evaluate forces on all distinct pairs of atoms:
loop $i \leftarrow 1 \ldots nAtom$
  loop $j \leftarrow i + 1 \ldots nAtom - 1$

Part 2.1 Determine pair separations ($rijx_i$, $rijy_i$ and $rijy_i$).
Apply periodic boundary conditions.
Determine forces between pairs ($fijx_i$, $fijy_i$, $fijz_i$).

Part 2.2 Calculate pressure tensor components:

$$p_{xx} \leftarrow p_{xx} + rijx_i \times fijx_i$$
$$p_{xy} \leftarrow p_{xy} + rijx_i \times fijy_i$$
$$p_{xz} \leftarrow p_{xz} + rijx_i \times fijz_i$$
$$p_{yx} \leftarrow p_{yx} + rijy_i \times fijx_i$$
$$p_{yy} \leftarrow p_{yy} + rijy_i \times fijy_i$$
$$p_{yz} \leftarrow p_{yz} + rijyi \times fijz_i$$
$$p_{zx} \leftarrow p_{zx} + rijz_i \times fijx_i$$
$$p_{zy} \leftarrow p_{zy} + rijz_i \times fijy_i$$
$$p_{zz} \leftarrow p_{zz} + rijz_i \times fijz_i$$

  end j **loop**

Part 2.3 Accumulate total forces and energies etc:

$$fx_i \leftarrow fx_i + fijx_i$$
$$fy_i \leftarrow fy_i + fijy_i$$
$$fz_i \leftarrow fz_i + fijz_i$$

end i **loop**

Part 3 Obtained corrected velocities.

Part 4 Add kinetic contributions from corrected velocities:
loop $i \leftarrow 1 \ldots nAtom$

$$p_{xx} \leftarrow p_{xx} + m_i \times vx_i \times vx_i$$
$$p_{xy} \leftarrow p_{xy} + m_i \times vx_i \times vy_i$$
$$p_{xz} \leftarrow p_{xz} + m_i \times vxi \times vz_i$$
$$p_{yx} \leftarrow p_{yx} + m_i \times vy_i \times vx_i$$
$$p_{yy} \leftarrow p_{yy} + m_i \times vy_i \times vy_i$$
$$p_{yz} \leftarrow p_{yz} + m_i \times vy_i \times vz_i$$
$$p_{zx} \leftarrow p_{zx} + m_i \times vz_i \times vx_i$$
$$p_{zy} \leftarrow p_{zy} + m_i \times vz_i \times vy_i$$
$$p_{zz} \leftarrow p_{zz} + m_i \times vzi \times vz_i$$

end i **loop**

Part 5 Divide by volume to determine the pressure:

$$p_{xx} \leftarrow p_{xx}/V$$
$$p_{xy} \leftarrow p_{xy}/V$$
$$p_{xz} \leftarrow p_{xz}/V$$
$$p_{yx} \leftarrow p_{yx}/V$$
$$p_{yy} \leftarrow p_{yy}/V$$
$$p_{yz} \leftarrow p_{yz}/V$$
$$p_{zx} \leftarrow p_{zx}/V$$
$$p_{zy} \leftarrow p_{zy}/V$$
$$p_{zz} \leftarrow p_{zz}/V$$

Viscosity has been studied widely by molecular simulation (Ashurst and Hoover, 1973; Gosling et al., 1973; Ciccotti et al., 1979; Hoover et al., 1980; Singer et al., 1980; Ladd, 1984; Evans and Morriss, 1984b). For shear viscosity, the most efficient NEMD algorithms are the *dolls* tensor method (Hoover et al., 1980) and the *sllod* algorithm (Ladd, 1984; Evans and Morriss, 1984b). It has been shown (Evans and Morriss, 2008) that the *sllod* algorithm is exact for arbitrarily large strain rates and it can be applied to nonlinear, non-Newtonian systems. Consequently, we will focus exclusively on the *sllod* algorithm.

The isokinetic *sllod* equations of motion are,

$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i} + (\mathbf{r}_i \cdot \nabla \mathbf{u}) \tag{8.8}$$

$$\dot{\mathbf{p}}_i = \mathbf{F}_i - \left(\mathbf{p}_i \cdot \nabla \mathbf{u}\right) - \alpha \mathbf{p}_i \tag{8.9}$$

where $\mathbf{u} = (u_x, 0, 0)$ and $u_x = \dot{\gamma} y$ is the velocity field corresponding to planar Couette flow with a strain rate denoted as[2] $\dot{\gamma}$. In Eq. (8.9), α is a Gaussian thermostating constant is given by:

$$\alpha = \frac{\sum_{i=1}^{N} \mathbf{F}_i \cdot \mathbf{p}_i - \dot{\gamma} p_{xi} p_{yi}}{\sum_{i=1}^{N} p_i^2} \tag{8.10}$$

Other non-Gaussian thermostats, such as the Nosé–Hoover thermostat (Chapter 9) can also be used with *sllod*. In contrast to EMD, which involves discontinuing the thermostat after the equilibration stage, NEMD requires a thermostat for the entire length of the simulation.

The calculation of α occurs following the determination of the forces and is quite straightforward as illustrated in Algorithm 8.3 (Parts 1 and 2). However, a common complication is that the accumulation of numerical error results in drift from the desired temperature. It has been reported (Baranyai and Evans, 1990) that this issue can be addressed by incorporating a proportional feedback term. For the Gaussian thermostat, this involves (Part 3) determining the proportional deviation (*T-dev*) between the kinetic temperature and the desired temperature. The value of α is then adjusted by $w \times$ *T-dev*, where w is obtained by trial and error and is typically in the range of 0 to 10. Initially, $w = 0$ should be used to test that the thermostat is working correctly.

8.2.1.1 *Cautionary note on modifications to* sllod

Tuckerman et al. (1997) proposed a generalized *sllod* algorithm (*gsllod*). Some theoretical aspects of the procedure have been debated from the outset (Evans et al., 1998; Tuckerman et al., 1998). Subsequently, different variants have been proposed (Edwards et al., 2006) that are commonly identified by other single letter prefixes to *sllod*. However, there are very detailed scientific arguments (Daivis and Todd, 2006; Todd and Daivis, 2017) to support both the validity and the primacy of the original approach. Therefore, we

2. This should not be confused with the strain, which is commonly denoted as γ.

will focus exclusively on the original *sllod* algorithm, which is implemented in widely used software packages.[3]

ALGORITHM 8.3 Calculation of the α thermostatic constant.

```
          α-den ← α-num ← 0
          loop i ← 1 ... nAtom
Part 1      Determine denominator and numerator parts of α:
            α-den ← mi × mi × (vxi × vxi + vyi × vyi + vzi × vzi)
            α-num ← mi × (fxi × vxi + fyi × vyi + fzi × vzi - γ̇ × vxi × vyi)
          end i loop
Part 2    α ← α-num/α-den
Part 3    Account for numerical drift:
          T-dev ← (α-den/mi – 3(nAtom – 1) × k × T)
                  /( 3(nAtom – 1) × k × T)
          α ← α + w × T-dev
```

Eqs (8.8)–(8.10) must be used in conjunction with suitable periodic boundary conditions such as the Lees–Edwards sliding brick boundary conditions (Lees and Edwards, 1972) as illustrated by Fig. 8.1.

In the Lees–Edwards periodic boundary conditions, the movement of particles outside the central simulation box is imaged in three distinct ways. If the particle passes through either face parallel to the *y*-axis, the new position vector is obtained from,

$$\mathbf{r}_i^{new} = (\mathbf{r}_i) \bmod L \tag{8.11}$$

where L is the length of the simulation box. If the particle moves out of the bottom of the simulation box, its position and velocity are calculated from,

$$\left.\begin{aligned} \mathbf{r}_i^{new} &= (\mathrm{r}_i + \mathbf{i}\gamma L t) \bmod L \\ \mathbf{v}_i^{new} &= \mathbf{v}_i + \mathbf{i}\gamma L \end{aligned}\right\} \tag{8.12}$$

where γ is the strain. If it moves out of the top of the box:

$$\left.\begin{aligned} \mathbf{r}_i^{new} &= (\mathbf{r}_i - \mathbf{i}\gamma L t) \bmod L \\ \mathrm{v}_i^{new} &= \mathbf{v}_i - \mathrm{i}\gamma L \end{aligned}\right\} \tag{8.13}$$

After the simulation reaches a steady state for a given $\dot{\gamma}$, the pressure tensor given by Eq. (8.7) is computed and averaged. The strain rate-dependent shear viscosity is obtained from Newton's law of viscosity.

$$\eta = -\frac{\langle P_{yx} \rangle + \langle P_{xy} \rangle}{2\dot{\gamma}} \tag{8.14}$$

The application of the *sllod* algorithm is handled as a modification of Newton's law in the integrator part of the MD algorithm. No modifications are required to

3. For example, the *sllod* algorithm is used in LAMMPS (https://www.lammps.org). This should be verified at the time of use because of changes made during periodical software updates.

the predictor step. However, an alternative predictor version to that introduced previously (Algorithm 7.2) is given in Algorithm 8.4. In this version, the corresponding corrector step (Algorithm 8.5) of the Gear method is applied independently to both the positions and velocities, solving two first-order differential equations.[4] In common with Algorithm 8.4, the solution illustrated by Algorithm 8.5 involves scaled time derivatives. The prediction of new positions (Part 1.1) involves three derivatives, which are in turn predicted in each time step (Parts 1.1−1.4). The second part (Part 2) of the algorithm involves applying the predictor method to determine new velocities, which involves other derivatives (Parts 2.2−2.4).

The corrector step (Algorithm 8.5) incorporates the effect of a streaming velocity (u_x) in the x direction and a constant strain rate ($\dot{\gamma} = \partial u_x/\partial y$). The correction to the positions (Δrx, Δry, Δrz) using Eq. (8.8) are given at the beginning of Part 1 and are used subsequently to update both positions and its derivatives (Parts 1.1−1.4). The involvement of the $\dot{\gamma}$ term is confined to the x direction (Δrx). The velocity corrections (Δvx, Δvy, Δvz) using Eq. (8.8) are given at the start of Part 2. In common with the positions, the $\dot{\gamma}$ term is only applied in the x direction. However, the α term is applied in all directions, which corresponds to the application of a thermostat. The subsequent parts of the algorithm (Parts 2.1−2.4) involve applying the corrections to both the velocity and its derivatives. It is only after the velocities have been corrected that both the momenta and kinetic energies (not shown in Algorithm 8.5) are calculated.

ALGORITHM 8.4 A four-value Gear predictor applied independently to both positions and velocities using scaled time derivatives.

loop $i \leftarrow 1 \ldots nAtom$

Part 1 Predict new position vectors using Gear method:

Part 1.1 Predict Positions:

$$rx_i \leftarrow rx_i + (drx_i/dt) + (d^2rx_i/dt^2) + (d^3rx_i/dt^3) + (d^4rx_i/dt^4)$$
$$ry_i \leftarrow ry_i + (dry_i/dt) + (d^2ry_i/dt^2) + (d^3ry_i/dt^3) + (d^4ry_i/dt^4)$$
$$rz_i \leftarrow rz_i + (drz_i/dt) + (d^2rz_i/dt^2) + (d^3rz_i/dt^3) + (d^4rz_i/dt^4)$$

Part 1.2 First (scaled time) derivatives of position:

$$(drx_i/dt) \leftarrow (drx_i/dt) + (d^2rx_i/dt^2) + 2(d^3rx_i/dt^3) + 3(d^4rx_i/dt^4)$$
$$(dry_i/dt) \leftarrow (dry_i/dt) + (d^2ry_i/dt^2) + 2(d^3ry_i/dt^3) + 3(d^4rx_i/dt^4)$$
$$(drz_i/dt) \leftarrow (drz_i/dt) \; v \; (d^2rz_i/dt^2) + 2(d^3rz_i/dt^3) + 3(d^4rz_i/dt^4)$$

Part 1.3 Second (scaled time) derivatives of position:

$$(d^2rx_i/dt^2) \leftarrow (d^2rx_i/dt^2) + (d^3rx_i/dt^3) + 3(d^4rx_i/dt^4)$$
$$(d^2ry_i/dt^2) \leftarrow (d^2ry_i/dt^2) + (d^3ry_i/dt^3) + 3(d^4ry_i/dt^4)$$
$$(d^2rz_i/dt^2) \leftarrow (d^2rz_i/dt^2) + (d^3rz_i/dt^3) + 3(d^4rz_i/dt^4)$$

Part 1.4 Third (scaled time) derivatives of position:

$$(d^3rx_i/dt^3) \leftarrow (d^3rx_i/dt^3) + (d^4rx_i/dt^4)$$
$$(d^3ry_i/dt^3) \leftarrow (d^3ry_i/dt^3) + (d^4ry_i/dt^4)$$
$$(d^3rz_i/dt^3) \leftarrow (d^3rz_i/dt^3) + (d^4rz_i/dt^4)$$

4. The Gear corrector coefficients for the first-order differential equations are $k_0 = 3/8$, $k_1 = 1$, $k_2 = 3/4$, and $k_3 = 1/6$.

Part 2 Predict new velocities using Gear algorithm:
Part 2.1 Predict velocities:

$$vx_i \leftarrow vx_i + (dvx_i/dt) + (d^2vx_i/dt^2) + (d^3vx_i/dt^3) + (d^4vx_i/dt^4)$$
$$vy_i \leftarrow vy_i + (dvy_i/dt) + (d^2vy_i/dt^2) + (d^3vy_i/dt^3) + (d^4vy_i/dt^4)$$
$$vz_i \leftarrow vz_i + (dvz_i/dt) + (d^2vz_i/dt^2) + (d^3vz_i/dt^3) + (d^4vz_i/dt^4)$$

Part 2.2 First (scaled time) derivatives of velocity:

$$(dvx_i/dt) \leftarrow (dvx_i/dt) + (d^2vx_i/dt^2) + 2(d^3vx_i/dt^3) + 3(d^4vx_i/dt^4)$$
$$(dvy_i/dt) \leftarrow (dvy_i/dt) + (d^2vy_i/dt^2) + 2(d^3vy_i/dt^3) + 3(d^4vx_i/dt^4)$$
$$(dvz_i/dt) \leftarrow (dvz_i/dt) + (d^2vz_i/dt^2) + 2(d^3vz_i/dt^3) + 3(d^4vz_i/dt^4)$$

Part 2.3 Second (scaled time) derivatives of velocity:

$$(d^2vx_i/dt^2) \leftarrow (d^2vx_i/dt^2) + (d^3vx_i/dt^3) + 3(d^4vx_i/dt^4)$$
$$(d^2vy_i/dt^2) \leftarrow (d^2vy_i/dt^2) + (d^3vy_i/dt^3) + 3(d^4vy_i/dt^4)$$
$$(d^2vz_i/dt^2) \leftarrow (d^2vz_i/dt^2) + (d^3vz_i/dt^3) + 3(d^4vz_i/dt^4)$$

Part 2.4 Third time (scaled time) derivatives of velocity:

$$(d^3vx_i/dt^3) \leftarrow (d^3vx_i/dt^3) + (d^4vx_i/dt^4)$$
$$(d^3vy_i/dt^3) \leftarrow (d^3vy_i/dt^3) + (d^4vy_i/dt^4)$$
$$(d^3vz_i/dt^3) \leftarrow (d^3vz_i/dt^3) + (d^4vz_i/dt^4)$$

end *i* **loop**

ALGORITHM 8.5 A four-value Gear corrector applied independently to both positions and velocities implementing *sllod* under constant strain rate, using scaled time derivatives.

loop $i \leftarrow 1 \ldots nAtom$
Part 1 Determine corrections to positions and its derivatives:

$$\Delta rx \leftarrow (vx_i + \dot{\gamma} \times ry_i) \times \Delta t - (drx_i/dt)$$
$$\Delta ry \leftarrow vy_i \times \Delta t - (dry_i/dt)$$
$$\Delta rz \leftarrow vz_i \times \Delta t - (drz_i/dt)$$

Part 1.1 Correct Positions:

$$rx_i \leftarrow rx_i + k0 \times \Delta rx$$
$$ry_i \leftarrow ry_i + k0 \times \Delta ry$$
$$rz_i \leftarrow rz_i + k0 \times \Delta rz$$

Part 1.2 Correct first (scaled time) derivatives of position:

$$(drx_i/dt) \leftarrow (drx_i/dt) + \Delta rx$$
$$(dry_i/dt) \leftarrow (dry_i/dt) + \Delta ry$$
$$(drz_i/dt) \leftarrow (drz_i/dt) + \Delta rz$$

Part 1.3 Second (scaled time) derivatives of position:

$$(d^2rx_i/dt^2) \leftarrow (d^2rx_i/dt^2) + k2 \times \Delta rx$$
$$(d^2ry_i/dt^2) \leftarrow (d^2ry_i/dt^2) + k2 \times \Delta ry$$
$$(d^2rz_i/dt^2) \leftarrow (d^2rz_i/dt^2) + k2 \times \Delta rz$$

```
Part 1.4    Third (scaled time) derivatives of position:
            (d³rxi/dt³) ← (d³rxi/dt³) + k3 × Δrx
            (d³ryi/dt³) ← (d³ryi/dt³) + k3 × Δrx
            (d³rzi/dt³) ← (d³rzi/dt³) + k3 × Δrx
Part 2      Determine corrections to velocity and its derivatives:
            Δvx ← (fxi/mi - γ̇ × vyi - α × vxi) × Δt - (dvxi/dt)
            Δvy ← (fyi/mi - α × vxi) × Δt - (dvyi/dt)
            Δvz ← (fzi/mi - α × vxi) × Δt - (dvzi/dt)
Part 2.1    Correct velocities:
            vxi ← vxi + k0 × Δvx
            vyi ← vyi + k0 × Δvy
            rzi ← vzi + k0 × Δvz
Part 2.2    Correct first (scaled time) derivatives of velocity:
            (dvxi/dt) ← (dvxi/dt) + Δvx
            (dvyi/dt) ← (dvyi/dt) + Δvy
            (dvzi/dt) ← (dvzi/dt) + Δvz
Part 2.3    Second (scaled time) derivatives of velocity:
            (d²vxi/dt²) ← (d²vxi/dt²) + k2 × Δvx
            (d²vyi/dt²) ← (d²vyi/dt²) + k2 × Δvy
            (d²vzi/dt²) ← (d²vzi/dt²) + k2 × Δvz
Part 2.4    Third (scaled time) derivatives of velocity:
            (d³vxi/dt³) ← (d³vxi/dt³) + k3 × Δvx
            (d³vyi/dt³) ← (d³vyi/dt³) + k3 × Δvx
            (d³vzi/dt³) ← (d³vzi/dt³) + k3 × Δvx
        end i loop
```

Hansen and Evans (1994) have discussed how shear flow can be simulated efficiently on a distributed memory parallel processor. Bhupathiraju et al. (1996) applied the *sllod* algorithm with a deforming cell method instead of the Lees−Edwards periodic boundary conditions. They concluded that the modification improved the accuracy of calculated viscosities for low shear rates. The modified algorithm was also more efficient computationally than the conventional Lees−Edwards approach.

Shear viscosity has been widely examined by NEMD techniques. Examples of systems studied include methane (Simmons and Cummings, 1986; Evans, 1979b; Cummings and Evans, 1992), carbon dioxide (Wang and Cummings, 1989a,b), water (Cummings and Varner, 1988; Balasubramanian et al., 1996), liquid metals (Cummings and Morriss, 1987, 1988; Cummings and Evans, 1992), and alkanes (Edberg et al., 1987; Morriss et al., 1991; Berker et al., 1992). These studies have used various intermolecular potentials, and comparison with experimental shear viscosity data has proved useful in identifying the relative accuracy of the different intermolecular potentials. Lee and Cummings (1994) investigated shear viscosity using both two-body and three-body intermolecular potentials. They concluded that three-body interactions make a contribution of only 3%. This relatively small contribution was confirmed by subsequent studies (Marcelli et al., 2001a,b). It is much less than the influence on nontransport properties such as

either phase equilibria (Marcelli and Sadus, 1999) or thermodynamic properties (Raabe et al., 2005; Vlasiuk and Sadus, 2017).

NEMD simulations have demonstrated (Marcelli et al., 2001a; Ge et al., 2001) that the pressure (p) and energy (U) of simple fluids under planer shear flow are only linear functions of $\dot{\gamma}^{2/3}$ in the vicinity of the triple point. More generally,

$$\left.\begin{aligned} U &= U_0 + a\dot{\gamma}^{\alpha} \\ p &= p_0 + b\dot{\gamma}^{\alpha} \end{aligned}\right\} \tag{8.15}$$

where the 0 subscript denotes an equilibrium property; a and b are density and temperature-dependent constants. Ge et al. (2003) determined the following relationship for α in terms of the reduced temperature (T^*) and reduced density (ρ^*).

$$\alpha = 3.67 + 0.69T^* - 3.35\rho^* \tag{8.16}$$

The constants in Eq. (8.16) are either universally the same for a one component fluid or dependent only on the intermolecular potential. The significance of Eqs. (8.15) and (8.16) is that the properties of normal nonequilibrium fluids as a function of strain rate could be predicted for any arbitrary state point. However, this does not appear to be the case (Ahmed et al., 2010) for the special case of intrapenetrable fluids such as the Gaussian core model fluid (Chapter 3).

The LJ potential has been the most widely studied intermolecular potential for shear viscosity. Ahmed and Sadus (2010) made use of extensive NEMD *sllod* simulations to determine an empirical equation of state for the LJ potential.

$$p = \rho T + \sum_{i=1}^{8} A_i(T)\rho^{i+1} + e^{-\delta\rho^2}\sum_{i=1}^{8} B_i(T)\rho^{2i+1}$$

$$+ \rho\sum_{i=0}^{3}\sum_{j}^{3} f_{ij}\dot{\gamma}^i X^{i+m} T^{1-j} \tag{8.17}$$

In Eq. (8.17), all quantities are in nondimensional reduced form and the first three terms on the right-hand side refer to LJ equilibrium properties (Nicolas et al., 1979). The contribution at nonzero strain rate is given be the third term, which involves parameters (f_{ij}) fitted to the simulation data with $X = \rho T^{-1/4}$. This can serve as an accurate reference equation of state for LJ at different values of $\dot{\gamma}$, which greatly reduces the need to repeat NEMD simulations for new cases.

The utility of NEMD is evident by the concept of electropumping of fluids (de Luca et al., 2013) in nanopores, which involves the application of an external electric field that induces enhanced flow rates. In particular, it has been reported (Ostler et al., 2020) that electropumping could be many

times more efficient than conventional Poiseuille flows in nanochannels such as carbon nanotubes (CNT).

8.2.2 Self-diffusion

The Green−Kubo relationship for the self-diffusion coefficient is:

$$D_s = \frac{1}{3}\int_0^\infty \left\langle \mathbf{v}_i(0)\cdot\mathbf{v}_i(t)\right\rangle dt \tag{8.18}$$

Evans et al. (1983) developed the "color conductivity" algorithm to calculate the self-diffusion coefficient. The system is described by the following Hamiltonian,

$$H = H_0 - \sum_{i=1}^{N} c_i x_i F(t) \tag{8.19}$$

where H_0 is the unperturbed Hamiltonian, F is a fictitious color field in the x direction, and $c_i = (-1)^i$ is the "color charge" on molecule i. The equations of motion derived from this Hamiltonian are:

$$\mathbf{r}_i = \frac{\mathbf{p}_i}{m_i} \tag{8.20}$$

$$\left.\begin{aligned} \dot{p}_{xi} &= F_{xi} - Fc_i \\ \dot{p}_{yi} &= F_{yi} - \lambda_s p_{yi} \\ \dot{p}_{zi} &= F_{zi} - \lambda_s p_{zi} \end{aligned}\right\} \tag{8.21}$$

In Eq. (8.21), λ_s is a thermostating constraint parameter. The applied color field F is obtained from:

$$F = \frac{1}{N}\sum_{i=1}^{N} c_i F_{xi} \tag{8.22}$$

This drives the color current (J_x).

$$J_x = \sum_{i=1}^{N} c_i \frac{p_{ix}}{m_i} \tag{8.23}$$

The constraint parameter can be calculated from:

$$\lambda_s = \frac{\sum_{i=1}^{N}\left(p_{iy}F_{iy} + p_{iz}F_{iz}\right)}{\sum_{i=1}^{N}\left(p_{iy}^2 + p_{iz}^2\right)} \tag{8.24}$$

The work W generated by the field (F) is given by:

$$W = F\sum_{i=1}^{N} c_i p_{xi} \tag{8.25}$$

The self-diffusion coefficient is calculated from:

$$D_s = -\frac{kT(N-1)J_x}{N^2\langle W\rangle} \tag{8.26}$$

Examples of the calculation of the self-diffusion coefficient have been reported for carbon dioxide (Wang and Cummings, 1989b) and mercury (Raabe et al., 2005).

8.2.3 Thermal conductivity

The Green−Kubo relationship for the thermal conductivity coefficient (λ) is,

$$\lambda = \frac{V}{3kT^2}\int_0^\infty \langle \mathbf{J}_Q(0)\cdot\mathbf{J}_Q(t)\rangle dt \tag{8.27}$$

where $\mathbf{J}_Q$ is the heat current. A Hamiltonian-based algorithm for thermal conductivity cannot be devised because the equations of motion obtained from the Hamiltonian are discontinuous when used with periodic boundary conditions. Evans (1982, 1986a,b) introduced a fictitious vector field (**F**) as the driving force for the heat flux ($\mathbf{J}_Q$). The equations of motion for atomic systems are,

$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i} \tag{8.28}$$

$$\begin{aligned}\dot{\mathbf{p}}_i = {} & \mathbf{F}_i + (E_i - E)\mathbf{F}(t) \\ & -\frac{1}{2}\sum_{j=1}^{N}\mathbf{F}_{ij}(\mathbf{r}_{ij}\cdot\mathbf{F}(t)) + \frac{1}{2N}\sum_{j=1}^{N}\sum_{k=1}^{N}\mathbf{F}_{jk}(\mathbf{r}_{jk}\cdot\mathbf{F}(t)) - \alpha\mathbf{p}_i\end{aligned} \tag{8.29}$$

where $\mathbf{F}_{ij}$ is the force exerted by particle j on particle i, E_i is the instantaneous energy of i, and E is the instantaneous average energy per particle. In Eq. (8.29), α is the thermostating multiplier.

$$\alpha = \frac{\sum_i \left[\mathbf{F}_i + (E_i - E)\mathbf{F}(t) - \frac{1}{2}\sum_{j=1}^{N}\mathbf{F}_{ij}(\mathbf{r}_{ij}\cdot\mathbf{F}(t)) + \frac{1}{2N}\sum_{j=1}^{N}\sum_{k=1}^{N}\mathbf{F}_{jk}(\mathbf{r}_{jk}\cdot\mathbf{F}(t))\right]\cdot\mathbf{p}_i}{\sum_i \mathbf{p}_i^2} \tag{8.30}$$

The heat current ($\mathbf{J}_Q$) is derived from,

$$\mathbf{J}_Q V = \sum_{i=1}^{N} E_i\frac{\mathbf{p}_i}{m_i} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\mathrm{r}_{ij}(\mathbf{p}_i\cdot\mathbf{F}_{ij}) \tag{8.31}$$

and the thermal conductivity is calculated using:

$$\lambda = \frac{\langle \mathbf{J}_Q\rangle}{T\mathbf{F}} \tag{8.32}$$

Evans (1986a,b) has demonstrated that this algorithm can be used to predict the thermal conductivity of argon at the triple point more precisely than experimental measurements. Gillan and Dixon (1983) have derived a similar algorithm from a different perspective.

Molecular simulations of thermal conductivity are reported less frequently than for other transport phenomena. The NEMD studies of carbon dioxide (Wang and Cummings, 1989b; Cummings and Evans, 1992) and butane (Daivis and Evans, 1994, 1995) are relatively rare examples. An investigation has been reported (Searles et al., 1998) of thermal conductivity near the critical point.

8.2.3.1 Cautionary note on the triple point

Studies of transport properties are often conducted near the triple point to provide examples of behavior in a dense liquid phase. In particular, the LJ triple point is a very common choice. There are varying estimates in the literature (Ahmed and Sadus, 2010) of the LJ triple point. Currently, the most precise value (Schultz and Kofke, 2018) of the reduced triple point is $T^* = 0.69455$, $p^* = 0.001267$, $\rho_l^* = 0.84535$, and $\rho_s^* = 0.960765$. This is for the full potential, and its value will be significantly affected by cut-off values, truncations, and potential shifts. Therefore, care is required when performing simulations in the vicinity of the triple point to ensure that the nominated state points are genuinely in the single-phase region. The same considerations also apply to other choices of intermolecular potential (Deiters and Sadus, 2022).

8.3 Application of NEMD algorithms to molecules

Early work on NEMD algorithms focused on either atomic systems or small molecules. For small polyatomic molecules, the *sllod* algorithm can be adapted (Edberg et al., 1987) in terms of a quaternionic framework. More generally, however, the synthetic NEMD algorithms for atomic systems can be used with the constraint dynamics techniques and realistic intermolecular potentials to predict the transport properties of multiatom molecules.

Daivis and Evans (1994, 1995) have investigated the thermal conductivity of butane using different models. Comparison with experiment indicates (Daivis and Evans, 1995) deficiencies in both the Ryckaert and Bellemans (1978) and anisotropic united-atom (Toxvaerd, 1990) models of butane. Allen and Rowley (1997) compared the viscosity of alkanes predicted by the Ryckaert−Bellemans, united-atom, and all-atom intermolecular potentials. The comparison indicated that results obtained using the all-atom model were generally superior to the results obtained with the other intermolecular potentials.

NEMD techniques have also been used to examine the rheology of both linear and branched alkanes (Berker et al., 1992; Morriss et al. 1991; Daivis et al., 1992). Mundy et al. (1995) investigated the shear viscosity of decane

using *sllod* algorithms in both the isothermal−isobaric and canonical ensembles. Good agreement between simulation and experimental data for the shear viscosity of alkanes has been reported (Mundy et al., 1996). Lahtela et al. (1998) reported NEMD simulations for branched alkanes, which indicate that viscosity is dominated by the size of the branch. Tang et al. (1998) has developed a theory for the dynamics of flexible chains. NEMD has also proved useful in providing insights for the behavior of macromolecules. For example, it has been used to determine the generation-dependence of both the structural properties (Bosko et al., 2004a) and shear-thing behavior (Bosko et al., 2004b) of dendrimers. The shape of the dendrimer and its shear viscosity are directly related (Bosko et al. 2006).

8.4 Application of NEMD algorithms to mixtures

The use of NEMD algorithms has been historically focused on pure component properties, but the prediction of the transport properties of fluid mixtures is becoming increasingly more important. Murad (1986) used the *sllod* algorithm to evaluate different mixture rules for the viscosity of binary LJ mixtures. Wang and Cummings (1996) performed NEMD *sllod* calculations for the shear viscosity for mixtures containing carbon dioxide and ethane. Both carbon dioxide and ethane were modeled as two-center LJ molecules yielding good agreement with experimental data. Wheeler and Rowley (1998) reported simulations for the water + methanol + acetone ternary mixture in addition to results for binary mixtures containing these components. Sarman and Evans (1992a,b) used NEMD to evaluate heat flow and mass diffusion in binary LJ mixtures. The thermal conductivity of mixtures of polyatomic fluids has also been studied by NEMD (Ravi and Murad, 1992). EMD has arguably been used more widely than NEMD for the calculation of the self-diffusion coefficient of mixtures (Jacucci and McDonald, 1975; Schoen and Hoheisel, 1984a,b; Jolly and Bearman, 1980; Vogelsang and Hoheisel, 1988; Heyes and Preston 1991a,b; Gardner et al., 1991). The shear viscosity of mixtures of dendrimers has also been studied (Bosko et al., 2005) in different ensembles.

A very important example (Losey et al., 2019) of the use of NEMD algorithms for mixtures is their usefulness for investigating transport phenomena in either pores or CNTs. This involves mixture properties in two ways: interactions between a pure confined fluid and the CNT; and interaction between both the CNT and a confined mixture and between the individual components of the confined fluid.

8.5 Comparison with EMD

As noted earlier, transport properties can also be calculated from EMD trajectories using either Einstein or Green−Kubo relationships. The Einstein relationships for diffusion, shear viscosity, and thermal conductivity are

given by Eq. (2.48), Eq. (2.49), and Eq. (2.52), respectively. When $F_e = 0$, the NEMD algorithms discussed earlier revert to their equilibrium form.

The main advantage of using EMD is that all the transport properties can be obtained from a single run. However, the disadvantages are that the equilibrium calculations are expensive computationally, they sometimes yield poor signal to noise ratios; and the results may depend strongly on the size of the simulated system. Consequently, NEMD is the preferred method for obtaining transport phenomena. Nonetheless, it is of interest to compare results obtained from NEMD with EMD calculations. A comparison between the two simulation approaches has been given by Cummings and Evans (1992). Hoheisel (1994) has reviewed the calculation of transport properties of EMD.

Holian and Evans (1983) reported a direct comparison of NEMD and EMD for shear viscosity calculated from the Green−Kubo relationship. They concluded that the NEMD results were size independent for a system of 100 or more particles, whereas EMD results were strongly size-dependent. Schoen and Hoheisel (1985) reported good agreement between EMD Green−Kubo and NEMD results. The EMD calculations were not significantly size-dependent beyond a system size of 500 particles. Chialvo and Debenedetti (1991) and Chialvo et al. (1989) used EMD with the Einstein formula to calculate the shear viscosity of linear molecules and LJ molecules, respectively. These results indicate that the Einstein formula gives low noise prediction. For linear molecules, there is a discrepancy between EMD results and the NEMD calculations of Wang and Cummings (1989a). For LJ molecules, the EMD results were strongly system dependent.

Cummings and Evans (1992) reported that EMD calculations for the self-diffusion coefficient using either the Einstein or Green−Kubo relationships yield similar results to NEMD calculations. Furthermore, the EMD method requires fewer time steps than the NEMD algorithm to obtain the same degree of accuracy.

A direct comparison between different EMD and NEMD techniques is not available for thermal conductivity. Levesque and Verlet (1987) observed a strong size dependence for the thermal conductivity calculated by the Green−Kubo EMD method in systems containing up to 4000 LJ molecules.

8.6 Summary

NEMD algorithms have been successful in extending the application of MD to phenomena in which thermodynamic equilibrium does not occur. The most widely used NEMD algorithms are synthetic and their development parallels the extension of MD to different thermodynamic ensembles (Chapter 9). The application of NEMD to mixtures is continually increasing and providing theoretical insights into the behavior of real systems. It can be anticipated that NEMD algorithms will play an increasing role in the investigation of the rheological properties of large molecules.

References

Ahmed, A., Sadus, R.J., 2010. Effect of potential truncations and shifts on the solid-liquid phase coexistence of Lennard-Jones fluids. J. Chem. Phys. 133, 124515.

Ahmed, A., Mausbach, P., Sadus, R.J., 2010. Pressure and energy behavior of the Gaussian core model fluid under shear. Phys. Rev. E 82, 011201.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Allen, W., Rowley, R.L., 1997. Predicting the viscosity of alkanes using nonequilibrium molecular dynamics: evaluation of intermolecular potential models. J. Chem. Phys. 106, 10273–10281.

Ashurst, W.T., Hoover, W.G., 1973. Argon shear viscosity via a Lennard-Jones potential with equilibrium and nonequilibrium molecular dynamics. Phys. Rev. Lett. 31, 206–208.

Balasubramanian, S., Mundy, C.J., Klein, M.L., 1996. Shear viscosity of polar fluids: molecular dynamics calculations of water. J. Chem. Phys. 105, 11190–11195.

Baranyai, A., Evans, D.J., 1990. New algorithm for constrained molecular-dynamics simulation of liquid benzene and naphthalene. Mol. Phys. 70, 53–63.

Berker, A., Chynoweth, S., Klomp, U.C., Michopoulos, Y., 1992. Non-equilibrium molecular dynamics (NEMD) simulations of the rheological properties of liquid n-hexadecane. J. Chem. Soc. Faraday Trans. 88, 1719–1725.

Bhupathiraju, R., Cummings, P.T., Cochran, H.D., 1996. An efficient parallel algorithm for non-equilibrium molecular dynamics simulations of very large systems in planar Couette flow. Mol. Phys. 88, 1665–1670.

Bosko, J.T., Todd, B.D., Sadus, R.J., 2004a. Internal structure of dendrimers in the melt: a molecular dynamics study. J. Chem. Phys. 121, 1091–1096.

Bosko, J.T., Todd, B.D., Sadus, R.J., 2004b. Viscoelastic properties of dendrimers in the melt from nonequilibrium molecular dynamics. J. Chem. Phys. 121, 12050–12059.

Bosko, J.T., Todd, B.D., Sadus, R.J., 2005. Molecular simulation of dendrimers and their mixtures under shear: comparison of isothermal-isobaric (NpT) and isothermal-isochoric (NVT) ensemble systems. J. Chem. Phys. 123, 034905.

Bosko, J.T., Todd, B.D., Sadus, R.J., 2006. Analysis of the shape of dendrimers under shear. J. Chem. Phys. 124, 044910.

Chialvo, A.A., Debenedetti, P.G., 1991. Use of the McQuarrie equation for the computation of shear viscosity via equilibrium molecular dynamics. Phys. Rev. A 43, 4289–4295.

Chialvo, A.A., Heath, D.L., Debenedetti, P.G., 1989. A molecular dynamics study of the influence of elongation and quadrupole moment upon some thermodynamic and transport properties of linear heteronuclear triatomic fluids. J. Chem. Phys. 91, 7818–7830.

Ciccotti, G., 1991. In: Meyer, M., Pontikis, V. (Eds.), Computer Simulation in Materials Science. Interatomic Potentials, Simulation techniques and Applications. Kluwer, Dordrecht.

Ciccotti, G., Jacucci, G., McDonald, I.R., 1979. "Thought-experiments" by molecular dynamics. J. Stat. Phys. 21, 199–226.

Cummings, P.T., Evans, D.J., 1992. Nonequilibrium molecular dynamics approaches to transport properties and non-Newtonian fluid rheology. Ind. Eng. Chem. Res. 31, 1237–1252.

Cummings, P.T., Morriss, G.P., 1987. Nonequilibrium molecular dynamics calculation of the shear viscosity of liquid rubidium. J. Phys. F. 17, 593–604.

Cummings, P.T., Morriss, G.P., 1988. Shear viscosity of liquid rubidium at the triple point. J. Phys. F. 18, 1439–1447.

Cummings, P.T., Varner Jr, T.L., 1988. Nonequilibrium molecular dynamics calculation of the shear viscosity of liquid water. J. Chem. Phys. 89, 6391–6398.

Daivis, P.J., Evans, D.J., 1994. Non-equilibrium molecular dynamics calculation of thermal conductivity of flexible molecules: butane. Mol. Phys. 81, 1289–1295.

Daivis, P.J., Evans, D.J., 1995. Temperature dependence of the thermal conductivity for two models of liquid butane. Chem. Phys. 198, 25–34.

Daivis, P.J., Todd, B.D., 2006. A simple, direct derivation and proof of the validity of the SLLOD equations of motion for generalized homogeneous flows. J. Chem. Phys. 124, 194103.

Daivis, P.J., Evans, D.J., Morriss, G.P., 1992. Computer simulation study of the comparative rheology of branched and linear alkanes. J. Chem. Phys. 97, 616–627.

de Luca, S., Todd, B.D., Hansen, J.S., Daivis, P.J., 2013. Electropumping of water with rotating electric fields. J. Chem. Phys. 138, 154712.

Deiters, U.K., Sadus, R.J., 2022. First-principles determination of the solid-liquid-vapor triple points: the noble gases. Phys. Rev. E 105, 054128.

Edberg, R., Morriss, G.P., Evans, D.J., 1987. Rheology of n-alkanes by nonequilibrium molecular dynamics. J. Chem. Phys. 86, 4555–4570.

Edwards, B.J., Baig, C., Keffer, D.J., 2006. A validation of the p-SLLOD equations of motion for homogeneous steady-state flows. J. Chem. Phys. 124, 194104.

Evans, D.J., 1979a. The non-symmetric pressure tensor in polyatomic fluids. J. Stat. Phys. 20, 547–555.

Evans, D.J., 1979b. The frequency dependent shear viscosity of methane. Mol. Phys. 37, 1745–1754.

Evans, D.J., 1982. Homogeneous NEMD algorithm for thermal conductivity -application of non-canonical linear response theory. Phys. Lett. A 91, 457–460.

Evans, D.J., 1986a. Thermal conductivity of the Lennard-Jones fluid. Phys. Rev. A 34, 1449–1453.

Evans, D.J., 1986b. In: Ciccotti, G., Hoover, W.G. (Eds.), Molecular Dynamics Simulation of Statistical-Mechanical Systems. North-Holland, Amsterdam.

Evans, D.J., Morriss, G.P., 1984a. Nonlinear response theory for steady planar Couette flow. Phys. Rev. A 30, 1528–1530.

Evans, D.J., Morriss, G.P., 1984b. Non-Newtonian molecular dynamics. Comp. Phys. Rep. 1, 297–343.

Evans, D.J., Morriss, G.P., 2008. Statistical Mechanics of Nonequilibrium Liquids, second ed. Cambridge University Press, Cambridge.

Evans, D.J., Hoover, W.G., Failor, B.H., Moran, B., Ladd, A.J.C., 1983. Nonequilibrium molecular dynamics via Gauss's principle of least constraint. Phys. Rev. A 28, 1016–1021.

Evans, D.J., Searles, D.J., Hoover, W.G., Hoover, C.G., Holian, B.L., Posch, H.A., Morriss, G.P., 1998. Comment on "Modified nonequilibrium molecular dynamics for fluid flows with energy conservation" [J. Chem. Phys. 106, 5615 (1997)]. J. Chem. Phys. 108, 4351–4352.

Ford, D.M., Heffelfinger, G.S., 1998. Massively parallel dual control grand canonical molecular dynamics with LADERA. II. Gradient driven diffusion through polymers. Mol. Phys. 94, 673–683.

Gardner, P.J., Heyes, D.M., Preston, S.R., 1991. Molecular dynamics computer simulations of binary Lennard-Jones fluid mixtures: Thermodynamics of mixing and transport coefficients. Mol. Phys. 73, 141–173.

Ge, J., Marcelli, G., Todd, B.D., Sadus, R.J., 2001. Energy and pressure of shearing fluids at different state points. Phys. Rev. E 64, 02120. Erratum: Phys. Rev. E. 65, 069901 (2002).

Ge, J., Todd, B.D., Wu, G.-W., Sadus, R.J., 2003. Scaling behavior for the pressure and energy of shearing fluids. Phys. Rev. E 67, 061201.

Gillan, M.J., Dixon, M., 1983. The calculation of thermal conductivities by perturbed molecular dynamics simulation. J. Phys. C. 16, 869–878.
Gosling, E.M., McDonald, I.R., Singer, K., 1973. On the calculation by molecular dynamics of the shear viscosity of a simple fluid. Mol. Phys. 26, 1475–1484.
Green, H.S., 1951. The quantum mechanics of assemblies of interacting particles. J. Chem. Phys. 19, 955–962.
Hansen, D.P., Evans, D.J., 1994. A parallel algorithm for nonequilibrium molecular dynamics simulation of shear flow on distributed memory machines. Mol. Sim. 13, 375–393.
Heffelfinger, G.S., Ford, D.M., 1998. Massively parallel dual control grand canonical molecular dynamics with LADERA. I. Gradient driven diffusion in Lennard-Jones fluids. Mol. Phys. 94, 659–671.
Heffelfinger, G.S., van Swol, F., 1994. Diffusion in Lennard-Jones fluids using dual control volume grand canonical molecular dynamics simulation (DCV-GCMD). J. Chem. Phys. 100, 7548–7552.
Heyes, D.M., Preston, S.R., 1991a. Transport coefficients of argon-krypton mixtures by molecular dynamics computer simulation. Phys. Chem. Liq. 23, 123–149.
Heyes, D.M., Preston, S.R., 1991b. Equilibrium molecular dynamics computer simulations of the transport coefficients of argon-methane mixtures. Mol. Sim. 7, 221–239.
Hoheisel, C., 1994. Transport properties of molecular liquids. Phys. Rep. 245, 111–157.
Holian, B.L., Evans, D.J., 1983. Shear viscosities away from the melting line: a comparison of equilibrium and nonequilibrium molecular dynamics. J. Chem. Phys. 78, 5147–5150.
Hoover, W.G., 1985. Canonical dynamics: equilibrium phase-space distributions. Phys. Rev. A 31, 1695–1697.
Hoover, W.G., 1991. Computational Statistical Mechanics. Elsevier, Amsterdam.
Hoover, W.G., Evans, D.J., Hickman, R.B., Ladd, A.J.C., Ashurst, W.T., Moran, B., 1980. Lennard-Jones triple-point bulk and shear viscosities. Green-Kubo theory, Hamiltonian mechanics, and nonequilibrium molecular dynamics. Phys. Rev. A 22, 1690–1697.
Hunt, T.A., Todd, B.D., 2003. On the Arnold cat map and periodic boundary conditions for planar elongational flow. Mol. Phys. 101, 3445–3454.
Hunt, T.A., Bernardi, S., Todd, B.D., 2010. A new algorithm foe extended nonequilibrium molecular dynamics simulations of mixed flow. J. Chem. Phys. 133, 154116.
Jacucci, G., McDonald, I.R., 1975. Structure and diffusion in mixtures of rare-gas liquids. Phys. A 80, 607–625.
Jolly, D.L., Bearman, R.J., 1980. Molecular dynamics simulation of the mutual and self diffusion in Lennard-Jones liquid mixtures. Mol. Phys. 41, 137–147.
Kubo, R., 1957. Statistical-mechanical theory of irreversible processes. I. General theory and simple applications to magnetic and conduction problems. J. Phys. Soc. Jpn. 12, 570–586.
Ladd, A.J.C., 1984. Equations of motion for nonequilibrium molecular dynamics simulations of viscous flow in molecular liquids. Mol. Phys. 53, 459–463.
Lahtela, M., Linnolahti, M., Pakkanen, T.A., Rowley, R.L., 1998. Computer simulations of branched alkanes: the effect of side chain and its position on rheological behavior. J. Chem. Phys. 108, 2626–2630.
Lee, S.H., Cummings, P.T., 1994. Effect of three-body forces on the shear viscosity of liquid argon. J. Chem. Phys. 101, 6206–6209.
Lees, A.W., Edwards, S.F., 1972. The computer study of transport processes under extreme conditions. J. Phys. C. 5, 1921–1929.
Levesque, D., Verlet, L., 1987. Molecular dynamics calculations of transport coefficients. Mol. Phys. 61, 143–159.

Losey, J., Kannam, S.K., Todd, B.D., Sadus, R.J., 2019. Flow of water through carbon nanotubes predicted by different atomistic models. J. Chem. Phys. 150, 194501.

Marcelli, G., Sadus, R.J., 1999. Molecular simulation of the phase behavior of noble gases using accurate two-body and three-body intermolecular potentials. J. Chem. Phys. 111, 1533–1540.

Marcelli, G., Todd, B.D., Sadus, R.J., 2001a. The strain rate dependence of shear viscosity, pressure and energy from two-body and three-body interactions. Fluid Phase Equilib. 183-184, 371–379.

Marcelli, G., Todd, B.D., Sadus, R.J., 2001b. Analytic dependence of the pressure and energy of an atomic fluid under shear. Phys. Rev. E 63, 021204.

Morriss, G.P., Evans, D.J., 1985. Isothermal response theory. Mol. Phys. 54, 629–636.

Morriss, G.P., Daivis, P.J., Evans, D.J., 1991. The rheology of n-alkanes: decane and eicosane. J. Chem. Phys. 94, 7420–7433.

Mundy, C.J., Siepmann, J.I., Klein, M.L., 1995. Decane under shear: a molecular dynamics study using reversible NVT-SLLOD and NPT-SLLOD algorithms. J. Chem. Phys. 103, 10192–10200.

Mundy, C.J., Balasubramanian, S., Bagchi, K., Siepmann, J.I., Klein, M.L., 1996. Equilibrium and Non-Equilibrium simulation studies of fluid alkanes in bulk and at interfaces. Faraday Discuss. 104, 17–36.

Murad, S., 1986. The viscosity of dense fluid mixtures: mixing rules reexamined using nonequilibrium molecular dynamics. AIChE J. 32, 513–516.

Nicolas, J.J., Gubbins, K.E., Street, W.B., Tildesley, D.J., 1979. Equation of state for the Lennard-Jones fluid. Mol. Phys. 37, 1429–1454.

Ostler, D., Kannam, S.K., Frascoli, F., Daivis, P.J., Todd, B.J., 2020. Efficiency of electropumping in nanochanels. Nano Lett. 20, 3396–3402.

Raabe, G., Todd, B.D., Sadus, R.J., 2005. Molecular simulation of the shear viscosity and the self diffusion coefficient of mercury along the vapor-liquid coexistence curve. J. Chem. Phys. 123, 034511.

Rapaport, D.C., 2004. The Art of Molecular Dynamics Simulation, second ed. Cambridge University Press, Cambridge.

Ravi, P., Murad, S., 1992. Thermal conductivity of mixtures of polyatomic fluids using nonequilibrium molecular dynamics. Mol. Sim. 9, 239–245.

Ryckaert, J.-P., Bellemans, A., 1978. Molecular dynamics of liquid alkanes. Discuss. Faraday Soc. 66, 95–106.

Sadus, R.J., 2006. Molecular simulation of the thermophysical properties of fluids: phase behaviour and transport properties. Mol. Sim. 32, 185–189.

Sarman, S., Evans, D.J., 1992a. Heat flow and mass diffusion in binary Lennard-Jones mixtures. Phys. Rev. A 45, 2370–2379.

Sarman, S., Evans, D.J., 1992b. Heat flow and mass diffusion in binary Lennard-Jones mixtures. II. Phys. Rev. A 46, 1960–1966.

Schoen, M., Hoheisel, C., 1984a. The mutual diffusion coefficient D_{12} in binary liquid model mixtures. Molecular dynamics calculations based on Lennard-Jones (12-6) potentials. I. The method of determination. Mol. Phys. 52, 33–56.

Schoen, M., Hoheisel, C., 1984b. The mutual diffusion coefficient D_{12} in binary liquid model mixtures. A molecular dynamics study based on Lennard-Jones (12-6) potentials. II. Lorentz-Berthelot mixtures. Mol. Phys. 52, 1029–1042.

Schoen, M., Hoheisel, C., 1985. The shear viscosity of a Lennard-Jones fluid calculated by equilibrium molecular dynamics. Mol. Phys. 56, 653–672.

Schultz, A.J., Kofke, D.A., 2018. Comprehensive high-precision high-accuracy equation of state and coexistence properties for classical Lennard-Jones crystals and low-temperature fluid phases. J. Chem. Phys. 149, 204508.

Searles, D.J., Evans, D.J., Hanley, H.J.M., Murad, S., 1998. Simulations of the thermal conductivity in the vicinity of the critical point. Mol. Sim. 20, 385–395.

Simmons, A.D., Cummings, P.T., 1986. Non-equilibrium molecular dynamics simulation of dense fluid methane. Chem. Phys. Lett. 129, 92–98.

Singer, K., Singer, J.V.L., Fincham, D., 1980. Determination of the shear viscosity of atomic liquids by non-equilibrium molecular dynamics. Mol. Phys. 40, 515–519.

Tang, W.H., Kostov, K.S., Freed, K.F., 1998. Theory for the nonequilibrium dynamics of flexible chain molecules: relaxation to equilibrium of pentadecane from an all-trans conformation. J. Chem. Phys. 108, 8736–8742.

Tenenbaum, A., Ciccotti, G., Gallico, R., 1982. Stationary non-equilibrium states by molecular dynamics. Fourier's law. Phys. Rev. A 25, 2778–2787.

Todd, B.D., Daivis, P.J., 1998. Nonequilibrium molecular dynamics simulations of planar elongational flow with spatially and temporally periodic boundary conditions. Phys. Rev. Lett. 81, 1118–1121.

Todd, B.D., Daivis, P.J., 2017. Nonequilibrium Molecular Dynamics: Theory, Algorithms and Applications. Cambridge University Press, Cambridge.

Toxvaerd, S., 1990. Molecular dynamics calculation of the equation of state of alkanes. J. Chem. Phys. 93, 4290–4295.

Trozzi, C., Ciccotti, G., 1984. Stationary nonequilibrium states by molecular dynamics. II. Newton's law. Phys. Rev. A 29, 916–925.

Tuckerman, M.E., Mundy, C.J., Balasubramanian, S., Klein, M.L., 1997. Modified nonequilibrium molecular dynamics for fluid flows with energy conservation. J. Chem. Phys. 106, 5615–5621.

Tuckerman, M.E., Mundy, C.J., Balasubramanian, S., Klein, M.L., 1998. Response to "Comment on 'Modified nonequilibrium molecular dynamics for fluid flows with energy conservation'" [J. Chem. Phys. 108, 4351 (1998)]. J. Chem. Phys. 108, 4353–4354.

Vlasiuk, M., Sadus, R.J., 2017. *Ab initio* interatomic potentials and the thermodynamic properties of fluids. J. Chem. Phys. 147, 024505.

Vogelsang, R., Hoheisel, C., 1988. The Dufour and Soret coefficients of isotopic mixtures from equilibrium molecular dynamics calculations. J. Chem. Phys. 89, 1588–1591.

Wang, B.Y., Cummings, P.T., 1989a. Non-equilibrium molecular dynamics calculation of the shear viscosity of carbon dioxide. Int. J. Thermophys. 10, 929–940.

Wang, B.Y., Cummings, P.T., 1989b. Non-equilibrium molecular dynamics calculation of the transport properties of carbon dioxide. Fluid Phase Equilib. 53, 191–198.

Wang, B.Y., Cummings, P.T., 1996. In: Gubbins, K.E., Quirke, N. (Eds.), Molecular Simulation and Industrial Applications: Methods, Examples and Prospects. Gordon and Breach, Amsterdam.

Wheeler, D.R., Rowley, R.L., 1998. Shear viscosity of polar liquid mixtures via non-equilibrium molecular dynamics: water, methanol, and acetone. Mol. Phys. 94, 555–564.

Wold, I., Hafskjold, B., 1999. Nonequilibrium molecular dynamics simulations of coupled heat and mass transport in binary fluid mixtures in pores. Int. J. Thermophys. 20, 847–856.

This page intentionally left blank

Chapter 9

Molecular simulation of ensembles

Previous chapters have described most of the fundamental aspects of molecular simulation methods. However, molecular simulations are almost invariably conducted in the context of an ensemble. The choice of the ensemble determines the overall simulation algorithm. Monte Carlo (MC) and molecular dynamics (MD) algorithms for a specified ensemble are very different, reflecting fundamental differences in the two simulation methods. In this chapter, MC and MD simulation algorithms for commonly used ensembles are outlined. This topic has also been discussed by Allen and Tildesley (2017), Heermann (1990), Rapaport (2004), Frenkel and Smit (2023), and Heyes (1998).

The original MC method (Metropolis et al., 1953) sampled the canonical (*NVT*) ensemble. Metropolis et al. (1953) introduced the concept of importance sampling to construct a Markov chain. The MC method can be extended relatively easily to other ensembles because importance sampling can be used for any Markov chain process. In general, the MC method can be extended to other ensembles by sampling the variable that is conjugate to the new fixed parameter. For example, McDonald (1972) reported MC simulations in the isobaric−isothermal (*NpT*) ensemble by sampling volume (the conjugate of pressure) in addition to molecular coordinates. In the grand canonical ensemble, the chemical potential is fixed and the total number of molecules N is the conjugate variable that is sampled (Valleau and Cohen, 1980). In the past, it has not been possible to perform MC simulations in the microcanonical ensemble. However, The MC method has been extended specifically (Ray, 1991; Lustig, 1998) for the microcanonical ensemble.

Newton's equations of motion lead naturally to the microcanonical (*NVE*) ensemble because the total energy is conserved. Therefore, in contrast to MC simulations, the microcanonical ensemble is the natural choice for MD. Extending MD to other ensembles requires some artificial tampering with the equations of motion. A canonical (*NVT*) ensemble can be obtained by either using velocity scaling or explicitly adapting the

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00003-4

equations of motion (Nosé, 1984; Hoover, 1985). Andersen (1980) formulated an isobaric−isoenthalpic (*NpH*) and isobaric−isothermal (*NpT*) MD algorithms by introducing additional degrees of freedom that are coupled to the particle coordinates. This work has been extended rigorously (Nosé, 1984; Hoover, 1985). Considerable progress has been achieved in the development of MD algorithms for the grand canonical ensemble (Çagin and Pettitt, 1991a & b; Lo and Palmer, 1995; Beutler and van Gunsteren, 1994). The calculation of thermodynamic properties for different ensembles is discussed in Chapter 2. MC and MD simulation algorithms for different ensembles in a single phase are discussed here, whereas techniques for multiphase equilibria are examined in Chapter 10.

9.1 Monte Carlo

All MC simulations use probabilities in conjunction with importance sampling to generate a Markov chain of events. The theoretical basis of the MC method, Markov chains, and importance sampling has been discussed extensively elsewhere (Allen and Tildesley, 2017; Sobol', 1994; Frenkel and Smit, 2023). From a practical point of view, these concepts can be best understood in the context of determining ensemble averages. Indeed, the concept of importance sampling was first introduced (Metropolis et al., 1953) in the context of simulation of the canonical ensemble, which will be the starting point for our discussion. In a later section, we will show how simulations in the canonical ensemble can be extended to other ensembles.

9.1.1 Canonical (*NVT*) ensemble

From the definition of the canonical (*NVT*) ensemble (Chapter 2), the canonical average of any function of particle coordinates $\langle A \rangle$ can be obtained from,

$$\langle A \rangle = \frac{\int A(\mathbf{r}^N)\exp\left(-\beta U(\mathbf{r}^N)\right)d\mathbf{r}^N}{\int \exp(-\beta U(\mathbf{r}^N))d\mathbf{r}^N} \tag{9.1}$$

where $\beta = 1/kT$ and U is the total potential energy. In principle, by generating a large number of configurations (M) of particles, $\langle A \rangle$ can be estimated by replacing the integrals with finite sums.

$$\langle A \rangle = \frac{\sum_{i=1}^{M} A(i)\exp[-\beta U(i)]}{\sum_{i=1}^{M} \exp[-\beta U(i)]} \tag{9.2}$$

However, in practice, this simple approach is very inefficient because many of the configurations sampled are unlikely to make a substantial

contribution to $\langle A \rangle$. To sample phase-space efficiently, a mechanism is required to ensure that the finite configurations sampled are genuinely representative of the system. This is the concept of importance sampling, which ensures that regions of configuration space that make the largest contribution to the integrals of Eq. (9.1) are also the regions that are sampled most frequently. However, importance sampling also introduces a bias to the sampling. This bias can be removed from the ensemble averages by weighting each configuration appropriately. If $W(i)$ is the probability of choosing a configuration i, Eq. (9.2) is replaced by:

$$\langle A \rangle = \frac{\sum_{i=1}^{M} \frac{A(i)\exp(-\beta U(i))}{W(i)}}{\sum_{i=1}^{M} \frac{\exp(-\beta U(i))}{W(i)}} \tag{9.3}$$

The Boltzmann distribution can be used as the weighting factor.

$$W(i) = \exp(-\beta U(i)) \tag{9.4}$$

By substituting Eq. (9.4) into Eq. (9.3), we obtain:

$$\langle A \rangle = \frac{1}{N} \sum_{i=1}^{M} A(i) \tag{9.5}$$

Eq. (9.5) indicates that the canonical ensemble is obtained as an unweighted average over configurations in the sample. Therefore, as will be illustrated subsequently, the trick of transforming the MC method to other ensembles is to choose an appropriate weighing factor such that Eq. (9.5) is obtained.

The procedure for a MC simulation in the canonical ensemble is to displace a particle (i) randomly and determine the change in potential energy $\Delta U = U_{trial} - U_i$. If $\Delta U < 0$, the move is accepted because it lowers the energy of the system, and the particle is placed in its new position. If $\Delta U > 0$, the move is allowed with a probability of $\exp(-\beta\Delta U)$. To decide the fate of the particle, a random number R is generated between 0 and 1. If $R > \exp(-\beta\Delta U)$, the particle remains to its original position otherwise the move is accepted. It is important to note that irrespective of the outcome of the move, that is, whether a different configuration has been generated or the original configuration has been retained, a new configuration is created for the purpose of the ensemble average. This process generates a so-called Markov chain in which the probability of obtaining any configuration is determined solely by the previous configuration. This procedure is illustrated in Algorithm 9.1.

In common with all MC simulations, Algorithm 9.1 involves an initialization phase during which ensemble and simulation parameters are specified

(Algorithm 9.1, Part 1) and an initial configuration is generated. Typically, the initial state is created by positioning particles on a face-centered cubic lattice common to many solids. If we assume pairwise interaction, the initial potential energy (*Utotal*) is evaluated by calculating the contribution of all distinct pair interactions using a suitable intermolecular potential. This is the only time that a double-loop is require in a typical *NVT* MC simulation. Initial values for other ensemble properties such as the virial and pressure are also evaluated.

The heart of a MC simulation is the generation of a Markov chain (Algorithm 9.2, Part 2). The Markov chain is generated typically for several thousand cycles (*NCycles*) with each cycle consisting of many MC moves. Each new link or configuration in the Markov chain is created by generating trial positions for all the atoms in turn. Each atom *i* is selected randomly, typically using a random number generator (rand()) that returns numbers on the interval from 0 to 1 (Algorithm 9.1, Part 2.1). Selecting the atoms randomly is best practice for MC, although simply cycling through all the full range is also commonly used. The coordinates of the chosen atom are displaced randomly within a predetermined maximum range of -*rMax* to *rMax*, adjusted for periodic boundary conditions(pbc(), Algorithm 5.2)

The contribution to the total potential energy (*Utriali*) of atom *i* in the trial position is evaluated (Algorithm 9.1, Part 2.2). The evaluation of *Utriali* involves using an intermolecular potential to calculate the energy of interaction between atom *i* and the remaining *N* - *1* atoms. Consequently, a MC algorithm for pairwise interaction is of order *N* (Algorithm 1.1). The change in potential energy (ΔU) is the difference in energy between atom *i* at its trial and original positions (Algorithm 9.1, Part 2.3). The next part of the algorithm (Algorithm 9.1, Part 2.4) is to determine whether or not the trial move is accepted. If the trial move is accepted, the position of atom *i* and the energy of the ensemble are updated. Otherwise, no change is made and the next atom is considered. Both changed and unchanged states contribute to the Markov chain.

During the equilibration period, the maximum displacements (*rMax*) used for the trail moves are continually adjusted (Part 2.5.1) to achieve a desired acceptance rate (*acceptR*), which involves continually updating the value of the progressive acceptance ratio (*acceptRatio*).

It should also be noted that the system is allowed to come to equilibrium (*NEquilibration*) before contributions to the ensemble average are accumulated (Part 2.5.2). The equilibration period is important for obtaining reliable ensemble averages because the initial state generally does not reflect accurately the configuration at equilibrium. Therefore, the equilibration period is used to "melt" the solid lattice to create a fluid state.

Algorithm 9.1 is the basis of MC simulations of a wide range of applications (Dubbeldam et al., 2013), including thermodynamic properties (Vlasiuk et al., 2016).

ALGORITHM 9.1 MC simulation in the canonical (*NVT*) ensemble.

Overall Algorithm

Part 1 Initialization

Assign values for N, V, T, *NCycles*, *DispCounter* = 0, *acceptR* = 0.5
Assign initial coordinates rx_i, ry_i, rz_i for the N atoms.
Evaluate and store *Utotal*, U_i, *ptotal*, p_i etc.

Part 2 Generate a Markov Chain

loop $n \leftarrow 1 \ldots$ *NCycles*
  $k \leftarrow 1$
  loop while ($k <=$ *NAtom*)
2.1 Generate a trial displacement.
2.2 Calculate energy of atom i at the trial position (*Utriali*).
2.3 Calculate change in the potential energy:
  $\Delta U \leftarrow$ *Utriali* - U_i
2.4 Displacement acceptance criterion.
  $k \leftarrow k + 1$
  end while loop
2.5 **if** (n <= *NEquilibration*)
2.5.1 Adjust maximum displacement.
  else
2.5.2 Accumulate averages. //post equilibration period
  end if
end n **loop**

Details of Some Key Sub-Parts

Part 2.1 Generate a trial displacement:
  Select i randomly, assuming indexes starting from 1:
  $i \leftarrow$ nearest integer value of rand() $\times$ *NAtom*
  rxTrial $\leftarrow$pbc(rx_i + *rMax* $\times$ (2 $\times$ rand() - 1))
  ryTrial $\leftarrow$pbc(ry_i + *rMax* $\times$ (2 $\times$ rand() - 1))
  rzTrial $\leftarrow$pbc(rz_i + *rMax* $\times$ (2 $\times$ rand() - 1))

Part 2.4 Displacement acceptance criterion:
  if ($\Delta U < 0$) //accept new positions
    $rx_i \leftarrow$ *rxTrial*
    $ry_i \leftarrow$ *ryTrial*
    $rx_i \leftarrow$ *rzTrial*
    $E_i \leftarrow$ *Etriali*
    Utotal $\leftarrow$ *Utotal* + ΔU //update energy, pressure etc
    DispCounter $\leftarrow$ *DispCounter* + 1
  else

```
        BF ← exp(-βΔU)                    //Boltzmann factor
      end if
      if (BF > rand())                    //accept new positions
        rx_i ← rxTrial
        ry_i ← ryTrial
        rz_i ← rzTrial
        U_i ← Utriali
        Utotal ← Utotal + ΔU              //update energy, pressure etc
        DispCounter ← DispCounter + 1
      end if
    end if

Part 2.5.1 Adjust maximum displacement:
    acceptRatio = DispCounter/n
    if (acceptRatio < acceptR)
      rMax ← 0.95 × rMax
    else
      rMax ← 1.05 × rMax
    end if
```

As discussed subsequently, other MC ensemble simulations can be performed by modifying the acceptance criterion used in Algorithm 9.1. In general, it is useful to define the following "pseudo-Boltzmann factor,"

$$W(i) = \exp(-\beta Y) \tag{9.6}$$

Where the nature of Y depends on the characteristics of the ensemble. For the canonical ensemble, Y is simply U; however, additional terms are required to calculate Y for other ensembles.

9.1.2 Isobaric–Isothermal (*NpT*) ensemble

At conditions of constant temperature and pressure, the equivalent of Eq. (9.2) (McDonald, 1972) is:

$$\langle A \rangle = \frac{\int_0^\infty \exp(-\beta pV) dV \int_V A(\mathbf{r}^N, V) \exp[-\beta U(\mathbf{r}^N)] d\mathbf{r}^N}{\int_0^\infty \exp(-\beta pV) dV \int_V \exp[-\beta U(\mathbf{r}^N)] d\mathbf{r}^N} \tag{9.7}$$

In a MC simulation, the particles are confined to a cube of fluctuating length (L). Consequently, we can make use of scaled coordinates ($\boldsymbol{\alpha}_i = \mathbf{r}_i/L$) such that the integrals over particle coordinates in Eq. (9.7) become integrals over the unit cube (Ω).

$$\langle A \rangle = \frac{\int_0^\infty V^N \exp(-\beta pV) dV \int_\Omega A([L\boldsymbol{\alpha}]^N, V) \exp\left(-\beta U([L\boldsymbol{\alpha}]^N, L)\right) d\boldsymbol{\alpha}^N}{\int_0^\infty V^N \exp(-\beta pV) dV \int_\Omega \exp\left(-\beta E([L\boldsymbol{\alpha}]^N, L)\right) d\boldsymbol{\alpha}^N} \tag{9.8}$$

By analogy with Eq. (9.4), we can define a pseudo-Boltzmann weighting factor by using (9.6) with:

$$Y = pV + U\left([L\alpha]^N, L\right) - NkT\ln V \tag{9.9}$$

The algorithm for performing the *NpT* simulation is therefore a relatively simple modification on the *NVT* algorithm. In addition to attempted particle displacements, which are accepted or rejected depending of the value of ΔU, attempted volume fluctuations are required to maintain constant pressure. The value of ΔY will determine whether or not an attempted volume change is accepted. In principle, attempting a volume change is more expensive computationally than particle displacement because it requires the re-evaluation of $N(N-1)$ interactions compared with $N-1$ calculations for particle displacement. However, this computational overhead can be often avoided by using a "scalable" intermolecular potential.

Formally, the nth derivative of U with respect to volume V can be calculated from (Lustig, 1994).

$$\frac{\partial^n U}{\partial V^n} = \frac{1}{2(3V)^n}\sum_{k=1}^{n} a_{nk} r_{ij}^k \left(\frac{\partial^k u(\mathrm{r}_{ij})}{\partial r_{ij}^k}\right) \tag{9.10}$$

In Eq. (9.10), u is the intermolecular potential and the coefficients a_{nk} are Sterling numbers of the second kind (Abramowitz and Stegun, 1970), which can be calculated from the following recursive algorithm.

$$\left.\begin{aligned} a_{n+1,k} &= a_{n,k-1} + k a_{nk} \\ a_{n0} &\equiv 0 \\ a_{nn} &= 1 \end{aligned}\right\} \tag{9.11}$$

When $n = 1$ we obtain:

$$\frac{\partial U}{\partial V} = \frac{1}{6V}\sum_{i}\sum_{i\neq j} r_{ij}\left(\frac{\partial u(r_{ij})}{\partial r_{ij}}\right) \tag{9.12}$$

The usefulness of these equations for determining thermodynamic properties is detailed in Chapter 2.

It is evident from the earlier equation that a volume fluctuation can be very expensive computationally because it affects all particles. The change in volume also means that long-range corrections must also be re-evaluated. However, for many simple intermolecular potentials, the need to re-calculate all the pairwise interactions can be avoided by using scaling. For example, if a Lennard-Jones (LJ)-type intermolecular potential is used, the energy of the trial state can be calculated from:

$$U = 4\varepsilon\sum_{i}\sum_{j>i}\left(\frac{\sigma_{ij}}{L\alpha_{ij}}\right)^{12} - 4\varepsilon\sum_{i}\sum_{j>i}\left(\frac{\sigma_{ij}}{L\alpha_{ij}}\right)^{6} \tag{9.13}$$

Equation (9.13) indicates that there are two distinct contributions to the energy with a different power dependence. To make use of scaling, the 12th and 6th dependence must be stored separately. Therefore, we define the potential energy as:

$$U = U(12) + U(6) \tag{9.14}$$

Consequently, the effect of changing the length of the box on the potential energy can be calculated from:

$$U_{trial} = U(12)\left(\frac{L_{trial}}{L}\right)^{12} + U(6)\left(\frac{L_{trial}}{L}\right)^{6} \tag{9.15}$$

The same procedure can be applied to obtain the virial.

The scaling procedure can only be applied if there is only one characteristic length in the potential function. Consequently, its usefulness is limited to relatively simple intermolecular potentials. It is not appropriate for any molecular models, which involve intramolecular bond lengths in addition to site−site interactions. For these cases, the computationally expensive $N(N-1)$ pair calculation may be unavoidable.

A possible implementation of the *NpT* MC procedure is illustrated by Algorithm 9.2. The initialization phase (Algorithm 9.2, Part 1) of Algorithm 9.2 is similar to the initialization phase of Algorithm 9.1 except that the "scalability" of the intermolecular potential is identified. Whether or not the potential is scalable will influence how the volume fluctuation is handled (Algorithm 9.2, Parts 2.3 and 2.5). As discussed previously, a scalable potential avoids the recalculation of the total energy and as such is very computationally advantageous.

Each configuration in the Markov chain for a *NpT* ensemble is the result of two different possible trial moves involving atomic displacement (Algorithm 9.2, Part 2.1) and volume change (Algorithm 9.2, Part 2.2), respectively. The displacement stage (Algorithm 9.2, Part 2.1) of the Markov process proceeds identically to the *NVT* ensemble (Algorithm 9.1, Parts 2.1 to 2.4). It involves the generation of trial coordinates, calculation of the energy experienced at the trial position and determining the overall change in the energy of the ensemble. At this stage, the volume is unaffected, and the acceptance criterion reduces to the *NVT* ensemble criterion (Algorithm 9.1, Part 2.4).

The next stage (Algorithm 9.2, Part 2.2) of the Markov process is to attempt a volume fluctuation. The trial volume is obtained by resizing randomly the length of the simulation box by a predefined amount with the range of *-LMax* to *LMax*. If the potential is scalable (Algorithm 9.2, Part 2.3), the energy of the ensemble is scaled by the new volume otherwise the trial energy is calculated by considering all pair interactions at the trial coordinates. The change in energy and volume of the ensemble (Algorithm 9.2, Part 2.4) are used to determine ΔY. The acceptance of the volume fluctuation (Algorithm 9.2, Part 2.5) is determined by the value of ΔY. If the trail volume

move is accepted, the assignment of the updated coordinates depends on whether or not the potential was scalable as illustrated in Part 2.5. In common with the displacement step, the acceptance of volume changes is continually monitored during the equilibration phase to achieve a specified acceptance value (acceptV). The nature of the adjustments (Algorithm 9.2, Part 2.5.1) follows the same procedure as for the displacement step.

ALGORITHM 9.2 MC simulation in the *NpT* ensemble.

Overall Algorithm

Part 1 Initialization

Assign values for N, p, T, *NCycles*, *DispCounter* = 0, *acceptR* = 0.5, VolCounter = 0, acceptV = 0.5 etc.
Assign initial coordinates rx_i, ry_i, rz_i for the N atoms.
Evaluate and store *Utotal*, U_i, *ptotal*, p_i etc.
if (intermolecular potential is scalable)
  scalable ← *true*
else
  scalable ← *false*
end if

Part 2 Generate a Markov Chain

loop $n \leftarrow 1 \ldots NCycles$
  $k \leftarrow 1$
  loop while ($k <= NAtom$)
2.1 Implement Parts 2.1-2.4 of Algorithm 9.1 (*NVT* Ensemble)
    $k \leftarrow k + 1$
  end while loop
2.2 Generate a trial move by attempting a volume change:
  $Ltrial \leftarrow L + LMax \times (2 \times \text{rand}() - 1)$
  $Vtrial \leftarrow Ltrial^3$
2.3 Evaluate Utrial:
  if (*scalable*)
    Scale *Utotal* to obtain *UtrialTotal*.
  else
    loop $i \leftarrow 1 \ldots NAtom$ //scale coordinates
      $rxTrial_i \leftarrow rx_i \times (L/Ltrial)$
      $ryTrial_i \leftarrow ry_i \times (L/Ltrial)$
      $rzTrial_i \leftarrow rz_i \times (L/Ltrial)$
    end *i* **loop**
    Perform N(N-1) calculations for *UtrialTotal, Utriali.*
  end if
2.4 Calculate change in energy and acceptance term.
  $\Delta U \leftarrow UtrialTotal - Utotal$
  $\Delta Y \leftarrow p \times (Vtrial - V) + \Delta E + N \times k \times T \times \ln(V/Vtrial)$
2.5 Volume change acceptance criterion.
  if ($n <= NEquilibration$)
2.5.1 Adjust maximum volume change.
    Adjust maximum displacement change (see Algorithm 9.1).
  else

```
            Accumulate averages.
        end if
    end n loop

                    Details of Some Key Sub-Parts

Part 2.5  Volume change acceptance criterion:
    if (ΔY < 0)                          //accept new configuration
        if (scalable)
            loop i ← 1 .. NAtom          //scale coordinates
                rx_i ← rx_i × (L/Ltrial)
                ry_i ← ry_i × ( L/Ltrial)
                rz_i ← rz_i × (L/Ltrial)
                Scale U_i
            end i loop
        else
            loop i ← 1 .. NAtom          //update coordinates
                rx_i ← rxTrial_i
                ry_i ← ryTrial_i
                rz_i ← rzTrial_i
                U_i ← Utriali
            end i loop
        end if
        L ←Ltrial                        //update properties
        V ←Vtrial
        Utotal ←UtrialTotal
        VolCounter ← VolCounter + 1
    else
        PBF ← exp(-βΔY)                  //pseudo-Boltzmann factor
        if (PBF > rand())                //accept new configuration
            if (scalable)
                loop i ← 1 .. NAtom      //scale coordinates
                    rx_i ← rx_i × (L/Ltrial)
                    ry_i ← ry_i × (L/Ltrial)
                    rz_i ← rz_i × (L/Ltrial)
                    Scale E_i
                end i loop
            else
                loop i ← 1 .. NAtom      //update coordinates
                    rx_i ← rxTrial_i
                    ry_i ← ryTrial_i
                    rz_i ← rzTrial_i
                    Scale U_i
                end i loop
            end if
                L ←Ltrial                //update properties
                V ←Vtrial
                Utotal ←Utotal + ΔU
                VolCounter ← VolCounter + 1
            end if
    end if

Part 2.5.1  Adjust maximum volume change:
    acceptVRatio = VolCounter/n
    if (acceptVRatio < acceptV)
        LMax ← 0.95 × LMax
    else
        LMax ← 1.05 × LMax
    end if
```

It should be noted that there are different possible combinations of trial moves. In Algorithm 9.2, a displacement is attempted for each atom followed by a single attempt to change the volume. However, both attempted displacements and volume changes could be combined in the same trial move but this would complicate the algorithm. It is also possible to choose randomly between particle displacement and volume change. The choice should be biased heavily toward particle displacement to ensure that equilibrium is achieved rapidly. In practice, one attempted volume change per cycle is often sufficient to bring the system to equilibrium. The efficiency of volume changes is discussed in greater detail by Brennan and Madden (1998). Some useful improvements to the basic algorithm have also been discussed (Okumura and Okamoto, 2004a &b; McGrath et al., 2005). The approach can also be adapted for MC simulations in the isobaric−isenthalpic (*NpH*) ensemble (Ströker and Meier, 2022).

9.1.3 Grand canonical (μVT) ensemble

MC simulation in the grand canonical ensemble has been discussed by several workers (Adams, 1974; Valleau and Cohen, 1980; Nicholson and Parsonage, 1982). The distinctive feature of the grand canonical ensemble is that number of particles fluctuates while maintaining a constant chemical potential (μ), volume, and temperature. The average of any property of the grand canonical ensemble can be obtained from

$$\langle A\rangle = \frac{\sum_{N=0}^{\infty} \frac{\Lambda^{-3N}}{N!} \exp(\beta N\mu) \int A(\mathbf{r}^N)\exp\left(-\beta U(\mathbf{r}^N)\right) d\mathbf{r}^N}{Z_{\mu VT}} \tag{9.16}$$

where $Z_{\mu VT}$ is the canonical partition function and Λ is the de Broglie thermal wavelength. Using scaled coordinates and making use of the relationship $\beta\mu_{\text{ideal}} = \log\,(N/V) + 3\log\,\Lambda$, Eq. (9.16) can be rewritten (Adams, 1974; Hansen and McDonald, 2013) as,

$$\langle A\rangle = \frac{\sum_{N=0}^{\infty} \exp(\beta N\mu^* - \ln N!) \int_{\Omega} A\exp(-\beta U) d\boldsymbol{\alpha}^{\mathrm{N}}}{Z_{\mu VT}} \tag{9.17}$$

where μ^* involves the contribution of the residual[1] chemical potential (μ^{r}).

$$\mu^* = \mu^r + kT\ln\langle N\rangle \tag{9.18}$$

1. As noted in Chapter 2, a residual quantity is sometimes mistakenly referred to as an excess property, whereas in chemical thermodynamics, the latter term should be applied only to mixture properties.

The pseudo-Boltzmann factor for this ensemble is given by Eq. (9.6) with:

$$Y = -N\mu^r + kT\ln N! + U\left(\mathbf{r}^N\right) \tag{9.19}$$

The grand canonical ensemble involves the possible removal and/or addition of particles. If there is no change in the number particles, it is apparent from Eq. (9.19) that $\Delta Y = \Delta U$ and Eq. (9.8) is simply the Boltzmann average used in the canonical *NVT* ensemble. Therefore, an algorithm for the grand canonical ensemble can be devised consisting of three attempted moves: particle displacement (identical to the canonical ensemble); particle insertion; and particle removal.

Algorithm 9.3 illustrates one possible procedure for conducting a grand canonical MC simulation. Following initialization (Algorithm 9.3, Part 1), a Markov chain is created (Algorithm 9.3, Part 2) by implementing molecular displacement (Algorithm 9.3, Part 2.1), particle creation (Algorithm 9.3, Part 2.2), or particle destruction (Algorithm 9.3, Part 2.3). The type of move implemented is chosen with equal probability. A particle displacement corresponds to a conventional MC *NVT* ensemble move. In Algorithm 9.3, when a displacement move is chosen, an attempt is made to displace all the particles. Alternatively, only one particle could be displaced per move and the number of cycles could be increased considerably. The particle insertion move (Algorithm 9.3, Part 2.2) involves selecting coordinates randomly and determining the energy that the atom would experience in the trial location. This evaluation of energy involves calculating the particle's interaction with the N particles. The interaction energy is used to calculate ΔY, which is used in evaluating the acceptance criterion (Algorithm 9.3, Part 2.2.1). The particle removal move (Algorithm 9.3, Part 2.3) proceeds in a similar fashion except that an existing particle must be selected randomly. The algorithm proceeds directly to the evaluation of ΔY and the application of the acceptance criterion (Algorithm 9.3, Part 2.3.1) because the energy of the selected particle is known.

In an object-oriented language such as C++ (Chapter 11), particle addition and removal can be achieved by creating and deleting an object, respectively. Alternately, if arrays are used, the addition of a particle can be handled by simply increasing the size of the array by 1 and assigning the coordinates of the added particle to rx_{m+1}, ry_{m+1}, and rz_{m+1}, where m is the last index of the original array. Particle deletion can be accomplished by assigning the array index of the deleted particle to the mth particle and then changing the last array index to $m - 1$.

The importance of particle creation and destruction in the μVT ensemble has been the impetus to development of novel approaches (Orkoulas, 2007; Shi and Maginn, 2007), which are particularly important when large macromolecules are involved. (Hu et al., 2005; Turesson et al., 2008). As discussed in Chapter 2, new approaches (Stutzman et al., 2018; Ströker et al., 2021)

for determining thermodynamic properties in the μVT ensemble have been developed. It should be noted that the μVT approach generally fails at high densities because insertion attempts are overwhelmingly rejected, which means it is inappropriate for solids. Suitable algorithms for solid−liquid equilibrium are discussed in Chapter 10.

ALGORITHM 9.3 MC simulation in the μVT ensemble.

Overall Algorithm

Part 1 Initialization

Assign values for μ, N, V, T, NCycles, *DispCounter* = 0,
acceptR = 0.5,etc.
Assign initial coordinates rx_i, ry_i, rz_i for the *N* atoms.
Evaluate and store *Utotal*, E_i, *ptotal*, p_i etc.

Part 2 Generate a Markov Chain

loop *n* ←*1... NCycles*
  choice ← rand()
  if (0 < *choice* ≤ 1/3)
    k ←*1*
    loop while (*k* <= *NAtom*)
2.1 Implement Parts 2.1-2.4 of Algorithm 9.1 (*NVT* Ensemble)
    k ← *k*+ *1*
    end while loop
  else if (1/3 > *choice* ≤ 2/3)
2.2 Attempt to insert an atom:
    Randomly select a position (*rxTrial, ryTrail, rxTrial*).
    Evaluate U_i.
    $\Delta Y \leftarrow -\mu^* + kT\ln(N+1) + U_i$
2.2.1 Insertion acceptance criterion.
  else
2.3 Attempt to remove an atom:
    Select randomly an atom (*i*) to be removed.
    $\Delta Y \leftarrow \mu^* - kT\ln(N) + U_i$
2.3.1 Removal acceptance criterion.
  end if
2.4 **if** (*n* <= *NEquilibration*)
2.4.1 Adjust maximum displacement change (see Algorithm 9.1).
  else
    Accumulate averages.
  end if
end *n* **loop**

Details of Some Key Sub-Parts

Part 2.2 Insertion acceptance criterion:
  if ($\Delta Y < 0$)
    Accept new atom.
    Utotal ←*Utotal* + U_i

```
    else
        PBF ← exp(-βΔY)
        if (PBF > rand())
            Accept new atom.
            Utotal ←Utotal + Ui
        end if
    end if

Part 2.3.1 Removal acceptance criterion:
    if (ΔY < 0)
        Remove selected atom.
        Utotal ←Utotal - Ui
    else
        PBF ← exp(-βΔY)
        if (PBF > rand())
            Remove selected atom.
            Utotal ←Utotal - Ui
        end if
    end if
```

9.1.4 Microcanonical (*NVE*) ensemble

The microcanonical ensemble conserves the number of particles, volume, and total energy (E). As discussed in the preceding sections, the basis of MC simulations is the algorithm of Metropolis et al. (1953), which generates the canonical (NVT) ensemble. This algorithm can be incorporated into simulations for the isothermal−isobaric and grand canonical ensembles but it cannot be used to generate a microcanonical ensemble. However, theoretical advances (Creutz, 1983; Ray, 1991; Lustig, 1998) have made MC simulations possible.

Creutz (1983) reported the first MC simulations in the microcanonical ensemble. This particular solution to the problem was confined to lattice-based systems and involves the use of an energy "demon" in conjunction with a random walk through possible configurations that satisfy the constraint of constant total energy. Later, Ray (1991) reported a general solution to the problem based on the statistical mechanical formulation of Pearson et al. (1985). Ray's procedure involves Metropolis-type moves and uses the acceptance criterion $P_E = \min(1, W_E(\mathrm{r}')/W_E(\mathrm{r}))$, where the probability density (W_E) plays the same role as the "pseudo-Boltzmann factor" in the other ensembles.

$$W_E(\mathrm{r}) = C\left[E - U(\mathbf{r}^N)\right]^{3N/2\ -1}\Theta\left[E - U(\mathbf{r}^N)\right] \tag{9.20}$$

In Eq. (9.20), C is a constant and $\Theta(x) = 1$ for $x > 0$ and $\Theta(x) = 0$ for $x \leq 0$. The corresponding partition function for the ensemble is:

$$Z_{NVE} = C(N) \int \left[E - U(\mathbf{r}^N)\right]^{3N/2} \Theta\left[E - U(\mathbf{r}^N)\right] d\mathbf{r}^N \tag{9.21}$$

The relationships $S = k\ln(Z_{NVE})$ and $T^{-1} = \partial S/\partial E$ enable us to determine the temperature.

$$T = \frac{2\left\langle E - U(\mathbf{r}^N)\right\rangle}{3Nk} = \frac{2\langle K\rangle}{3Nk} \tag{9.22}$$

In Eq. (9.22), we have identified the kinetic energy (K) as simply $E - U(\mathbf{r}^N)$. Strictly, the constraints imposed (Lustig, 1998; Hüneneberger, 2005) by the MD simulation should be subtracted from the *3 N* contribution in Eq. (9.22). However, in practice this has a negligible effect because of large values of N typically used in modern simulations. The pressure is obtained by differentiating the entropy with respect to volume.

$$p = \frac{NkT}{V} - \left\langle \frac{\partial U}{\partial V} \right\rangle \tag{9.23}$$

If the entropy is differentiated with respect to energy, the specific heat at constant volume (C_V) can be calculated (Chapter 2).

$$\frac{Nk}{C_V} = 1 - \left(1 - \frac{2}{3N}\right)\langle K\rangle\left\langle K^{-1}\right\rangle \tag{9.24}$$

Application of MC simulations for the microcanonical ensemble has been discussed by Lustig (1998).

Equation (9.22) is significant because it enables the calculation of temperature from a MC simulation. The temperature depends on the momenta of the atoms and it can be calculated easily in a MD simulation. In contrast, in a MC simulation, the temperature is a fixed variable, which cannot be obtained from the simulation because the momenta of the atoms cannot be calculated in MC procedure. Equation (9.22) overcomes this problem by calculating temperature from the configurational properties of the ensemble without requiring any dynamical information. Butler et al. (1998) have shown that the temperature can be also calculated from the following formula,

$$\frac{1}{kT} = \left\langle \frac{\nabla_q^2 U}{\left|\nabla_q U\right|^2} \right\rangle \tag{9.25}$$

where$\nabla_q$is the gradient operator with respect to the particle positions (q). Equation (9.25) implies that the temperature can be calculated by evaluating

the first and second derivatives of the intermolecular potential during the course of the simulation. For a pairwise potential the scalar separation between particle i and j in terms of the x, y, and z components is given by $r_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2}$ and the derivatives that are required to evaluate the contributions to Eq. (9.25) can be obtained from:

$$\left.\begin{aligned} \frac{\partial U}{\partial \mathbf{r}_i} &= -\frac{\partial U}{\partial \mathbf{r}_j} = \frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\partial U}{\partial r_{ij}} \\ \frac{\partial^2 U}{\partial \mathbf{r}_i^2} &= \frac{\partial^2 U}{\partial \mathbf{r}_j^2} = \frac{\partial^2 U}{\partial x_i^2} + \frac{\partial^2 U}{\partial y_i^2} + \frac{\partial^2 U}{\partial z_i^2} \\ \frac{\partial^2 U}{\partial x_i^2} &= \left(\frac{1}{r_{ij}} - \frac{x_{ij}^2}{r_{ij}^3}\right)\frac{\partial U}{\partial r_{ij}} + \frac{x_{ij}^2}{r_{ij}^2}\frac{\partial^2 U}{\partial r_{ij}^2} \end{aligned}\right\} \tag{9.26}$$

Using Eq. (9.25) provides an additional check on the accuracy of the simulation, as the configurational temperature must equal the fixed value of temperature specified at the outset of the MC simulation.

Algorithm 9.4 illustrates an implementation of a MC simulation in the microcanonical ensemble. The procedure is similar to Algorithm 9.1 except that the acceptance or rejection of a move depends on the ratio of kinetic energies rather than the change in potential energy. The initial kinetic energy is determined in the initialization phase (Algorithm 9.4, Part 1). In common with the canonical ensemble, a trial move is attempted (Algorithm 9.4, Part 2.1) and the energy experienced by the atom is determined. The change in energy is used to calculate the new kinetic energy. The acceptance of a move (Algorithm 9.4, Part 2.2) depends on the ratio of the new and old kinetic energies (W). The value of W plays the analogous role to ΔU in the canonical ensemble. The allowed values of Θ mean that the only moves that can be accepted are those for which the energy of interaction is lower than the total energy. Configurations with $W > 1$ can be accepted immediately because they have a lower potential energy than the current configuration. It is also important to update the maximum displacement periodically (Algorithm 9.4, Part 2.3) to avoid a low acceptance rate. An alternative implementation of Algorithm 9.4 is to attempt random displacements for every atom and apply the acceptance criterion for the combined move. This alternative is described in Chapter 6 and an implementation in C++ is given in Chapter 11.

It should be noted that a similar procedure can also be used to generate an *NpH* ensemble. For the *NpH* ensemble, the appropriate probability density is (Ray, 1991):

$$W_E(\mathrm{r}) = C\left[H - pV - U(\mathbf{r}^N)\right]^{3N/2\,-1}\Theta\left[H - pV - U(\mathbf{r}^N)\right] \tag{9.27}$$

Algorithm 9.4 can be modified easily to use Eq. (9.27) instead of Eq. (9.20), however in this case volume fluctuations are also required periodically to maintain constant pressure.

ALGORITHM 9.4 MC simulation in the *NVE* ensemble.

Overall Algorithm

Part 1 Initialization

Assign values for N, V, *Efixed*, *NCycles*, *DispCounter* = 0, *acceptR* = 0. 5etc.
Assign initial coordinates rx_i, ry_i, rz_i for the N atoms.
Evaluate and store *Utotal*, U_i, *ptotal*, p_i etc.
$F \leftarrow 3N/2 - 1$
Kold ← *Efixed* – *Utotal*

Part 2 Generate a Markov Chain

loop *n* ←*1... NCycles*
  k ←*1*
  loop while (*k* <= *NAtom*)
2.1 Implement Parts 2.1-2.3 of Algorithm 9.1 (*NVT* Ensemble).
  Knew ←*Efixed* - (*Utotal* + ΔU) //kinetic energy
2.2 NVE acceptance criterion.
  k ← *k* +*1*
  end while loop
2.3 **if** (*n* <= *NEquilibration*)
2.3.1 Adjust maximum displacement change (see Algorithm 9.1).
  else
  Accumulate averages.
  end if
end *n* **loop**

Details of Some Key Sub-Parts

Part 2.2 NVE acceptance criterion:
if(*Knew* > *0*)
  $W \leftarrow (Knew/Kold)^F$
  if (*W* > 1 or *W* > rand())
    $rx_i \leftarrow rxTrial$ //accept new positions
    $ry_i \leftarrow ryTrial$
    $rz_i \leftarrow rzTrial$
    $U_i \leftarrow Utriali$
    Utotal ← *Utotal* + ΔU //update energy
    Kold ← *Knew*
  end if
end if

Some examples of the use of the *NVE* approach include investigations of the high temperature behavior of square well fluids (Sastre et al., 2015 & 2018) and lattice gas systems (Shida et al., 2003). An alternative *NVE* implementation that makes use of Eq. (9.25) has been proposed (Palma and Riveros, 2021).

As discussed in Chapter 2, properties obtained from different ensembles should yield the same values in the thermodynamic limit. Comparisons of properties obtained for common state points using different ensembles are relatively scarce in the literature. Assembling a comparison from different sources is also subject to confounding factors such as the differences in the number of particles, cut-off values, and other differences in the simulation settings. Table 9.1 compares MC simulations in the *NVE*, *NVT* (Lustig, 2011), *NpT* (Ströker et al., 2021), and *NpH* (Ströker and Meier, 2022) ensembles for various thermodynamic properties at close to a common state point for the LJ potential (Chapter 3). The difficulty in obtaining E and H to exactly generate common values of p, V, and T for a finite number of particles means that identical results cannot be realistically expected. Nonetheless, reasonably close agreement for the thermodynamic quantities obtained using different ensembles is apparent from the data in Table 9.1. This conclusion is supported by other work (Nitzke and Vrabec, 2023), which also indicates that the microcanonical ensemble yields results with the best overall statistical quality. In contrast, the choice of *NVT* and *NpT* ensembles in nonequilibrium MD simulations (Chapter 8) has a noticeable effect (Bosko et al., 2005) on the observation of shear thickening at high strain rates.

TABLE 9.1 Comparison of MC simulations for the isobaric heat capacity (C_p), Joule–Thomson coefficient (μ_{JT}), and zero frequency speed of sound (w_0) for the Lennard-Jones potential in the *NVE*, *NVT*, *NpT*, and *NpH* ensembles (Lustig, 2011; Ströker et al., 2021; Ströker and Meier, 2022). The values in brackets are the reported uncertainties in the final digit.

Ensemble/ Reduced Property	*NVE*	*NVT*	*NpT*	*NpH*
kT/ε	1.015	1.015	1.0	1.05781(7)
$p\sigma^3/\varepsilon$	1.071(5)	1.070(6)	1.0	1.0
$\rho\sigma^3$	0.8	0.8	0.79878(48)	0.780391(13)
C_p/k	4.8(1)	4.8(2)	4.79(7)	4.820(21)
$k\mu_{JT}/\sigma^3$	−0.16(1)	−0.15(1)	−0.1573(43)	−0.1493(14)
$w_0/\sqrt{\varepsilon/m}$	5.563(3)	5.557(4)	5.5159(24)	5.3479(46)

9.2 Molecular dynamics

9.2.1 Microcanonical (*NVE*) ensemble

The microcanonical ensemble is the default ensemble for MD because Newton's equations of motion conserve energy. The MD algorithm for the

NVE ensemble also conserves linear momentum (Chapter 2). As was discussed in Chapter 2, the motion of particles does not affect the energy of the ensemble, which remains constant. In principle, a microcanonical (*NVE*) ensemble is generated automatically by solving Newton's equations of motion. The general procedure for obtaining a microcanonical ensemble is illustrated by Algorithm 9.5.

The initialization phase (Algorithm 9.5, Part 1.1) of Algorithm 9.5 involves assigning initial velocities to the atoms. The velocities can take any initial values because they will be corrected quickly in the integrator phase (Algorithm 9.5, Part 2.1.2) of the main body of the simulation. Consequently, initial velocities can be either assigned randomly or set to zero. Similarly, with the exception of acceleration, any other time derivatives required by the integrator algorithm can be assigned to zero. The initial forces (Algorithm 9.5, Part 1.2) on the atoms must be calculated accurately by considering all $N(N-1)$ interactions. During the initial force evaluations, the initial accelerations are determined accurately from the relationship $a_i = f_i/m_i$. The total potential energy is also calculated during the force evaluation. The main body (Algorithm 9.5, Part 2) of the algorithm is a loop, which executes repeatedly the key simulation strategies. The heart of any MD program is the calculation of the force (Algorithm 9.5, Part 2.1.1) experienced by each atom as the result of interaction with all other atoms. This invariably involves using an appropriate intermolecular potential to evaluate particle interactions. Examples of intermolecular potentials are discussed in Chapters 3 and 4. The newly calculated forces are used in conjunction with a finite-difference algorithm (Chapter 7) to evaluate Newton's equations of motion, and thereby determine new atomic coordinates, velocities, accelerations, and any other time derivatives (Algorithm 9.5, Part 2.1.2). Each pass through the loop represents the advancement of time t by an amount Δt. Consequently, after M cycles, the total duration or elapsed time of the simulation is equal to $M\Delta t$. Contributions to ensemble averages are only accumulated after a period of equilibration (*tEquilibration*) (Algorithm 9.5, Part 2.2).

It is apparent from Algorithm 9.5 that the key feature of a MD algorithm is the evaluation of force, which via a suitable integrator algorithm (Chapter 7) determines new molecular coordinates and other dynamical properties in accordance with Newton's laws of motion. Unlike MC simulations, chance plays no part in MD. Although the potential energy is determined during the force evaluations, it plays no part in determining molecular displacements. The need to re-evaluate all forces at each time step means that MD algorithms are intrinsically order N^2 compared with order N for *MC* algorithms.[2] Considerable attention has been directed toward computation-saving strategies (Chapter 5).

2. However, the displacement step for each MC cycle is typically repeated N times, which means that both MC and MD are overall of order N^2 for a given cycle/time step.

ALGORITHM 9.5 General MD *NVE* ensemble strategy.

Part 1 Initialization

1.1 Assign number of particles, energy, volume, etc.
Assign coordinates (r_i) at time (t) = 0.
Assign molecules an initial velocity (v_i).
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion:
2.1.1 Calculate f_i on each atom from all other j atoms.
2.1.2 Apply integrator to update r_i, a_i, v_i, $(\partial^n \mathbf{r}/\partial t^n)_i$.
2.2 **If** ($t > tEquilibration$)
Accumulate averages.
end if
$t \leftarrow t + \Delta t$
while ($t \leq tMax$)

In the microcanonical ensemble, the number of particles, volume, and energy are constant. Keeping the number of particles and volume constant is straightforward. They are specified during the initialization phase and they cannot change because the algorithm does not involve volume fluctuations, particle removal, or particle addition. Control of the total energy is achieved via the contribution from kinetic energy. At the start of the simulation, the desired energy per particle is specified (E_D), however the actual energy (E_A) depends on the actual kinetic energy (K_A) and the calculated potential energy (U).

$$E_A = K_A + U \tag{9.28}$$

The desired kinetic energy (K_D) is obtained from the calculated potential energy.

$$K_D = E_D - U \tag{9.29}$$

The desired energy can be achieved by scaling the initial velocities (v_i) of each molecule with respect to the desired and actual kinetic energies.

$$\left.\begin{aligned} v_{ix}^{new} &= v_{ix}\sqrt{\frac{K_D}{K_A}} \\ v_{iy}^{new} &= v_{iy}\sqrt{\frac{K_D}{K_A}} \\ v_{iz}^{new} &= v_{iz}\sqrt{\frac{K_D}{K_A}} \end{aligned}\right\} \tag{9.30}$$

At the start of MD simulation, the linear momentum is usually set to zero. In principle, it is only necessary to scale the velocities once at the beginning of the simulation and thereafter, the energy conservation properties of the canonical ensemble should ensure that the energy remains constant. However, in practice a drift in energy is observed caused partly by inaccuracies in the calculation of forces. Therefore, to keep the calculations on track, the velocities are scaled periodically during the equilibration period. This scaling must not be continued after the equilibration period because it would introduce an artificial discontinuity in energy.[3] This strategy is illustrated by Algorithm 9.6.

ALGORITHM 9.6 MD in the *NVE* ensemble using periodic scaling of velocities.

Part 1 Initialization

1.1 Assign number of particles, energy, volume, etc.
Assign coordinates (r_i) at time (t) = 0.
Assign molecules an initial velocity (v_i).
Scale v_i consistent with the energy of the ensemble.
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion:
2.1.1 Calculate f_i on each atom from all other j atoms.
2.1.2 Apply integrator to update r_i, a_i, v_i, $(\partial^n \mathbf{r}/\partial t^n)_i$.
2.2 **If** ($t > tEquilibration$)
Accumulate averages.
else if (mod(m, *scalingInterval*) = 0)
Scale velocities.
end if
$t \leftarrow t + \Delta t$
$m \leftarrow m + 1$
while ($t \leq tMax$)

Algorithm 9.6 follows closely Algorithm 9.5 with the exception that the velocities are scaled in Parts 1.1 and 2.2. The choice of *scalingInterval* is arbitrary and depends on the behavior of the integrator. Typically, a scaling interval of 10−100 time steps is usually sufficient. An implementation of Algorithm 9.6 is described in Chapter 11. MC and MD implementations of

3. As noted in Chapter 8, this does not necessarily apply to NEMD algorithms because the modification of the Newton's law introduces additional heating that requires the application of a thermostat for the full length of the simulation.

the *NVE* ensemble have been directly compared of a restricted range of applications (Schierz et al., 2015).

9.2.2 Canonical (*NVT*) ensemble

Microcanonical ensemble simulations are not used routinely because we generally have a much better idea of temperature than energy. If the number of particles, volume, and temperature are constant, a canonical ensemble is generated. There are a few different options for generating a canonical ensemble. The simplest methods to obtain constant temperature involve either velocity scaling or heat-bath coupling. Alternatively, the thermostats of Andersen (1980), Nosé (1984), and Hoover (1985) or a general constraint (Hoover et al., 1982; Evans, 1983) approach can be used. These latter alternatives involve modifying the equations of motion. The implementation and merits of the various strategies are discussed subsequently. A general review of thermostats used in MD is available elsewhere (Hüneneberger, 2005).

9.2.2.1 Simple velocity scaling

In principle, the simplest method to maintain a constant temperature is to rescale periodically the particle velocities. The kinetic energy per particle for the ensemble can be calculated from,

$$\langle K \rangle = \frac{1}{2N} \left\langle \sum_i m_i \mathbf{v}_i \cdot \mathbf{v}_i \right\rangle \tag{9.31}$$

or from the kinetic theory of gases.

$$\langle K \rangle = \frac{3kT}{2} \tag{9.32}$$

Equating Eqs. (9.31) and (9.32), we obtain the following relationship for temperature.

$$T = \frac{1}{3Nk} \left\langle \sum_i m_i \mathbf{v}_i \cdot \mathbf{v}_i \right\rangle \tag{9.33}$$

Equation (9.33) enables[4] us to determine the actual temperature (T_A) for the ensemble at any time. Consequently, the velocities can be scaled with respect to the actual and desired temperatures (T_D).

4. For MD, Eq. (9.33) should be modified to account for the reduction in the number of degree of freedom due to the conservation of linear momentum. That is, $3N$ should be replaced with $3N - 3$.

$$\left.\begin{aligned} v_{ix}^{new} &= v_{ix}\sqrt{\frac{T_D}{T_A}} \\ v_{iy}^{new} &= v_{iy}\sqrt{\frac{T_D}{T_A}} \\ v_{iz}^{new} &= v_{iz}\sqrt{\frac{T_D}{T_A}} \end{aligned}\right\} \tag{9.34}$$

ALGORITHM 9.7 Molecular dynamics in the *NVT* ensemble using simple velocity scaling.

Part 1 Initialization

1.1 Assign number of particles, temperature, volume, etc.
Assign coordinates (r_i) at time (t) = 0.
Assign molecules an initial velocity (v_i).
Scale v_i consistent with the temperature of the ensemble.
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion:
2.1.1 Calculate f_i on each atom from all other j atoms.
2.1.2 Apply integrator to update r_i, a_i, v_i, $(\partial^n \mathbf{r}/\partial t^n)_i$.
2.2 **if** ($t > tEquilibration$)
Accumulate averages.
else if (mod(m, $scalingInterval$) = 0)
Scale velocities according to temperature.
end if
$t \leftarrow t + \Delta t$
$m \leftarrow m + 1$
while ($t \leq tMax$)

The algorithm for the canonical ensemble (Algorithm 9.7) is similar to the velocity-scaled microcanonical ensemble (Algorithm 9.6) except that the velocities are scaled via temperature (Algorithm 9.7, Part 2.2) rather than energy. In the velocity-scaled microcanonical ensemble algorithm, velocity scaling after the initialization phase is optional. However, for the canonical algorithm, periodic re-scaling in the loop is essential. Velocity scaling is less effective in achieving a set-point temperature than energy scaling to obtain a set-point energy because kinetic energy is not a constant of motion. Energy is exchanged between the potential and

kinetic energy contributions during equilibration causing a drift away from the desired temperature. Re-scaling during this period is required to compensate for this drift.

Velocity scaling is appealing because it is simple to implement in diverse range of different situations (Zhou et al., 2010; Kouza and Hansmann, 2011), although other approaches have arguably a stronger theoretical basis (Kutteh, 1999a & b; Page et al., 2012). A relatively simple modification has been proposed (Morishita, 2003a) to improve its usefulness in the velocity-Verlet algorithm (Chapter 7). It has used to optimize for replica exchange MD (REMD) (Kouza and Hansmann, 2011; Yu et al., 2015) and applied to NEMD (Chapter 8) simulations (Kuang and Gezelter, 2010 & 2012).

9.2.2.2 Heat-bath coupling

An alternative to simple velocity scaling is to couple the system to an external heat-bath corresponding to the desired temperature. The bath acts as either a heat source or heat sink by providing or removing heat to the system. The rate of change of the actual temperature is related to the desired temperature (the bath temperature) by,

$$\frac{dT_A}{dt} = \frac{T_D - T_A}{\tau} \tag{9.35}$$

where τ is a constant, which determines how tightly the bath and the system are coupled together. From Eq. (9.35), the change in temperature between successive time intervals is

$$\Delta T = \frac{\Delta t(T_D - T_A)}{\tau} \tag{9.36}$$

and the following velocity scaling applies.

$$\left.\begin{aligned} v_{ix}^{new} &= v_{ix}\sqrt{1 + \frac{\Delta t}{\tau}\left(\frac{T_D}{T_A} - 1\right)} \\ v_{iy}^{new} &= v_{iy}\sqrt{1 + \frac{\Delta t}{\tau}\left(\frac{T_D}{T_A} - 1\right)} \\ v_{iz}^{new} &= v_{iz}\sqrt{1 + \frac{\Delta t}{\tau}\left(\frac{T_D}{T_A} - 1\right)} \end{aligned}\right\} \tag{9.37}$$

When $\tau = \Delta t$, Eq. (9.37) is identical to the simple velocity scaling method. This form of scaling is implemented in the same way as illustrated in Algorithm 9.7. Typically, $\Delta t/\tau = 0.0025$ gives satisfactory results.

The advantage of Algorithm 9.7 is that it is an intuitively simple extension of the velocity-scaling microcanonical ensemble. However, it should be noted that strictly speaking, temperature is not fixed exactly but instead fluctuates around the desired value. It is legitimate to use a scaling procedure during the equilibration period to bring the system to the desired temperature but until this is achieved the system may not have a genuinely canonical distribution. Issues regarding heat baths have been studied extensively (Baldovin et al., 2009; Cano and Stuart, 2001; Morishita, 2000 & 2003b).

9.2.2.3 Andersen thermostat

Andersen (1980) proposed an alternative to velocity scaling, which combines MD with stochastic processes and guarantees a canonical distribution. At constant temperature, the energy of a system of N particles must fluctuate. This fluctuation can be introduced by changing the kinetic energy via periodic stochastic collisions. At the outset of the simulation, the temperature (T) and the frequency (ν) of stochastic collisions are specified. At any time interval (Δt), the probability that a particular particle is involved in a stochastic collision is $\nu\Delta t$. A suitable value of ν is,

$$\nu = \frac{\nu_c}{N^{2/3}} \tag{9.38}$$

where ν_c is the collision frequency.

During the simulation, random numbers can be used to determine which particles undergo stochastic collision at any small time interval. The simulation proceeds as follows. Initial values of positions and momenta are chosen and the equations of motion are integrated in the normal way until the time is reached for the first stochastic collision. The momentum of the particle chosen for the stochastic collision is chosen randomly from a Boltzmann distribution at temperature, T. The collision does not effect any of the other particles and the Hamiltonian equations for the entire particles are integrated until the next stochastic collision occurs. The process is then repeated. This strategy is summarized by Algorithm 9.8.

ALGORITHM 9.8 MD in the *NVT* ensemble using the Andersen thermostat.

Part 1 Initialization

1.1 Assign number of particles, temperature, volume, ν etc.
Assign coordinates (r_i) at time (t) = 0.
Assign molecules an initial velocity (v_i).
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion:
2.1.1 Calculate f_i on each atom from all other j atoms.
2.1.2 Apply integrator to update r_i, a_i, v_i, $(\partial^n \mathbf{r}/\partial t^n)_i$.
2.2 Andersen thermostat:
2.2.1 **if** (rand() < $\nu \Delta t$)
i ←*nearest integer* ∈ [1,N] //select molecule randomly
v_i ← *random value from a Boltzmann distribution*
end if
2.3 **If** ($t > tEquilibration$)
Accumulate averages.
end if
$t \leftarrow t + \Delta t$
while ($t \leq tMax$)

In Algorithm 9.8, the initialization phase (Algorithm 9.8, Part 1) proceeds as normal except that no velocity scaling occurs. In Part 2, after the application of the integrator algorithm (Algorithm 9.8, Part 2.1.2), the Andersen thermostat (Algorithm 9.8, Part 2.2.1) is applied periodically. The molecule that undergoes the collision is selected randomly after it has been determined that a collision has occurred (Algorithm 9.8, Part 2.2). Depending on ν, this may not generate a sufficient number of collisions during the course of the simulation. If the collision rate is too low, the system will not sample from a canonical distribution of energies. An alternative would be to check for collisions from i equals 1 to N times. If a collision is found, then the collision is with the ith molecule. This procedure is likely to result in a far greater number of collisions because N is typically 500 or more. However, it should be noted that if the collision rate is too high, temperature control dominates and the desired fluctuations in kinetic energy are suppressed.

The use of random numbers makes the Andersen thermostat a hybrid of MC and MD. There is evidence to suggest that the use of stochastic

processes may compromise the integrity of the calculation of dynamical quantities and that the frequency of stochastic collisions must be chosen carefully (Tanaka et al., 1983). A detailed theoretical discussion of the Andersen thermostat is available (Weinan and Dong, 2008).

9.2.2.4 Nosé thermostat

Nosé (1984) showed that the canonical (*NVT*) ensemble could be obtained rigorously by an unaided MD strategy. In Nosé's approach, the thermal reservoir is an integral part of the system and it is manifested as an additional degree of freedom (s). The potential energy of the reservoir (res) is,

$$U^{res} = gkT_D \ln s \tag{9.39}$$

where g is the number of degrees of freedom of the physical system. The reservoir also has kinetic energy, which can be obtained from,

$$K^{res} = \frac{Q}{2}\left(\frac{ds}{dt}\right)^2 = \frac{Q}{2}\mathbf{p}_s^2 \tag{9.40}$$

where Q is a parameter which couples the reservoir to the real system and which affects temperature fluctuations. It has the dimensions of energy $\times$ time^2 but it can be considered usefully as the imaginary mass of the extra degree of freedom. Typically, Q is proportional to gkT_D with the proportionality constant determined by trial simulations to observe whether or not the desired temperature is maintained satisfactorily. If Q is too large, there is no energy flow between the reservoir and the system, whereas if Q is too small, energy oscillations occur inhibiting equilibration.

Each state that can be generated for the extended system corresponds to a unique state in the real system. The velocities of the atoms in the real system are given by:

$$\mathbf{v} = s\frac{d\mathrm{r}}{dt} = \frac{\mathbf{p}}{ms} \tag{9.41}$$

Consequently, the kinetic energy of the real system is:

$$K^{real} = \frac{\mathbf{p}^2}{2ms^2} \tag{9.42}$$

Therefore, the Hamiltonian for the extended system is:

$$H = \sum_i \frac{p_i^2}{2m_i s^2} + U(q) + \frac{p_s^2}{2Q} + gkT_D \ln s \tag{9.43}$$

The key feature of Nosé's approach (Hoover, 1991) is the use of an extended Hamiltonian (Chapter 2). Any change in the Hamiltonian will be

reflected by changes in the equations of motion. The equations of motion obtained from Eq. (9.43) are:

$$\dot{\mathbf{q}} = \frac{\mathbf{p}}{ms^2} \tag{9.44}$$

$$\dot{\mathbf{p}} = \mathbf{F}(\mathbf{q}) \tag{9.45}$$

$$\dot{\mathbf{s}} = \frac{\mathbf{p}_s}{Q} \tag{9.46}$$

$$\dot{\mathbf{p}}_s = \sum_i \frac{p_i^2}{m_i s^3} - \frac{gkT_D}{s} \tag{9.47}$$

9.2.2.5 Nosé–Hoover equations

The equations for the Nosé thermostat contain the inconvenient s variable. Hoover (1985) demonstrated that Nosé's equations of motion could be reformulated to remove the s variable. These equations, hereafter referred to as the Nosé–Hoover equations, are:

$$\dot{\mathbf{q}} = \frac{\mathbf{p}}{m} \tag{9.48}$$

$$\dot{\mathbf{p}} = \mathbf{F}(\mathbf{q}) - \zeta\mathbf{p} \tag{9.49}$$

$$\dot{\zeta} = \frac{\sum_i \frac{p_i^2}{m_i} - gkT_D}{Q} \tag{9.50}$$

In the Nosé–Hoover equations, ζ is a "friction" coefficient that evolves with time according to Eq. (9.50). The value of ζ is determined by using Eq. (9.50) as a feedback equation. The overall algorithm for generating a *NVT* ensemble using the Nosé–Hoover equation is a straightforward adaptation of Algorithm 9.5. The only difference is that the ζ parameter must be re-evaluated at each time step. The integration of the equations of motion can be most simply handled by predictor–corrector (see discussion below) or Runge–Kutta (Chapter 7) algorithms. The appearance of a velocity in the acceleration (Eq. (9.49)) poses problems for the velocity-Verlet algorithm but this can be overcome (Frenkel and Smit, 2002). The predictor–corrector calculation strategy is illustrated by Algorithm 9.9.

In the initialization phase (Algorithm 9.9, Part 1), values are assigned to the m-derivatives of position with respect to time required by the predictor–corrector strategy and ζ is initially assigned to zero. During the simulation process (Algorithm 9.9, Part 2), the equations of motion are solved using a predictor–corrector calculation (Algorithm 9.9, Part 2.1). The

force on each atom is calculated (Algorithm 9.9, Part 2.1.1) and it is used to predict (Algorithm 9.9, Part 2.1.2) new positions and all of the time derivatives except acceleration. The acceleration (Algorithm 9.9, Part 2.1.3) is predicted using the latest predicted velocity. The newly predicted positions and all time derivatives are subsequently refined in the corrector step (Algorithm 9.9, Part 2.1.4). To update the ζ term (Algorithm 9.9, Part 2.1.5), it is necessary to determine the sum of the squared velocities. This calculation is performed conveniently during the corrector step of the integration.

ALGORITHM 9.9 MD in the *NVT* ensemble using the Nosé–Hoover equations of motion and a predictor–corrector strategy.

Part 1 Initialization

1.1 Assign number of particles, temperature, volume, ν etc.
Assign coordinates (r_i) at time (t) = 0.
Assign molecules an initial velocity (v_i).
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$ ... m).
$g \leftarrow 3N$ //3-D unconstrained system
$\zeta \leftarrow 0$
1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion using a PC method:
2.1.1 Calculate f_i on each atom from all other j atoms.
2.1.2 Predict new positions and $(\partial^n \mathbf{r}/\partial t^n)_i$ ($n \neq 2$).
2.1.3 $a_i \leftarrow f_i/m_i - \zeta v_i$
2.1.4 Correct new positions and $(\partial^n \mathbf{r}/\partial t^n)_i$ ($n = 1$ ... m).
2.1.5 $\zeta \leftarrow \zeta + \Delta t(\Sigma v_i^2/m_i - gkT_D)/Q$
2.2 **If** ($t > tEquilibration$)
Accumulate averages.
end if
$t \leftarrow t + \Delta t$
while ($t \leq tMax$)

It has been observed that the Nosé–Hoover equations are not genuinely ergodic for either small or stiff systems. To correct this, a modification has been proposed (Martyna et al., 1992) that replaces the single thermostat with a chain of variables. The Nosé–Hoover equations using M thermostats are:

$$\dot{\mathbf{q}} = \frac{\mathbf{p}}{m} \tag{9.51}$$

$$\dot{\mathbf{p}} = \mathbf{F}(\mathbf{q}) - \zeta_1 \mathbf{p} \tag{9.52}$$

$$\zeta_1 = \frac{\sum_i \frac{p_i^2}{m_i} - gkT_D}{Q_1} - \zeta_1\zeta_2 \tag{9.53}$$

$$\zeta_j = \frac{Q_{j-1}\zeta_{j-1}^2 - kT_D}{Q_j} - \zeta_j\zeta_{j+1} \tag{9.54}$$

$$\zeta_M = \frac{Q_{M-1}\zeta_{M-1}^2 - kT_D}{Q_M} \tag{9.55}$$

It is apparent that the definition of ζ is recursive. This natural recursion can be exploited to simplify the coding in languages such as C++, particularly if M is a large number. Reversible MD algorithms for Nosé–Hoover chains have also been developed (Jang and Voth, 1997).

9.2.2.6 Constraint methods

In general, constant temperature can be achieved by introducing a temperature constraint into the equations of motion. A constant kinetic temperature dynamics is generated by the following modified equations of motion (Hoover et al., 1982; Evans, 1983).

$$\dot{\mathbf{r}} = \frac{\mathbf{p}}{m} \tag{9.56}$$

$$\dot{\mathbf{p}} = \mathbf{f} - \zeta(\mathbf{r},\mathbf{p})\mathbf{p} \tag{9.57}$$

In Eq. (9.57), $\zeta(\mathbf{r},\mathbf{p})$ acts as a "friction coefficient," which varies to constrain the instantaneous temperature to a constant value. The ζ term plays a similar role to the corresponding term in the Nosé–Hoover thermostat (Eqs. (9.48) and (9.50)). However, by applying Gauss's principle of least constraint (Chapter 2), it can be shown (Hoover, 1983; Evans and Morriss, 1984) that:

$$\zeta = \frac{\sum_i \mathbf{p}_i \cdot \mathbf{f}_i}{\sum_i |\mathbf{p}_i|^2} \tag{9.58}$$

Equations (9.56) and (9.57) can be most easily solved using a Gear predictor–corrector algorithm but a variant of the leap-frog scheme can also be used (Brown and Clarke, 1984). The use of a Gaussian constraint is likely to lead to a drift in T with time caused by rounding errors. This can be mitigated by factors such as the choice of integrator (Chapter 7) or using a linear proportional feedback mechanism (Travis et al., 1995).

9.2.3 Isobaric–isoenthalpic (*NpH*) ensemble

The isobaric–isoenthalpic ensemble is not used commonly in molecular simulations. However, *NpH* ensemble algorithms are worthy of consideration because they introduce techniques for obtaining constant pressure, and they can be extended easily to generate the more useful *NpT* ensemble. The *NpH* ensemble is particularly useful for calculation of the Joule–Thomson coefficient (Kioupis and Maginn, 2002; Travis, 2023). Obtaining thermodynamic properties from the *NpH* ensemble is discussed in Chapter 2.

9.2.3.1 Andersen's algorithm

The general approach used by Andersen (1980) is to introduce additional degrees of freedom to the physical system. Specifically, the volume is considered as a dynamical variable that is coupled to particle coordinates. Newton's equations of motion are solved for the "extended system" involving both particle coordinates and volume.

Using scaled coordinates $x_i = r_i/V^{1/3}$, Andersen (1980) proposed the following Lagrangian at constant pressure, which introduces a new variable, Q.

$$L = \frac{Q^{2/3}}{2}\sum_{i=1}^{N} m_i \dot{\mathbf{x}}\cdot\dot{\mathbf{x}} - \sum_{j>i}^{N} v(Q^{1/3}\mathbf{x}_{ij}) + \frac{M\dot{Q}^2}{2} - \alpha Q \tag{9.59}$$

The variable Q can be interpreted as the volume, and the first two terms represent the conventional Lagrangian expressed in terms of new variables. The third term is the kinetic energy of Q, and the fourth term is the potential energy associated with Q. The terms α and m are constants. A physical interpretation of the Lagrangian can be gained by considering the system to be a container of variable volume controlled by a piston. In this context, the volume Q of the system is determined by the coordinates of the piston, αQ is a potential derived from an external pressure α acting on the piston, and m is the mass of the piston. The Hamiltonian for this system is,

$$H = \frac{1}{2Q^{2/3}}\sum_{i=1}^{N}\frac{\pi_i\cdot\pi_i}{m_i} + \sum_{j>i}^{N} v(Q^{1/3}x_{ij}) + \frac{\Pi^2}{2M} + \alpha Q \tag{9.60}$$

where π and Π are the momenta conjugate to x and Q, respectively. We define the following relationships between the scaled and unscaled coordinates.

$$\left.\begin{aligned} V &= Q \\ \mathbf{r} &= Q^{1/3}\mathbf{x} \\ \mathbf{p} &= \frac{\pi}{Q^{1/3}} \end{aligned}\right\} \tag{9.61}$$

Using Eq. (9.59) in conjunction with Eq. (9.61), we obtain the following equations of motion.

$$\dot{\mathbf{r}} = \frac{\mathbf{p}}{m} + \frac{\mathbf{r}}{3}\frac{d\ln V}{dt} \tag{9.62}$$

$$\dot{\mathbf{p}} = \mathbf{F}(\mathbf{r}_{ij}) - \frac{\mathbf{p}}{3}\frac{d\ln V}{dt} \tag{9.63}$$

$$\frac{Md^2V}{dt^2} = -\alpha + \frac{2}{3V}\sum_{i=1}^{N}\frac{\mathbf{p}_i\cdot\mathbf{p}_i}{2m_i} - \frac{1}{3V}\sum_{j>i}^{N}\mathbf{r}_{ij}\frac{\partial u(\mathbf{r}_{ij})}{\partial \mathbf{r}_{ij}} \tag{9.64}$$

By recognizing that α is the desired pressure (p_D), and that the actual pressure (p_A) is obtained from,

$$p_A = \frac{2}{3V}\sum_{i=1}^{N}\frac{\mathbf{p}_i\cdot\mathbf{p}_i}{2m_i} - \frac{1}{3V}\sum_{j>i}^{N}\mathbf{r}_{ij}\frac{\partial u(\mathbf{r}_{ij})}{\partial \mathbf{r}_{ij}} \tag{9.65}$$

Equation (9.65) can be expressed simply as:

$$\frac{d^2V}{dt^2} = \frac{p_A - p_D}{M} \tag{9.66}$$

The change in volume with time in Eqs. (9.62) and (9.63) is determined by using Eq. (9.64) as a "feedback" equation. In turn, the volume is adjusted depending on the value of dV/dt.

It can be shown (Andersen, 1980) that this approach generates an isobaric–isoenthalpic (*NpH*) ensemble. The enthalpy of the system is the energy (E) associated with the scaled system minus the time average kinetic energy associated with the motion of Q. Parrinello and Rahman (1980, 1981 & 1982) generalized Andersen's approach to allow for changes in the shape of the simulation box, which is particularly useful for solid-state calculations. Alternative approaches (Kioupis and Maginn, 2002; Travis, 2023) are also possible.

An implementation of the isobaric–isoenthalpic ensemble is illustrated by Algorithm 9.10. In the initialization phase (Algorithm 9.10, Part 1), the volume derivatives can be set to zero. Their values will be corrected rapidly in the simulation phase (Algorithm 9.10, Part 2). The actual pressure of the ensemble (p_A) is calculated conveniently during the force evaluation. The integrator (Algorithm 9.10, Part 2.1.3) subsequently updates the coordinates, velocities, accelerations, and other time derivatives. The first time through the simulation loop, we are effectively applying Newton's equations of motion because the volume derivatives in

Eqs. (9.62) to (9.64) are zero. However, updating the volume derivatives (Algorithm 9.10, Part 2.1.4) results subsequently in the application of Andersen's equations.

ALGORITHM 9.10 MD in the *NpH* ensemble using Andersen's method.

Part 1 Initialization

1.1 Assign M, N, V, v, T_D, p_D, etc.
Assign coordinates (r_i) at time (t) = 0.
Assign molecules an initial velocity (v_i).
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
$dlnV/dt \leftarrow 0$
$dV/dt \leftarrow 0$
$d^2V/dt^2 \leftarrow 0$
$Vold \leftarrow V$
1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion:
2.1.1 Calculate f_i on each atom from all other j atoms.
2.1.2 Calculate p_A.
2.1.3 Apply integrator to update r_i, a_i, v_i, $(\partial^n \mathbf{r}/\partial t^n)_i$.
2.1.4 $dV^2/dt^2 \leftarrow dV^2/dt^2 + \Delta t \times (p_D - p_A)/M$
$dV/dt \leftarrow dV/dt + \Delta t \times (dV^2/dt^2)$
$V \leftarrow V + \Delta t \times (dV/dt)$
$dlnV/dt \leftarrow (\ln(V) - \ln(Vold))/\Delta t$
$Vold \leftarrow V$
2.2 **If** ($t > tEquilibration$)
Accumulate averages.
end if
$t \leftarrow t + \Delta t$
while ($t \leq tMax$)

9.2.3.2 Constraint methods

Gauss's principle of least constraint (Chapter 2) can be used to obtain modified equations of motion that are appropriate at constant pressure and enthalpy (Evans and Morriss, 1984),

$$\dot{\mathbf{r}} = \frac{\mathbf{p}}{m} + \chi(\mathbf{r}, \mathbf{p})\mathbf{r} \tag{9.67}$$

$$\dot{\mathbf{p}} = \mathbf{f} - \chi(\mathbf{r}, \mathbf{p})\mathbf{p} \tag{9.68}$$

$$\dot{V} = 3V\chi(\mathbf{r}, \mathbf{p}) \tag{9.69}$$

where χ is a Lagrange multiplier, which represents the rate of dilation of the system.

$$\chi = -\frac{2\sum_i \frac{\mathbf{p}_i \cdot \mathbf{f}_i}{m_i} - \sum_i \sum_{j>i} \frac{(\mathbf{r}_{ij} \cdot \mathbf{p}_{ij})}{m_i r_{ij}} \left(\frac{dw(r_{ij})}{dr_{ij}}\right)}{2\sum_i \frac{\mathbf{p}_i^2}{m_i} + \sum_i \sum_{j>i} r_{ij} \left(\frac{dw(r_{ij})}{dr_{ij}}\right) + 9pV} \tag{9.70}$$

Long-range corrections (Chapter 5) must be included in the calculation of both the virial and pressure terms in Eq. (9.70).

9.2.4 Isothermal–isobaric (*NpT*) ensemble

There are three distinct methods for simulating an *NpT* ensemble by MD, which involve extending the approach used for *NVT* ensemble MD. The first is the hybrid MC and MD approach used by Andersen (1980). The Andersen thermostat can be modified to include constant pressure resulting in an isobaric–isoenthalpic (*NpH*) ensemble, which can be transformed to an *NpT* ensemble via a stochastic process. Alternatively, either Hoover's equations for the *NVT* ensemble can be modified to incorporate a pressure constraint or the general constraint procedure can be extended. The development of MD *NpT* algorithms remains an active area of interest (Braga and Travis, 2005; Bussi et al., 2009; Huang et al., 2011; Kim et al., 2019).

9.2.4.1 Hybrid MD/MC

To modify Andersen's algorithm for the *NpH* ensemble to generate an algorithm for the *NpT* ensemble, the energy and enthalpy of the system must be allowed to fluctuate. Fluctuations can be achieved by the introduction of stochastic collisions that affect the momentum of one particle at a time. The stochastic collisions are instantaneous events and they occur at a particular value of Q. The effect of each stochastic collision is to replace the momentum of the affected particle by a new value chosen at random from a distribution given by:

$$\exp\left(-\frac{\pi_i \cdot \pi_i}{2m_i Q^{2/3} kT}\right) \tag{9.71}$$

The implementation of the *NpT* ensemble algorithm proceeds in a similar fashion to the *NpH* ensemble except that allowance is made for stochastic collisions that result in a constant temperature. The procedure is illustrated by Algorithm 9.11.

ALGORITHM 9.11 MD in the *NpT* ensemble using Andersen's method.

Part 1 Initialization

1.1 Assign M, N, V, v, T_D, p_D, etc.
Assign coordinates (r_i) at time (t) = 0.
Assign molecules an initial velocity (v_i).
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
$dlnV/dt \leftarrow 0$
$dV/dt \leftarrow 0$
$d^2V/dt^2 \leftarrow 0$
$Vold \leftarrow V$
1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion:
2.1.1 Calculate f_i on each atom from all other j atoms.
2.1.2 Calculate P_A.
2.1.3 Apply integrator to update r_i, a_i, v_i, $(\partial^n \mathbf{r}/\partial t^n)_i$.
2.1.4 Change momenta via stochastic collision:
loop $i \leftarrow 1 \ldots N$
if (rand() $< \nu \Delta t$)
$p_i \leftarrow$ random value from Boltzmann distribution
end if
end i **loop**
2.1.5 $dV^2/dt^2 \leftarrow dV^2/dt^2 + \Delta t \times (p_D - p_A)/M$
$dV/dt \leftarrow dV/dt + \Delta t \times (dV^2/dt^2)$
$V \leftarrow V + \Delta t \times (dV/dt)$
$dlnV/dt \leftarrow (\ln(V) - \ln(Vold))/\Delta t$
$Vold \leftarrow V$
2.2 **If** ($t >$ *tEquilibration*)
Accumulate averages.
end if
$t \leftarrow t + \Delta t$
while ($t \le$ *tMax*)

Algorithm 9.11 is similar to Algorithm 9.10 except for the presence of stochastic collisions (Algorithm 9.11, Part 2.1.4). After the integrator has updated the coordinates, velocities, accelerations, and the like, N random numbers are generated. If *rand()* $< \nu \Delta t$, a collision is deemed to have taken place and the momentum of the atom (p_i) is assigned a random value from a Boltzmann distribution. In the absence of any stochastic collisions Algorithm 9.11 generates a *NpH* ensemble.

9.2.4.2 Extension of the Nosé–Hoover equations

The equations of motion for the *NpT* ensemble can be obtained as a straightforward extension of the Nosé–Hoover equations for the *NVT* ensemble. In

terms of reduced coordinate ($x \equiv q/V^{1/d}$; d is the number of dimensions), Hoover (1985) defined,

$$\dot{\mathbf{x}} = \frac{\mathbf{p}}{mV^{1/d}} \tag{9.72}$$

$$\dot{\mathbf{p}} = \mathbf{F} - (\dot{\varepsilon} + \zeta)\mathbf{p} \tag{9.73}$$

$$\dot{\zeta} = \frac{\sum_i \frac{\mathbf{p}_i^2}{m_i} - gkT_D}{Q} \tag{9.74}$$

$$\dot{\varepsilon} = \frac{\dot{V}}{dV} \tag{9.75}$$

$$\ddot{\varepsilon} = \frac{p_A - p_D}{\tau^2 kT_D} \tag{9.76}$$

where p_A and p_D are the actual and desired pressure, respectively, and τ is the relaxation time. Equations (9.72–9.75) make use of an additional dilation rate $\dot{\varepsilon}$, which is determined by using Eq. (9.76) as a feedback equation in a similar way to the calculation of ζ in the *NVT* ensemble. The definition and role of Eq. (9.76) is similar to Eq. (9.66) for Andersen's *NpH* ensemble algorithm.

The overall *NpT* ensemble algorithm proceeds in a similar way to the *NVT* ensemble for the Nosé–Hoover equations except that both $\dot{\varepsilon}$ and ζ must be calculated for each time step via the feedback equations. The simulation strategy is illustrated by Algorithm 9.12. To update $\dot{\varepsilon}$and ζ, both the sum of the squared velocities and the actual pressure must be calculated. The actual pressure, calculated from the virial relationship (Eq. (9.65)), can be obtained conveniently during the force evaluation (Part 2.1.1) and the squared velocities can be calculated during the corrector step (Algorithm 9.12, Part 2.1.4). The values of $\dot{\varepsilon}$ and ζ are updated after the corrector step (Algorithm 9.12, Part 2.1.5). It should be noted that because the actual pressure is calculated before the application of either the corrector or the predictor, its value at Part 2.1.5 effectively lags the simulations by one step. However, the imprecision this introduces is likely to be very small. If the intermolecular potential is scalable, an accurate value could be obtained after the corrector step via a simple scaling procedure. If the intermolecular potential is not scalable, an accurate value can only be obtained by performing a computationally expensive $N(N-1)$ re-evaluation.

The Nosé–Hoover approach is a topic of active interest, which have proved particularly useful (Hoover, 2007) in NEMD (Chapter 8) simulations. Issues addressed include reversibility (Jiang and Voth, 1997; Tchouar et al., 2007), temperature–energy–space sampling (Fukuda and Moritsugu, 2020), relationship to a fluctuating heat-bath (Fukuda and Moritsugu, 2016), multiple thermostat systems (Morishita, 2010; Fukuda, 2016), the removal of time

scaling (Fukuda and Moritsugu, 2017), muticanonical ensembles (Jang et al., 2002), and dissipative particle dynamics (Allen and Schmid, 2007).

Traditionally, MD thermostats have controlled the kinetic temperature. However, the concept of the configurational temperature embodied in Eq. (9.25) provides an alternative. Braga and Travis (2005) have utilized Eq. (9.25) to develop a configurational Nosé–Hoover thermostat, which has proved (Travis and Braga, 2006) particularly useful for NEMD. The basic concept has also been extended (Travis and Braga, 2008; Beckedahl et al., 2016) to other situations.

ALGORITHM 9.12 MD in the *NpT* ensemble using the Nosé–Hoover equations of motion and a corrector–predictor strategy.

Part 1 Initialization

1.1 Assign Q, N, T_D, p_D, τ etc.
Assign coordinates (r_i) at time (t) = 0.
Scale coordinates ($x = q/V^{1/d}$).
Assign molecules an initial velocity (v_i).
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
$g \leftarrow 3N$
$\zeta \leftarrow 0$
$\varepsilon Dot \leftarrow 0$
1.2 Initial force calculations:
Calculate forces (f_i) on from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
2.1 Integrate equations of motion using a PC method:
2.1.1 Calculate f_i on each atom from all other j atoms.
Calculate p_A.
2.1.2 Predict new positions and $(\partial^n \mathbf{r}/\partial t^n)_i$ ($n \neq 2$).
2.1.3 $a_i \leftarrow f_i/m_i - (\varepsilon Dot + \zeta)v_i$
2.1.4 Correct new positions and $(\partial^n \mathbf{r}/\partial t^n)_i$ ($n = 1 \ldots m$).
2.1.5 $\zeta \leftarrow \zeta + \Delta t(\Sigma v_i^2/m_i - gkT_D)/Q$
$\varepsilon Dot \leftarrow \varepsilon Dot + \Delta t(P_A - P_D)/(\tau^2 kT_D)$
2.2 **If** ($t > tEquilibration$)
Accumulate averages.
end if
$t \leftarrow t + \Delta t$
while ($t \leq tMax$)

9.2.4.3 Constraint methods

To obtain an *NpT* ensemble, the following modifications are made to the constrained *NVT* ensemble equations of motion (Evans and Morriss, 1984).

$$\dot{\mathbf{r}} = \frac{\mathbf{p}}{m} + \chi(\mathbf{r}, \mathbf{p})\mathbf{r} \tag{9.77}$$

$$\dot{\mathbf{p}} = \mathbf{f} - \chi(\mathbf{r}, \mathbf{p})\mathbf{p}\text{-}\zeta(\mathbf{r}, \mathbf{p})\mathbf{p} \tag{9.78}$$

$$\dot{V} = 3V\chi(\mathbf{r}, \mathbf{p}) \tag{9.79}$$

These equations introduce Lagrange multipliers for both the "rate of dilation" of the system (χ) and "friction" (ζ).

$$\chi = -\frac{\sum_i \sum_{j>i} \frac{(\mathbf{r}_{ij} \cdot \mathbf{p}_{ij})}{m_i r_{ij}} \left(\frac{dw(r_{ij})}{dr_{ij}}\right)}{\sum_i \sum_{j>i} r_{ij} \left(\frac{dw(r_{ij})}{dr_{ij}}\right) + 9pV} \tag{9.80}$$

where w is the instantaneous virial and:

$$\zeta = \frac{\sum_i \mathbf{p}_i \cdot \mathbf{f}_i}{\sum_i |\mathbf{p}_i|^2} - \chi \tag{9.81}$$

The formulation of alternative NpT ensembles is an active area of research. Melchionna et al. (1993) have reported that the trajectories of isobaric ensembles may have an unphysical dependence on the choice of basis lattice vectors. To compensate, modifications to the Nosé−Hoover equations have been proposed (Martyna et al., 1994).

9.2.5 Grand canonical (μVT) ensemble

In the grand canonical ensemble, the chemical potential is constant and the number of particles changes. In contrast to the MC method, which handles particle variations by straightforward addition or removal, it is considerably more difficult to apply MD to the grand canonical ensemble. The problem has been addressed by several workers (Lupkowski and van Swol, 1991; Çagin and Pettitt, 1991a & b; Ji et al., 1992; Lo and Palmer, 1995; Papadopoulou et al., 1993; Lynch and Pettitt, 1997; Kuznetsova and Kvamme, 1999; Boinepalli and Attard, 2003; Samways et al., 2020). A feature of many MD implementations of the grand canonical ensemble is the use of a stochastic process for particle fluctuation.

9.2.5.1 Lupkowski−van Swol method

The method proposed by Lupkowski and van Swol (1991) is a simple combination of MD and MC methods. In their approach, a conventional MD NVT ensemble simulation is interrupted periodically by an attempt to either create or destroy a particle. A particle is created with a probability of,

$$P^+ = \min\left[1, \exp\left(\frac{\mu - \Delta U_{trial}}{kT} + \ln\left(\frac{V}{\Lambda^{3N}(N+1)}\right)\right)\right] \tag{9.82}$$

whereas a particle is removed with a probability of:

$$P^{-} = \min\left[1, \exp\left(-\frac{\mu + \Delta U_{trial}}{kT} + \ln\left(\frac{N\Lambda^{3N}}{V}\right)\right)\right] \tag{9.83}$$

The particle creation and removal steps are conducted according to the MC grand canonical algorithm (Algorithm 9.3). The velocity of the newly created particle is obtained from a Maxwell−Boltzmann distribution. The algorithm is relatively simple to implement. However, the particle creation and removal procedure may affect the velocity autocorrelation functions required for the calculation of transport properties.

9.2.5.2 Çagin−Pettitt method

Çagin and Pettitt (1991a & b) used the concept of "fractional particles" to formulate a MD method for the grand canonical ensemble. The number of particles in a system is considered to be a function of time. During the simulation, the system has fractional particles at position $\mathbf{r}_f$, which are in the process of either becoming full particles or vanishing. The Nosé−Hoover thermostat for an *NVT* ensemble exploits parameters that couple the system to a heat-bath. To couple the system to a chemical potential reservoir, Çagin and Pettitt introduced a real continuous variable (c), whose integer part represents the number of particles in the system (N). The kinetic and potential energies of the system are subsequently scaled by $c - N$. Either an adiabatic constant chemical ensemble or the grand canonical ensemble can be formulated by keeping the chemical potential constant. The Lagranian for the grand canonical ensemble is:

$$\begin{aligned} L = {} & \frac{1}{2}\sum_{a=1}^{N} ms^2\dot{\mathbf{x}}_a^2 + \frac{c-N}{2} m_f s^2 \dot{\mathbf{x}}_f^2 + \frac{W\dot{c}^2}{2} + \frac{Q\dot{s}^2}{2} \\ & - \sum_{a=1}^{N-1}\sum_{b>a}^{N} u(r_{ab}) - (c-N)\sum_{a=1}^{N} u(r_{fa}) + \mu c - gkT\ln s \end{aligned} \tag{9.84}$$

In common with the work of Andersen (1980), Nosé (1984), and Hoover (1985) for the *NVT* and *NpT* ensembles, Eq. (9.84) is the Lagrangian for the extended system. The variable x represents the coordinates of the extended system, s is the parameter that couples the system to a heat-bath, and Q is the mass associated with s. Equation (9.84) introduces the new variable W, which can be interpreted as the mass associated with the new dynamical variable c. The equations of motion for this Lagrangian are:

$$ms^2\ddot{x}_{ai} = -\sum_{b=1}^{N}\left(\frac{\partial u(r_{ab})}{\partial r_{ab}}\right)\frac{x_{abi}}{r_{ab}} - (c-N)\left(\frac{\partial u(r_{af})}{\partial r_{af}}\right)\frac{x_{afi}}{r_{af}} - 2ms\dot{s}\dot{x}_{ai} \tag{9.85}$$

$$(c-N)ms^2\ddot{x}_{fi} = -(c-N)\sum_{a=1}^{N}\left(\frac{\partial u(r_{fa})}{\partial r_{fa}}\right)\frac{x_{fai}}{r_{fa}} - ms^2\dot{c}\dot{x}_{fi} - 2(c-N)ms\dot{s}\dot{x}_{fi} \tag{9.86}$$

$$W\ddot{c} = \mu + \frac{ms^2x_f^2}{2} - \sum_{a=1}^{N}u(r_{fa}) \tag{9.87}$$

$$Q\ddot{s}s = \sum_a \frac{1}{2}ms^2\dot{x}_a^2 + \frac{1}{2}(c-N)ms^2\dot{x}_f^2 - gkT \tag{9.88}$$

The method has been applied successfully to determine the density dependence of the chemical potential of a three-site water model (Ji et al., 1992). A limitation of (Eqs. 9.85–9.88) is that they do not reproduce correctly the behavior of the ideal gas. The algorithm can be modified (Weerasinghe and Pettitt, 1994) to include the corrected ideal gas behavior. However, Lo and Palmer (1995) observed that the modified algorithm may compromise the accuracy of the predicted densitytemperature–chemical potential behavior for the LJ fluid.

9.2.5.3 Lo–Palmer method

The method of Lo and Palmer (1995) is of particular interest because it can be used to reproduce correctly the density–temperature–chemical potential behavior of the LJ fluid; it can be extended to the Gibbs ensemble (Chapter 10), and in the limit, the properties of the ideal gas are described properly. Following the general approach of Nosé (1984), Lo and Palmer (1995) proposed an extended Hamiltonian for the grand canonical ensemble of the form:

$$\begin{aligned} H = &\sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{2m_is^2} + U(\mathbf{q}^N) + \frac{\mathbf{p}_f^2}{2m_fs^2} + U_f(\mathbf{q}^N,\mathbf{q}_f,c) + \frac{\mathbf{p}_s^2}{2Q} \\ &+ gkT\ln s + \frac{\mathbf{p}_v^2}{2W} + ckT\ln[(N+1)!] \\ &+ (1-c)kT\ln\left(\frac{VN!}{\Lambda_f^3}\right) - (N+c)\mu \end{aligned} \tag{9.89}$$

The earlier Hamiltonian distinguishes between the properties of ordinary particles and "fractional" particles denoted by the subscript *f*. Therefore, there is a contribution to kinetic energy and potential energy from both ordinary and fractional particles. The variable c is the coupling coordinate that couples the fractional particle to the rest of the system. It plays a similar role to the corresponding parameter in the Çagin–Pettitt algorithm and it can

have any value between 0 and 1. The constant W is the mass associated with the variable c and Λ_f is the de Brogile wavelength of the fractional particle. The remaining terms have the same meaning as in the Nosé–Hamiltonian (Eq. (9.43)). When $c = 1$, the fractional particle is completely coupled to the rest of the system, whereas a value of $c = 0$ indicates that the fractional particle is totally decoupled from the system. When $c = 1$, Eq. (9.89) is formally identical to Nosé's Hamiltonian except for the addition of the $-N\mu$ term. In terms of real variables, the equation of motion resulting from this Hamiltonian are:

$$\dot{\mathbf{p}} = \mathbf{F} + \mathbf{F}_f - \frac{\dot{s}\mathbf{p}}{s} \tag{9.90}$$

$$\dot{\mathbf{p}}_f = \mathbf{F}_f - \frac{\dot{s}\mathbf{p}_f}{s} \tag{9.91}$$

$$\ddot{s} = \frac{\dot{s}^2}{s} + \frac{s}{Q}\left(\sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{m_i} + \frac{\mathbf{p}_f^2}{m_f} - gkT\right) \tag{9.92}$$

$$\ddot{c} = \frac{\dot{s}\dot{c}}{s} + \frac{s^2}{W}\left[-\frac{\partial U_f}{\partial c} + kT\ln\left(\frac{V}{\Lambda_f^3(N+1)}\right) + \mu\right] \tag{9.93}$$

Equations (9.92) and (9.93) are feedback equations that enable the calculation of c and s and their time derivatives at each time step. A possible implementation is illustrated by Algorithm 9.13.

The algorithm starts with a normal initialization phase (Algorithm 9.13, Part 1). The first phase of the simulation process (Algorithm 9.13, Part 2) is to create an equilibrated *NVT* ensemble (Algorithm 9.13, Part 2.1). This can be achieved by applying any *NVT* ensemble algorithm; however, the Nosé–Hoover algorithm is the most obvious choice because in the absence of a constant chemical potential, the Lo–Palmer equations reduce to the Nosé–Hoover algorithm.

After the *NVT* ensemble is generated, it is transformed to a μVT ensemble via a stochastic process (Algorithm 9.13, Part 2.2). A random choice is made to either create (Algorithm 9.13, Part 2.2.1) or remove a particle (Algorithm 9.13, Part 2.2.2). Creating a new particle is quite straightforward. In contrast, removing a particle requires reordering of indices. Alternatively, the deleted particle can be simply tagged as being deleted and ignored in subsequent force evaluations. If a particle is created (Algorithm 9.13, Part 2.2.1), c is set to zero, the coordinates of the fractional particle are selected randomly, and a Maxwell–Boltzmann distribution is used to obtain the initial velocity and $\dot{c}$. If a particle is deleted (Algorithm 9.13, Part 2.2.2), the deleted particle is chosen randomly, c is set to 1, and $\dot{c}$ is obtained from a Maxwell–Boltzmann distribution. The equations of motion are solved repeatedly (Algorithm 9.13, Part 2.3) until c is either 0 or 1. If c equals 1 (Algorithm 9.13, Part 2.4), the fractional coordinates and velocities are reassigned as ordinary particles.

ALGORITHM 9.13 MD in a μVT ensemble using the Lo–Palmer algorithm.

Part 1 Initialization

1.1 Assign values of *Q, W, N, V, T*, etc.
Assign coordinates (r_i) at time (t) = 0.
Scale coordinates ($x = q/V^{1/d}$).
Assign molecules an initial velocity (v_i).
Assign any other $(\partial^n \mathbf{r}/\partial t^n)_i$ values ($n > 2$).
$g \leftarrow 3N$
$s \leftarrow 0$
$sDot \leftarrow 0$

1.2 Initial force calculations:
Calculate forces (f_i) on each atom from all other j atoms.
Calculate acceleration (a_i) for each atom.

Part 2 Simulation Process

loop
  if ($t \leq tNVTEquilibration$)
2.1 Create an equilibrated NVT-ensemble:
  Nosé-Hoover algorithm (e.g., Algorithm 9.9).
  else
2.2 Transform NVT-ensemble to a μVT-ensemble:
  if (rand() < 0.5)
2.2.1 Particle addition:
  $c \leftarrow 0$
  Create new particle *i.*
  Assign coordinates for new particle *i*, randomly.
  Assign particle *i* a velocity from a
  Maxwell-Boltzmann distribution.
  Assign *cDot* a value from a Maxwell-Boltzmann distribution.
  else
2.2.2 Particle removal:
  $i \leftarrow \in$[1, N] //select a particle randomly
  Delete all reference to particle *i.*
  $c \leftarrow 1$
  Assign *cDot* from a Maxwell-Boltzmann distribution.
  end if
  loop
2.3 Integrate equation of motion (eqs (9.91)-(9.94)).
  Evaluate *c2Dot* and *s2Dot* from eqs (9.93) and (9.94).
  $sDot \leftarrow sDot + \Delta t \times s2Dot$
  $cDot \leftarrow cDot + \Delta t \times c2Dot$
  $s \leftarrow s + \Delta t \times sDot$
  $c \leftarrow c + \Delta t \times cDot$
2.4 **if** $(c = 1)$
  Make fractional particle a real particle.
  end if
  if ($t > tEquilibration$)
  Accumulate averages.
  end if
  $t \leftarrow t + \Delta t$
  while $(0 > c \not< 1$
  end if
while $(t \leq t_{max})$

9.2.5.4 Beuter–van Gunsteren method

Beutler and van Gunsteren (1994) developed a grand canonical MD algorithm by weakly coupling the system to a bath of constant chemical potential. Their chemical potential weak coupling (CPWC) algorithm is based on the concepts behind the thermostat of Berendsen et al. (1984). The CPWC algorithm relies on an estimate of the instantaneous chemical potential to drive the system to the correct number of particles. A mean Boltzmann factor (MBF) can be obtained by performing Widom test particle insertions.

$$MBF(N) = \left\langle \exp\left(-\frac{U_{test}}{kT}\right)\right\rangle_N \tag{9.94}$$

The MBF is related to the residual chemical potential via the relationship:

$$\mu^r = -kT\ln(MBF(N)) \tag{9.95}$$

The correct number of particles is estimated using,

$$N(t+\Delta t) = N(t) + round[\alpha_N(MBF(N) - MBF(N(t)))] \tag{9.96}$$

where α_N is a coupling constant and *round* yields the nearest integer of the term in square brackets. Application of the CPWC algorithm has three distinct stages.

Stage 1. Sampling. The value of $MBF(N(t))$ is determined by performing n_w Widom particle insertions for each of the n_1 MD time steps. At the end of this sampling period, Eq. (9.96) is used to determine the change in the number of particles.

Stage 2. Growing/shrinking. A total of n_2 time steps are involved in this stage. If particles are added, a MC estimate is used to determine the energetically most favorable starting points for new particles. The particles are given an initial velocity of zero and the size of their interactions are scaled progressively from zero to their full values. If particles are removed, the particles with the highest potential energy are chosen to be scaled down to zero interaction in n_2 time steps.

Stage 3. Relaxation. The sub-ensemble created following the growing/shrinking phase is simulated for n_3 MD time steps.

9.2.5.5 Parallel algorithm

Vega et al. (1994) reported an algorithm which exploits parallelism. The distinguishing feature of the parallel grand canonical MD (PGCMD) algorithm is that several different *NVT* ensembles are generated with different number of particles. At each time step, a decision is made to change to an ensemble with either greater or fewer number of particles. The change in ensemble, which effectively corresponds to either a particle addition or removal, is accepted with the following probability

$$P^{+} = \min\left[1, \frac{1}{N+1}\exp\left(\frac{\mu}{kT} + \ln\left(\frac{V}{\Lambda^3}\right)\right)\left\langle \exp\left(\frac{-U_{test}}{kT}\right)\right\rangle_N\right] \tag{9.97}$$

$$P^{-} = \min\left[1, \frac{N\exp\left(-\frac{\mu}{kT} - \ln\left(\frac{V}{\Lambda^3}\right)\right)}{\left\langle \exp\left(\frac{-U_{test}}{kT}\right)\right\rangle_{N-1}}\right] \tag{9.98}$$

where P^{+} represents the probability of moving between an ensemble containing N particles to an ensemble containing $N+1$ particles whereas P^{-} is the probability of moving from an ensemble containing N particles to an ensemble containing $N-1$ particles. The implementation of this method is illustrated by Algorithm 9.14.

ALGORITHM 9.14 Parallel grand canonical molecular dynamics (PGCMD) algorithm.

Part 1 Initialization

1.1 Assign values of *N, V, T.*
Assign values required for *NVT* ensemble algorithm.
Assign maximum fluctuation in N (*fluctuation*).
1.2 Generate several ensembles:
loop $i \leftarrow$ (*N - fluctuation*) .... (*N + fluctuation*)
Assign coordinates (r_i) at time (t) = 0 for $ensemble_i$.
Assign molecules initial velocities for $ensemble_i$.
1.2.1 Calculate forces (f_i) in $ensemble_i$.
Calculate acceleration (a_i) for each atom.
1.2.2 Calculate <exp(-*Utest*/*kT*)>$_i$
end *i* **loop**
$i \leftarrow N$

Part 2 Simulation Process

loop
2.1 Solve NVT-ensemble equation of motions for $ensemble_i$.
2.2 Attempt change in ensemble:
if (rand() ≥ 0.5)
2.2.1 Attempt to move to an ensemble with more particles:
Apply eq. (9.96).
if (attempt is successful)
$i \leftarrow i + 1$
end if
else
2.2.2 Attempt to move to an ensemble with fewer particles:
Apply eq. (9.97).
If (attempt is successful)
$i \leftarrow i - 1$
end if
end if
2.3 **if** ($t >$ *tEquilibration*)
Accumulate averages.
end if
$t \leftarrow t + \Delta t$
while ($t \leq t_{max}$)

The initialization phase (Part 1) of Algorithm 9.14 is considerably more complicated than the initialization phase of a normal MD simulation. It involves creating several different ensembles (Algorithm 9.14, Part 1.2) at the same temperature and volume but with a different number of particles. For each ensemble, initial coordinates and forces (Algorithm 9.14, Part 1.2.1) must be calculated in addition to the Widom test particle term (Algorithm 9.14, Part 1.2.2). In Part 2, a particular *NVT* ensemble is selected and the particle coordinates are evolved (Algorithm 9.14, Part 2.1) until a successful attempt is made to change the ensemble (Algorithm 9.14, Part 2.2). The attempt to change to an ensemble with more or fewer particles is made with equal probability. Equation (9.97) is used to determine the success of a move to an ensemble with more particles (Algorithm 9.14, Part 2.2.1), otherwise Eq. (9.98) is used (Algorithm 9.14, Part 2.2.2). The new *NVT* ensemble is in turn evolved until a new ensemble is selected. Consequently, at any time step, only one ensemble is evolving and the other ensembles remain dormant. Ensemble averages are accumulated from the current ensemble in the usual way (Algorithm 9.14, Part 2.3). Any *NVT* ensemble algorithm can be used to evolve the ensemble.

It is apparent that implementing the PGCMD algorithm in conventional procedural languages is awkward, requiring a considerable amount of bookkeeping to keep track of the properties of the various ensembles. In contrast, we note that implementation of PGCMD in an object-oriented language is simply involves using different objects of type ensemble (Chapter 11).

9.3 Summary

The choice of ensemble governs the nature of the simulation algorithm. The natural choice for MC simulations is the canonical ensemble but it is relatively simple to construct MC algorithms for other ensembles. In contrast, the equations of motion lead naturally to a microcanonical ensemble and MD simulation in other ensembles involves altering the equations of motions. The MD implementation of the grand canonical ensemble invariably involves the introduction of stochastic elements. An implementation of both MC and MD simulation in the microcanonical ensemble is given in Chapter 11.

References

Abramowitz, A., Stegun, I.A., 1970. Handbook of Mathematical Functions. Dover, New York.

Adams, D.J., 1974. Chemical potential of hard-sphere fluids by Monte Carlo methods. Mol. Phys. 28, 1241–1252.

Allen, M.P., Schmid, F., 2007. A thermostat for molecular dynamics of complex fluids. Mol. Sim. 33, 21–26.

Allen, M.P., Tildesley, D.J., 2017. Computer Simulation of Liquids, second ed. Oxford University Press, Oxford.

Andersen, H.C., 1980. Molecular dynamics at constant pressure and/or temperature. J. Chem. Phys. 72, 2384–2393.

Baldovin, F., Chavanis, P.-H., Orlandini, E., 2009. Microcanonical quasistationarity of long-range interacting systems in contact with a heat bath. Phys. Rev. E 79, 011102.

Beckedahl, D., Obaga, E.O., Uken, D.A., Sergi, A., Ferrario, M., 2016. On the configurational temperature Nosé-Hoover thermostat. Phyica A 461, 19–35.

Berendsen, H.J.C., Postma, J.P.M., van Gunsteren, W.F., DiNola, A., Haak, J.R., 1984. Molecular dynamics with coupling to an external bath. J. Chem. Phys. 81, 3684–3690.

Beutler, T.C., van Gunsteren, W.F., 1994. Molecular dynamics simulations with first order coupling to a bath of constant chemical potential. Mol. Sim. 14, 21–34.

Boinepalli, S., Attard, P., 2003. Grand canonical molecular dynamics. J. Chem. Phys. 119, 12769–12775.

Bosko, J., Todd, B.D., Sadus, R.J., 2005. Molecular simulation of dendrimers and their mixtures under shear: Comparison of isothermal-isobaric (*NpT*) and isothermal-isochoric (*NVT*) ensemble systems. J. Chem. Phys. 123, 034905.

Braga, C., Travis, K.P., 2005. A configurational temperature Nosé-Hoover thermostat. J. Chem. Phys. 123, 134101.

Brennan, J.K., Madden, W.G., 1998. Efficient volume changes in constant-pressure Monte Carlo simulations. Mol. Sim. 20, 139–157.

Brown, D., Clarke, J.H.R., 1984. A comparison of constant energy, constant temperature, and constant pressure ensembles in molecular dynamics simulations of atomic liquids. Mol. Phys. 51, 1243–1252.

Bussi, G., Zykova-Timan, T., Parrinello, M., 2009. Isothemal-isobaric molecular dynamics using stochastic velocity rescaling. J. Chem. Phys. 130, 074101.

Butler, B.D., Atyon, G., Jepps, O.G., Evans, D.J., 1998. Configurational temperature verification of Monte Carlo simulations. J. Chem. Phys. 109, 6519–6522.

Çagin, T., Pettitt, B.M., 1991a. Grand molecular dynamics: a method for open systems. Mol. Sim. 6, 5–26.

Çagin, T., Pettitt, B.M., 1991b. Molecular dynamics with a variable number of molecules. Mol. Sim. 72, 169–175.

Cano, B., Stuart, A.M., 2001. Underresolved simulations of heat baths. J. Comput. Phys. 169, 193–214.

Creutz, M., 1983. Microcanonical Monte Carlo simulation. Phys. Rev. Lett. 50, 1411–1414.

Dubbeldam, D., Torres-Knoop, A., Walton, K.S., 2013. Monte Carlo codes, tools and algorithms: on the inner workings of Monte Carlo codes. Mol. Sim. 39, 1253–1292.

Evans, D.J., 1983. Computer "experiment" for nonlinear thermodynamics of Couette flow. J. Chem. Phys. 78, 3297–3302.

Evans, D.J., Morriss, G.P., 1984. Non-Newtonian molecular dynamics. Comp. Phys. Rep. 1, 297–344.

Frenkel, D., Smit, B., 2023. Understanding, Molecular Simulation. From Algorithms to Applications, third ed. Academic Press, San Diego.

Fukuda, I., 2016. Coupled Nosé-Hoover lattice: a set of the Nosé-Hoover equations with different temperatures. Phys. Lett. A 380, 2465–2474.

Fukuda, I., Moritsugu, K., 2016. Coupled Nosé-Hoover equations of motion to implement a fluctuating heat-bath temperature. Phys. Rev. E 93, 033306.

Fukuda, I., Moritsugu, K., 2017. Coupled Nosé-Hoover equations of motion without time scaling. J. Phys. A: Math. Theor. 50, 015002.

Fukuda, I., Moritsugu, K., 2020. Temperature-energy-space sampling molecular dynamics: deterministic and single-replica method utilizing continuous temperature system. J. Phys. A: Math. Theor. 53, 375004.

Hansen, J.-P., McDonald, I.R., 2013. Theory of Simple Liquids, fourth ed. Academic Press, Amsterdam.

Heermann, D.W., 1990. Computer Simulation Methods in Theoretical Physics, second ed. Springer-Verlag, Heidelberg.

Heyes, D.M., 1998. The Liquid State: Applications of Molecular Simulations. John Wiley & Sons, Chichester.

Hoover, W.G., 1983. Nonequilibrium molecular dynamics. Ann. Rev. Phys. Chem. 34, 103–127.

Hoover, W.G., 1985. Canonical dynamics: equilibrium phase-space distribution. Phys. Rev. A. 31, 1695–1697.

Hoover, W.G., 2007. Nosé-Hoover nonequilibrium dynamics and statistical mechanics. Mol. Sim. 33, 13–19.

Hoover, W.G., 1991. Computational Statistical Mechanics. Elsevier, Amsterdam.

Hoover, W.G., Ladd, A.J.C., Moran, B., 1982. High strain rate plastic flow studied via nonequilibrium molecular dynamics. Phys. Rev. Lett. 48, 1818–1820.

Hu, J., Ma, A., Dinner, A.R., 2005. Monte Carlo simulations of biomolecules: the MC module in CHARMM. J. Comput. Chem. 27, 203–216.

Huang, C., Li, C., Choi, P.Y.K., Nandakumar, K., Kostiuk, L.W., 2011. A novel method for molecular dynamics simulation in the isothermal-isobaric ensemble. Mol. Phys. 109, 191–202.

Hüneneberger, P.H., 2005. Thermostat algorithms for molecular dynamics simulations. Adv. Polym. Sci. 173, 105–149.

Jang, S., Voth, G., 1997. Simple reversible molecular dynamics algorithms for Nosé-Hoover chain dynamics. J. Chem. Phys. 107, 9514–9526.

Jang, S., Pak, Y., Shin, S., 2002. Multicanonical ensemble with Nosé-Hoover molecular dynamics simulation. J. Chem. Phys. 116, 4782–4786.

Ji, J., Çagin, T., Pettitt, B.M., 1992. Dynamic simulations of water at constant chemical potential. J. Chem. Phys. 96, 1333–1342.

Jiang, S., Voth, G.A., 1997. Simple reversible molecular dynamics algorithms for Nosé-Hoover chain dynamics. J. Chem. Phys. 107, 9514–9526.

Kim, M., Kim, E., Lee, S., Kim, J.S., Lee, S., 2019. J. Phys. Chem. A 123, 1689–1699.

Kioupis, L.I., Maginn, E.J., 2002. Pressure-enthalphy driven molecular dynamics for thermodynamic property calculation. I. Methodology. Fluid Phase Equilib. 200, 75–92.

Kouza, M., Hansmann, U.H.E., 2011. Velocity scaling for optimizing replica exchange molecular dynamics. J. Chem. Phys. 134, 044124.

Kuang, S., Gezelter, J.D., 2010. A gentler approach to RNEMD: nonisotropic velocity scaling for computing thermal conductivity and shear viscosity. J. Chem. Phys. 133, 164101.

Kuang, S., Gezelter, J.D., 2012. Velocity shearing and scaling RNEMD: a minimally perturbing method for simulating temperature and momentum gradients. Mol. Phys. 110, 691–701.

Kutteh, R., 1999a. New approaches for molecular dynamics simulations with nonholonomic constraints. Comp. Phys. Commun. 119, 159–168.

Kutteh, R., 1999b. New methods for incorporating nonholonomic constraints into molecular dynamics simulations. J. Chem. Phys. 111, 1394–1406.

Kuznetsova, T., Kvamme, B., 1999. Grand canonical molecular dynamics for TIP4P water systems. Mol. Phys. 97, 423–431.

Lo, C., Palmer, B., 1995. Alternative Hamiltonian for molecular dynamics simulations in the grand canonical ensemble. J. Chem. Phys. 102, 925–931.

Lupkowski, M., van Swol, F., 1991. Ultrathin films under shear. J. Chem. Phys. 95, 1995–1998.

Lustig, R., 1994. Statistical thermodynamics in the classical molecular dynamics ensemble. II. Application to computer simulation. J. Chem. Phys. 100, 3060–3067.

Lustig, R., 1998. Microcanonical Monte Carlo simulation of thermodynamic properties. J. Chem. Phys. 109, 8816–8828.

Lustig, R., 2011. Direct molecular NVT simulation of the isobaric heat capacity, speed of sound and Joule-Thomson coefficient. Mol. Sim. 37, 457–465.

Lynch, G.C., Petititt, B.M., 1997. Grand canonical ensemble molecular dynamics simulations: Reformation of extended system dynamics approaches. J. Chem. Phys. 107, 8594–8610.

Martyna, G.J., Klein, M.L., Tuckerman, M., 1992. Nosé-Hoover chains: The canonical ensemble via continuous dynamics. J. Chem. Phys. 97, 2635–2643.

Martyna, G.J., Tobias, D.J., Klein, M.L., 1994. Constant pressure molecular dynamics algorithms. J. Chem. Phys. 101, 4177–4189.

McDonald, I.R., 1972. NpT-ensemble Monte Carlo calculations for binary liquid mixtures. Mol. Phys. 23, 41–58.

McGrath, M.J., Siepmann, J.I., Kuo, I.-F.W., Mundy, C.J., VandeVondele, J., Hutter, J., Mohamed, F., Krack, M., 2005. Isobaric-isothermal Monte Carlo simulations from first principles: application to liquid water at ambient conditions. ChemPhysChem 6, 1894–1901.

Melchionna, S., Ciccotti, G., Holian, B.L., 1993. Hoover NPT dynamics for systems varying in shape and size. Mol. Phys. 78, 533–544.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., 1953. Equation of state calculations by fast computing machines. J. Chem. Phys. 21, 1087–1092.

Morishita, T., 2000. Fluctuation formulas in molecular-dynamics simulations with weak coupling heat bath. J. Chem. Phys. 113, 2976–2982.

Morishita, T., 2003a. Modified velocity scaling for molecular dynamics at constant temperature and/or pressure. Mol. Sim. 29, 63–69.

Morishita, T., 2003b. Generalized coupling to a heath bath: extension of the Gaussian isokinetic dynamics and effect of time scaling. J. Phys. Chem. 119, 7075–7082.

Morishita, T., 2010. From Nosé-Hoover chain to Nosé-Hoover network: design on non-Hamiltonian equations of motion for molecular dynamics with multiple thermostats. Mol. Phys. 108, 1137–1347.

Nicholson, D., Parsonage, N.G., 1982. Computer Simulation and the Statistical Mechanics of Adsorption. Academic Press, New York.

Nitzke, I., Vrabec, J. 2023. Numerical discrimination of thermodynamic Monte Carlo simulations in all eight statistical ensembles. J. Chem. Theory Comput. 19, 3460–3468.

Nosé, S., 1984. A unified formulation of the constant temperature molecular dynamics methods. J. Chem. Phys. 81, 511–519.

Okumura, H., Okamoto, Y., 2004a. Molecular Carlo simulations in the multibaric–multithermal ensemble. Chem. Phys. Lett. 383, 391–396.

Okumura, H., Okamoto, Y., 2004b. Monte Carlo simulations in generalized isobaric-isothermal ensembles. Phys. Rev. E 70, 026702.

Orkoulas, G., 2007. Acceleration of Monte Carlo simulations through spatial updated in the grand canonical ensemble. J. Chem. Phys. 127, 084106.

Page, A.J., Isomoto, T., Knaup, J.M., Irle, S., Morokuma, K., 2012. Effects of molecular dynamics thermostats on descriptions of chemical nonequilibrium. J. Chem. Theory Comput. 8, 4019–4028.

Palma, G., Riveros, A., 2021. General method to sample systems in the microcanonical ensemble using Monte Carlo simulations. Eur. Phys. J. B 94, 23.

Papadopoulou, A., Becker, E.D., Lupkoski, M., van Swol, F., 1993. Molecular dynamics and Monte Carlo simulations in the grand canonical ensemble: Local versus global control. J. Chem. Phys. 98, 4897–4908.

Parrinello, M., Rahman, A., 1980. Crystal structure and pair potentials: a molecular dynamics study. Phys. Rev. Lett. 45, 1196–1199.

Parrinello, M., Rahman, A., 1981. Polymorphic transitions in single crystals: a new molecular dynamics method. J. Appl. Phys. 52, 7182–7190.

Parrinello, M., Rahman, A., 1982. Strain fluctuations and elastic constants. J. Chem. Phys. 76, 2662–2666.

Pearson, E.M., Halicioglu, T., Tiller, W.A., 1985. Laplace-transform technique for deriving thermodynamic equations from the classical microcanonical ensemble. Phys. Rev. A 32, 3030–3039.

Rapaport, D.C., 2004. The Art of Molecular Dynamics Simulation, second ed. Cambridge University Press, Cambridge.

Ray, J.R., 1991. Microcanonical ensemble Monte Carlo method. Phys. Rev. A. 44, 4061–4064.

Samways, M.L., Bruce Macdonald, H.E., Essex, J.W., 2020. Grand: A python module for grand canonical water sampling in OpenMM. J. Chem. Info. Model. 60, 4436–4441.

Sastre, F., Benavides, A.L., Torres-Areanas, J., Gil-Villegas, A., 2015. Microcanonical ensemble simulation method applied to discrete potential fluids. Phys. Rev. E 92, 033303.

Sastre, F., Moreno-Hilario, E., Sotelo-Serna, M.G., Gil-Villegas, A., 2018. Microcanonical-ensemble computer simulation of the high-temperature expansion coefficients of the Helmholtz free energy of a square-well fluid. Mol. Phys. 116, 351–360.

Schierz, P., Zierenberg, J., Janke, W., 2015. Molecular dynamics and Monte Carlo simulations in the microcanonical ensemble: quantitative comparison and reweighting techniques. J. Chem. Phys. 143, 134114.

Shi, W., Maginn, E.J., 2007. Continuous fractional component Monte Carlo: An adaptive biasing method for open atomistic simulations. J. Chem. Theory Comput. 3, 1451–1463.

Shida, C.S., Henriques, V.B., de Oliveira, M., 2003. Microcanonical Monte Carlo simulation of lattice gas models. Phys. Rev. E 68, 066125.

Sobol', I.M., 1994. A Primer for the Monte Carlo Method. CRC Press, Boca Raton.

Ströker, P., Meier, K., 2022. Rigorous expressions for thermodynamic properties in the *NpH* ensemble. Phys. Rev. E 105, 035301.

Ströker, P., Hellmann, R., Meier, K., 2021. Systematic formulation of thermodynamic properties in the *NpT* ensemble. Phys. Rev. E 103, 023305.

Stutzman, L.B., Escobedo, F.A., Tester, J.W., 2018. Heat capacities of supercritical fluids via grand canonical ensemble simulations. Mol. Sim. 44, 147–155.

Tanaka, H., Nakanishi, K., Watanabe, N., 1983. Constant temperature molecular dynamics calculation on Lennard-Jones fluid and its application to water. J. Chem. Phys. 78, 2626–2634.

Tchouar, N., Benyettou, M., Baghli, H., 2007. A reversible algorithm for Nosé molecular dynamics simulations. Equilibrium properties of liquids methane. J. Mol. Liq. 136, 5–10.

Travis, K.P. (2023), in preparation.

Travis, K.P., Braga, C., 2006. Configurational temperature and pressure molecular dynamics: review of current methodology and applications to the shear flow of a simple fluid. Mol. Phys. 104, 22–24.

Travis, K.P., Braga, C., 2008. Configurational temperature control for atomic and molecular systems. J. Chem. Phys. 128, 014111.

Travis, K.P., Daivis, P.J., Evans, D.J., 1995. Computer simulation algorithms for molecules undergoing planar Couette flow: a nonequilibrium molecular dynamics study. J. Chem. Phys. 103, 1109–1118.

Turesson, M., Woodward, C.E., Åkesson, T., Forsman, 2008. Simulating equilibrium surface forces in polymer solutions using a canonical grid method. J. Phys. Chem. B 112, 9802–9809.

Valleau, J.P., Cohen, L.K., 1980. Primitive model electrolytes. I. Grand canonical Monte Carlo computations. J. Chem. Phys. 72, 5935–5941.

Vega, L.F., Shing, K.S., Rull, L.F., 1994. A new algorithm for molecular dynamics simulations in the grand canonical ensemble. Mol. Phys. 82, 439–453.

Vlasiuk, M., Frascoli, F., Sadus, R.J., 2016. Molecular simulation of the thermodynamic, structural and vapor-liquid equilibrium properties of neon. J. Chem. Phys. 145, 104501.

Weerasinghe, S., Pettitt, B.M., 1994. Ideal chemical potential contribution in molecular dynamics simulations of the grand canonical ensemble. Mol. Phys. 82, 897–911.

Weinan, E., Dong, E., 2008. The Andersen thermostat in molecular dynamics. Commun. Pure Appl. Math. 61, 96–136.

Yu, Y., Wang, J., Shao, Q., Shi, J., Zhu, W., 2015. Increasing the sampling efficiency of protein conformational transition using velocity-scaling optimized hybrid explicit/implicit solvent REMD simulation. J. Chem. Phys. 142, 125105.

Zhou, Y., Huai, X., Lin, L., 2010. Molecular dynamics simulation for homogeneous nucleation of water and liquid nitrogen in explosive boiling. Appl. Therm. Eng. 30, 859–863.

Chapter 10

Molecular simulation of phase equilibria

Molecular simulation is performed most commonly for a single isolated system characterized by certain physical properties that define the nature of the ensemble (Chapter 2). In effect, these simulations represent the properties of the fluid for a single isolated phase. However, the techniques of molecular simulation can be extended to multiple phases, providing a powerful tool for the investigation of the phase coexistence of both pure fluids and fluid mixtures. Historically, the application of molecular simulation to phase equilibria has lagged applications to other areas. This can be attributed partly to the difficulty in obtaining free energy or chemical potential data required for phase coexistence. Phase transitions for elastic disks were first reported by Alder and Wainwright (1962), and Hoover and Ree (1967) located a melting transition via molecular simulation. Hansen and Verlet (1969) reported simulations of vapor−liquid equilibria (VLE) for a Lennard-Jones (LJ) fluid (Chapter 3). Later, Adams (1976, 1979) reported extensive Monte Carlo (MC) (Chapter 6) calculations for the vapor line of a LJ fluid. The advent of the Gibbs ensemble (Panagiotopoulos, 1987) has been the impetus for a massive increase in the number, range, and scope of phase equilibria simulations.

Conceptually, the most straightforward method for simulating coexisting phases is to divide the simulation box into distinct regions. For example, VLE can be simulated in a box containing three distinct regions that correspond the liquid phase, vapor phase, and an interfacial region, respectively. The simulation is initiated by introducing a slab of previously equilibrated liquid into the center of the simulation box with a vacuum on either side. If the density of the liquid is close to coexisting density, the liquid will evaporate into the vacuum to form a stable two-phase system. This approach is particularly useful for interfacial studies (Rowlinson and Widom, 1982). A disadvantage of this approach is that the density must be known in advance. However, Chapela et al. (1987) have reported a variation of the traditional approach, which overcomes this difficulty.

The two-phase method is conceptually appealing because it corresponds directly with experimental observation. However, the disadvantages of the method are numerous. The simulations are slow because a large number of molecules are required and the equilibration of the interfacial region is a slow process. It is difficult to simulate stable phase equilibria between phases of similar densities.

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00013-7

Furthermore, the confinement of the fluid between parallel walls can potentially influence both the density and pressure of the coexisting phases resulting in a simulation of the properties of confined fluids rather than a bulk fluid.

The limitations of the traditional two-phase method are overcome by using the Gibbs ensemble (Panagiotopoulos, 1987, 1992a). The Gibbs ensemble method is characterized by two homogeneous phases that are in thermodynamic equilibrium but not in physical contact. Unlike the two-phase method, there is no interface thus resulting in much shorter run times. The Gibbs ensemble was formulated originally as a MC method, however, a few molecular dynamics (MD) (Chapter 9) implementations of the Gibbs ensemble have been proposed (Palmer and Lo, 1994; Vega et al., 1994; Kotelyanskii and Hentschke, 1995; Baranyai and Cummings, 1996).

Indirect methods provide an alternative to direct methods of simulating phase equilibria. Indirect methods work by calculating the chemical potential for a series of state points. Phase coexistence can be determined by finding points in the state space where two phases have the same chemical potential, pressure, and temperature. Conventional molecular simulations do not provide information on thermal properties such as free energies but instead yield mechanical properties such as the energy and pressure. Consequently, special techniques are required such as test particle methods, umbrella sampling, grand canonical and semigrand ensembles, and thermodynamic integration. Comprehensive reviews of these methods are available elsewhere (Gubbins et al., 1983; Gubbins, 1989, 1994; Frenkel, 1986). Generally, indirect methods are less efficient computationally than direct methods.

It should be noted that progress in the simulation of phase equilibria is being made continually (Kofke, 1993a; de Pablo et al., 1999; Bruce and Wilding, 2003; Panagiotopoulos, 2000). For example, efficient histogram reweighting algorithms have been reported (Kiyohara et al., 1997, 1998), which can in turn be used in conjunction with transition-matrix MC (Shen and Errington, 2005; Singh, 2018). The focus has been mainly on VLE, however, improved algorithms for solid−liquid equilibria (SLE) have also been reported (Ge et al., 2003; Singh et al., 2021; Deiters and Sadus, 2022a,b). We will largely restrict our attention to advances made post-1985. Comprehensive reviews of earlier work are available elsewhere (Rowlinson and Widom, 1982; Gubbins et al., 1983; Frenkel, 1986; Gubbins, 1989, 1994; Frenkel and Smit, 2023).

10.1 Calculating the chemical potential

The criteria for phase equilibria requires the equivalence of temperature (thermal equilibrium), pressure (mechanical equilibrium), and chemical potential (material equilibrium) for every particle in the coexisting phases. Consequently, the chemical potential is a very important quantity, particularly for indirect methods that rely on an accurate determination of the chemical potential. In other methods such as the Gibbs ensemble, which does not

rely on knowledge of the chemical potential, calculating the chemical potential can serve as independent verification that equilibrium has been attained.

The most common approach for calculating the chemical potential is the Widom test particle method (Widom, 1963). The method involves inserting a "ghost" particle (i) randomly into the ensemble and calculating the energy of its interaction ($U_{\text{i,test}}$) with the particles of the ensemble. For a canonical ensemble, the residual chemical potential (i.e., the chemical potential minus the contribution from the ideal gas) is obtained subsequently from the following ensemble average:

$$\mu_{ir} = -kT\ln\left\langle \exp(-\beta U_{itest})\right\rangle_N \tag{10.1}$$

It should be stressed that because the test particle is a "ghost," it does not affect the properties of the N real molecules of the ensemble. Eq. (10.1) can be applied to every type of molecule and it is completely independent of any assumption concerning pairwise additivity or the nature of intermolecular forces. Eq. (10.1) can be incorporated easily into a conventional molecular simulation. In principle, it can be used to calculate the chemical potential at any density, however, in practice some difficulties are encountered at high density. At high density, attractive configurations make the dominant contribution to Eq. (10.1) but they are sampled less frequently as the density is increased because of the $\exp(-\beta U_{itest})$ term.

Strictly, Eq. (10.1) is only valid for the canonical ensemble. In the NpT ensemble, density variations mean that Eq. (10.1) should be replaced with:

$$\mu_{ir} = -kT\ln\left[\frac{\left\langle V\exp(-\beta U_{itest})\right\rangle}{\langle V\rangle}\right] \tag{10.2}$$

Generally, either Eqs. (10.1) or (10.2) yield similar results. However, there are significant differences near the critical region.

For the Gibbs ensemble, the correct theoretical value of the chemical potential can be calculated from (Smit and Frenkel, 1989):

$$\mu_{ir} = -kT\ln\left\langle \frac{V}{N+1}\exp(-\beta U_{itest})\right\rangle \tag{10.3}$$

In practice, the values obtained from Eq. (10.3) only vary significantly from the results of Eq. (10.1) in the region of the critical point.

As noted earlier, the simple test particle method fails at high densities. Several strategies have been proposed to improve the calculation of the chemical potential. One possibility is to use umbrella sampling (Torrie and Valleau, 1974). The principle behind umbrella sampling is to assign a weight (w) to the configurational energy (U). The weighting function is chosen to sample regions of E that are important to the chemical potential. For example, in a MC simulation, configurations are chosen with a probability of $w(U_{test})\exp(-\beta U)$ instead of $\exp(-\beta U)$. Consequently, the ensemble average in Eq. (10.1) is obtained from:

$$\left\langle \exp(\beta U_{itest})\right\rangle = \frac{\left\langle \exp(-\beta U_{itest})/w\right\rangle}{\langle 1/w\rangle} \tag{10.4}$$

This procedure works well for LJ fluids (Torrie and Valleau, 1977) at high liquid densities. However, the limitations of the procedure are that appropriate weights can only be obtained by trial and error and, unlike the test particle method, it requires considerable modification to the conventional MC simulation algorithm.

An alternative to "ghost" particle insertions is to use real molecules to evaluate the chemical potential. When real molecules are used as test particles, the equation for the chemical potential becomes (Shing and Gubbins, 1982).

$$\mu_{ir} = kT\ln\left\langle \exp(-\beta U_{itest})\right\rangle_{N+1} \tag{10.5}$$

This equation can be rewritten (Shing and Gubbins, 1982) as,

$$\mu_{ir} = kT\ln \int_{-\infty}^{\infty} g(U_{itest})\exp(-\beta U_{itest})dU_{itest} \tag{10.6}$$

where $g(U_{itest})dU_{itest}$ is the probability that U_{itest} for a real test molecule lies in the range U_{itest} to $U_{itest} + dU_{itest}$. For a given U_{itest}, the distribution functions f and g are related by:

$$g(U_{itest}) = \exp(\beta\mu_{ir})\exp(-\beta U_{itest})f(U_{itest}) \tag{10.7}$$

The averages in Eqs. (10.5) and (10.6) are dominated by repulsive configurations that are likely to be positive even at moderate densities. Therefore, it should be noted that special sampling techniques are required (Shing and Gubbins, 1982; Kumar, 1992) to take advantage of these equations.

The calculation of the chemical potential is an area of continuing research interest. Rowley et al. (1994, 1995) proposed an osmotic MD method for calculating the chemical potential. Their method involves establishing an osmotic equilibrium across a semipermeable membrane. The chemical potential is calculated using the mechanical properties of the fluid on either side of the membrane. Swope and Andersen (1995) reported calculations of the chemical potential in a "bicanonical ensemble." The bicanonical ensemble is characterized by a temperature and a volume in which all systems have either N or $N-1$ particles. The chemical potential is related to the fraction of the systems that have N particles. Sokhan (1997) developed a coupled test particle approach for determining the chemical potential at high liquid densities. Soto-Campos et al. (1998) have discussed the calculation of chemical potentials using a small system grand ensemble method.

10.2 Gibbs ensemble Monte Carlo

In principle, because the grand canonical ensemble allows chemical potential to be specified, it can be used to determine phase coexistence. However, the grand canonical ensemble is not a practical method for calculating phase coexistence because a large number of simulations are required to determine the phase envelope, and the computational cost increases dramatically for two or more

components. The chemical potential can also be calculated by using Widom's test particle approach in conjunction with a *NVT* ensemble simulation. The most convenient and widely used method for calculating phase coexistence is to use the Gibbs ensemble (Panagiotopoulos, 1987; Panagiotopoulos et al., 1988).

Panagiotopoulos (1987) proposed that phase coexistence could be simulated by following the evolution in phase space of a system composed of two distinct regions. Equilibrium between the different region or phases is achieved by performing three distinct MC moves as illustrated by Fig. 10.1. Molecular displacements are attempted to obtain internal equilibrium resulting in equivalence of temperatures. Mechanical equilibrium requires that the pressure in the two regions are identical and this is obtained by allowing fluctuations in volume. Attempts are made to exchange particles between the regions to obtain material equilibrium. The acceptance of the three moves are governed by the normal acceptance criterion for MC moves involving the "pseudo-Boltzmann term" (min [1, exp($-\beta\Delta Y$)]) discussed in Chapter 6. In the case of molecular displacement,

$$\Delta Y_{disp} = \Delta U_{\alpha} + \Delta U_{\beta} \tag{10.8}$$

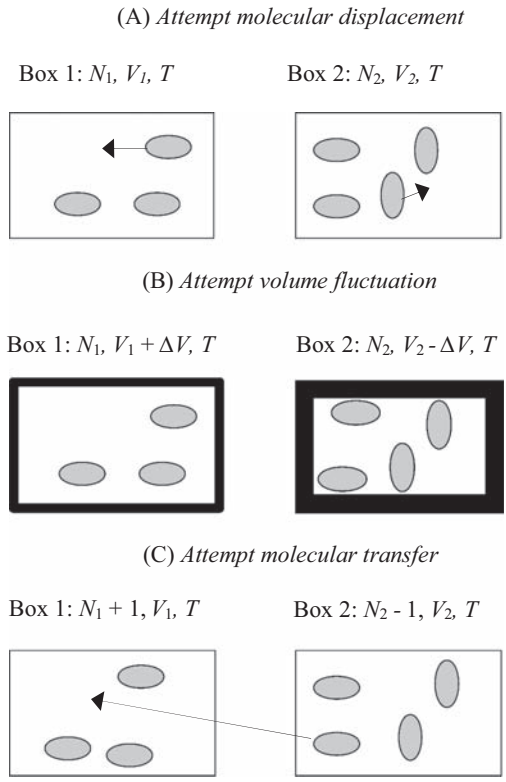


FIGURE 10.1 Illustration of MC moves used in the Gibbs ensemble.

where the subscripts α and β denote the two different regions or simulation boxes. For volume changes, we have:

$$\Delta Y_{volume} = \Delta U_\alpha + \Delta U_\beta - N_\alpha kT \ln \frac{V_\alpha + \Delta V_\alpha}{V_\alpha} - N_\beta kT \ln \frac{V_\beta + \Delta V_\beta}{V_\beta} + p(\Delta V_\alpha + \Delta V_\beta) \tag{10.9}$$

In the Gibbs ensemble, the total number of particles is conserved. Consequently, a successful particle transfer from box β to box α implies a particle destruction in box β. For particle transfers, we obtain:

$$\Delta Y_{trans} = \Delta U_\alpha + \Delta U_\beta + kT \ln \frac{V_\beta (N_\alpha + 1)}{V_\alpha N_\beta} \tag{10.10}$$

A general scheme for a Gibbs ensemble MC (GEMC) calculation is illustrated by Algorithm 10.1. A feature of Algorithm 10.1 is that the sequence of moves is determined randomly instead of being predetermined. It is possible to use a predetermined sequence, however, when the probability of particle transfer exceeds 3%, erroneous results may be obtained (Smit, 1992).

In the particle displacement step (Algorithm 10.1, Part 2.1), new particle coordinates are chosen randomly, whereas in the volume change step (Algorithm 10.1, Part 2.2), the length of the simulation box is either decreased or increased by a random amount. Typically, at least a 50% acceptance rate is desirable to achieve equilibrium efficiently, however, this is only a guide and it does not have any theoretical justification. The acceptance rate can be adjusted by altering the values of the maximum possible displacement and maximum change in box length. For many intermolecular potentials (Chapters 3 and 4), the effect of a change in volume can be calculated simply by scaling the molecular coordinates (see Chapter 8).

The particle transfer step (Algorithm 10.1, Part 2.3) involves selecting randomly a molecule in one box and attempting to insert it randomly in the other box. It is difficult to prescribe exactly a desirable acceptance rate for particle transfer. In the case of mixtures, it is important that each particle is selected with equal probability to satisfy the requirements of thermodynamic reversibility. Typically, an acceptance rate of a few percent is usually sufficient for material equilibrium. During the particle transfer test, the chemical potential can be easily calculated using Widom's test particle approach. Identical, or at least very similar chemical potentials in the boxes, indicate that the acceptance rate is satisfactory and that material equilibrium has been achieved.

Eqs. (10.8)–(10.10) can be used to generate either a *NVT* Gibbs ensemble or an *NpT* ensemble depending on the treatment of volume fluctuations in Eq. (10.9). For the *NVT* Gibbs ensemble, the total volume of the two boxes is conserved, that is, $\Delta V_\alpha = -\Delta V_\beta$, whereas for the *NpT* ensemble, the volume fluctuation of the two boxes are independent of each other. An alternative implementation of the Gibbs ensemble, which allows heat exchange between the phases, is discussed elsewhere (Kristóf and Liszi, 1998).

ALGORITHM 10.1 Gibbs ensemble MC.

```
Part 1    Initialization:
          Specify the number of cycles (nCycles), number of particles in
          the boxes (nBox1, nBox2), number of volume (nVol), transfer
          attempts (nTrans), and the total number of moves
          (nTotal = nBox1 + nBox2 + nVol + nTrans).

Part 2    Generate Markov chain:
          loop i ←1 ... nCycles
             loop j ←1 ... nTotal
                if (rand() ≤ nBox1/nTotal)     //Decide which move to perform
Part 2.1           Displace particle in box 1.
                else if(rand() ≤ (nBox1 + nBox2)/nTotal)
                   Displace particle in box 2.
                else if(rand() ≤ (nBox1 + nBox2 + nVol)/nTotal)
Part 2.2           Change volume.
                else if(rand() ≤
                         (nBox1 + nBox2 + nVol + 0.5nTrans)/nTotal)
Part 2.3           Transfer particle from box 1 to box 2.
                else
                   Transfer particle from box 2 to box 1.
                end if
             end j loop
          end i loop
```

GEMC is arguably the most commonly used algorithm for fluid phase equilibria (Panagiotopoulos, 1992a; Panagiotopoulos, 2000; Silva Fernandes and Fartaria, 2015). It has been used widely to predict VLE in both conventional one-component fluids and binary mixtures and extended to cover quantum systems (Kowalczyk et al., 2013; Fantoni and Moroni, 2014). It has been used in conjunction with many empirical intermolecular potentials (Chapters 3 and 4) including the LJ (Panagiotopoulos, 1987; Plačkov and Sadus, 1997; Potoff and Panagiotopoulos, 1998), square-well (Vega et al., 1992), Stockmayer (Smit et al., 1989), restrictive primitive (Panagiotopoulos, 1992b), Yukawa (Rudisill and Cummings, 1989), Gay−Berne (de Miguel et al., 1990), Keesom (Sadus, 1996a), Mie (Mick et al., 2015; Sadus, 2020a,b), and double-Gaussian core model (Losey and Sadus, 2019) potentials. The VLE behavior of various water models (Chapter 4) have been evaluated (Raabe and Sadus, 2007) and it has been used (Vogt et al., 2001; Vlasiuk et al., 2016; Vlasiuk and Sadus, 2017; Deiters and Sadus, 2019) to evaluate the pure component VLE behavior displayed by ab initio potentials. GEMC has also been applied (Sadus, 1998a; Marcelli and Sadus, 1999; Wang and Sadus, 2006a; Sadus, 2020c; Raabe and Sadus, 2003) to study the relative effect of two- and three-body interaction on the VLE of pure fluids. Contrary to some earlier indications (Valleau, 2003), there is evidence (Dinpajooh et al., 2015) that GEMC can be used to accurately determine VLE critical points using relatively small system sizes.

Examples of GEMC binary VLE simulations include mixtures modeled via the LJ (van Leeuwen et al., 1991), hard-sphere/soft-sphere (Amar, 1989), Keesom (Sadus, 1996b), and Mie (Potoff and Kamath, 2014; Mick et al., 2015) potentials. The VLE of hydrocarbon mixtures (de Pablo and Prausnitz, 1989), polydisperse fluids (Stapleton et al., 1990), and the effect of three-body interactions in mixtures (Marcelli and Sadus, 2001) have also been determined using GEMC. In general, the results are remarkably insensitive to the number of molecules used with the possible exception of near-critical equilibria (Valleau, 1998). The method has been applied to the calculation of Henry's constant (Sadus, 1997) at vapor−liquid and liquid−liquid phase boundaries. The Gibbs ensemble can also be used in conjunction with a "reaction ensemble" to simulate phase-dependent chemical reactions (Johnson et al., 1994; Smith and Triska, 1994).

Binary mixtures also provide the first opportunity to study liquid−liquid equilibria (LLE). GEMC can also be used for LLE, although greater difficulties are encountered in the particle exchange step, which now involve the insertion of atoms or molecules between phases of similar densities. Nonetheless, successful GEMC for LLE in binary mixtures have been reported for several different cases (Sadus, 1996b, 1998b; Bergermann et al., 2021a,b).

As discussed previously, GEMC can be easily applied to one, two, three, or more components. However, applications of the GEMC have been overwhelmingly restricted to either pure components or binary mixtures. This situation, which is common to other phase behavior algorithms, reflects the relative interest in such systems rather than any theoretical impediment. Indeed, the ease of extending GEMC to multicomponent mixtures would make it arguably the method of choice for systems containing three or more components. A few examples of GEMC simulations for ternary mixtures have been reported (Tsang et al., 1995; Sadus, 1999).

The main limitation of the MC Gibbs ensemble method is that it is difficult to apply successfully for highly nonspherical, multisegment, or strongly interacting molecules because the particle transfer step has a very low probability of acceptance for these molecules. In this context, the use of the Gibbs ensemble algorithm with configurational bias methods is advantageous (see Chapter 6). Gromov et al. (1998) have reported calculations of polymer-supercritical solvent mixtures using an "expanded" Gibbs ensemble in conjunction with a multiple-time-step hybrid MC technique, which alleviates this difficulty. This issue has been addressed using a continuous fractional component (CFC) MC move (Shi and Maginn, 2008), which involves assigning the strength of particle interaction on a scale of 0 (ideal gas) to 1 (normal). The CFC MC approach results in increased efficiency (Rahbari et al., 2021) of particle exchanges between phases. Another possible solution to this problem is the formulation of a MD equivalent of the Gibbs ensemble (Palmer and Lo, 1994; Hentschke et al., 1996). The MD adaption (discussed below) also makes use of fractional particles to facilitate particle exchanges.

The difficulty of particle transfer means that GEMC is not a practical algorithm for SLE, although an attempt (Sweatman and Quirke, 2004) has

been made to address this issue. Eike et al. (2005) have discussed the challenges posed by SLE for simulation algorithms in general.

10.3 MD Gibbs ensemble

The Gibbs ensemble was developed initially as a MC technique for calculating phase equilibria. The main limitation of the MC implementation of the Gibbs ensemble is that it does not handle large molecules efficiently. It is difficult to design MC moves for molecules with large or complicated molecular structures that sample configuration space efficiently, and successful particle exchanges become increasingly rare. To overcome these difficulties, some MD Gibbs ensemble methods have been developed (Palmer and Lo, 1994; Kotelyanskii and Hentschke, 1995; Baranyai and Cummings, 1996) as detailed below. Very little use has been made of these MD adaptations. These methods have also be supplemented by MD simulations (Kuznetsova and Kvamme, 2005; Eslami and Müller-Plathe, 2007) of the grand canonical ensemble.

10.3.1 Lo–Palmer method

A MD implementation of the Gibbs ensemble has been proposed based on the following Hamiltonian (Palmer and Lo, 1994).

$$\begin{aligned} H = & \sum_{i=1}^{N_\alpha} \frac{\mathbf{p}_i^2}{2m_i} + u_\alpha(\mathbf{q}^{N_\alpha}) + \sum_{j=1}^{N_\beta} \frac{\mathbf{p}_j^2}{2m_j} + u_\beta(\mathbf{q}^{N_\beta}) \\ & + \frac{\mathbf{p}_{\alpha f}^2}{2m_f} + u_{\alpha f}(\mathbf{q}^{N_\alpha}, \mathbf{q}_{\alpha f}, c_\alpha) + \frac{\mathbf{p}_{\beta f}^2}{2m_f} + u_{\beta f}(\mathbf{q}^{N_\beta}, \mathbf{q}_{\beta f}, c_\beta) \\ & + \frac{\mathbf{p}_{s_1}^2}{2Q_1} + \frac{\mathbf{p}_{s_2}^2}{2Q_2} + g_1 kT \ln s_1 + g_2 kT \ln s_2 + \frac{\mathbf{p}_{V_\alpha}^2}{2W} + \frac{\mathbf{p}_{c_\alpha}^2}{2X} \\ & + c_\alpha kT \ln\left[V_\beta (N_\alpha + 1)! N_\beta!\right] + c_\beta kT \ln\left[V_\alpha (N_\beta + 1)! N_\alpha!\right] \end{aligned} \tag{10.11}$$

Eq. (10.11) is an extension of Eq. (9.89) (Lo and Palmer, 1995) for two different phases denoted by α and β of volume V_α and V_β, respectively. It is written in terms of real variables. The total volume of the system is conserved ($V_{\text{total}} = V_\alpha + V_\beta$). The subscripts i and j refer to the ordinary particles in boxes α and β, respectively, whereas the label f denotes a fractional particle that exists simultaneously in both boxes. We can distinguish between the coordinates $\mathbf{q}$ and momenta $\mathbf{p}$ of the ordinary particles ($\mathbf{q}_{\alpha i}$, $\mathbf{p}_{\alpha i}$, etc.) and fractional particles ($\mathbf{q}_{\alpha f}$, $\mathbf{p}_{\alpha f}$, etc.). The terms c_α and c_β are coupling parameters for the two boxes, which satisfy the relationship $c_\alpha + c_\beta = 1$. Consequently, when $c_\alpha = 1$, the fractional particle is coupled completely to box α, whereas when $c_\beta = 1$, the fractional particle is coupled completely to box β. The temperatures of the two boxes are controlled separately using two Nosé thermostats with the velocity scaling parameters s_1 and s_2. The variables $\mathbf{p}_{s1}$, $\mathbf{p}_{s2}$, $\mathbf{p}_{V\alpha}$, and $\mathbf{p}_{c\alpha}$ are the momenta conjugate to s_1, s_2, V_α, and c_α, respectively. The constants Q_1, Q_2, W, and X are the masses

associated with the velocity scaling variables s_1 and s_2, the volume V_α, and the coupling parameter c_α, respectively. The constants g_1 and g_2 denote the total number of degrees of freedom of each box, and the u terms are the potential energy functions of the two boxes.

The equations of motion derived from this Hamiltonian are:

$$\dot{\mathbf{p}} = \frac{\mathbf{F}+\mathbf{F}_f}{s_1^2} - \frac{2m\dot{s}_1}{s_1}\left[\mathbf{v} - \frac{1}{3}\left(\frac{\dot{V}_\alpha}{V_\alpha}\right)\mathbf{q}\right] - \frac{2m}{9}\left(\frac{\dot{V}_\alpha}{V_\alpha}\right)^2\mathbf{q} + \frac{m}{3}\left(\frac{\ddot{V}}{V_\alpha}\right)\mathbf{q} \quad (10.12)$$

$$\ddot{s}_1 = \frac{s_1}{Q_1}\left[\sum_{i=1}^{N_\alpha} m_i\left(\mathbf{v}_i - \frac{1}{3}\left(\frac{\dot{V}_\alpha}{V_\alpha}\right)\mathbf{q}_i\right)^2 + m_f s_1\left(\mathbf{v}_{\alpha f} - \frac{1}{3}\left(\frac{\dot{V}_\alpha}{V_\alpha}\right)\right)^2 - \frac{g_1}{s_1}kT\right] \quad (10.13)$$

$$\begin{aligned} W\ddot{V}_\alpha = &\sum_{i=1}^{N_\alpha}\frac{m_i s_1^2}{3V_\alpha}\left(\mathbf{v}_i - \frac{1}{3}\left(\frac{\dot{V}_\alpha}{V_\alpha}\right)\mathbf{q}_i\right)^2 + \frac{m_f s_1^2}{3V_\alpha}\left(\mathbf{v}_{f\alpha} - \frac{1}{3}\left(\frac{\dot{V}_\alpha}{V_\alpha}\right)\mathbf{q}_{f\alpha}\right)^2 \\ &- \sum_{j=1}^{N_\beta}\frac{m_j s_2^2}{3V_\beta}\left(\mathbf{v}_j - \frac{1}{3}\left(\frac{\dot{V}_\beta}{V_\beta}\right)\mathbf{q}_j\right)^2 - \frac{m_f s_2^2}{3V_\beta}\left(\mathbf{v}_{f\beta} - \frac{1}{3}\left(\frac{\dot{V}_\beta}{V_\beta}\right)\mathbf{q}_{f\beta}\right)^2 \\ &- \frac{\partial u_\alpha}{\partial V_\alpha} - \frac{\partial u_{\alpha f}}{\partial V_\alpha} + \frac{\partial u_{\beta f}}{\partial V_\beta} - c_\beta\frac{kT}{V_\alpha} + c_\alpha\frac{kT}{V_\beta} \end{aligned} \quad (10.14)$$

$$X\ddot{c}_\alpha = -\frac{\partial u_{\alpha f}}{\partial c_\alpha} + \frac{\partial u_{\beta f}}{\partial c_\beta} + kT\ln\frac{V_\alpha(N_\beta+1)}{V_\beta(N_\alpha+1)} \quad (10.15)$$

The equations of motion for the remaining coordinates and s_2 can be obtained by manipulating the indices of Eqs. (10.12) to (10.15).

An implementation of the Lo–Palmer method for a multiphase system is illustrated by Algorithm 10.2. The first step in the simulation process (Algorithm 10.2, Part 2) involves generating an equilibrated *NVT* ensemble for each phase (Algorithm 10.2, Part 2.1). The phases are initially characterized by an identical temperature but different number of particles, volumes, and pressure. The *NVT* ensembles can be generated by any suitable algorithm; however, it is convenient to use the Nosé–Hoover algorithm (Algorithm 10.2, Part 2.1.1) because, when there are no particle fluctuations, the Lo–Palmer algorithm reduces to the Nosé–Hoover algorithm. After the *NVT* ensemble equilibration period (Algorithm 10.2, Part 2.2), a particle is chosen randomly from one of the phases α (Algorithm 10.2, Part 2.2.1). The chosen particle represents the fractional particle f_α for phase α, $c_\alpha = 1$, and $\dot{c}_\alpha$ is obtained from a Boltzmann distribution. A particle is inserted into another phase β (Algorithm 10.2, Parts 2.2.2 and 2.2.3) with momenta obtained from a Boltzmann distribution. The inserted particle f_β represents the fractional particle in phase β with $c_\beta = 0$. Solutions to the equations of motion [Eqs. (10.12) to (10.15)] are obtained until c_α equals 0 or 1 (Algorithm 10.2, Part 2.2.4). A value of $c_\alpha = 1$ indicates that the fractional particle has become coupled to phase α. Consequently, it is transformed (Algorithm 10.2, 2.2.5) into a normal particle of phase α. If $c_\alpha = 0$, the fractional particle has become coupled to phase β and it is transformed to a normal particle of phase β. The other fractional particle f_β is removed and the

entire procedure is repeated until the simulation is completed. Averages are accumulated following an equilibration period (Algorithm 10.2, Part 2.2.6).

ALGORITHM 10.2 Gibbs ensemble MD using the Lo–Palmer algorithm.

```
Part 1     Initialisation:
           Assign values of W, Q, X, N1, N2, V1, V2, T, etc.
Part 1.1   Initialise all phases:
           loop i ← 1 ... nPhases
              Assign coordinates (ri) at time (t) = 0.
              Assign molecules initial velocities.
              Scale coordinates.
              gi ←3N
              si ←0
              siDot ← 0
           end loop

Part 2     Simulation Process:
Part 2.1   Create equilibrated NVT-ensembles for nPhases:
           loop
              loop i ← 1 ... nPhases
Part 2.1.1       Implement Nosé-Hoover algorithm for ensemblei
                 (e.g. Algorithm 9.9).
              end loop
              t ← t + Δt
           while (t ≤ tNVTEquilibration)
Part 2.2   Implement Lo-Palmer Algorithm:
           loop
              α ← ∈ integer[1, nPhases]      // Choose a phase randomly
Part 2.2.1    Choose a fractional particle randomly for the phase:
              fα ← ∈ integer[1, Nα]
              cα ← 1
              cαDot ← Boltzmann distribution
Part 2.2.2    Choose the insertion phase randomly:
              loop
                 β ← ∈ integer[1, nPhases]
              while (β ≠ α)
Part 2.2.3    Insert a fractional particle (fβ) into phase β.
              pfβ ←Boltzmann distribution
              cβ ← 0
Part 2.2.4    loop
                 Evaluate eqs. (10.12) to (10.15).
                 t ← t + Δt
              while ( cα < 1)
Part 2.2.5    if (cα = 1)
                 Re-assign fα as a normal particle of phase α.
              else
                 Re-assign fα as a normal particle of phase β.
              end if
              Remove fβ.
Part 2.2.6    if (t > tEquilibration)
                 Accumulate averages.
              end if
           while (t ≤ tMax)
```

10.3.2 Kotelyanskii–Hentschke method

Kotelyanskii and Hentschke (Kotelyanskii and Hentschke, 1995, 1996; Hentschke et al., 1996) have also used a fractional particle approach to derive a MD version of the Gibbs ensemble. To simulate a variable number of molecules in the simulation boxes, they introduced an additional degree of freedom ξ_i for each molecule. The value of ξ_i can vary between 0 and 1. When $\xi_i = 1$, molecule i is in box 1 whereas when $\xi_i = 0$, the molecule is in box 2. At intermediate values of ξ_i, the molecule is in a transition state and it is felt by both boxes. Consequently, the potential energy of the system is a function of both ξ_i and the intermolecular coordinates,

$$\begin{aligned} U(\mathbf{r}_{ij}, \xi_i, V_1, V_2) &= \sum_{i<j} \left[u(\mathbf{r}_{ij}, V_1)\xi_i\xi_j + u(\mathrm{r}_{ij}, V_2)(1-\xi_i)(1-\xi_j) \right] \\ &\quad + \sum_i g(\xi_i) \\ &= U_1 + U_2 + \sum_i g(\xi_i) \end{aligned} \tag{10.16}$$

where V_i and U_i are the volumes and energies, respectively, of the boxes indicated by the subscript. In Eq. (10.16), $g(\xi_i)$ is a potential function, which accounts for the presence of transition state molecules. The value of g must equal zero at either $\xi_i = 0$ or $\xi_i = 1$ but it is greater than zero at intermediate values of ξ_i, which correspond to any transition molecule. Kotelyanskii and Hentschke (1995) proposed the following g function,

$$g(\xi_i) = \begin{cases} w\left[\tanh(v\xi_i) + \tanh\left(v(1-\xi_i)\right) - 1\right] & 0 \le \xi_i \le 1 \\ \infty & otherwise \end{cases} \tag{10.17}$$

where w and v represent a height barrier and steepness parameter, respectively between the states corresponding to the real particles. The effect of the w and v parameters is to make the transition state unfavorable energetically.

The conditions for phase coexistence require that the temperature, pressure, and chemical potential of the coexisting phases are identical. For a one-component system, this is achieved by ensuring that every change of volume in one of the boxes is accompanied by an opposite but equal change in volume in the other box. Therefore the total volume of the system remains constant, whereas the volumes of the individual boxes are variable. The MD equation of motion for a one-component system becomes:

$$\mathbf{p} = m\dot{\mathbf{r}} \tag{10.18}$$

$$\dot{\mathbf{p}} = -\frac{\partial U}{\partial \mathbf{r}} = \eta \mathbf{p} \tag{10.19}$$

$$\dot{\eta} = \frac{1}{Q_T}\left[\sum_i \frac{\mathbf{p}_i^2}{m_i} - XkT\right] \tag{10.20}$$

$$\mathbf{p}_{\xi_i} = m_{\xi_i}\dot{\xi}_i \tag{10.21}$$

$$\dot{\mathbf{p}}_{\xi_i} = -\frac{\partial U}{\partial \xi_i} = -\sum_{i<j}\left[u(\mathbf{r}_{ij}, V_1)\xi_j - u(\mathbf{r}_{ij}, V_2)(1-\xi_j)\right] - \sum_i \frac{\partial}{\partial \xi_i} g(\xi_i) \tag{10.22}$$

$$\mathbf{p}_{V_1} = Q_p \dot{V}_1 \tag{10.23}$$

$$\dot{\mathbf{p}}_{V_1} = p_1^e - p_2^e \tag{10.24}$$

Together, Eqs. (10.22) to (10.24) determine the evolution of particle coordinates, particle transfers, and volume fluctuations. The variables $\mathbf{p}_i$ and $\mathbf{p}_\xi$ are the momenta conjugate to the normal Cartesian coordinates, and the transfer coordinate, respectively. Eqs. (10.18) and (10.19) represent the evolution of a system that is coupled to an external heat-bath of temperature T. In the case of only one simulation box they correspond to a *NVT* simulation controlled with the Nosé–Hoover thermostat (Chapter 9). The transfer of molecules between the boxes is controlled by Eqs. (10.21) and (10.22). Eqs. (10.23) and (10.24) are the equations of motion of the box of volume V_1, where $\mathbf{p}_{V_1}$ is a momentum variable conjugate to V_1 and Q_p is a parameter governing volume relaxation. The volume changes are controlled by the difference between the instantaneous values of the "external" pressures p_1^e and p_2^e in the two boxes.

The method can be extended easily to multicomponent fluids. For example, to simulate two-phase coexistence in a two-component system, Eqs. (10.23) and (10.24) are replaced by,

$$\dot{\mathbf{p}}_{V_k} = Q_p \dot{V}_k \tag{10.25}$$

$$\dot{\mathbf{p}}_{V_k} = p_k^e - p \tag{10.26}$$

where p is the predetermined value of pressure and $k = 1,2$.

10.3.3 Baranyai–Cummings method

Baranyai and Cummings (1996) have proposed a MD implementation of the Gibbs ensemble in which the conditions for mechanical and thermal equilibrium are obtained from the equations of motion via a Hoover-type deterministic feedback approach, whereas a MC move is used to determine the equivalence of the chemical potentials.

The system is considered to be constrained by the following conditions:

$$V = V_1 + V_2 \tag{10.27}$$

$$N = N_1 + N_2 \tag{10.28}$$

The equations of motion are derived to satisfy the above constraint while guiding the system to mechanical and thermal equilibrium. The starting points are the equations of motion in the Hoover-type form,

$$\dot{\mathbf{q}} = \frac{\mathbf{p}}{m} + \varepsilon(\mathbf{q} - \mathbf{R}_0) \tag{10.29}$$

$$\dot{\mathbf{p}} = \mathbf{F} - \varepsilon\mathbf{p} - \alpha\mathbf{p} \tag{10.30}$$

where $\mathbf{R}_0$ is the center of mass of the system, ε is a strain rate factor that couples the system to a barostat, and α is the feedback multiplier that couples the system to a heat-bath. These equations are supplemented by the Nosé−Hoover thermostat (Chapter 9),

$$\dot{\alpha} = \frac{1}{Q_T}\left(\frac{T(t)}{T_0} - 1\right) \tag{10.31}$$

where Q_T is a constant and T_0 is the reservoir temperature.

The change in volume can be obtained using ε.

$$\dot{V} = 3V\varepsilon \tag{10.32}$$

To obtain equations for $\mathrm{d}\varepsilon/\mathrm{d}t$ while maintaining the restriction of constant volume, Eq. (10.27) is differentiated twice with respect to time.

$$V_1\left(3\varepsilon_1^2 + \dot{\varepsilon}_1\right) + V_2\left(3\varepsilon_2^2 + \dot{\varepsilon}_2\right) = 0 \tag{10.33}$$

To relate $\mathrm{d}\varepsilon/\mathrm{d}t$ to the pressure of the two phases, we have,

$$\dot{\varepsilon}_1 = \frac{p_1 - p_2}{V_1 Q_V} - \frac{3}{2}\left(\varepsilon_1^2 + \varepsilon_2^2\frac{V_2}{V_1}\right) \tag{10.34}$$

$$\dot{\varepsilon}_2 = \frac{p_1 - p_2}{V_2 Q_V} - \frac{3}{2}\left(\varepsilon_2^2 + \varepsilon_1^2\frac{V_1}{V_2}\right) \tag{10.35}$$

where Q_V is a constant. If Eqs. (10.34) and (10.35) are substituted into Eq. (10.33), it becomes apparent that the relative volume changes are determined by the difference in the instantaneous pressures.

$$V_1\dot{\varepsilon}_1 - V_2\dot{\varepsilon}_2 = \frac{2(p_1 - p_2)}{Q_V} \tag{10.36}$$

Eqs. (10.29)–(10.35) govern the dynamics of both phases and bring the system to mechanical and thermal equilibrium. To obtain the equivalence in the chemical potentials, the MD run is halted after n time steps and a particle interchange is attempted in the same way as the MC Gibbs ensemble. The only difference is that we must also choose a random velocity from a Maxwell–Boltzmann distribution corresponding to the fixed internal temperature.

10.4 *NpT* + test particle

VLE can be studied using the NpT + test particle method (Möller and Fischer, 1990; Lotfi et al., 1992). The basic approach is to determine the phase coexistence properties at a given temperature from the intersection of the vapor and liquid branches in the chemical potential versus pressure diagram. The chemical potential is written in terms of a Taylor series expansion in pressure. The coefficients of the Taylor series are determined by MD simulations in the liquid and vapor phases. The technique has been developed further (Boda et al., 1995a,b; Kronome et al., 2000), and it can also be used in conjunction with the NVT (Szalai et al., 1995; Okumura and Yonezawa, 2001) and grand canonical ensembles (Boda et al., 1996, 1997). It has been applied to determine (Okumura and Yonezawa, 2000) the pure VLE behavior of different intermolecular potentials. Studies involving binary mixtures have also been reported (Vrabec et al., 1995; Vrabec and Gross, 2008).

10.5 Gibbs–Duhem integration

Kofke (1993a,b) proposed a Gibbs–Duhem simulation method, which combines elements of the Gibbs ensemble with integration of the Clapeyron equation. It is commonly referred to as Gibbs–Duhem integration (GDI). For a one-component system, the Gibbs–Duhem equation is

$$d(\beta\mu) = hd\beta + \beta v dp \tag{10.37}$$

where μ is the chemical potential, h is the molar enthalpy, v is the molar volume, p is the pressure, and $\beta = 1/kT$. The Clapeyron equation is obtained by writing Eq. (10.37) for two coexisting phases (I and II)

$$\left(\frac{dp}{d\beta}\right)_s = -\frac{\Delta h}{\beta \Delta v} \tag{10.38}$$

where $\Delta h = h_{\mathrm{I}} - h_{\mathrm{II}}$ and $\Delta v = v_{\mathrm{I}} - v_{\mathrm{II}}$ are the differences in the molar enthalpies and molar volumes, respectively, of the coexisting phases.

The subscript s indicates that the derivative is taken along the saturation line. Alternatively:

$$\left(\frac{d\ln p}{d\beta}\right)_s = -\frac{\Delta h}{\beta p \Delta v} = f(\beta, p) \tag{10.39}$$

Eqs. (10.38) and (10.39) represent first-order differential equations that regulate the change in pressure required to maintain two-phase coexistence. If the pressure at a given coexistence point is known, either Eqs. (10.38) or (10.39) can be integrated numerically to obtain the entire coexistence curve. In practice, it is advantageous to use Eq. (10.39) rather than Eq. (10.38) because the right-hand side of Eq. (10.38) varies only slowly.

Many different strategies are available for solving differential equations (Chapter 7). A simple and convenient method is the Adams predictor–corrector algorithm

$$\left.\begin{aligned} P:&\quad y_{i+1} = y_i + \frac{\Delta\beta(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3})}{24} \\ C:&\quad y_{i+1} = y_i + \frac{\Delta\beta(9f_{i+1} + 19f_i + 5f_{i-1} + f_{i-2})}{24} \end{aligned}\right\} \tag{10.40}$$

where y represents ln p, f is $f(\beta, P)$, $\Delta\beta$ is the difference in β between the current simulation and the initial condition, and P and C denote the predictor and corrector parts of the algorithm, respectively. The Adams algorithm is chosen because it can be used to solve accurately (Kofke, 1993b) the Clapeyron equation. However, the algorithm is not self-starting requiring four previous values of f. The initial estimate can be obtained from the trapezoid predictor–corrector.

$$\left.\begin{aligned} P:&\quad y_1 = y_0 + \Delta\beta f_0 \\ C:&\quad y_1 = y_0 + \frac{\Delta\beta(f_1 + f_0)}{2} \end{aligned}\right\} \tag{10.41}$$

The second point can be determined from the midpoint predictor–corrector.

$$\left.\begin{aligned} P:&\quad y_2 = y_0 + 2\Delta\beta f_1 \\ C:&\quad y_2 = y_0 + \frac{\Delta\beta(f_2 + 4f_1 + f_0)}{3} \end{aligned}\right\} \tag{10.42}$$

The third point can be obtained by combining the midpoint predictor with the Adams corrector.

$$\left.\begin{aligned} P:&\quad y_3 = y_1 + 2\Delta\beta f_2 \\ C:&\quad y_3 = y_2 + \frac{\Delta\beta(9f_3 + 19f_2 - 5f_1 + f_0)}{24} \end{aligned}\right\} \tag{10.43}$$

In the above equations, it should be stressed that the values of f refer to results of previous simulations at different values of β.

The starting point for the simulation is a known coexistence point. This can be obtained by performing a conventional GEMC simulation. These data are used to calculate f_0, and thereby obtain an estimate of the pressure at the new temperature. For example, applying the trapezoid predictor we obtain:

$$p = p_o \exp(f_0 \Delta\beta) \tag{10.44}$$

An *NpT* ensemble simulation is conducted in both phases for a short period using the earlier estimate of pressure. At the end of this period, the average energy and volume of the phases are calculated and used to update the pressure using the corrector formula. Using the trapezoid-corrector yields,

$$p = p_o \exp\left(\frac{\Delta\beta(f_0 + f_1)}{2}\right) \tag{10.45}$$

where f_1 is the estimate of f obtained from the simulation in progress. The simulation is continued using the new pressure and the process is repeated after another sampling interval. It should be noted that either MC or MD simulations can be used. A MD implementation of GDI has been reported by Lísal and Vacek (1996a,b, 1997).

Algorithm 10.3 illustrates a possible implementation of GDI for a one-component fluid at several different temperatures. The first stage (Algorithm 10.3, Part 1) involves a GEMC simulation to obtain the coexistence properties for an initial value of β. These GEMC data are used (Algorithm 10.3, Part 2) to estimate the pressure at a new value of β, which differs from the initial value by an amount $\Delta\beta$. This new value of β is used for the first GDI simulation (Algorithm 10.3, Part 3). Several GDI simulations are performed until the desired β*End* value is reached. At the start of each new simulation (Algorithm 10.3, Part 3), the average values of the properties required to calculate f are initialized to zero. Each simulation is conducted for nCycles (typically several thousand cycles are required). During each cycle, a *phase* is selected randomly (Algorithm 10.3, Part 3.1) and a random choice is made to attempt either a molecular displacement or volume change (Algorithm 10.3, Part 3.2). The decision to accept the move is made according to the standard MC acceptance criteria as illustrated in Algorithm 9.2 for a *NpT* ensemble. Following the MC move, if the update frequency has been reached (*update* is assigned typically a value of 10), the pressure is re-evaluated using an appropriate predictor–corrector method (Algorithm 10.3, Part 3.3). This involves evaluating f from the running averages of enthalpy and volume

differences. If the *update* interval has not been reached, the running averages are updated (Algorithm 10.3, Part 3.4). During each simulation cycle, a check is performed to determine whether the equilibration period (*nEquilibration*) has been reached. When the system has been equilibrated, the running averages are re-initialized to zero (Algorithm 10.3, Part 3.5) and contributions to the ensemble averages are accumulated (Algorithm 10.3, Part 3.6). At the end of the simulation (Algorithm 10.3, Part 3.7), the properties of the ensemble and f are stored and β is changed by a further amount of $\Delta\beta$.

The main advantage of GDI is that it avoids the difficult particle insertion step required by GEMC. Consequently, unlike the GEMC method, it can be used to predict equilibria involving a solid phase (Lísal and Vacek, 1997; Poison and Frenkel, 1998). It may also be useful in predicting the phase behavior of polymers and other large molecules for which it is difficult to obtain successful particle insertions. However, GDI relies heavily on having an accurately determined starting point that must be obtained by other simulation techniques such as GEMC. If the starting point is inaccurate, the rest of coexistence curve predicted by GDI will also be obtained inaccurately (Kofke, 1993b).

Mehta and Kofke (1994) extended the GDI method to binary mixtures. The Gibbs–Duhem equation for binary mixtures (Denbigh, 1961) is,

$$x_1 d\ln f_1 = h_r d\beta + Z d\ln p - x_2 d\ln f_2 \tag{10.46}$$

where x_i is the mole fraction of component i with fugacity f_i, h_r is the residual molar enthalpy (i.e., the enthalpy excluding the ideal gas contribution), and $Z = pv/RT$ is the compressibility factor. It can be shown (Mehta and Kofke, 1994) that the Clapeyron relationship arising from Eq. (10.46) is,

$$\left(\frac{\partial p}{\partial f_2}\right)_{\beta,s} = \frac{1}{x_1^v Z^l - x_1^l Z^v}\left(\frac{x_1^v}{\phi_2^l} - \frac{x_1^l}{\phi_2^v}\right) \tag{10.47}$$

where the superscripts l and v refer to the liquid and vapor phases, respectively, and $\phi_2 = f_2/(px_2)$ is the fugacity coefficient of component 2. Eq. (10.47) prescribes simulation in an osmotic ensemble in which the independent variables are temperature, pressure, and the fugacity of component 2.

ALGORITHM 10.3 MC implementation of GDI for an one-component fluid at several different temperatures.

```
Part 1    GEMC for initial coexistence point (Algorithm 10.1).

Part 2    Initialisation for GDI:
          f_o ← - ΔEnthalpy/(pressure × ΔVolume)
          Calculate p from trapezoidal-predictor.
          β ← β + Δβ                    //value for first Gibbs-Duhem simulation
          simulation ←1
          offset ← 0
Part 3    GDI/Markov Chain:
          loop                          //simulation for several temperatures
            accumΔEnthalpy ← 0                    //initialise accumulators
            accumΔVolume ← 0
            loop n ← 1 ... nCycles                //Markov process
Part 3.1      Select a phase:
              if(rand() ≤0.5)
                phase ←1
              else
                phase ←2
              end if
Part 3.2      Select MC move:
              if(rand() ≤0.5)
                Random displacement (Algorithm 9.2, Part 2.1).
              else
                Random volume fluctuation (Algorithm 9.2, Part 2.2).
              end if
Part 3.3      Update pressure:
              if(mod(nCycles,update) = 0)
                if (simulation = 1)
                  Update pressure using trapezoidal-PC.
                else if (simulation = 2)
                  Update pressure using midpoint-PC.
                else if (simulation = 3)
                  Update pressure using midpoint-P/Adams-C.
                else
                  Update pressure using Adams-PC.
                end if
Part 3.4      else                          //update accumulators
                accumΔEnthalpy ← accumΔEnthalpy + ΔEnthalpy
                accumΔVolume ← accumΔVolume + ΔVolume
```

```
                    averageΔEnthalpy ← accumΔEnthalpy/(n - offset)
                    averageΔVolume ← accumΔVolume/(n - offset)
                end if
Part 3.5        if (n = nEquilibration)              //re-initialise accumulators
                    accumΔEnthalpy ← 0
                    accumΔVolume ← 0
                    offset ← nEquilibration
                end if
Part 3.6        if (n ≥ nEquilibration)
                    Accumulate averages.
                end if
            end n loop
Part 3.7    Store pressure, enthalpy, etc. data for completed simulation.
            f_simulation ← - ΔEnthalpy/(pressure × ΔVolume)
            β← β+ Δβ
            simulation ← simulation + 1
        while (β does not equal βEnd)
```

It is also possible to obtain a Clapeyron relationship for the semi-grand ensemble. The following fugacity fraction (ξ_2) for component 2 is defined.

$$\xi_2 = \frac{f_2}{f_1 + f_2} \tag{10.48}$$

Substituting Eq. (10.48) into Eq. (10.46) yields:

$$d\ln(f_1 + f_2) = h_r d\beta + Zd\ln p - \frac{x_2 - \xi_2}{\xi_2(1-\xi_2)} d\xi_2 \tag{10.49}$$

Mehta and Kofke (1994) showed that Eq. (10.49) leads to the following Clapeyron relationship:

$$\left(\frac{\partial \ln p}{\partial \xi_2}\right)_{\beta,s} = \frac{x_2^l - x_2^v}{\xi_2(1-\xi_2)(Z^l - Z^v)} \tag{10.50}$$

Eq. (10.50) prescribes a simulation in a semi-grand ensemble (Briano and Glandt, 1984; Kofke and Glandt, 1988), which has the temperature, pressure, fugacity fraction, and the total number of molecules as the independent variables.

Mehta and Kofke (1994) have reported details of the MC implementation of GDI in both the osmotic and semi-grand ensembles. The method has been used to calculate three-phase coexistence lines (Agrawal et al., 1994). Lísal and Vacek (1997) reported a MD version of GDI for the semi-grand ensemble. For mixtures, some of the advantage of GDI over the Gibbs ensemble method is lost. Attempted particle deletions and insertions are required for the osmotic ensemble, whereas the implementation of the algorithm for the semigrand ensemble requires attempts to change particle identity.

The semigrand ensemble implementation has the advantage over the Gibbs ensemble that it avoids particle interchange attempts. In principle, the

MC acceptance criteria for particle displacement and volume change are identical to the criteria used in the Gibbs ensemble. In practice, it is useful to couple the volume moves in both phases to prevent unilateral condensation or vaporization of either phase. The acceptance criterion for volume changes must be modified (Mehta and Kofke, 1994) to account for the effects of volume coupling. The acceptance criterion for changing the identity of particles is relatively simple. If the identity of particle of component 2 is changed to component 1, the change in identity is accepted with a probability of:

$$\min\left[1, \exp\left\{-\beta\Delta U - \ln\left(\frac{\xi_2}{1-\xi_2}\right)\right\}\right] \tag{10.51}$$

If the identity of particle of component 1 is changed to component 2, the change in identity is accepted with a probability of:

$$\min\left[1, \exp\left\{-\beta\Delta U + \ln\left(\frac{\xi_2}{1-\xi_2}\right)\right\}\right] \tag{10.52}$$

Swapping particle identities is likely to be most effective when the particles are of similar size and shape. In contrast, the acceptance rate for attempts to change the identities of very dissimilar molecules is likely to be similar to the Gibbs ensemble deletion/insertion move. In particular, swapping the identity of a small molecule for a large molecular is likely to fail because of particle overlap with neighboring molecules.

Escobedo and de Pablo (1997) have reformulated the GDI method. In the reformulated approach, phase coexistence of a one-component system is governed by,

$$\left(\frac{d(\beta\mu)}{d\beta}\right)_s = \frac{\rho^{II}h^{II} - \rho^{I}h^{I}}{\rho^{II} - \rho^{I}} \tag{10.53}$$

where μ is the chemical potential and the superscripts I and II distinguish between the different coexisting phases. For a two-component system,

$$d\beta\mu = \frac{\left(1-x_i^I\right)h^{II} - \left(1-x_i^{II}\right)h^{I}}{\left(x_i^{II} - x_i^{I}\right)}d\beta + \frac{\left(1-x_i^I\right)v^{II} - \left(1-x_i^{II}\right)v^{I}}{\left(x_i^{II} - x_i^{I}\right)}\beta dp \tag{10.54}$$

where $i = 1, 2$. Eq. (10.54) allows the generation of either isothermal ($d\beta = 0$) or isobaric ($dp = 0$) diagrams. The simulations can be performed osmotically with T, p, N_1, and μ_2 being held constant. Alternatively, a grand canonical-type simulation can be conducted at constant μ_1, μ_2, V, and T. The $\mu_1\mu_2VT$ method is potentially useful for simulations of polymers because volume moves are avoided (for simulations involving large molecules, volume moves have a low probability of acceptance) and the simulations can be performed using an expanded grand canonical approach.

An improvement to GDI (van 't Hof et al., 2006a), particularly for mixtures, is to combine it with histogram reweighting (discussed below). This

provides greater computational efficiency without sacrificing accuracy. The issue of obtaining a reliable VLE starting point has also been addressed (van 't Hof et al., 2006b), which may be particularly useful for determining the properties of mixtures of chain molecules.

If a suitable starting point can be determined, a significant benefit of GDI is the ability to determine SLE, which is practically impossible to achieve using GEMC. The SLE starting point is often obtained from the evaluation of free energies but a combined equilibrium/nonequilibrium method (Ge et al., 2003, discussed below) provides a useful alternative in many cases. The SLE behavior of pure fluids represents the simplest case. GDI has been used to determine (Fartaria et al., 2002) the SLE and triple points on (n−6) LJ fluids (Ahmed and Sadus, 2009a), C_{60} (Fartaria et al., 2001, 2002), and the noble gases (Deiters and Sadus, 2021; Singh et al., 2021) using ab initio potentials (Chapter 4).

Application of GDI for the SLE of mixtures has been confined largely to binary systems. The SLE of different LJ atoms (Hitchcock and Hall, 1999; Lamm and Hall, 2001, 2002) and both LJ diatomic (Galbraith and Hall, 2007) and chain (Poison and Frenkel, 1998) molecules have been studied. SLE diagrams for binary metallic alloys have also been reported (Nam et al., 2007). The GDI method has also been extended (Attwood and Hall, 2008) to ternary systems, which allowed the determination of SLE of mixtures of three different LJ components.

10.6 Thermodynamic scaling

10.6.1 Temperature and density MC

Valleau (1991) developed a thermodynamic-scaling MC method, which can be used to calculate phase coexistence. The difference in the residual Helmholtz function (ΔA_r) at two different densities, characterized by box lengths L_1 and L_2 is given by,

$$\Delta A_r = -kT\ln\left\{\frac{Q^*(N,L_2,T)}{Q^*(N,L_1,T)}\right\} \tag{10.55}$$

where Q^* is the reduced configurational integral. The ratio of the reduced configurational integrals can be obtained from,

$$\frac{Q^*(N,L_2,T)}{Q^*(N,L_1,T)} = \frac{\left\langle \exp\left(-\beta U(\mathbf{r}^N,L_2)\right)/\pi(\mathbf{r}^N)\right\rangle_\pi}{\left\langle \exp(-\beta U(\mathbf{r}^N,L_1))/\pi(\mathbf{r}^N)\right\rangle_\pi} \tag{10.56}$$

where $<\ldots>_\pi$ indicates an average over the sampling distribution $\pi(\mathbf{r}^N)$. Consequently, the average of any thermodynamic property X at each density ρ_k of the canonical ensemble can be obtained from:

$$\langle X\rangle_{\rho_k} = \frac{\left\langle X(\mathbf{r}^N,L_k)\exp\left(-\beta U(\mathbf{r}^N,L_k)\right)/\pi(\mathbf{r}^N)\right\rangle_\pi}{\left\langle \exp(-\beta U(\mathbf{r}^N,L_k))/\pi(\mathbf{r}^N)\right\rangle_\pi} \tag{10.57}$$

This "density-scaling" procedure has been used successfully to calculate the properties of hard spheres and electrolytes described by the restricted primitive model (Valleau, 1991) and subcritical LJ fluids (Valleau, 1993). A "temperature-scaling" version of Eq. (10.57) has been used (Graham and Valleau, 1990) to determine the phase coexistence of electrolytes. The general principles been applied to obtain thermodynamic properties in the *NVT* (Valleau, 2005a) and *NpT* (Valleau, 2005b) ensembles.

Thermodynamic scaling can provide significantly more accurate results than the Gibbs ensemble. The disadvantages of the method are that some of the thermodynamic states sampled are not of any interest and additional composition variables are required to apply the method to the phase behavior of multicomponent mixtures. The thermodynamic-scaling MC algorithm has also been used to predict the phase diagram of a two-dimensional Coulomb gas (Orkoulas and Panagiotopoulos, 1996) and a square-well fluid (Brilliantov and Valleau, 1998a,b).

10.6.2 Thermodynamic-scaling Gibbs ensemble MC

Kiyohara et al. (1996) proposed a procedure for combining the thermodynamic-scaling procedure with the Gibbs ensemble. In the *NVT* Gibbs ensemble, the average of any thermodynamic property X of the system interacting with the Hamiltonian n at an inverse temperature $\beta_i = 1/kT_i$ can be calculated from,

$$\langle X\rangle_{\beta_i,n} = \frac{\left\langle X(\mathbf{r}^N;n)\frac{\Psi(\mathbf{r}^N;\beta_i,n)}{\pi(\mathbf{r}^N)}\right\rangle_{\pi(\mathbf{r}^N)}}{\left\langle \frac{\Psi(\mathbf{r}^N;\beta_i,n)}{\pi(\mathbf{r}^N)}\right\rangle_{\pi(\mathbf{r}^N)}} \tag{10.58}$$

where:

$$\psi(\mathbf{r}^N;\beta_i,n) = \frac{V_I^{N_I}V_{II}^{N_{II}}}{N_I!N_{II}!}\exp\left(-\beta_i U_n^I(\mathbf{r}^{N_I}) - \beta_i U_n^{II}(\mathbf{r}^{N_{II}})\right) \tag{10.59}$$

In Eq. (10.59), $U_n^I(\mathbf{r}^{N_I})$ *and* $U_n^{II}(\mathbf{r}^{N_{II}})$ are the potential energies in regions *I* and *II*, respectively.

The Gibbs ensemble generates a Markov chain by using particle displacements, volume changes, or particle interchanges to obtain a trial configuration ($\mathbf{r}^N{}_{trial}$) from the existing configuration ($\mathbf{r}^N{}_i$). The trial configuration is accepted with a probability:

$$p_{i\to trial} = \min\left(1, \frac{\pi(\mathbf{r}^N_{trial})}{\pi(\mathbf{r}^N_i)}\right) \tag{10.60}$$

To use thermodynamic scaling in conjunction with the Gibbs ensemble, configurations relevant to several Hamiltonians at different temperatures must be sampled in a single simulation. Therefore it is important that the distribution $\pi(\mathbf{r}^N)$ samples configurations relevant to all thermodynamic states with equal

frequency. A convenient distribution is a weighted linear combination of the Boltzmann factors of all the thermodynamic states of interest,

$$\pi(\mathbf{r}^N) = \sum_i \sum_n W_{\beta_i,n} \psi(\mathbf{r}^N; \beta_i, n) \tag{10.61}$$

where $W_{\beta_i,n}$ is the sampling weight for the thermodynamic state characterized by inverse temperature β_i, and Hamiltonian n. A possible distribution of weights between two thermodynamic states (β_i,n) and (β_j,m) can be obtained (Kiyohara et al., 1996) from:

$$\frac{W_{\beta_i,n}}{W_{\beta j,m}} = \frac{\left\langle \frac{\Psi(\mathbf{r}^N;\beta_i,n)}{\pi(\mathbf{r}^N)} \right\rangle_{\pi(\mathbf{r}^N)}}{\left\langle \frac{\Psi(\mathbf{r}^N;\beta_j,m)}{\pi(\mathbf{r}^N)} \right\rangle_{\pi(\mathbf{r}^N)}} \tag{10.62}$$

However, the proper weights are obtained from an iterative process because the estimate of the ratio of weights becomes more accurate as the weights used in the distribution $(\mathbf{r}^N)$ are improved.

The formulation of thermodynamic scaling for the NpT Gibbs ensemble is almost identical. The only modification required is that Eq. (10.59) is replaced with,

$$\psi(\mathbf{r}^N; \beta_i, n) = \frac{V_I^{N_I} V_{II}^{N_{II}}}{N_{I\alpha}! N_{II\beta}!} \exp\left(-\beta_i U_n^I(\mathbf{r}^{N_I}) - \beta_i U_n^{II}(\mathbf{r}^{N_{II}}) - \beta_i p V_I - \beta_i p V_{II}\right) \tag{10.63}$$

where p is the imposed value of pressure and $N_{I\alpha}$ is the number of molecules of species α in region I.

Kiyohara et al. (1996) used thermodynamic-scaling GEMC to calculate VLE for a $(n-6)$ LJ fluid where $n = 10$, 11, and 12. The different values of n for the potential correspond to different Hamiltonians for the fluid. The thermodynamic-scaling Gibbs ensemble method was found to be more efficient than the conventional Gibbs ensemble for the calculation of the properties the related potentials at many different thermodynamic states. This method has also been applied successfully to phase equilibria involving the exp-6 potential (Errington and Panagiotopoulos, 1998a). Valleau (2003) has addressed some issues concerning the effectiveness of thermodynamic-scaling GEMC.

10.7 Pseudo-ensemble methods

A general feature of the simulation of phase equilibria is the insertion and deletion or particles associated with either the NpT ensemble or the grand canonical ensemble. The particle insertion is the most difficult MC move to implement successfully, particularly at high density or if the insertion involves a large molecule such as a polymer. A low success rate for particle insertions or deletions slows dramatically the attainment of thermodynamic

equilibrium. To overcome this problem, Mehta and Kofke (1995) proposed a procedure in which particle insertion and deletions are replaced by volume fluctuations. The acceptance criterion for the volume changes (ΔV) that are designed to mimic particle insertion or deletion attempts is given by,

$$\min\left[1, \exp\left(\Delta N\left(\beta\mu_{GC} - \beta\mu(\rho) + Z(\rho)\right) + N\ln\frac{V+\Delta V}{V} - \beta\Delta U\right)\right] \quad (10.64)$$

where:

$$\Delta N = -\Delta V\frac{N}{V+\Delta V} \quad (10.65)$$

In Eq. (10.64), μ_{GC} is the predetermined grand canonical chemical potential and ΔU is the change in potential energy resulting from the change in volume. Eq. (10.64) requires the calculation of the chemical potential $\mu(\rho)$ and the compressibility factor $Z(\rho)$ at the instantaneous density of the system ρ. The chemical potential can be calculated from ghost particle insertions, and the compressibility can be obtained from the virial theorem. As an alternative to these traditional calculations, Mehta and Kofke (1995) accumulated instantaneous values of the chemical potential and pressure into density histograms. Subsequently, $\mu(\rho)$ and $Z(\rho)$ required in Eq. (10.64) were obtained from empirical correlations fitted to the histogram data. Incorporating Eq. (10.64) into a grand canonical ensemble effectively results in a pseudo-grand canonical simulation.

Phase coexistence between different phases can be simulated directly by replacing the particle/insertion move of the Gibbs ensemble algorithm with a pseudo-exchange move. Camp and Allen (1996) showed that the attempted pseudo-exchange of ΔN particles from phase II to phase I is accepted with a probability of,

$$\min\left[1, \exp\left(\begin{array}{l}\Delta N\left(Z^{I} + Z^{II} - \beta\left(\mu^{I} - \mu^{II}\right)\right) + N^{I}\ln\dfrac{V^{I}+\Delta V^{I}}{V^{I}} \\ + N^{II}\ln\dfrac{V^{II}+\Delta V^{II}}{V^{II}} - \beta\left(\Delta U^{I} + \Delta U^{II}\right)\end{array}\right)\right] \quad (10.66)$$

where the compressibility factors and chemical potentials are evaluated for the density corresponding to the volume change, and the volume increments are related to ΔN by:

$$\left.\begin{array}{l}\Delta V^{I} = -\dfrac{V^{I}\Delta N}{N^{I}+\Delta N} \\ \Delta V^{II} = +\dfrac{V^{II}\Delta N}{N^{II}+\Delta N}\end{array}\right\} \quad (10.67)$$

If the $\mu(\rho)$ and Z(ρ) functions are known accurately, the limiting distribution probed by the pseudo-grand canonical ensemble is that of the *NpT*

ensemble (Camp and Allen, 1996). Escobedo and de Pablo (1997) used this observation to formulate an alternative criterion for the volume move,

$$\min\left[1, \exp\left(-\beta p_{eq}\Delta V + N\ln\frac{V+\Delta V}{V} - \beta\Delta U\right)\right] \tag{10.68}$$

where p_{eq} is an initial estimate of the equilibrium pressure, which is refined progressively during the simulation. During the simulation, p_{eq} can be refined using:

$$p_{eq} = p(\rho) + \rho[\mu_{GC} - \mu(\rho)] \tag{10.69}$$

In the context of a pseudo-Gibbs ensemble simulation, Eq. (10.68) becomes (Escobedo and de Pablo, 1997):

$$\min\left[1, \exp\left(\begin{array}{l} -\beta p_{eq}\left(\Delta V^I + \Delta V^{II}\right) + N^I\ln\dfrac{V^I+\Delta V^I}{V^I} \\ + N^{II}\ln\dfrac{V^{II}+\Delta V^{II}}{V^{II}} - \beta\left(\Delta U^I + \Delta U^{II}\right)\end{array}\right)\right] \tag{10.70}$$

Escobedo and de Pablo (1997) have also applied this technique to an expanded grand canonical ensemble. It has been argued (Escobedo, 1998) that pseudo-ensembles provide connections between histogram reweighting, Gibbs ensemble, Gibbs–Duhem, and thermodynamic integration algorithms.

10.8 Histogram reweighting algorithms

If, in a MC simulation, σ represents a configuration, the Hamiltonian (H) can be obtained from,

$$-\beta H(\sigma) = Kl(\sigma) \tag{10.71}$$

where K is a dimensionless coupling constant associated with the operator $l(\sigma)$. A MC simulation of N steps at an inverse temperature of β_0 produces a histogram $H(S, \beta_0, N)$ for values S of the operator $l(\sigma)$. The normalized distribution function,

$$P_K(S) = \frac{H(S,\beta_0,N)\exp((K-K_0)S)}{\sum_{\{S\}} H(S,\beta_0,N)\exp((K-K_0)S)} \tag{10.72}$$

can be used to calculate the expectation value of any function of S in the neighborhood of β_0.

$$\langle f(S)\rangle_K = \sum_{\{s\}} f(S)P_K(S) \tag{10.73}$$

The theory of histograms was developed by Salsburg et al. (1959) and later implemented in a simulation by McDonald and Singer (1967). Valleau and Card (1972) introduced multistage sampling (MSS) to broaden the range of temperatures. MSS has been used (Bhanot et al., 1987a,b,c,d) to produce graphs of

thermodynamic quantities over a wide range of parameters. Several workers (Falcioni et al., 1982; Marinari, 1984; Ferrenberg and Swendsen, 1989) applied histograms to phase transitions. Ferrenberg and Swendsen (1988) introduced a multihistogram method that combined the advantages of multistage sampling with an improved method (Bennett, 1976) for estimating free energy differences.

Histogram methods, which are continually being made more efficient (Lidmar, 2012), have been employed very successfully for phase equilibria (Kiyohara et al., 1997, 1998; Conrad and de Pablo, 1998; Panagiotopoulos et al., 1998; Kamath and Potoff, 2006). The approach has been used for identifying the VLE of alkynes (Barhaghi et al., 2017) and the determination of the critical properties of binary mixtures (Lenart and Panagiotopoulos, 2006). The investigation of the SLE in protein solutions (Chang et al., 2004) has been facilitated by the histogram reweighting approach.

10.8.1 Grand canonical MC

In a MC simulation for a grand canonical ensemble, Kiyohara et al. (1997) showed that the average of any thermodynamic property X can be obtained from,

$$\langle X\rangle_{\mu VT} = \frac{\sum_N \sum_{E_N} X(N,U_N) f_{\mu VT}(N,U_N)}{\sum_N \sum_{E_N} f_{\mu VT}(N,U_N)} \tag{10.74}$$

where $\sum_N$ denotes the summation of N from 0 to infinity and $\sum_{E_N}$ is a summation over all energy levels for each N. In Eq. (10.74), $f_{\mu VT}(N,U_N)$ is a two-dimensional histogram, which is related to the components of the grand canonical partition function by,

$$f_{\mu VT}(N,U_N)\cdot C = \exp[\beta(N\mu - U_N)]\Omega(N,V,U_N) \tag{10.75}$$

where C is a simulation-specific simulation constant and Ω (N, V, U_N) is the number of microstates for number of particles N, volume V, and energy E_N. The grand canonical partition function for different states (μ', V, T') can be calculated from,

$$Z_{\mu' VT'} = \sum_N \sum_{U_N} W f_{\mu VT}(N,U_N) C \tag{10.76}$$

where use is made of the following reweighting term:

$$W = \exp(N(\beta'\mu' - \beta\mu) - (\beta' - \beta)U_N) \tag{10.77}$$

Therefore the average of any thermodynamic property X at a different temperature (T') and chemical potential (μ') can be calculated from:

$$\langle X\rangle_{\mu' VT'} = \frac{\sum_N \sum_{U_N} X(N,U_N) W f_{\mu VT}(N,U_N)}{\sum_N \sum_{U_N} W f_{\mu VT}(N,U_N)} \tag{10.78}$$

It is evident from Eqs. (10.74)–(10.78) that if the components of the grand canonical partition function ($f_{\mu VT}$ (N,U_N)) are known at one particular

state (μ, V, T), the grand canonical partition function at any other state (μ', V, T') can be obtained by reweighting each component using W.

The histogram reweighting technique can be applied (Kiyohara et al., 1997) to subcritical vapor−liquid coexistence by finding the chemical potential that gives the same pressure for both phases. In terms of the grand canonical partition function ($Z_{\mu VT}$), the pressure for a given thermodynamic state (μ, V, T) is calculated from:

$$p_{\mu VT} = \frac{\ln Z_{\mu VT}}{V} \tag{10.79}$$

If a histogram is made for the state (μ, V, T) from Eqs. (10.77) and (10.79), the pressure at (μ', V, T') is given by:

$$p_{\mu' VT'} = \frac{1}{V} \ln \sum_{N} \sum_{E_N} W f_{\mu VT}(N, U_N) C \tag{10.80}$$

The value of C is determined by calculating the pressure at (μ, V, T) from the virial theorem, that is,

$$p = \frac{\langle N \rangle_{\mu VT} kT}{V} + \left\langle \frac{dU}{dV} \right\rangle_{\mu VT} \tag{10.81}$$

and equating this value with the right-hand side of Eq. (10.80).

The grand canonical MC reweighting procedure has been applied to both polar fluids (Kiyohara et al., 1997, 1998; Errington et al., 1998; Errington and Panagiotopoulos, 1998b) and lattice polymers (Panagiotopoulos et al., 1998). The main advantage of the technique is that the coexistence properties can be obtained very accurately. In particular, vapor−liquid coexistence in the vicinity of the critical point can be obtained much more accurately than could be otherwise calculated using the Gibbs ensemble.

10.8.2 Isothermal−isobaric MC

Conrad and de Pablo (1998) have applied histogram reweighting techniques for the NpT ensemble. They compared results obtained from the NpT and grand canonical histogram reweighting. Both approaches yielded results of similar accuracy.

10.8.3 GEMC

Boulougouris et al. (2010) have combined histogram reweighting with GEMC. The combination of the two methods improves the accuracy of predications both close to and away from the pure fluid critical point.

10.9 Finite-size scaling

Generally, conventional molecular simulations that typically involve only a few hundred particles are not accurate in the region of the critical point. Near the critical point, the correlation length becomes very large and often exceeds the size of the simulation. The result is that the discontinuities and singularities that characterize critical phenomena in the thermodynamic limited are shifted. Such finite-size effects can affect adversely the accuracy of the estimated critical point.

Finite-size-scaling (FSS) methods have been developed (Wilding, 1995, 1997) to overcome these problems. FSS methods are typically conducted in the context of grand canonical MC simulations, although application to the canonical and isothermal–isobaric ensembles is possible (Wilding and Binder, 1996). The finite-size behavior in the Gibbs ensemble has been studied (Bruce, 1997). The advantage of FSS methods is that they allow the critical point to be estimated with a high degree of accuracy (Kim et al., 2003; Binder, 2010). For example, the critical temperature and density of a three-dimensional LJ fluid has been calculated (Wilding, 1995; Caillol, 1998; Rżysko and Borówko, 2011) to a high degree of accuracy. FSS has been used to investigate accurately the critical exponent of both square-well (de Miguel, 1997) and dipolar square-well fluids (Martín-Betancourt et al., 2009). In contrast to earlier studies, the FSS results indicate that the square-well fluid does not exhibit mean-field behavior. The method is often used in conjunction with algorithms such as histogram reweighting (Shi and Johnson, 2001) and applied to real fluids (Bianco and Franzese, 2014).

Applications of FSS to mixtures have also been reported. Systems studied include electrolytes (Hynninen et al., 2005), mixtures of colloids and polymers (Chou et al., 2006), rods and spheres (Jungblut et al., 2008), nonadditive hard spheres (Góźdź, 2003), different LJ atoms (Pérez-Pellitero et al., 2006), and polydisperse (Kristóf and Liszi, 2001) fluids. The application to binary mixtures means that LLE can be studied (Jagannathan and Yethiraj, 2005; Das, 2015; Góźdź, 2017; Ricci and Debenedetti, 2017).

10.10 Empirically based simulation algorithms

The previous examples of molecular simulation algorithms typically have well-articulated theoretical foundations. In contrast, a much less-studied route is to determine an algorithm by actually observing the effect of a phase change on either measurable or calculated thermodynamic proprieties. The synthetic method (Sadus, 2012) is an example of the former whereas the combined equilibrium/nonequilibrium MD approach (Ge et al., 2003) is an example of the latter.

10.10.1 Synthetic method

There is a wide variety of ways for experimentally observing phase equilibria. However, these experimental methods are totally disconnected from the simulation algorithms used for phase equilibria. The disconnect between experiment and simulation is understandable because it is generally impossible to directly translate actual laboratory practice into a usable algorithm. An exception is the "synthetic" experimental method pioneered by Lentz (1969) and Lentz and Franck (1969) for high-pressure phase equilibria. The method, which was specifically designed for mixtures, involves an autoclave of variable volume with known amounts of two components that are initially immiscible. For any given constant volume, the temperature is increased gradually while monitoring the increase in pressure until the meniscus separating the two phases disappears resulting in a single phase. The point of transition is a coexistence point on the three-dimensional pTx surface, where x represents the composition.

Of course, it is not possible to observe a meniscus in a molecular simulation, but the transition is also apparent as a "knick point" of the pT curve. An experimental example of such a knick point for the methane + methanol mixture (Francesconi, 1978) for one isochore is illustrated in Fig. 10.2. The volume of the autoclave can be adjusted and by repeating the procedure at different constant volumes. The locus of the resulting different knick points

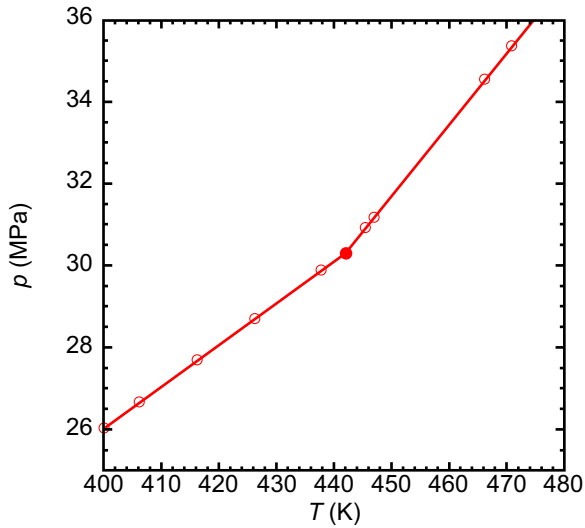


FIGURE 10.2 An experimentally determined (Francesconi, 1978) isochore (○) for methane + methanol ($x_2 = 0.4511$, $V = 81.94$ cm^3/mol) showing the location of a knick point (●) at $T = 442.15$ K and $p = 30.35$ MPa. For a given composition, an isopleth is the locus of knick points obtained from different isochores. The line through the points is only for guidance.

for a given composition is an isopleth. The high temperature of the isopleths is the critical curve for the mixture. It is apparent that the conditions experienced in the autoclave could be represented via an *NVT* ensemble, resulting in Algorithm 10.4.

ALGORITHM 10.4 Synthetic algorithm for fluid phase equlibria.

```
Part 1      Initialization:
            Specify a fixed composition (x2 = N2/N) involving N1 and N2
            Particles of components 1 and 2 respectively such that
            N = N1 + N2. Initialize other settings etc.

Part 2      Locate knick points at different isochores:
            loop
Part 2.1      Look for knick point:
              T = Tiintial
              knickPoint = false
              knickPointRef = false
              notFound = true
              first = true
              i = 1
              loop
Part 2.1.1      Perform NVT MC/MD simulation (Algorithm 9.1/9.7).
                Determine if knick point is crossed.
                if (knickPoint and first)
                  Tlower = T - ΔT
                  Tupper = T
                  notFound = false
                  first = false
                end if
Part 2.1.2      Refine knick point location:
                If (knickPoint)
                  T = Tlower
                  loop
                    Perform NVT MC/MD simulation (Algorithm 9.1/9.7).
                    Determine if knick point is crossed.
                    If (knickPointRef)
                      Tkpi = T
                      pkpi = p
                      i = i + 1
                    end if
                    T = T + ΔT/factor
                  while (knickPointRef = false or T <= Tupper)
                end if
              while (notFound and T <= Tmax)
              V = V + ΔV
            while (V ≤ Vmax)
```

Algorithm 10.4 commences (Part 1) in the usual way with the initialization of simulation settings. As it is normally applied to mixtures, care is required to specify the number of each component such that the sum does not exceed *N*.

The example is for a binary mixture but there is no theoretical limitation on the number of components. The main part of the algorithm (Part 2) is a loop that sweeps over a range of different V to determine knick points at different isochores. This involves performing *NVT* simulations (Part 2.1.1) for a range of T. This can be implemented in either MC (Algorithm 9.1) or MD (Algorithm 9.7). Alternatively an *NVE* implementation in either MC (Algorithm 9.4) or MD (Algorithm 9.5) can be used. In each case, this will involve a mixture implementation of the simulation algorithms. Following the *NVT* simulation, a check is made to determine whether the knick point has been located. In the original implementation (Sadus, 2012), this involved a visual inspection by producing a figure equivalent to Fig. 10.2. A more sophisticated alternative would be to calculate $\partial p/\partial T$, which should change abruptly either side of the knick point. This could be facilitated by the relationships for $\partial p/\partial T$ detailed in Chapter 2. It is unlikely that the initial determination would be sufficiently precise, which means repeating (Part 2.1.2) the process over a narrow range of T, over a desired increment size (factor).

Although intended for mixtures, the algorithm can be applied to pure systems (Sadus, 2012). The main disadvantages of the algorithm are (1) it requires many *NVT* simulations to locate a coexistence point; (2) the location of the knick point is hard to automate; and (3) the coexistence densities are not determined. Statistical uncertainties in the simulation may also mean that the knick point is not as clear as the experimental case (Fig. 10.2). The advantages are (1) implementation in either MC/MD using either *NVT/NVE*; (2) discrimination of very small differences in composition (Sadus, 2012) is possible; and (3) direct estimation of the critical locus as the high temperature envelope of the isopleths.

10.10.2 Combined NEMD/EMD method for SLE

As noted previously, a limitation of most simulation algorithms is that they are difficult to apply to SLE, which is usually addressed using specialized free energy techniques. GDI is very useful for SLE but it requires a suitable starting point that is commonly obtained from free energy calculations. In the course of an nonequilibrium MD (NEMD) investigation (Chapter 8) of the properties of LJ fluids at different strain rates ($\dot{\gamma}$) and constant isochores, Ge et al. (2003) observed that the onset of SLE was associated with an abrupt discontinuity in the pressure between $\dot{\gamma} = 0$ (equilibrium cases) and $\dot{\gamma} > 0$ (nonequilibrium case). This is illustrated in Fig. 10.3.

In Fig. 10.3, the isochores corresponding to a single liquid phase (black line) are almost linear irrespective of the value of $\dot{\gamma}$, whereas isochores (red lines) involving a solid−liquid transition display a sudden drop in pressure at zero strain rate. This feature can be used to combine equilibrium MD (EMD) and NEMD simulations to determine SLE as outlined in Algorithm 10.5

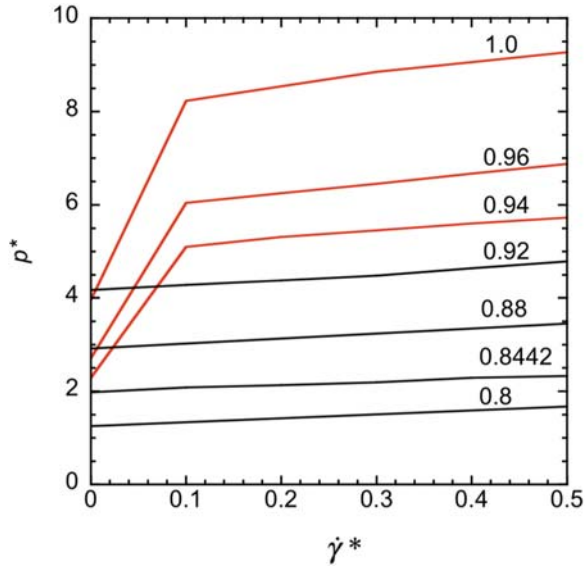


FIGURE 10.3 Reduced pressure versus reduced strain rate at different reduced isochores (as labeled) for a LJ fluid when $T^* = 1.0$ (Ge et al., 2003). The abrupt changes indicate the presence of a solid phase.

ALGORITHM 10.5 Combined EMD/NEMD algorithm for SLE.

Part 1 Determine the freezing density (ρ_f) and pressure (p_f) for a given T_f:
discontinuity = false
loop
Part 1.1 NVT MD simulation ($\dot{\gamma} = 0$).
NEMD NVT simulation (Algorithm 8.1) ($\dot{\gamma}_1 > 0$).
NEMD NVT simulation (Algorithm 8.1) ($\dot{\gamma}_2 \neq \dot{\gamma}_1$).
Part 1.2 Check for discontinuity in p:
if (*discontinuity* = true)
$\rho_f = \rho$
$p_f = p$
discontinuity = true
end if
increment ρ.
while (ρ <= ρMax and discontinuity = false)

Part 2 Repeat Part 1 as necessary to refine accuracy of ρ_f and p_f.

Part 3 Determine p-ρ curves:
Specify initial and final ρ values (ρMax).
loop
NVT simulation (Algorithm 9.9) to determine p.
increment ρ.
while (ρ <= ρMax)

Part 4 Determining melting density (ρ_m) using ρ_f and p_f
and data from Part 3.

The first part of Algorithm 10.5 (Part 1) is to determine a serious of isochores by performing *NVT* EMD and *NVT* NEMD simulations in parallel (Part 1.1). The number of nonzero values of $\dot{\gamma}$ is arbitrary and one well-chosen value could be sufficient. The discontinuity in p (Part 1.2), which signifies the values of the freezing pressure (p_f) and density (ρ_f), can either be obtained graphically by reproducing a plot such as Fig. 10.3 or it could be detected by monitoring for an abrupt change in $\partial p/\partial\dot{\gamma}$. In either case, this is likely to be very easy to detect, as it is apparent from Fig. 10.3 that isochores exclusively in the liquid phase will have only a slight gradient. Depending on the initial spacing between the isochores, the procedure may need to be repeated (Part 2) to obtain sufficiently precise values of p_f and ρ_f. The goal is to locate the first time this transition occurs. Part 3 involves determining the freezing and melting lines using conventional *NVT* simulations. It should be noted that this stage could be performed using either MC or MD simulations. However, it is convenient to use MD as an EMD algorithm can be always obtained by running an NEMD algorithm with $\dot{\gamma} = 0$. By definition, freezing and melting pressures are identical, which means that the melting density (ρ_m) can be obtained by extending a constant p_f tie line from the freezing curve to the melting curve data obtained in Part 3. Strictly, Part 3 only needs to generate the melting curve, which could be obtained by restricting the simulations to only high densities.

The main advantage of Algorithm 10.5 is that it can be used in all situations and it utilizes existing EMD and NEMD techniques. A disadvantage is that it requires several simulations per state point and coding NEMD is largely unfamiliar to researchers interested in phase equilibria, which by definition is an equilibrium property. Nonetheless, it can be used to obtain the entire SLE behavior and as such is particularly useful for difficult situations (Wang and Sadus, 2006b; Mausbach et al., 2009). Alternatively, it can provide a highly reliable starting point for extensive simulations using GDI (Ahmed and Sadus, 2009a,b; Ahmed and Sadus, 2010).

10.11 Accurate determination of SLE

As detailed earlier, VLE properties can be accurately determined via a variety of simulation algorithms. SLE properties can also be evaluated; however, the accuracy of previously discussed approaches is very limited. Deiters and Sadus (2022a,b) have addressed the issue of accuracy via the entropy correlation method (ECM). For a fixed displacement (α) in a MC simulation, the residual entropy (S^r) for dense phases is an inverse function value of a displacement parameter (λ_α), i.e.,

$$S^r \simeq -Nk\left(\frac{c_1}{\lambda_\alpha}\left(1 + c_3 e^{-c_2/\lambda_\alpha}\right) + c_0\right) \tag{10.82}$$

where N is the number of particles; c_0, c_1, c_2, and c_3 are coefficients that depend on the intermolecular potential; and c_0 also depends on α.

The change in the residual molar entropy (ΔS^r_m) along an isothermal compression from the molar density ρ_0 to ρ_1 depends of the enthalpy (H), compressibility factor (Z), and the universal gas constant (R) via Eq. (10.83),

$$\Delta S^r_m = S^r_m(\rho_1) - S^r_m(\rho_0) = \frac{H^r_m(\rho_1) - H^r_m(\rho_0)}{T} - R\left(\int_{\rho_0}^{\rho_1} \frac{Z-1}{\rho} d\rho + Z(\rho_1) - Z(\rho_0)\right) \tag{10.83}$$

which can be transformed to obtain:

$$\Delta S^r_m = S^r_m(\rho_1) - S^r_m(\rho_0) = -Rc_1\left(\frac{1 + c_3 e^{-c_2/\lambda_\alpha(\rho_1)}}{\lambda_\alpha(\rho_1)} - \frac{1 + c_3 e^{-c_2/\lambda_\alpha(\rho_1)}}{\lambda_\alpha(\rho_0)}\right) \tag{10.84}$$

The residual molar chemical potential ($\Delta\mu^r$) at a given T is obtained from the residual molar enthalpy ($H^{r,f}_m$ and $H^{r,s}_m$) of the fluid (f) and solid (s) phases and the corresponding molar volumes (V^f and V^s).

$$\Delta\mu^r = H^{r,f}_m - H^{r,s}_m + RT\left[c_1\left(\frac{1 + c_3 e^{-c_2/\lambda^f_\alpha}}{\lambda^f_\alpha} - \frac{1 + c_3 e^{-c_2/\lambda^s_\alpha}}{\lambda^s_\alpha}\right) - \ln\left(\frac{V^f}{V^s}\right) - 1\right] \tag{10.85}$$

Using these thermodynamic relationships, the ECM is a six-step process as illustrated in Algorithm 10.6.

ALGORITHM 10.6 ECM algorithm for SLE.

Part 1	Starting form either a gas or liquid state, perform *NpT* MC simulations (Algorithm 9.2) for a given T and a series of p
Part 2	Interpolate the compression factors obtained in Part 1 using a quadratic polynomial to facilitate integration in Part 3.
Part 3	Evaluate Eq. (10.83).
Part 4	Starting from an fcc lattice, tentatively determine the values of H^r, V and λ_α form a series of *NpT* MC simulations and the same values of p.
Part 5	Use Eq. (10.89) to obtain $\Delta\mu^r$ from the simulation results for the liquid and solid phases. A 2nd or 3rd order polynomial can be used to fit the differences a function of p. The equilibrium p is the nearest root of this polynomial.
Part 6	Using the p obtained in Part 5, the values of V for the liquid and solid phases are obtained via interpolation.

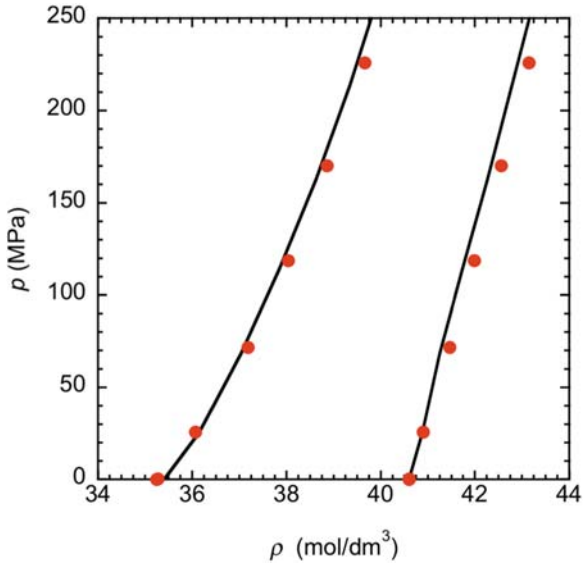


FIGURE 10.4 Comparison of ECM molecular simulation (red circles) with experimental data (Crawford, 1977) (solid lines) for the SLE properties of argon obtained for the SAAP + ATM + FH-1-2 potential (Chapter 4) (Deiters and Sadus, 2022b).

It is apparent from the description given in Algorithm 10.6 that the ECM process requires repeated *NpT* MC simulations and a considerable amount of postsimulation analysis. The reward for this effort is that SLE properties can be evaluated form low to very high pressures with an unprecedented degree of accuracy. For example, the triple point T of argon can be determined (Deiters and Sadus, 2022a) to an accuracy of within approximately 0.1 K of its experimental value using a combination of SAAP + ATM + FH-1-2 potentials (Chapter 4). This is a considerable improvement in accuracy of the previous discrepancy of 7 K reported by an alternative method (Pahl et al., 2008). This exceptional agreement with experiment data (Crawford, 1977) continues from the triple point to vary high values of p as is illustrated in Fig. 10.4.

10.12 Summary

MC algorithms are used overwhelmingly to simulate the phase behavior of fluids. The GEMC algorithm is the standard algorithm for phase equilibria simulation. However, more accurate MC techniques based on histogram rescaling are being developed and improved continually. The options for MD simulations are more limited. A promising route, which is arguably underutilized, is provided by the MD implementation of the Gibbs ensemble. Simulations regarding SLE are considerably more challenging than VLE.

The GDI method is very useful for SLE but requires initial starting point that can be difficult to obtain. Perhaps somewhat unexpectedly a combined EMD/NEMD has proved useful for SLE. It can be used both independently of other algorithms or provide reliable initial values for GDI simulations. However, SLE properties can be obtained with an unprecedented level of accuracy using the ECM simulations.

References

Adams, D.J., 1976. Calculating the low temperature vapour line by Monte Carlo. Mol. Phys. 32, 647–657.

Adams, D.J., 1979. Calculating the high-temperature vapour line by Monte Carlo. Mol. Phys. 37, 211–221.

Agrawal, R., Mehta, M., Kofke, D.A., 1994. Efficient evaluation of three-phase coexistence lines. Int. J. Thermophys. 15, 1073–1083.

Ahmed, A., Sadus, R.J., 2009a. Solid-liquid equilibria and triple points of n-6 Lennard-Jones fluids. J. Chem. Phys. 131, 174504. Erratum: *J. Chem. Phys.* 133, 229902 (2010).

Ahmed, A., Sadus, R.J., 2009b. Phase diagram of the Weeks-Chandler-Andersen potential from very low to high temperatures and pressures. Phys. Rev. E 80, 061101. Erratum: *Phys. Rev. E* 80, 029901 (2019).

Ahmed, A., Sadus, R.J., 2010. Effect of potential truncations and shifts on the solid-liquid phase coexistence of Lennard-Jones fluids. J. Chem. Phys. 133, 124515.

Alder, B.J., Wainwright, T.E., 1962. Phase transition in elastic disks. Phys. Rev. 127, 359–361.

Amar, J.G., 1989. Application of the Gibbs ensemble to the study of fluid-fluid phase equilibrium in a binary mixture of symmetric non-additive hard spheres. Mol. Phys. 67, 739–745.

Attwood, B.C., Hall, C.K., 2008. Solid-liquid phase behavior of ternary mixtures. AIChE J. 54, 1886–1894.

Baranyai, A., Cummings, P.T., 1996. On the molecular dynamics algorithm for Gibbs ensemble simulation. Mol. Sim. 17, 21–25.

Barhaghi, M.S., Mick, J.R., Potoff, J.J., 2017. Optimised Mie potentials for phase equilibria: application to alkynes. Mol. Phys. 115, 1378–1388.

Bennett, C.H., 1976. Efficient estimation of free energy differences from Monte Carlo data. J. Comput. Phys. 22, 245–268.

Bergermann, A., French, Redmer, R., 2021a. Gibbs-ensemble Monte Carlo simulation of H_2-H_2O. Phys. Chem. Chem. Phys. 23, 12637–12643.

Bergermann, A., French, M., Schöttler, M., Redmer, R., 2021b. Gibbs-ensemble Monte Carlo simulation of H_2-He mixtures. Phys. Rev, E. 103, 013307.

Bhanot, G., Black, S., Carter, P., Salvador, R., 1987a. A new method for the partition function of discrete systems with applications to the 3D Ising model. Phys. Lett. B 183, 331–336.

Bhanot, G., Bitar, K., Black, S., Carter, P., Salvador, R., 1987b. The partition function of Z(2) and Z(8) lattice gauge theory in four dimensions: a novel approach to simulations of lattice systems. Phys. Lett. B 187, 381–386.

Bhanot, G., Bitar, K., Salvador, R., 1987c. On solving four-dimensional SU(2) gauge theory by numerically finding its partition function. Phys. Lett. B 188, 246–252.

Bhanot, G., Salvador, R., Black, S., Carter, P., Toral, R., 1987d. Accurate estimate of ν for the three-dimensional Ising model from a numerical measurement of its partition function. Phys. Rev. Lett. 59, 803–806.

Bianco, V., Franzese, G., 2014. Critical behavior of a water monolayer under hydrophobic confinement. Sci. Rep. 4, 4440.

Binder, K., 2010. Computer simulations of critical phenomena and phase behavior of fluids. Mol. Phys. 108, 1797–1815.

Boda, D., Liszi, J., Szalai, I., 1995a. An extension of the NpT plus test particle method for the determination of the vapour-liquid equilibria of pure fluids. Chem. Phys. Lett. 235, 140–145.

Boda, D., Liszi, J., Szalai, I., 1995b. Dielectric constant of a Stockmayer fluid along the vapour-liquid coexistence curve. Mol. Phys. 85, 429–444.

Boda, D., Liszi, J., Szalai, I., 1996. A new simulation method for the determination of the vapour-liquid equilibria in the grand canonical ensemble. Chem. Phys. Lett. 256, 474–482.

Boda, D., Chan, K.-Y., Szalai, I., 1997. Determination of vapour-liquid equilibrium using cavity-biased grand canonical Monte Carlo method. Mol. Phys. 92, 1067–1072.

Boulougouris, G.C., Peristeras, L.D., Economou, I.G., Theodorou, D.N., 2010. J. Suprcrit. Fluids 55, 503–509.

Briano, J.G., Glandt, E.D., 1984. Statistical thermodynamics of polydisperse fluids. J. Chem. Phys. 80, 3336–3343.

Brilliantov, N.V., Valleau, J.P., 1998a. Thermodynamic scaling Monte Carlo study of the liquid-gas transition in the square-well fluid. J. Chem. Phys. 108, 1115–1122.

Brilliantov, N.V., Valleau, J.P., 1998b. Effective Hamiltonian analysis of fluid criticality and application to the square-well fluid. J. Chem. Phys. 108, 1123–1130.

Bruce, A.D., 1997. Finite-size critical behavior in the Gibbs ensemble. Phys. Rev. E 55, 2315–2320.

Bruce, A.D., Wilding, N.B., 2003. Computational strategies for mapping equilibrium phase diagrams. Adv. Chem. Phys. 127, 1–64.

Caillol, J.M., 1998. Critical-point of the Lennard-Jones fluid: a finite-size scaling study. J. Chem. Phys. 109, 4885–4893.

Camp, P.J., Allen, M.P., 1996. Phase coexistence in a pseudo Gibbs ensemble. Mol. Phys. 88, 1459–1469.

Chang, J., Lenhoff, A.M., Sandler, S.I., 2004. Determination of fluid-solid transitions in model protein solutions using the histogram rewighting method and expanded ensemble simulations. J. Chem. Phys. 120, 3003–3014.

Chapela, G.A., Martinez-Casas, S.E., Varea, C., 1987. Square well orthobaric densities via spinodal decomposition. J. Chem. Phys. 86, 5683–5688.

Chou, C.-Y., Vo, T.T.M., Panagiotopoulos, A.Z., Robert, M., 2006. Phys. A 369, 275–290.

Conrad, P.B., de Pablo, J.J., 1998. Comparison of histogram reweighting techniques for a flexible water model. Fluid Phase Equilib. 150, 51–61.

Crawford, R.K., 1977. Melting, vaporization and sublimation. In: Klein, M.L., Venables, J.A. (Eds.), Rare Gas Solids, Vol. 2. Academic Press, London, pp. 663–728.

Das, S.K., 2015. Atomistic simulations of liquid-liquid coexistence in confinement: comparison of thermodynamics and kinetics with bulk. Mol. Sim. 41, 382–401.

de Miguel, E., 1997. Critical behavior of the square-well fluid with $\lambda = 2$: a finite-size-scaling study. Phys. Rev. E 55, 1347–1354.

de Miguel, E., Rull, L.F., Chalam, M.K., Gubbins, K.E., 1990. Liquid-vapour coexistence of the Gay-Berne fluid by Gibbs-ensemble simulation. Mol. Phys. 71, 1223–1231.

de Pablo, J.J., Prausnitz, J.M., 1989. Phase equilibria for fluid mixtures from Monte Carlo simulation. Fluid Phase Equilib. 53, 177–189.

de Pablo, J.J., Yann, Q., Escobedo, F.A., 1999. Simulation of phase transitions in fluids. Annu. Rev. Phys. Chem. 50, 377–411.

Deiters, U.K., Sadus, R.J., 2019. Fully a priori prediction of the vapor-liquid equilibria of Ar, Kr, and Xe from ab initio two-body plus three-body interatomic potentials. J. Chem. Phys. 151, 034509.

Deiters, U.K., Sadus, R.J., 2021. Interatomic interactions responsible for the solid-liquid and vapor-liquid phase equilibria of neon. J. Phys. Chem. B 125, 8522–8531.

Deiters, U.K., Sadus, R.J., 2022a. First-principles determination of the solid-liquid-vapor triple point: the noble gases. Phys. Rev. E. 105, 054128.

Deiters, U.K., Sadus, R.J., 2022b. Accurate determination of solid-liquid equilibria by molecular simulation: behavior of Ne, Ar, Kr, and Xe from low to high pressures. J. Chem. Phys. 157, 204504.

Denbigh, K., 1961. Principles of Chemical Equilibrium. Cambridge University Press, Cambridge.

Dinpajooh, M., Bai, P., Allan, D.A., Siepmann, J.I., 2015. Accurate and precise determination of critical properties from Gibbs ensemble Monte Carlo simulations. J. Chem. Phys. 143, 114113.

Eike, D.M., Brennecke, J.F., Maginn, E.J., 2005. Toward a robust and general molecular simulation method for computing solid-liquid coexistence. J. Chem. Phys. 122, 014155.

Errington, J.R., Panagiotopoulos, A.Z., 1998a. Phase equilibria of the modified Buckingham exponential-6 potential from Hamiltonian scaling grand canonical Monte Carlo. J. Chem. Phys. 109, 1093–1100.

Errington, J.R., Panagiotopoulos, A.Z., 1998b. A fixed point charge model for water optimized to vapor-liquid coexistence properties. J. Phys. Chem. B 102, 7470–7475.

Errington, J.R., Kiyohara, K., Gubbins, K.E., Panagiotopoulos, A.Z., 1998. Monte Carlo simulation of high-pressure phase equilibria in aqueous systems. Fluid Phase Equilib. 150, 33–40.

Escobedo, F.A., 1998. Novel pseudoensembles for simulation of multicomponent phase equilibria. J. Chem. Phys. 108, 8761–8772.

Escobedo, F.A., de Pablo, J.J., 1997. Pseudo-ensemble simulations and Gibbs-Duhem integrations for polymers. J. Chem. Phys. 106, 2911–2923.

Eslami, H., Müller-Plathe, F., 2007. Molecular dynamics simulation in grand canonical ensemble. J. Comput. Chem. 28, 1763–1773.

Falcioni, M., Marinari, E., Paciello, M.L., Parisi, G., Taglienti, B., 1982. Complex zeros in the partition function of the four-dimensional SU(2) lattice gauge model. Phys. Lett. B 108, 331–332.

Fantoni, R., Moroni, S., 2014. Quantum Gibbs ensemble Monte Carlo. J. Chem. Phys. 141, 114110.

Fartaria, R.P.S., Silva Fernades, F.M.S., Freitas, F.F.M., Rodrigues, P.C.R., 2001. Phase behavior of C_{60} by computer simulation using ab-initio interaction potential. Int. J. Quant. Chem. 84, 375–387.

Fartaria, R.P.S., Silva Fernades, F.M.S., Freitas, F.F.M., 2002. Monte Carlo simulation of the phase diagram of C-60 using two interaction potentials. Enthalpies of sublimation. J. Chem. Phys. B 106, 10227–10232.

Ferrenberg, A.M., Swendsen, R.H., 1988. New Monte Carlo technique for studying phase transitions. Phys. Rev. Lett. 61, 2635–2638.

Ferrenberg, A.M., Swendsen, R.H., 1989. Optimized Monte Carlo data analysis. Phys. Rev. Lett. 63, 1195–1198.

Francesconi, A.Z., Kritische Kurve, Phasengleichewichte und PVT-Daten im System Methanol-Methan bis 3 bar und 240°C, Doctoral thesis (University of Karlsruhe, Germany, 1978).

Frenkel, D., 1986. In: Ciccotti, G., Hoover, W.G. (Eds.), Molecular Dynamics Simulation of Statistical Mechanical Systems. North-Holland, Amsterdam.

Frenkel, D., Smit, B., 2023. Understanding Molecular Simulation. From Algorithms to Applications, third ed. Academic Press, San Diego.

Galbraith, A.L., Hall, C.K., 2007. Solid-liquid phase equilibria for mixtures containing diatomic Lennard-Jones molecules. Fluid Phase Equilib. 262, 1–13.

Ge, J., Wu, G.-W., Todd, B.D., Sadus, R.J., 2003. Equilibrium and nonequilibrium molecular dynamics methods for determining solid-liquid phase coexistence at equilibrium. J. Chem. Phys. 119, 11017–11023.

Góźdź, W.T., 2003. Critical-point and coexistence curve properties of a symmetric mixture of nonadditive hard spheres: a finite size scaling study. J. Chem. Phys. 119, 3309–3315.

Góźdź, W.T., 2017. Investigation of fluid-fluid and solid-solid phase separation of symmetric nonadditive hard spheres at high density. Langmuir 23, 11727–11732.

Graham, I.S., Valleau, J.P., 1990. A Monte Carlo study of the coexistence region of the restricted primitive model. J. Phys. Chem. 94, 7894–7898.

Gromov, D.G., de Pablo, J.J., Luna-Bárcenas, G., Sanchez, I.C., Johnston, K.P., 1998. Simulation of phase equilibria for polymer-supercritical mixtures. J. Chem. Phys. 108, 4647–4653.

Gubbins, K.E., 1989. The role of computer simulation in studying fluid phase equilibria. Mol. Sim. 2, 223–252.

Gubbins, K.E., 1994. In: Sandler, S.I. (Ed.), Models for Thermodynamic and Phase Equilibria Calculations. Marcel Dekker, New York.

Gubbins, K.E., Shing, K.S., Streett, W.B., 1983. Fluid phase equilibria: computer simulation, and theory. J. Phys. Chem. 87, 4573–4585.

Hansen, J.P., Verlet, L., 1969. Phase transitions of the Lennard-Jones system. Phys. Rev. 184, 151–161.

Hentschke, R., Bast, T., Aydt, E., Kotelyanskii, M., 1996. Gibbs-ensemble molecular dynamics: a new method for simulations involving particle exchange. J. Mol. Model. 2, 319–326.

Hitchcock, M.R., Hall, C.K., 1999. Solid-liquid phase equilibrium for binary Lennard-Jones mixtures. J. Chem. Phys. 110, 11433–11444.

Hoover, W.G., Ree, F.H., 1967. Use of computer experiments to locate the melting transition and calculate the entropy in the solid phase. J. Chem. Phys. 47, 4873–4878.

Hynninen, A.-P., Dijkstra, M., Panagiotopoulos, A.Z., 2005. Critical point of electrolyte mixtures. J. Chem. Phys. 123, 084903.

Jagannathan, K., Yethiraj, A., 2005. Dynamics of fluids near the consolute critical point: a molecular-dynamics study of the Widom-Rowlinson mixture. J. Chem. Phys. 122, 244506.

Johnson, J.K., Panagiotopoulos, A.Z., Gubbins, K.E., 1994. Reactive canonical Monte Carlo: a new simulation technique for reacting or associating fluids. Mol. Phys. 81, 717–733.

Jungblut, S., Binder, K., Schilling, T., 2008. Isotropic-isotopic phase separation in mixtures of rods and spheres: some aspects of Monte Carlo simulation in the grand canonical ensemble. Comput. Phys. Commun. 179, 13–16.

Kamath, G., Potoff, J.J., 2006. Monte Carlo predictions for the phase behavior of the H_2S + n-alkane, H_2S + CO_2, CO_2 + CH_4 and H_2S + CO_2 + CH_4 mixtures. Fluid Phase Equilib. 246, 71–78.

Kim, Y.C., Fisher, M.E., Luijten, E., 2003. Precise simulation of near-critical fluid coexistence. Phys. Rev. Lett. 91, 065701.

Kiyohara, K., Spyriouni, T., Gubbins, K.E., Panagiotopoulos, A.Z., 1996. Thermodynamic scaling Gibbs ensemble Monte Carlo: a new method for determination of phase coexistence properties of fluids. Mol. Phys. 89, 965–974.

Kiyohara, K., Gubbins, K.E., Panagiotopoulos, A.Z., 1997. Phase coexistence properties of polarizable Stockmayer fluids. J. Chem. Phys. 106, 3338–3347.

Kiyohara, K., Gubbins, K.E., Panagiotopoulos, A.Z., 1998. Phase coexistence properties of polarizable water models. Mol. Phys. 94, 803–808.

Kofke, D.A., 1993a. Gibbs-Duhem integration: a new method for direct evaluation of phase coexistence by molecular simulation. Mol. Phys. 78, 1331–1336.

Kofke, D.A., 1993b. Direct evaluation of phase coexistence by molecular simulation via integration along the saturation line. J. Chem. Phys. 98, 4149–4162.

Kofke, D.A., Glandt, E.D., 1988. Monte Carlo simulation of multicomponent equilibria in a semigrand canonical ensemble. Mol. Phys. 64, 1105-1031.

Kotelyanskii, M.J., Hentschke, R., 1995. Gibbs-ensemble molecular dynamics: liquid-gas equilibrium in a Lennard-Jones system. Phys. Rev. E 51, 5116–5119.

Kotelyanskii, M.J., Hentschke, R., 1996. Gibbs-ensemble molecular dynamics: liquid-gas equilibria for Lennard-Jones spheres and n-hexane. Mol. Sim. 17, 95–112.

Kowalczyk, P., Gauden, P.A., Terzyk, A.P., Pantatosaki, E., Papadopoulos, G.K., 2013. J. Chem. Theory Comput. 9, 2922–2929.

Kristóf, T., Liszi, J., 1998. Alternative Gibbs ensemble Monte Carlo implementations: application in mixtures. Mol. Phys. 94, 519–525.

Kristóf, T., Liszi, J., 2001. Phase coexistence and critical point determination in polydisperse fluids. Mol. Phys. 99, 167–173.

Kronome, G., Szalai, I., Wendland, M., Fischer, J., 2000. Extension of the NpT + test particle method for the calculation of phase equilibria of nitrogen + ethane. J. Mol. Liq. 85, 237–247.

Kumar, S.K., 1992. A modified real particle method for the calculation of the chemical potentials of molecular systems. J. Chem. Phys. 97, 3550–3556.

Kuznetsova, T., Kvamme, B., 2005. Grand canonical molecular dynamic simulations for polar systems. Chem. Eng. Commun. 192, 189–197.

Lamm, M.H., Hall, C.K., 2001. Molecular simulation of complete phase diagrams for binary mixtures. AIChE J. 47, 1664–1675.

Lamm, M.H., Hall, C.K., 2002. Equilibria between solid, liquid, and vapor phases in binary Lennard-Jones mixtures. Fluid Phase Equilib. 194–197, 197–206.

Lenart, P.J., Panagiotopoulos, A.Z., 2006. Tracing the critical loci of binary fluid mixtures using molecular simulation. J. Phys. Chem. B 110, 17200–17206.

Lentz, H., 1969. A method of studying the behavior of fluid phases at high pressures and temperatures. Rev. Sci. Intstrum. 40, 371–372.

Lentz, H., Franck, E.U., 1969. Das System Water-Argon bei hohen Drucken und Temperaturen. Ber. Busnen-Ges. Physik. Chem. 73, 28–35.

Lidmar, J., 2012. Improving the efficiency of extended ensemble simulations: the accelerated weight histogram method. Phys. Rev. E 85, 056708.

Lísal, M., Vacek, V., 1996a. Direct evaluation of vapour-liquid equilibria by molecular dynamics using Gibbs-Duhem integration. Mol. Sim. 17, 27–39.

Lísal, M., Vacek, V., 1996b. Direct evaluation of vapour-liquid equilibria of mixtures by molecular dynamics using Gibbs-Duhem integration. Mol. Sim. 18, 75–99.

Lísal, M., Vacek, V., 1997. Direct evaluation of solid-liquid equilibria by molecular dynamics using Gibbs-Duhem integration. Mol. Sim. 19, 43–61.

Lo, C., Palmer, B., 1995. Alternative Hamiltonian for molecular dynamics simulations in the grand canonical ensemble. J. Chem. Phys. 102, 925–931.

Losey, J., Sadus, R.J., 2019. Thermodynamic properties and anomalous behavior of double-Gaussian core model potential fluids. Phys. Rev. E 100, 012112.

Lotfi, A., Vrabec, J., Fischer, J., 1992. Vapour liquid equilibria of the Lennard-Jones fluid from the NpT + test particle method. Mol. Phys. 76, 1319–1333.

Marcelli, G., Sadus, R.J., 1999. Molecular simulation of the phase behavior of noble gases using accurate two-body and three-body intermolecular potentials. J. Chem. Phys. 111, 1533–1540.

Marcelli, G., Sadus, R.J., 2001. Three-body interactions and the phase equilibria of mixtures. High. Temp.-High-Press 33, 111–118.

Marinari, E., 1984. Complex zeroes of the d = 3 Ising model: finite-size scaling and critical amplitudes. Nucl. Phys. B 235, 123–134.

Martín-Betancourt, M., Romero-Enrique, J.M., Rull, L.F., 2009. Finite-size scaling study of the liquid-vapour critical point of dipolar square-well fluids. Mol. Phys. 107, 563–570.

Mausbach, P., Ahmed, A., Sadus, R.J., 2009. J. Chem. Phys. 131, 184507. Erratum: *J. Chem. Phys.* 132, 019901 (2010).

McDonald, I.R., Singer, K., 1967. Calculation of thermodynamic properties of liquid argon from Lennard-Jones parameters by a Monte Carlo method. Discuss. Faraday Soc. 43, 40–49.

Mehta, M., Kofke, D.A., 1994. Coexistence diagrams of mixtures by molecular simulation. Chem. Eng. Sci. 49, 2633–2645.

Mehta, M., Kofke, D.A., 1995. Molecular simulation in a pseudo grand canonical ensemble. Mol. Phys. 86, 139–147.

Mick, J.R., Barhahi, M.S., Jackman, B., Rushaidat, K., Schwiebert, L., Potoff, J.J., 2015. Optimized Mie potentials for phase equlibria: Apllication to noble gases and their mixtures with n-alkanes. J. Chem. Phys. 143, 114504.

Möller, D., Fischer, J., 1990. Vapour liquid equilibrium of a pure fluid from test particle method in combination with NpT molecular dynamics simulations. Mol. Phys. 69, 463–473.

Nam, H.-S., Mendelev, M.I., Srolocitz, D.J., 2007. Solid-liquid phase diagrams for binary metallic alloys: adjustable interatomic potentials. Phys. Rev. B 75, 014204.

Okumura, H., Yonezawa, F., 2000. Liquid-vapor coexistence curves of several interatomic model potentials. J. Chem. Phys. 113, 9162–9168.

Okumura, H., Yonezawa, F., 2001. Method for liquid-vapor coexistence curves by test-particle insertions in the canonical ensemble. J. Non-Cryst. Solids 293–295, 715–718.

Orkoulas, G., Panagiotopoulos, A.Z., 1996. Phase diagram of the two-dimensional Coulomb gas: a thermodynamic scaling Monte Carlo study. J. Chem. Phys. 104, 7205–7209.

Pahl, E., Calvo, F., Koči, L., Schwerdtfeger, P., 2008. Accurate melting temperatures for neon and argon from ab inito Monte Carlo Simulations. Angew. Chem. Int. Ed. 47, 8207–8210 (2008).

Palmer, B.J., Lo, C., 1994. Molecular dynamics implementation of the Gibbs ensemble calculation. J. Chem. Phys. 101, 10899–10907.

Panagiotopoulos, A.Z., 1987. Direct determination of phase coexistence properties of fluids by Monte Carlo simulation in a new ensemble. Mol. Phys. 61, 813–826 (1987).

Panagiotopoulos, A.Z., 1992a. Direct determination of fluid phase equilibria by simulation in the Gibbs ensemble: a review. Mol. Sim. 9, 1–23.

Panagiotopoulos, A.Z., 1992b. Molecular simulation of fluid-phase equilibria: simple, polymeric and ionic fluids. Fluid Phase Equilib. 76, 97–112.

Panagiotopoulos, A.Z., 2000. Monte Carlo methods for phase equilibria of fluids. J. Phys.: Condens. Matt. 12, R25–R52.

Panagiotopoulos, A.Z., Quirke, N., Stapleton, M., Tildesley, D.J., 1988. Phase equilibria by simulation in the Gibbs ensemble: alternative derivation, generalization and application to mixture and membrane equilibria. Mol. Phys. 63, 527–545.

Panagiotopoulos, A.Z., Wong, V., Floriano, M.A., 1998. Phase equilibria of lattice polymers form histogram reweighting Monte Carlo simulations. Macromolecules 31, 912–918.

Pérez-Pellitero, J., Ungerer, P., Orkoulas, G., Mackie, A.D., 2006. Critical point estimation of the Lennard-Jones pure fluid and binary mixtures. J. Chem. Phys. 125, 054515.

Plačkov, Đ., Sadus, R.J., 1997. Molecular simulation of intermolecular attraction and repulsion in coexisting liquid and vapour phases. Fluid Phase Equilib. 134, 77–95.

Poison, J.M., Frenkel, D., 1998. Calculation of solid-fluid phase equilibria for systems of chain molecules. J. Chem. Phys. 109, 318–328.

Potoff, J.J., Kamath, G., 2014. Mie potentials for phase equilibria: applications to alkenes. J. Chem. Eng. Data 59, 3144–3150.

Potoff, J.J., Panagiotopoulos, A.Z., 1998. Critical point and phase behavior of the pure fluid and a Lennard-Jones mixture. J. Chem. Phys. 109, 10914–10920.

Raabe, G., Sadus, R.J., 2003. Molecular simulation of the vapor-liquid coexistence of mercury. J. Chem. Phys. 119, 6691–6697.

Raabe, G., Sadus, R.J., 2007. Influence of bond flexibility on the vapor-liquid phase equilibria of water. J. Chem. Phys. 126, 044701.

Rahbari, A., Hens, R., Ramdin, M., Moultos, O.A., Dubbeldam, D., Vlugt, T.J.H., 2021. Recent advances in the continuous fractional component Monte Carlo methodology. Mol. Sim. 47, 804–823.

Ricci, F., Debenedetti, P.G., 2017. A free energy study of the liquid-liquid phase transition of the Jagla two-scale potential. J. Chem. Sci. 129, 801–823.

Rowley, R.L., Shupe, T.D., Schuck, M.W., 1994. A direct method for determination of chemical potential with molecular dynamics simulation. 1. Pure components. Mol. Phys. 82, 841–855.

Rowley, R.L., Schuck, M.W., Perry, J.C., 1995. A direct method for determination of chemical potential with molecular dynamics simulations. Part 2. Mixtures. Mol. Phys. 86, 125–137.

Rowlinson, J.S., Widom, B., 1982. Molecular Theory of Capillarity. Clarendon Press, Oxford.

Rudisill, E.N., Cummings, P.T., 1989. Gibbs ensemble simulation of phase equilibrium in the hard core two-Yukawa fluid for the Lennard-Jones fluid. Mol. Phys. 68, 629–635.

Rżysko, W., Borówko, M., 2011. A critical behavior of the Lennard-Jones dimeric fluid in two-dimensions. A Monte Carlo study. Surf. Sci. 605, 1219–1223.

Sadus, R.J., 1996a. Molecular simulation of the vapour-liquid equilibria of pure fluids and binary mixtures containing dipolar components: the effect of Keesom interactions. Mol. Phys. 87, 979–990.

Sadus, R.J., 1996b. Molecular simulation of the liquid-liquid equilibria of binary mixtures containing dipolar and non-polar components interacting via the Keesom potential. Mol. Phys. 89, 1187–1194.

Sadus, R.J., 1997. Molecular simulation of Henry's constant at vapor-liquid and liquid-liquid phase boundaries. J. Phys. Chem. B. 101, 3834–3838.

Sadus, R.J., 1998a. Exact calculation of the effect of three-body Axilrod-Teller interactions on vapour-liquid phase coexistence. Fluid Phase Equilib. 144, 351–359.

Sadus, R.J., 1998b. Effect of three-body interactions between dissimilar molecules on the phase behaviour of binary mixtures: the transition from vapor-liquid equilibria to type III behaviour. Ind. Eng. Chem. Res. 37, 2977–2982.

Sadus, R.J., 1999. Molecular simulation of the phase behaviour of ternary fluid mixtures: the effect of a third component on vapour-liquid and liquid-liquid coexistence. Fluid Phase Equilib. 157, 169–180.

Sadus, R.J., 2012. Molecular simulation of the phase behavior of fluids and fluid mixtures using the synthetic method. J. Chem. Phys. 137, 054507.

Sadus, R.J., 2020a. Vapor-liquid equilibria and cohesive r^{-4} interactions. J. Chem. Phys. 153, 204504.

Sadus, R.J., 2020b. Effect of the range of particle cohesion on the phase behavior on thermodynamic properties of fluids. J. Chem. Phys. 153, 244502.

Sadus, R.J., 2020c. Combining intermolecular potentials for the prediction of fluid properties: two-body and three-body interactions. J. Chem. Phys. 153, 214509.

Salsburg, Z.W., Jackson, J.D., Fickett, W., Wood, W.W., 1959. Application of the Monte Carlo method to the lattice-gas model. I. Two-dimensional triangular lattice. J. Chem. Phys. 30, 65–72.

Shen, V.K., Errington, J.R., 2005. Determination of fluid-phase behavior using transition-matrix Monte Carlo: Binary Lennard-Jones mixtures. J. Chem. Phys. 122, 064508.

Shi, W., Johnson, J.K., 2001. Histogram reweighting and finite-size scaling study of the Lennard-Jones fluids. Fluid Phase Equilib. 187–188, 171–191.

Shi, W., Maginn, E.J., 2008. Improvement in molecule exchange efficiency in Gibbs ensemble Monte Carlo: development and implentation of the continuous fractional component move. J. Comput. Chem. 29, 2520–2530.

Shing, K.S., Gubbins, K.E., 1982. The chemical potential in dense fluids and fluid mixtures via a computer simulation. Mol. Phys. 46, 1109–1128.

Silva Fernandes, F.M.S., Fartaria, R.P.S., 2015. Gibbs ensemble Monte Carlo. Amr. J. Phys. 83, 809–816.

Singh, S.K., 2018. Critical temperature estimation of bulk and confined atomic fluid using vapour-liquid interfacial free energy. Mol. Sim. 44, 156–163.

Singh, A.N., Dyre, J.C., Pedersen, U.R., 2021. Solid-liquid coexistence of neon, argon, krypton and xenon studied by simulations. J. Chem. Phys. 154, 134501.

Smit, B., 1992. In: Allen, M.P., Tildesley, D.J. (Eds.), Computer Simulation in Chemical Physics. Kluwer, Dordrecht.

Smit, B., Frenkel, D., 1989. Calculation of the chemical potential in the Gibbs ensemble. Mol. Phys. 68, 951–958.

Smit, B., Williams, C.P., Hendriks, E.M., de Leeuw, S.W., 1989. Vapour-liquid equilibria for Stockmayer fluids. Mol. Phys. 68, 765–769.

Smith, W.R., Triska, B., 1994. The reaction ensemble method for the computer simulation of chemical and phase equilibria. I. Theory and basic examples. J. Chem. Phys. 100, 3019–3027.

Sokhan, V.P., 1997. The chemical potential of dense liquids by the couples test particle method. Mol. Sim. 19, 181–204.

Soto-Campos, G., Corti, D.S., Reiss, H., 1998. A small system grand ensemble method for the study of hard-particle systems. J. Chem. Phys. 108, 2563–2570.

Stapleton, M.R., Tildesley, D.J., Quirke, N., 1990. Phase equilibria in polydisperse fluids. J. Chem. Phys. 92, 4456–4467.

Sweatman, M.B., Quirke, N., 2004. Simulating fluid-solid equilibrium with the Gibbs ensemble. Mol. Sim. 30, 23–28.

Swope, W.C., Andersen, H.C., 1995. A computer simulation method for the calculation of chemical potentials of liquids and solids using the bicanonical ensemble. J. Chem. Phys. 102, 2851–2863.

Szalai, I., Liszi, J., Boda, D., 1995. The NVT plus test particle method for the determination of the vapour-liquid equilibria of pure fluids. Chem. Phys. Lett. 246, 214–220.

Torrie, G.M., Valleau, J.P., 1974. Monte Carlo free energy estimates using non-Boltzmann sampling: application to the sub-critical Lennard-Jones fluid. Chem. Phys. Lett. 28, 578–581.

Torrie, G.M., Valleau, J.P., 1977. Monte Carlo study of a phase-separating mixture by umbrella sampling. J. Chem. Phys. 66, 1402–1408.

Tsang, P.C., White, O.N., Perigard, B.Y., Vega, L.F., Panagiotopoulos, A.Z., 1995. Phase equilibria in ternary Lennard-Jones systems. Fluid Phase Equilib. 107, 31–43.

Valleau, J.P., 1991. Density-scaling: a new Monte Carlo technique in statistical mechanics. J. Comput. Phys. 96, 193–216.

Valleau, J.P., 1993. Density-scaling Monte Carlo study of subcritical Leannard-Jonesium. J. Chem. Phys. 99, 4718–4728.

Valleau, J.P., 1998. Number-dependence concerns in Gibbs-ensemble Monte Carlo. J. Chem. Phys. 108, 2962–2966.

Valleau, J.P., 2003. A thermodynamic-scaling study of Gibbs-ensemble Monte Carlo. Mol. Sim. 29, 627–642.

Valleau, J.P., 2005a. Temperature-and-density-scaling Monte Carlo: methodology and the canonical thermodynamics of Lennard-Jonesium. Mol. Sim. 31, 223–253.

Valleau, J.P., 2005b. Temperature-and-density-scaling Monte Carlo: isothermal-isobaric thermodynamics of Lennard-Jonesium. Mol. Sim. 31, 255–275.

Valleau, J.P., Card, D.N., 1972. Monte Carlo estimation of the free energy by multistate sampling. J. Chem. Phys. 57, 5457–5462.

van 't Hof, A., de Leeuw, S.W., Peters, C.J., 2006a. An advanced Gibbs-Duhem integration method: theory and applications. J. Chem. Phys. 124, 054906.

van 't Hof, A., de Leeuw, S.W., Peters, C.J., 2006b. Computing the starting state for Gibbs-Duhem integration. J. Chem. Phys. 124, 054905.

van Leeuwen, M.E., Peters, C.J., de Swaan Arons, J., Panagiotopoulos, A.Z., 1991. Investigation of the transition to liquid-liquid immiscibility for Lennard-Jones (12,6) systems using Gibbs-ensemble molecular simulations. Fluid Phase Equilib. 66, 57–75.

Vega, L., de Miguel, E., Rull, L.F., Jackson, G., McLure, I.A., 1992. Phase equilibria and critical behavior of square-well fluids of variable width by Gibbs ensemble Monte Carlo simulation. J. Chem. Phys. 96, 2296–2305.

Vega, L.F., Shing, K.S., Rull, L.F., 1994. A new algorithm for molecular dynamics simulations in the grand canonical ensemble. Mol. Phys. 82, 439–453.

Vlasiuk, M., Sadus, R.J., 2017. Predicting vapour-liquid phase equilibria with augmented *ab initio* interatomic potentials. J. Chem. Phys. 146, 244504.

Vlasiuk, M., Frascoli, F., Sadus, R.J., 2016. Molecular simulation of the thermodynamic, structural, and vapor-liquid equilibrium properties of neon. J. Chem. Phys. 145, 104501.

Vogt, P.S., Liapine, E., Kirchner, B., Dyson, A.J., Huber, H., Marcelli, G., Sadus, R.J., 2001. Molecular simulation of the vapour-liquid phase coexistence of neon and argon using ab initio potentials. Phys. Chem. Chem. Phys. 3, 1297–1302.

Vrabec, J., Gross, J., 2008. Vapor-liquid equilibria simulations and an equation of state contribution for dipole-quadrupole interactions. J. Phys. Chem. B 112, 51–60.

Vrabec, J., Lofti, A., Fischer, J., 1995. Vapour liquid equilibria of Lennard-Jones model mixtures from the NpT plus test particle method. Fluid Phase Equilib. 112, 173–197.

Wang, L., Sadus, R.J., 2006a. Effect of three-body interactions on the vapor-liquid phase equilibria of binary fluid mixtures. J. Chem. Phys. 125, 074503.

Wang, L., Sadus, R.J., 2006b. Three-body interactions and solid-liquid phase equilibria: application of a molecular dynamics algorithm. Phys. Rev. E 74, 031203.

Widom, B., 1963. Some topics in the theory of fluids. J. Chem. Phys. 39, 2808–2812.

Wilding, N.B., 1995. Critical-point and coexistence-curve properties of the Lennard-Jones fluid: a finite-size scaling study. Phys. Rev. E 52, 602–611.

Wilding, N.B., 1997. Simulation studies of fluid critical behaviour. J. Phys.: Condens. Matter 9, 585–612.

Wilding, N.B., Binder, K., 1996. Finite-size scaling for near-critical continuum fluids at constant pressure. Phys. A 231, 439–477.

Chapter 11

Molecular simulation and object-orientation

In common with most scientific or engineering programming applications, molecular simulation algorithms have been traditionally coded in procedural languages such as FORTRAN or C that emphasize the decomposition of a problem into various functions. Confronted with a problem to solve, a procedural language programmer typically performs a functional analysis, decomposing the problem into several composite algorithms that are implemented as either subroutines or functions. This algorithm-centered approach is a natural choice for problem solving for many scientific applications, and it usually works well. When used properly, modern procedural languages allow the programmer to write modularized code that can be easily reused, changed, and minimize errors. However, software developers are moving increasingly away from the procedural approach in favor of object-orientation (OO). In contrast to the procedural approach involving a set of functions sharing a global state, OO uses a set of interacting individual objects with their own private states. When used properly, OO offers enhanced flexibility in code design, error minimization, and software that can be more easily adapted to meet new situations and challenges (Amorsy et al., 2013).

At the outset, it should be emphasized that functional design and OO are complementary approaches. It is possible to include elements of OO in a procedural language. For example, two features of OO are information hiding and encapsulation. Information hiding in a procedural language such as C can be achieved via the use of library interfaces, and structures can be employed to encapsulate related variables together. Similarly, the structured programming strategies that are essential in good procedural code are also necessary to write good code for the methods contained in objects. In a sense, OO is an evolution from the procedural approach rather than a revolutionary step. Khan et al. (1995) have provided a detailed comparison of the similarities and differences between structural programming and OO.

The OO approach offers significant benefits for the development of molecular simulation codes, particularly for simulations involving complicated molecules. Historically, the priority for molecular simulation codes has been overwhelming computational efficiency. There is an ever increasing

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00010-1

interest in studying complicated molecules such as proteins and massive polymers. Apart from the computational difficulties involved in modeling complicated molecules, the structured programming approach alone is not ideally suited to meet the challenge of increasing molecular complexity. Converting a procedure-based program developed for simulating atoms to work for protein molecules is a daunting task. It may involve the development of new algorithms which, in part, are likely to be motivated by the limitation imposed by the structured programming approach rather than the molecule itself. It can be argued that the features of structured programming have partly dictated the nature of some molecular simulation algorithms. In contrast, OO has features, which can make the task much less daunting by promoting code reuse. Nelson et al. (1996) have discussed a parallel implementation of molecular dynamics (MD) using OO. Baekdal et al. (1996) have discussed the use of OO in protein dynamics. The OO approach forms the basis of commonly used open source molecular simulation software packages (Abraham et al., 2015; Phillips et al., 2020; Thompson et al., 2022) that are in widespread use.

Relatively few molecular simulators have formal training in software engineering and learn their craft "on the job." In the process of developing code using an OO programming language, aspects of the object-oriented approach are often not fully appreciated. In particular, coding often takes precedence over object-oriented modeling, which is contrary to good software development practice. The aim of this chapter is to attempt to bridge this gap by explaining and applying object-oriented concepts for both MD and Monte Carlo (MC) simulations via detailed case studies.

The outcomes of the object-oriented approach are implemented using C++. The choice[1] of C++ has the advantage that the code can be easily modified for parallel execution on graphical processor units (Chapter 12). The widely used and highly optimized C++ simulation codes are available (Abraham et al., 2015; Phillips et al., 2020; Thompson et al., 2022). Interestingly, there is evidence (Pereira et al., 2021) that it is energy efficient, which is an increasingly important consideration. Details of programming in C++ can be found in well-known texts (Deitel and Deitel, 2002; Stroustrup, 2013). A good description of the use of OO and C++ has been given by Parsons (1997). Haney (1994) has discussed some aspects of the computational speed of C++ code.

Code availability

The C++ source code for the MD, MC, and combined MC and MD programs are available for download from the book's website at: https://www.elsevier.com/books-and-journals/book-companion/9780323853989. A software user's guide is given in the Appendix.

1. OO principles are entirely independent of the programming language and can be applied in other OOP languages such as python or Java. Too much time and effort is often devoted to championing a preferred language.

11.1 Fundamental concepts of OO

The strategy of OO involves OO analysis (OOA), OO design (OOD), and OO programming (OOP). The first step in OOP is to analyze the problem and identify the composite classes. In the design stage, the relationships between the classes are identified. Finally, the design is implemented in a programming language that supports object-oriented concepts. In reality, there is often a seamless boundary between these processes, and we will not dwell on the distinction.

In practice, OO is built on the principles of abstraction, encapsulation, inheritance, aggregation, and polymorphism. Abstraction and encapsulation are essential to an object-oriented program whereas other features may or may not be present. It quickly becomes evident that OO has a vocabulary, which is largely foreign to structural programming approaches. Therefore our starting point is to explain the key concepts of OO. At the outset, it should be noted that OO is not confined to programming. As explained subsequently, developing object-oriented code is the end process of the object-oriented approach. Indeed, many object-oriented practitioners do not complete the object-oriented processes by actually creating a program! OO provides a new perspective on problem solving in which algorithmic development is deferred until the final stages of the process. This is possibly the biggest difference from the structural programming approach in which the choice of algorithm is made at the outset and as such, dictates the nature of the code.

11.1.1 Classes and objects

OO is an attempt to represent real-world objects in software. Some objects are concrete things whereas others are more abstract. However, the most important aspect of OO is that objects can be composed of both data (variables) and processes (methods), whereas in procedural programming, data and processes are always treated separately. The concept of a class and that of an object are closely linked. The first step in the object-oriented approach is to identify classes from which objects can be created. In the jargon of OO, an object is *instantiated* from a particular class. An obvious example of a class in the context of MD is an atom. The *Atom* class contains the attributes or properties of the atom such as the mass, position, velocity, energy, and the like. Argon is a concrete example of the class *Atom*. Therefore, we can create an object to represent argon from the class *Atom*. Krypton, xenon, and the like are other examples of atoms, and objects representing these atoms can be created from the *Atom* class. Indeed, we can create different objects to represent all the elements of the periodic table from the same *Atom* class.

It is convenient and very useful to make use of a diagrammatic notation to represent both classes and interactions between classes. Here, we will

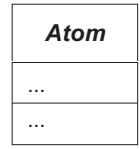


FIGURE 11.1 UML diagram representation of an *Atom* class. The middle and lower divisions are for private and public properties, respectively.

mainly use the broad features of the unified modeling language (UML) as described by Eriksson et al. (2003). The UML approach is an attempt to unify various object notations and it has many similarities with the object modeling technique (OMT) (Rumbaugh et al., 1991). It is possibly the most widely used form of object representation.

Fig. 11.1 is an UML diagram for the *Atom* class. The class is represented by a rectangle with three distinct divisions separated by solid lines. The name of the class is given as a heading in the top division. As discussed subsequently, the class can have both private and public information. The private properties are listed subsequently the class heading, whereas the bottom division is reserved for public information.

11.1.2 Abstraction and encapsulation

In general terms, abstraction is the attempt to define all the essential features of an object and group or encapsulate them with the object itself. In the earlier example, the class *Atom* was associated with various attributes, namely, position, mass, velocity, etc. Linking data with processes is an example of encapsulation and it allows the implementation of information hiding. Everything the object needs is encapsulated within the object and therefore hidden from all other objects. The definition of member variables inside structures in C is also an example of information hiding. However, unlike structures, encapsulation within classes is not limited to variables. Functions (or methods in object-oriented terminology) can also be encapsulated in a class. The encapsulated methods determine the behavior of the class and any object instantiated from it. Objects have a private part of hidden internal detail and a public interface, which defines the possible behavior of the object. An UML diagram for the *Atom* class with some possible attributes (mass, position, and velocity) and behavior (getMass, getVelocity, getPosition, setMass, setVelocity and setPosition) is illustrated by Fig. 11.2.

11.1.3 Methods and message passing

In general terms, an OO program works by passing messages between various objects using methods. Method is the OO terminology for

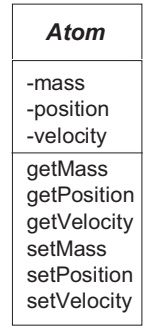


FIGURE 11.2 UML diagram for an Atom class showing private attributes (mass, velocity, and position) and public methods (getMass, getVelocity, getPosition, setMass, setVelocity, and setPosition).

functions/subroutines in procedural programming languages such as C and FORTRAN. As noted earlier, methods are encapsulated within the object. Methods can be divided into two broad types responsible for setting and getting values of attributes, respectively. Figure 11.2 illustrates some potential 'set' and 'get' methods for the attributes of the *Atom* class. The way 'set' and 'get' methods work in practice depends in part on how the objects are interrelated. This in turn depends on the inheritance, aggregation, and association characteristics of the objects.

11.1.4 Inheritance

A feature of OO is that one class can inherit properties from other classes. Inheritance is a powerful device for extending the functionality of a class and sharing common behaviors and properties between objects. However, it is important to note that inheritance only makes sense if a class is 'a kind of' another class. For example, polymers, proteins, and surfactants are different kinds of molecules. Therefore if we have a class *Molecule,* which represents the properties of molecules, we can have classes *Polymer*, *Protein*, and *Surfactant* that can legitimately inherit the properties and behavior of the *Molecule* class. An UML diagram representing inheritance between classes is illustrated in Fig. 11.3.

In UML, notation inheritance between classes is represented by arrows linking the classes. The head of the arrow points to the superclass (in this case the *Molecule* class) from which properties are inherited. The *Polymer*, *Protein,* and *Surfactant* classes are said to be derived from the *Molecule* superclass. Alternatively, they can be viewed as child classes obtained

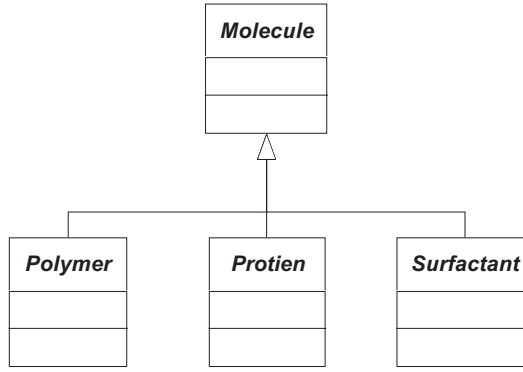


FIGURE 11.3 UML diagram illustrating the notation for inheritance from the superclass (*Molecule*) to the derived classes (*Polymer*, *Protein*, and *Surfactant*).

from a parent class. It should be noted that inheritance is a one-way process from parent class to the child classes. The parent class cannot inherit from a child class.

11.1.5 Aggregation

Some objects are 'part of' another object rather than being a 'kind of' a particular object. This important distinction allows us to create objects that are aggregations of other objects. In molecular simulation, there are many opportunities to exploit aggregations and/or associations (defined subsequently). Perhaps the most important candidate for aggregation is the *Molecule* class. By definition, an atom is part of a molecule. The UML diagram illustrating the aggregation relationship between *Molecule* and *Atom* is illustrated in Fig. 11.4. Notice that the diamond is attached to aggregated class (the *Molecule* class in this case). In general, a molecule is composed of two or more atoms, and therefore the cardinality of the relationship between *Molecule* and *Atom* is one to many represented in the UML diagram by the notation 1 ... *N*.

Of course, atoms are themselves composed of subatomic particles such as neutrons, electrons, and protons. Therefore if it is desired, the *Atom* class itself can be represented as an aggregation of objects representing subatomic properties as illustrated by Fig. 11.5.

It is quite common to combine aggregation and inheritance in an object-oriented analysis. Using the preceding examples, a complete description of polymers, surfactants, or polymers spanning the molecular level to the subatomic level can be obtained by the addition (Figs. 11.3 to 11.5), resulting in Fig. 11.6. *Polymer*, *Protein*, and *Surfactant* inherit from *Molecule*, giving them access to all the properties and behavior of *Molecule* including *Atom*, *Neutron*, *Proton*, and *Electron*.

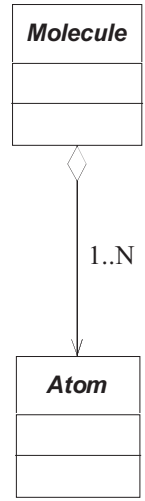


FIGURE 11.4 UML diagram representing the aggregation relationship between *Molecule* and *Atom*. The diagram indicates that one *Molecule* is composed of many *Atom*s.

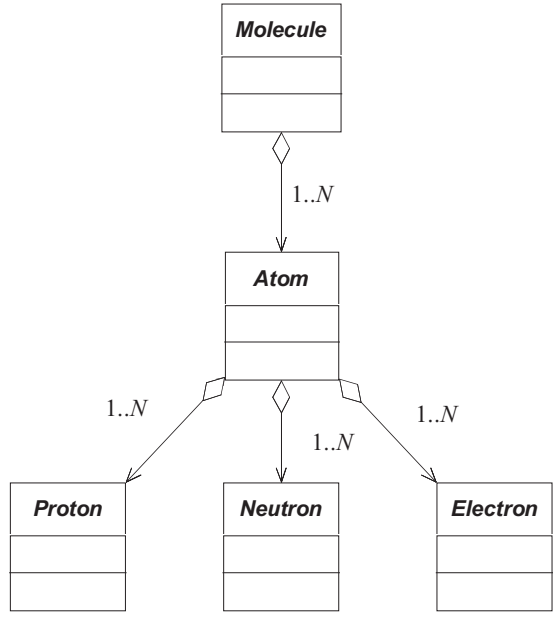


FIGURE 11.5 UML diagram illustrating different levels of aggregation. *Molecule* is an aggregation of *Atom*s, which in turn is an aggregation of different subatomic particles.

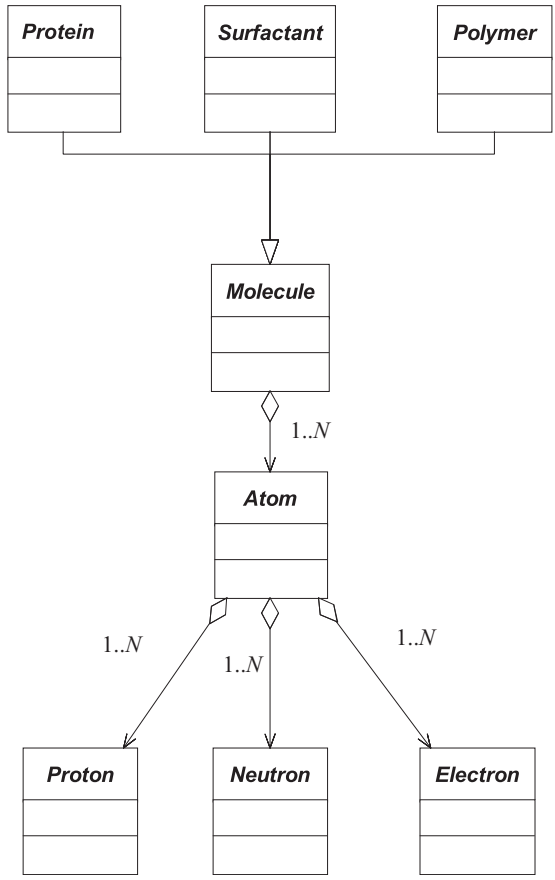


FIGURE 11.6 UML diagram illustrating the combined use of inheritance and aggregation.

11.1.6 Association

Aggregation is a form of association. However, more generally, an object may require another object without it being a 'part of' the object. This type of association is identified by the 'has a,' 'uses a,' 'needs a,' or any other similar dependency relationship. For example, in a molecular simulation, atomic interactions are typically described by an intermolecular potential. Therefore we can identify that the *Atom* class needs an intermolecular potential. In UML notation this is illustrated by a line between the class diagrams (Fig. 11.7).

It is important to note that the attributes and methods of an associated object could be also encapsulated directly into the object with which it is associated. For example, the attributes and methods of the *Intermolecular_Potential* class could become additional attributes and methods

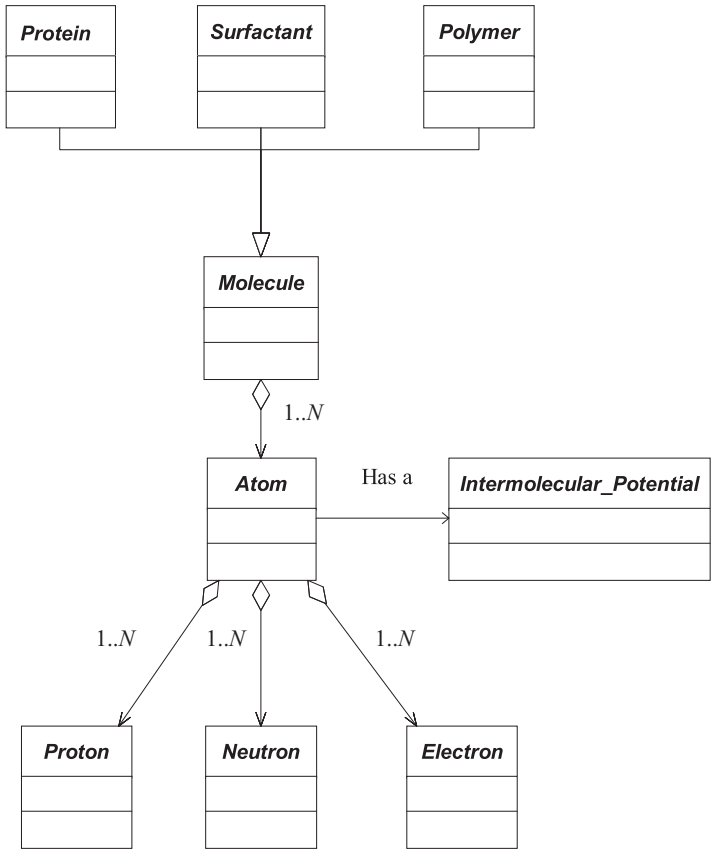


FIGURE 11.7 UML diagram showing the association (has a) between *Atom* and *Intermolecular_Potential* in addition to aggregation and inheritance relationships.

of the *Atom* class without having to create the *Intermolecular_Potential* class! Therefore the decision to create an association should be considered carefully.

11.1.7 Polymorphism

An advantage of OOP programming languages is that the same symbol can have different meanings depending on the context that it is used. Polymorphism is used to describe the ability to have many different forms. For example, in C the "*" operator is used to denote both multiplication and a pointer. There is no conflict between its different behaviors because the intended behavior is clear from the context that it is used. When multiplication is intended, "*" is used as a binary operator, whereas

it is used as a unary operator to denote pointer behavior. Languages such as C++ which support OOP allow us to define polymorphic behavior as required.

11.2 Case study: application of OO to the microcanonical MD simulation of Lennard-Jones atoms

In this section we illustrate the use of OO to the MD (Chapter 8) simulation of the microcanonical ensemble for Lennard-Jones (LJ) atoms. The three distinct stages of the processes—analysis, design, and programming—are illustrated separately. However, it should be noted that in reality the distinction is blurred. The design stage often results in a re-evaluation of the analysis. Similarly, the programming component can result in a re-think of the analysis and design. Commonly, the best software solution typically requires several iterations of the analysis, design, and programming stages.

It should be noted that the analysis provided here is exemplary only and intended to illustrate the key processes involved. It is likely that the same outcome can be obtained by adopting different strategies. Alternative outcomes are also valid; there are many ways to solve a problem and many different software solutions!

11.2.1 OOA and OOD

The starting basis for any object-oriented analysis is to provide a description of the problem to be solved. This problem description is used to identify potential classes, which will form the backbone of the object-oriented description. There are many ways to identify classes. Normally we can use our understanding of the problem to identify candidate classes. However, an effective and widely used technique is to identify major nouns. An example problem description is given subsequently; the nouns have been underlined as potential classes.

> Molecular dynamics simulation can be performed in a microcanonical (NVE) ensemble, in which the number of atoms, volume, and energy are held constant. The simulation proceeds for many time steps (Δt) until a predetermined duration (t_{total}) is reached. At the outset ($t = 0$) of the simulation, the atoms are placed on a face-centered cubic lattice and given initial values of velocity and accelerations. The nature of the atoms and initial values of density, energy, and/or temperature are specified. Temperature scaling of the velocities ensures that the ensemble has a constant energy. The force and energy interacting on each *atom* is calculated assuming pairwise interaction and an intermolecular potential such as the Lennard-Jones potential. Using the initial values of forces, velocities, and acceleration, Newton's equations of motion are applied to determine the next position at time $t = t + \Delta t$. In practice, this requires an integrator algorithm such as the Gear predictor method. If the Gear predictor method is used additional time derivatives with respect to position are calculated for each atom. The result of applying

Newton's equations of motion is to update the positions, velocities, acceleration, and force. The density and energy of the ensemble are stored at each time steps. Prior to equilibrium (t_{eq}), temperature scaling of the velocities is applied periodically to maintain the ensemble at constant energy. After the equilibrium period, ensemble averages of energy, temperature, and density are accumulated and statistics regarding these properties are kept. The statistical data for the ensemble is given at the end.

The potential candidates for classes arising from the problem description are listed in Table 11.1. In Table 11.1 some obviously related or identical potential classes (e.g., statistics/statistical data/averages) have been grouped together. The next step is to decide which of the potential classes from this noun-based analysis form useful classes. Of course, nouns can also be potential attributes. How do we distinguish between class and attributes? This part of the analysis benefits greatly from insights we have regarding the problem.

TABLE 11.1 Distinction between classes and attributes.

Potential class	Does the potential class have attributes/classes itself?	Class or attribute
acceleration	No, but vector components	Attribute
atom	Yes, type, position, etc.	Class
NVE ensemble	Yes, atoms, temperature, etc.	Class
density	No	Attribute
energy	Yes, but not useful in MD	Attribute
fcc lattice	Yes, positions	Class
force	Yes	Class
Gear Predictor-Corrector Newton's equations	Yes, force, acceleration, etc.	Class
integrator	Yes	Class
intermolecular potential Lennard-Jones potential	Yes, potential parameters but these are also part of atom	Attribute
molecular dynamics	Yes, all of the above	Class
statistical data, statistics	Yes, many!	Class
simulation	Yes, for example, molecular dynamics	Class
temperature	No	Attribute
time, step, time derivative	No	Attribute
velocity	No, but vector components	Attribute
volume	No	Attribute

However, as a rule of thumb useful classes are usually composed of other attributes and/or other classes. Therefore, for each potential class we ask the question: does this class contain attributes or other classes? If the answer is 'yes' then it may be a useful class otherwise it is an attribute. Of course, classes are also likely to contain methods but this aspect is excluded from our criterion because simple 'set' and 'get' methods can be devised for trivial classes, which are not useful for the problem analysis. The results of this analysis are summarized in Table 11.1. The assignment of velocity, acceleration, and energy as attributes rather than classes was the result of our insight into the problem as will become apparent later. However, because these properties each have vector components they could also be designated legitimately as classes.

In general, apart from making the distinction between classes and attributes, elimination of potential classes can be made by other criteria (Rumbaugh et al., 1991) such as redundancy, vagueness, irrelevancy, and operations. Redundant classes are classes that have an identical meaning but are expressed differently, for example, statistics, statistical data, and averages. We may also decide that some potential classes are defined too vaguely to have any practical meaning whereas others are irrelevant. Some nouns may also represent dynamic processes that are better viewed as operations rather than classes. Furthermore, some potential classes may arise as an artefact of the implementation rather than as part of the description of the problem. For example, it may be helpful to define classes to handle such items as data input, output, and array allocation, which arise from the implementation but they are clearly not part of the problem description.

The major classes are identified in Fig. 11.8 using short-hand UML notation. The short-hand notation is used because at this early stage we are not interested in either the attributes or methods. The next step is to identify the relationships or associations between the various classes. This is achieved by identifying either dependency (needs-a, has-a, requires-a, etc.) or inheritance (is-a kind of) relationship between the classes. *Molecular_Dynamics* is a type of *Simulation*. *Molecular_Dynamics* is performed in the context of the *NVE_Ensemble*. Consequently, there is a well-defined association between

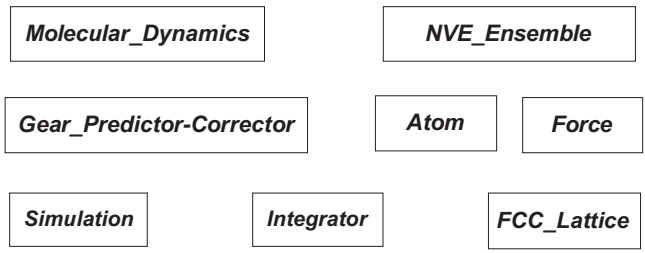


FIGURE 11.8 Short-hand UML notation for the preliminary major classes identified from object-oriented analysis.

these two classes. A *NVE_Ensemble* is composed of many *Atoms* indicating an aggregation relationship between these two classes. In turn each *Atom* has a *Force*. In *Molecular_Dynamics* the position, velocity, and acceleration of the *Atoms* is updated by a differential equation *Integrator* algorithm. Therefore *Molecular_Dynamics* requires an *Integrator*. The *Gear_Predictor-Corrector* is an example of an *Integrator*. The usefulness of the *FCC_Lattice* class is questionable because it has no independent existence of the atoms. Instead, the creation of a lattice is a candidate method of the ensemble. Therefore we eliminate it at this stage. These relationships are summarized in the UML diagram given in Fig. 11.9.

At this stage the UML diagram (Fig. 11.9) should be examined to identify any weakness in the design. One potential problem in the design is the relationship between *Atom* and *Force*. Although each atom has-a force, the calculation of force is not independent of each atom but rather it is the result of all pairwise interaction between the atoms. Therefore, it is preferable to make *Force* a property of the *NVE_Ensemble* rather than the *Atom* class as illustrated in Fig. 11.10. In Fig. 11.10 an association has been maintained between force and atoms. To work effectively, *Force* needs *Atoms*. However, as represented in Fig. 11.10 there is only one *Force* object, compared with *N* *Force* objects implied by Fig. 11.9. The weakness in the original design (Fig. 11.9) would have become very apparent in the programming phase because of the large overhead imposed by instantiating objects of type *Force* and *Gear_Predictor-Corrector* for each object of type *Atom*.

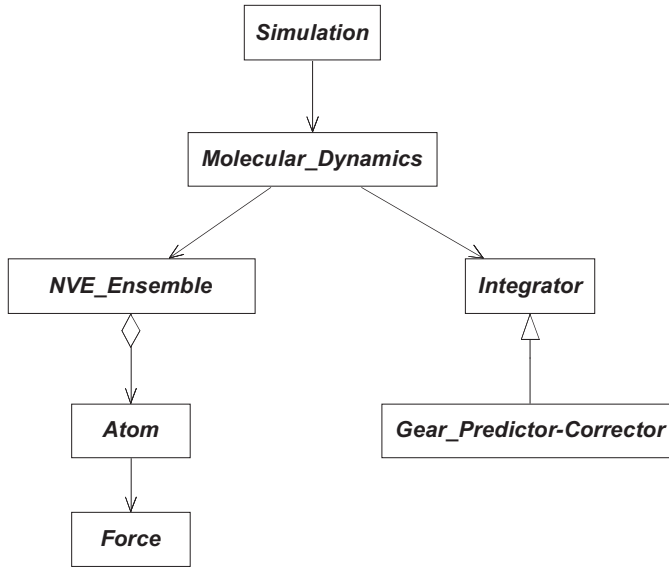


FIGURE 11.9 Preliminary UML diagram for MD in the microcanonical ensemble.

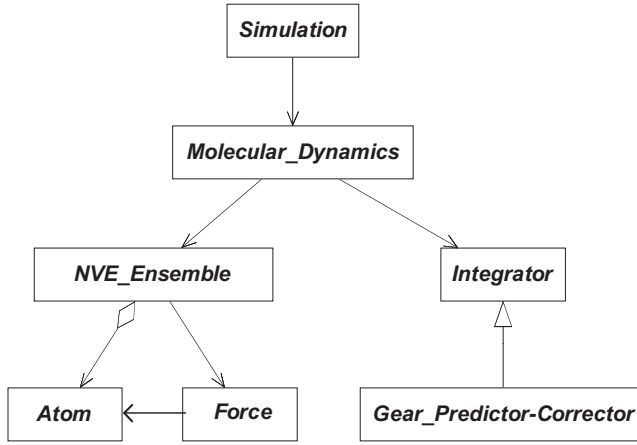


FIGURE 11.10 First revision of the UML diagram for the MD simulation in the microcanonical ensemble. Atom and force are now both created from *NVE_Ensemble*.

By examining Fig. 11.10 and considering the problem description, further refinements to the analysis are suggested. We know that the microcanonical ensemble is only one possible ensemble. This suggests that the creation of a generic *Ensemble* class may be useful. Conversely, the *Atom* and *Force* classes are generic, whereas the problem description deals specifically with atoms interacting via the Lennard-Jones potential. These insights suggest the use of *LJ_Atom* and *LJ_Force* classes. The revised UML diagram incorporating these additional classes is illustrated in Fig. 11.12. In Fig. 11.11, the *Ensemble*, *Force*, and *Atom* classes are denoted as abstract classes. In effect, abstract classes serve as placeholders for other classes. The benefit of using abstract classes is that they facilitate expansion of the analysis at a later date (see Chapter 11.5). The *NVE_Ensemble* is an example of an *Ensemble* class so it inherits from *Ensemble*. Similarly, *LJ_Atom* and *LJ_Force* inherit from the *Atom* and *Force* classes, respectively.

Reflecting on the association between classes in Fig. 11.11 suggests the possibility of a further refinement. In Fig. 11.11, there is no direct interaction between the *Atom* and *Integrator* classes. To operate effectively, *Integrator* needs to access *Atom* because part of its role is to update the properties of *Atom*. The additional association between *Integrator* and *Atom* is illustrated in Fig. 11.12. An alternative and equally valid mechanism is to define an association between the *Integrator* and *Molecular_Dynamics* classes. However, this would be less direct involving *Molecular_Dynamics* to provide *Atom* to *Integrator*.

The process of identifying associations between classes, determining their attributes, and behavior can be facilitated by maintaining and updating a data dictionary. The dictionary describes the classes, detailing associations, attributes, and operations. A preliminary dictionary summarizing the details to date is illustrated in Table 11.2.

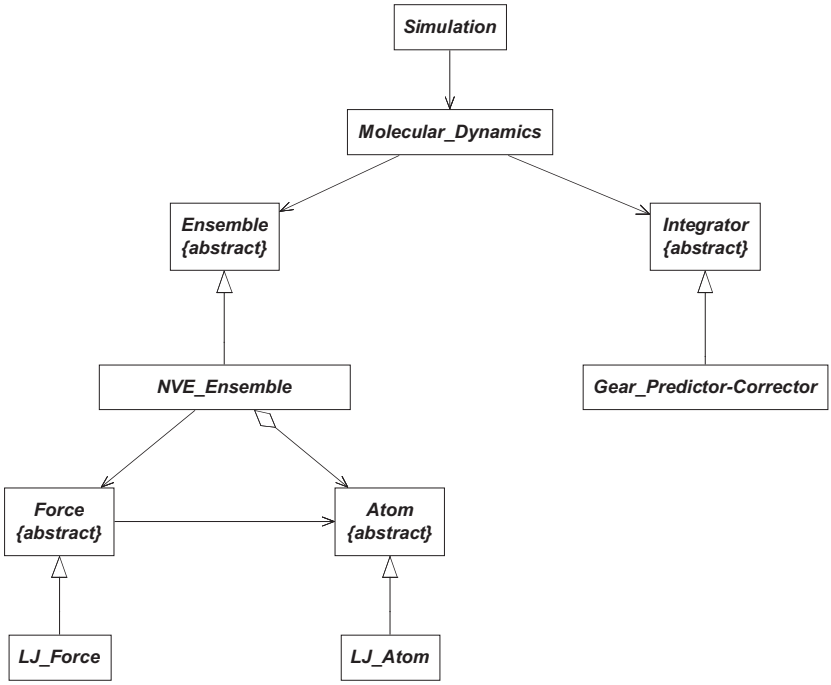


FIGURE 11.11 Second revision of UML diagram showing the use of abstract classes.

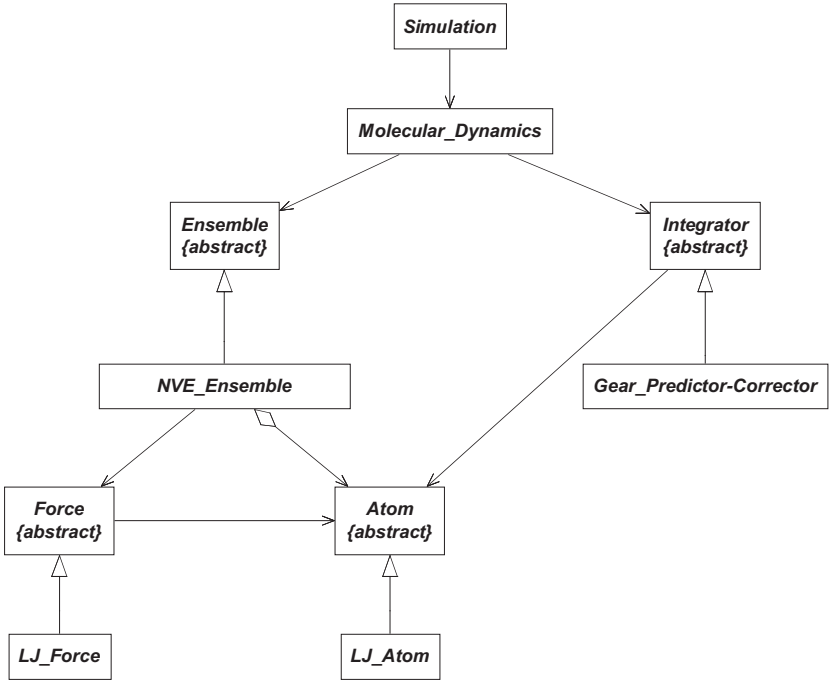


FIGURE 11.12 Third revision of UML diagram showing the added association between the *Integrator* and *Atom* classes.

TABLE 11.2 Preliminary data dictionary describing classes.

Class	Description
Atom	Abstract class serving as a placeholder for the *LJ_Atom* class. Created by *NVE_Ensemble*.
NVE_Ensemble	An example of an ensemble used to generate all the *Atoms* in the *Ensemble* and the *Force* class. Created by *Ensemble*.
Ensemble	Abstract class created by *Molecular_Dynamics*. Generates *NVE_Ensemble*.
Force	Abstract class serving as a placeholder for *LJ_Force* class. Force needs Atom. Created by *NVE_Ensemble*.
Gear_Predictor-Corrector	Contains information related to the Gear PC method. Updates positions, velocities, and acceleration. Created by *Molecular_Dynamics*.
LJ_Atom	Contains details of the Lennard-Jones atom. Created by *Atom*.
LJ_Force	Performs the force calculation. Created by *Force*.
Molecular_Dynamics	Creates the *Ensemble* and the *Integrator* classes and drives the simulation. It requires *Atom*.
Simulation	Creates the *Molecular_Dynamics* and handles I/O.

11.2.2 Detailed class description

Determining the attributes (both variables and methods) is very much an iterative process. Some attributes are fairly obvious, many attributes are apparent from the OOA but the remainder will require further analysis of the problem. Often, 'missing' attributes will only become apparent at the programming phase. Here we will present the 'final' result, which has undergone the process of iterative refinement. The process of determining attributes and methods is similar to the process that is undertaken in conventional structural programming in identifying functions and appropriate variables for functions. The main difference is that the methods and attributes both belong to a class. To create a working object or class, two distinct methods are usually required. The 'set' methods perform calculations and are responsible for assigning values to class attributes. They correspond to either functions or subroutines in structural programming language. The 'get' methods are used to retrieve the values of the class's attributes determined by the 'set' methods. They involve typically one line of code containing an appropriate 'return' statement. Later, the attributes and workings of specific classes must be identified. In C++, the class description can be mapped very closely into header files and we will

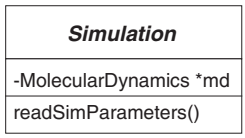
therefore provide the header files side by side with the UML diagrams. The use of header files[2] is advantageous because it separates the programming interface from the working program. In some small number of cases, the source code contains additional methods not given in the UML diagrams, which post-design coding decision, for example, different implementations of a particular method.

The C++ 20 standard released in 2021 provides the ability to express modularity using the module keyword. This can now be used as an alternative to header files. Details of the latest changes to C++ have been summarized by Stroustrup (2023).

11.2.2.1 Simulation

The *Simulation* class is the driver class for the whole program. It has one method *readSimParameters()* responsible for determining the choice of simulation. After the *Molecular_Dynamics* object is created the *run()* method of that class is called.

Simulation Class



md_Ver2.0\simMD.h

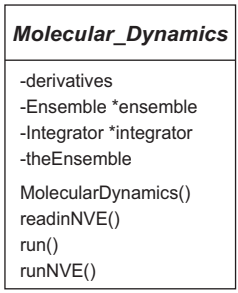
```
// File:  simMD.h
// -----------------
// This file contains the definition of
// the Simulation class, which defines
// all methods and data for MD or MC.

#ifndef _sim_h_
#define _sim_h_

#include "md.h"

class Simulation{
  private:
    MolecularDynamics *md;
  public:
    void readSimParameters();
};
#endif
```

11.2.2.2 Molecular_Dynamics

The *Molecular_Dynamics* class controls the simulation process via the *runNVE()* method, which is called by the *run()* method. It contains a reference to the *Ensemble* and *Integrator* classes.

2. The naming of the header file is arbitrary but it is sensible to adhere to the convention of source-file-name.h. Some C++ programmers prefer to use the .hpp extension for differentiation with C header files. However, we prefer .h because .hpp can be easily mistaken for .cpp increasing the likelihood of the wrong file being inadvertently deleted or overwritten.

Simulation Class



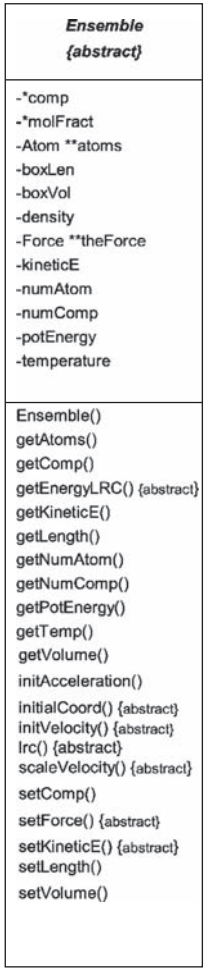
md_Ver2.0\md.h

```
//  File:  md.h
// -----------------
// This file contains the definition of
// the MolecularDynamics class, which
// defines all methods and data for a
// Molecular Dynamics simulation.

#ifndef _md_h_
#define _md_h_

#include "integrtMD.h"
#include "ensembleMD.h"

class MolecularDynamics
{
  private:
    Ensemble *ensemble;
    Integrator *integrator;
    int theEnsemble,derivatives;
  protected:
    int nSize, nEquil, nStep;
    double tStep;
  public:
    MolecularDynamics();
    int getnSize();
    int getnEquil();
    int getnStep();
    double gettStep();
    void runNVE();
    void readInNVE(int );
    void run();
};

#endif
```

11.2.2.3 *Ensemble*

The *Ensemble* class contains most of the required variables and methods of the simulation. It has references to the *Force* object, an array of *Atoms* and the simulation parameters. *Ensemble* also has abstract methods that are defined in the derived class *NVE_Ensemble*.

Simulation Class



md_Ver2.0\ensembleMD.h

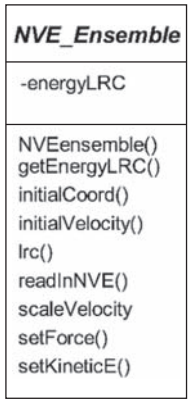
```
// File:  ensembleMD.h
// -----------------
// This file contains the definition of the
// abstract class Ensemble that defines all
// methods and data common to different
// ensemble types (eg derived classes), as
// well as foreshadowing methods of those
// derived classes through the use of abstract
// methods.

#ifndef _ensembleMD_h
#define _ensembleMD_h

#include "atomMD.h"
#include "forceMD.h"

// Class: EnsembleMD
// ---------------
// The EnsembleMD class contains the physical
// attributes of the abstract ensemble, from
// which instantiable ensemble classes are
// derived, such as the NVE ensemble.

class EnsembleMD
{
 protected:
   Atom  **atoms;      //array of atoms
   Force *theForce;    //the force
   int numComp;        //no. of components
   int numAtom;        //no. of atoms
   int *comp;          //no. comp. each type
   double temperature; //temperature
   double density;     //density
   double boxVol;      //box volume
   double boxLen;      //box length
   double *molFract;   //mole fractions
   double kineticE;    //kinetic energy
   double potEnergy;   //potential energy
 public:
   Ensemble(Atom **, int, int,
     double, double, double *, int *);
   virtual ~Ensemble();
   Atom **getAtoms();   //return atom array
```

```
    void setVolume();      //set box volume
    void setLength();      //set box length
    void setComp();        //set atom type
    void initAcceleration(); //initial accel.
    int getNumAtom();     //get total atoms
    int *getComp();       //get no.components
    int getNumComp();     //get total comp.
    double getTemp();     //get temperature
    double getVolume();   //get box volume
    double getLength();   //get box length
    double getPotEnergy();//get pot energy
    double getVirial();   //get virial
    getKineticE();        //get kinetic energy

    // Abstract Methods
    // ----------------
    // The following methods are defined in
    // the NVEensembleMD class(nve.h &nve.cpp).

    virtual void initialVelocity()=0;
    //assign the initial atom velocities

    virtual void initialCoord()=0;
    //place atoms on lattice

    virtual void scaleVelocity()=0;
    //scale velocities to temperature

    virtual void setForce()=0;
    //instigate force calculations

    virtual void setKineticE()=0;
    //set ensemble Kinetic Energy

    virtual void setKineticE(double) = 0;
    //set the Kinetic Energy;

    virtual double getEnergyLRC()=0;
    //get energy long range correction

    virtual double getVirialLRC() = 0;
    //get virial long range correction

     virtual void lrc()=0;
    //trigger the long range corrections
};

#endif
```

11.2.2.4 NVE_Ensemble

The *NVE_Ensemble* class contains attributes and methods that are specific to the microcanonical ensemble and other methods that are required for the simulation. The constant energy is maintained via velocity scaling.

Simulation Class



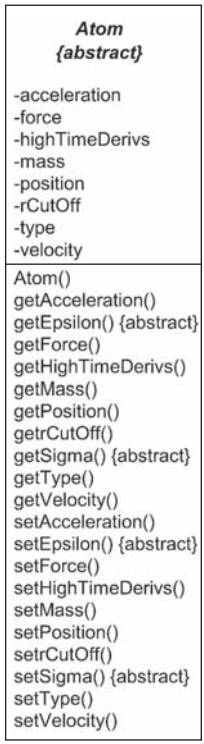
md_Ver2.0\nveMD.h

```
// File:  nveMD.h
// -----------------
// This file contains the definition of the
// NVEensembleMD class, which defines all
// methods and data for the microcanonical
// ensemble.

#ifndef _nveMD_h
#define _nveMD_h

#include "ensembleMD.h"

class NVEensemble : public EnsembleMD
{
 private:
   double energyLRC;   //long range E.corr.
   double virialLRC;   //long range vir. corr
 public:
   NVEensemble(AtomMD **, int, int, double,
       double, double *, int *);
   void readInNVE();
   //read in ensemble data
   virtual void initialVelocity();
   //assign the initial atom velocities
   virtual void initialCoord();
   //place atoms on lattice
   virtual void scaleVelocity(double);
   //scale velocities to temperature
   virtual void setForce();
   //instigate force calculations
   virtual void setKineticE();
   //set ensemble Kinetic Energy
   virtual void setKineticE(double);
   //set the ensmble kinetic energy
   virtual double getEnergyLRC();
   //get energy long range correction
    virtual double getVirialLRC();
   //get virial long range correction
   virtual void lrc();
   //trigger the long range corrections
};
#endif
```

11.2.2.5 Atom

The class *Atom* is an abstract container class for all data and methods common to all atom types such as position and mass. *Atom* contains a two-dimensional array, *highTimeDerivs*, which stores values of $(\partial^n \mathrm{r}/\partial t^n)$ where $n > 2$ required by the Gear predictor−corrector algorithm. The *velocity* $(n = 1)$ and *acceleration* $(n = 2)$ are given separately.

Simulation Class



md_Ver2.0\atomMD.h

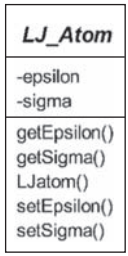
```
// File:  atomMD.h
// -------------
// This file contains the definition of
// abstract class Atom, which defines all
// methods and data common to different
// atom types (eg derived classes), as well as
// foreshadowing methods of those derived
// classes through the use of abstract
// methods.

#ifndef _atomMD_h_
#define _atomMD_h_

class AtomMD
{
 protected:
   int type;                //type of atom
   double mass;             //mass of the atom
   double **rCutOff;        //cut off distance
   double *position;        //position
   double *velocity;        //velocity
   double *acceleration;    //acceleration
   double **highTimeDerivs; //high. time der.
   double *force;           //force
 public:
   Atom(int, double, int, int);
   Atom();
   void setType(int);
   //assign type to atom
   void setrCutOff(double **);
   //assign cut off point for atom
   void setMass(double);
   //assign mass to atom
   void setAcceleration(double *);
   //assign acceleration to the atom
   void setPosition(double *);
   //assign position of atom
   void setVelocity(double *);
   //assign velocity to atom
   void setHighTimeDerivs(double **);
   //set derivatives of time
```

```
   void setForce(double *);
   void setForce(double,double,double);
   void setForce(int, double *);
   //set the force of the atom
   double **getrCutOff();
   //return cut off point for atom
   double *getAcceleration();
   //get acceleration of atom
   double *getPosition();
   //get position of atom
   double *getVelocity();
   //get velocity of atom
   double **getHighTimeDerivs();
   //get derivatives of time
   double *getForce();
   double *getForce(int);
   //get the force of the atom
   double getMass();
   //get mass of atom
   int getType();
   //get type of atom

   // Abstract Virtual Methods
   // ------------------------
   // LJatom derived class methods
   // ----------------------------

   virtual void setSigma(double **)=0;
   //set sigma for atom pairs
   virtual void setEpsilon(double **)=0;
   //set epsilon for atom pairs
   virtual double **getEpsilon()=0;
   //get epsilon for atom pairs
   virtual double **getSigma()=0;
   //get sigma for atom pairs
};

#endif
```

11.2.2.6 LJ_Atom

The *LJatom* class inherits from Atom and defines those attributes and methods that are specific to atoms interacting via the LJ potential. In particular, it contains the ε and σ parameters.

Simulation Class md_Ver2.0\ljatomMD.h

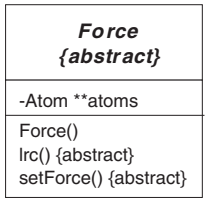



```
// File:  ljatomMD.h
// -----------------
// This file contains the definition of the
// LJatomMD class, which defines all methods a
// and data for an LJatomMD  interacting via
// the Lennard-Jones 12-6 intermolecular
// potential.

#ifndef _ljatomMD_h
#define _ljatomMD_h

#include "atomMD.h"

class LJatom : public Atom
{
 private:
   double **epsilon;             //LJ epsilon
   double **sigma;               //LJ sigma
 public:
   LJatom(int , double , double **,
     double **, double **, int, int);
   LJatom();
   virtual void setSigma(double **);
   //set sigma for atom pairs
   virtual void setEpsilon(double **);
   //set epsilon for atom pairs
   virtual double **getEpsilon();
   //get epsilon for atom pairs
   virtual double **getSigma();
   // get sigma for atom pairs
};

#endif
```

11.2.2.7 Force

Force is an abstract class that maintains a reference to the array of atoms. It is a container class for the methods in its derived classes that are responsible for performing the *setForce* calculations and the long-range corrections.

Simulation Class md_Ver2.0\forceMD.h

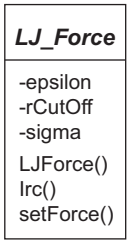



```
// File: forceMD.h
//----------------
// This file contains the definition of
// abstract class ForceMD, which defines all
// methods and data common to different
// force types (eg derived classes), as well
// as foreshadowing methods of those derived
// classes through the use of abstract
// methods.

#ifndef _forceMD_h
#define _forceMD_h

#include "atomMD.h"

class Force
{
 protected:
   Atom **atoms;   //reference to atom array
 public:
   Force(Atom **);
   virtual ~Force();

   // Abstract Methods for subclasses
   // -------------------------------
   // for LJforce:
   virtual void setForce(int, double,
              double *, double *)=0;
   //set force calculations
   virtual void lrc(int, int *, double,
              double *, double)=0;
   //long range corrections
};

#endif
```

11.2.2.8 LJ_Force

LJ_Force is the subclass of *Force* that is responsible for performing the long-range corrections and force calculations using the LJ potential.

Simulation Class



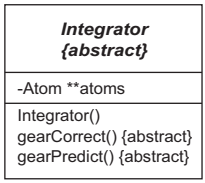
md_Ver2.0\ljforceMD.h

```
// File:  ljforceMD.h
// -------------
// This file contains the definition of the
// Ljforce class, which defines all methods
// and data for a ForceMD object which
// interacts according to the Lennard-Jones
// intermolecular potential.

#ifndef _ljforceMD_h_
#define _ljforceMD_h_

#include "forceMD.h"

class LJforce : public Force
{
 private:
   double **epsilon, **sigma, **rCut;
   //reference to epsilon, sigma and rCut
 public:
   LJforce (Atom **);

   virtual void setForce(int, double,
           double *, double *);
   //set force calculations
   virtual void lrc(int, int *, double,
            double *, double *);
   //long range corrections
};

#endif
```

11.2.2.9 Integrator

The *Integrator* class is a container class for all derived integrator methods. Apart from its constructor, it only contains abstract methods that will be implemented in its subclasses.

Simulation Class md_Ver2.0\integrtMD.h

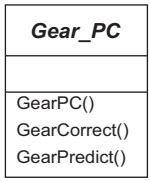



```
// File:  integrtMD.h
// -----------------
// This file contains the definition of the
// abstract class IntegratorMD, which defines
// all methods and data common to integrator
// methods.

#ifndef _integrtMD_h_
#define _integrtMD_h_

#include "atomMD.h"

class Integrator
{
 protected:
   Atom **atoms;
 public:
   Integrator(Atom **);
   ~Integrator();
   virtual void gearPredict(int,
            double, double, double)=0;
   //Gear's predictor
   virtual void gearCorrect(int, double,
             double)=0;
   //Gear's Corrector
};

#endif
```

11.2.2.10 Gear_Predictor−corrector

The *Gear_PC* class implements the methods of the Gear predictor−corrector integrator algorithm (Algorithm 7.2). It inherits a reference to the array of atoms from its base class, which allows calculations to be performed with little overhead from parameter passing.

Simulation Class



md_Ver2.0\gpcMD.h

```
// File:  gpcMD.h
// -----------------
// This file contains the definition of the
// GearPCMD class, which defines all methods
// data for the Gear predictor-corrector
// implementation of the integrator method.

#ifndef _gpcMD_h_
#define _gpcMD_h_

#include "integrtMD.h"

class GearPC : public Integrator
{
 public:
   GearPC(Atom **);
   virtual void gearPredict(int, double,
           double);
   //Gear's predictor
   virtual void gearCorrect(int, double,
            double);
  //Gear's Corrector
};

#endif
```

The Gear predictor method relies on a number of constants, which could have been also included in the gpcMD.h file. However, in this case, a separate gpcConstants.h file is used, as illustrated below.

md_ver2.0/gpcConstantsMD.h

```
// File:  gpcConstantsMD.h
// -----------------
// This file contains the constants required for
// the Gear-Predictor method.

#ifndef _gpcConstantsMD_h_
#define _gpcConstantsMD_h_

const double G0 = 0.1875;
const double G1 = 0.6972222222;
const double G3 = 0.6111111111;
const double G4 = 0.1666666667;
const double G5 = 0.0166666667;

#endif
```

11.2.2.11 Auxiliary methods

In common with many applications, there are a subset of methods and utilities that are not specific to a particular but are required by several classes. In

effect, these methods supplement the methods in the standard libraries that are provided with the compiler. The header file for these functions is auxfunc.h.

md_Ver2.0\auxfunc.h

```
// File: auxfunc.h
// -----------------
// Library interface for general purpose methods
// that are not specific to any class.

#ifndef _auxfunc_h
#define _auxfunc_h

double **getMemory(int num);
//Allocates memory for a one dimensional array
double **getMemory(int xNum, int yNum);
//Allocates memory for a two dimensional array
double nearestInt(double vector, double boxL);
//Finds the nearest integer
double random(int *idum);
//Finds a random number on the interval  0 to 1
void freeMemory(double **);
void freeMem(double **, int);
//Methods to free memory

#endif
```

11.2.3 Object-oriented programming in C++

After defining the class diagrams and prototyping the methods and variables required for the various classes, we enter the programming phase. The variables and methods for each class are written in a source file with the same name as the header file. This aspect of the object-oriented process involves using the principles of structural programming to design robust and well-structured methods. The design of methods is the algorithm phase of OO. Notice that it comes naturally as the last step in the object-oriented process, whereas the structural programming approach is algorithm-centered from the outset.

11.2.3.1 Code for the main() function

The previous object-oriented analysis has not addressed the role of the main() function. All C or C++ programs must contain the main() function. Where is the main() function in our analysis? The answer to this question is that it is effectively outside of the analysis. In the context of any object-oriented analysis, main() has a minimal role involving typically only the instantiation of one object. Thereafter, the program proceeds in accordance with the

instructions in the various classes, which control the creation of objects and message passing between objects via method calls. In view of the object-oriented analysis summarized by Fig. 11.18, the role of main() is clearly to instantiate an object of type *Simulation* as illustrated in the code subsequently.

md_Ver2.0\mainMD,cpp

```
// File mainMD.cpp
// -----------
// This file contains the main() function
// responsible for starting the simulation
/  process.

#include "simMD.h"

int main(int argc, char **arv)
{
  Simulation sim;
  sim.readSimParameters();
  return 0;
}
```

An object of type Simulation is created.

Call to readSimParameters() invokes the simulation process.

11.2.3.2 Code for the simulation class

md_Ver2.0\simMD.cpp

The simulation class is responsible for reading in the simulation parameters.

In the main() an object of type Simulation is instantiated and the readSimparameters() method is called.

The simulation settings are read from the "simMD.dat" file.

If the MD option is chosen a reference (md) to an object of MolecularDynamics is created.

The run() method of MolecularDynamics is called which starts the simulation process.

The Simulation class can be expanded later to include MC.

```
// File:  simMD.cpp
// -----------------
// This file contains the implementation of
// the Simulation class, which defines all
// methods and data for a molecular
// simulation, using either MD or MC.

#include "simMD.h"
#include <fstream>
#include <iostream>

using namespace std;

// Method:  readSimParameters
// Usage:   readSimParameters();
// ---------------------
// Reads in parameters for NVE ensemble

void Simulation::readSimParameters()
{
  int simulation;
  //open the sim.dat file

  ifstream in;
  in.open("simMD.dat");

  if(in.fail())
  {
    cout << "Cannot open simMD.dat!\n";
    return;
  }

  in >> simulation;
  in.close();

  switch(simulation)
  {
    case 1:  //Molecular Dynamics
      md = new MolecularDynamics();
      md->run();
      break;
    case 2:  //Monte Carlo
      cout << "Not Yet Implemented" << endl;
      break;
    default:
       cout << "Invalid! Aborting!" << endl;
       break;
  }
  return;
}
```

11.2.3.3 Code for the Molecular_Dynamics class

md_Ver2.0\md.cpp

```
// File:  md.cpp
// Class: MolecularDynamics
// ------------------------
// This file contains the implementation
// details for the Molecular dynamics class
// to perform an MD simulation

#include "md.h"
#include "nveMD.h"
#include "gpcMD.h"
#include "ljatomMD.h"
#include "auxfunc.h"
#include <math.h>
#include <fstream>
#include <iostream>

using namespace std;
```

The constructor is called when a MolecularDynamics object (md) is created in the Simulation object (sim).

```
// Constructor
// -----------
// Accesses parameter file md.dat, which
// identifies the choice of intermolecular
// potential, ensemble, and integrator method,
// & constructs the MolecularDynamics object.

MolecularDynamics::MolecularDynamics()
{
  int integratorM, InterPotential;
  ifstream in;

  in.open("md.dat");

  if(in.fail())
  {
    cout << "Unable to open md.dat
        for MD parameters" << endl;
    return;
  }

  in >> nStep;
  in >> nEquil;
  in >> nSize;
  in >> tStep;
  in >> integratorM;
```

The number of derivatives depends on the order of the Gear

```
  if(integrator ==1)                  //Gear PC
    in >> derivatives;
```

PC (5 in this case).

```
  in >> theEnsemble >> InterPotential;
  in.close();
```

The appropriate read-in method is called.

```
switch(theEnsemble)
  {
    case 1:                      //NVE ensemble
      readInNVE(potential);
      break;
    //other ensembles here
  }
```

Integrator points to an object of type GearPC.

```
  switch(integratorM)
  {
    case 1:
      integrator = new
                GearPC(ensemble->getAtoms());
      break;
   }

}
```

The run method is invoked by md->run() from the sim object.

```
// Method: run
// Usage:  run();
// -------------
// Runs the appropriate simulation run method
// determined by the choice of ensemble.
```

Different ensembles/integrator combinations will require different run() methods. This method controls the execution of the simulation.

```
void MolecularDynamics::run()
{
  switch(theEnsemble)
  {
    case 1: // NVE ensemble
      runNVE();
      break;
    //other ensembles can be inserted here
  }
}
```

Simulation settings.

```
void MolecularDynamics::runNVE()
{
  ofstream out;
  out.open("md_nve.out", ios::out);

  if(out.fail()) {
     cout << "could not open output file!"
     << endl;
     return;
  }

  int i, j = 0, n, nTotal;
  int nStep = getnStep();
  int nEquil = getnEquil();
  int nSize = getnSize();
```

Initialising averages.

```
int num       = ensemble->getNumAtom();
double length = ensemble->getLength();
double volume = ensemble->getVolume();
double tStep = gettStep();
double potE, kinE, totE, temp, lrc, lrcw;
double avPotEnergy = 0.0;
double avKinEnergy = 0.0;
double avTotEnergy = 0.0;
double avVirial = 0.0;
double avTemperature = 0.0;
double erKinEnergy = 0.0;
double erPotEnergy = 0.0;
double erTotEnergy = 0.0;
double erTotEnergy = 0.0;
double erVirial = 0.0;
double erTemperature = 0.0;
double pressure, erPress;
double *accumPotE, *accumKinE;
double *accumTotE, *accumVir, *accumTemp;
```

Determine size of simulation blocks. Properties to be averaged are accumulated in blocks of size n.

```
nTotal = nStep - nEquil;
n = nTotal/nSize;
accumPotE = new double [n];
accumKinE = new double [n];
accumTotE = new double [n];
accumTemp = new double [n];
accumVir  = new double [n];
accumTemp = new double [n];
```

Initialization of the properties of ensemble prior to the simulation.

```
for (i = 0; i < n; i++)
{
  accumPotE[i] = 0.0;
  accumKinE[i] = 0.0;
  accumTotE[i] = 0.0;
  accumTemp[i] = 0.0;
  accumVir[i]  = 0.0;
  accumTemp[i] = 0.0;
}

ensemble->lrc();
//long range corrections
lrc = ensemble->getEnergyLRC();
lrcW = ensemble->getVirialLRC();
ensemble->setForce();
//set initial force
ensemble->initAcceleration();
//initial acceleration
ensemble->initialVelocity();
//initial velocity
ensemble->setKineticE();
//determine kinetic energy
ensemble->scaleVelocity();
//scale velocities to coincide with energy
ensemble->setKineticE();
//Re-determine KE with scaled velocities
```

```
cout << "Results will be directed to the
       file \"md_nve.out\"" << endl;

for(i = 0; i < nStep; i++)
{
  integrator->gearPredict(num, length,
              tStep);
  ensemble->setForce();
  ensemble->setKineticE();
  kinE = integrator->
    gearCorrect(num, length, tStep);
  ensemble->setKineticE(kinE);

  if(((i % 10) == 0) && i < nEquil)
       ensemble->scaleVelocity(tStep);

  if(i >= nEquil)
  {
    potE = ensemble->getPotEnergy() + lrc;
    kinE = ensemble->getKineticE();
    virial = ensemble->getVirial() + lrcW;
    totE = potE + kinE;
    temp = 2.0*kinE/(3.0*num -3);
    avPotEnergy += potE;
    avKinEnergy += kinE;
    avTotEnergy += totE;
    avVirial    += virial;
    avTemperature += temp;

    if((i !=nEquil) && ((i % nSize)==0))
    {
      accumPotE[j] /= nSize;
      accumKinE[j] /= nSize;
      accumTotE[j] /= nSize;
      accumVir[j]  /= nSize;
      accumTemp[j] /= nSize;
     j++;
    }
    accumPotE[j] += potE;
    accumKinE[j] += kinE;
    accumTotE[j] += totE;
    accumVir[j]  += virial;
    accumTemp[j] += temp;
  }
}

avPotEnergy   /= nTotal;
avKinEnergy   /= nTotal;
avTotEnergy   /= nTotal;
avVirial      /= nTotal;
avTemperature /= nTotal;
```

Start of simulation using Gear PC.

Velocity scaling at every 10 time steps to keep energy constant.

Accumulation of averages after equilibration period.

Calculation of ensemble averages and standard deviations.

```
 for(i = 0; i < n; i++)
  {
    erPotEnergy +=
        pow((accumPotE[i] - avPotEnergy),2);
    erKinEnergy +=
        pow((accumKinE[i] - avKinEnergy),2);
    erTotEnergy +=
        pow((accumTotE[i] - avTotEnergy),2);
    erVirial    +=
         pow((accumVir[i] - avVirial),2);
    erTemperature +=
        pow((accumTemp[i] - avTemperature),2);
  }

  pressure = avVirial/volume +
             num*ensemble->getTemp()/volume;
  erPress  = erVirial/volume;
```

Printing out ensemble averages at the end of the simulation.

```
  out <<"Average Potential Energy/N:\t"
      << avPotEnergy/num  <<"  +/-
      "<<sqrt(erPotEnergy)/(nTotal*num)<<endl;
  out <<"Average Kinetic Energy/N:\t "
      << avKinEnergy/num <<"  +/-
      "<<sqrt(erKinEnergy)/(nTotal*num)<<endl;
  out <<"Average Total Energy/N:\t\t"
      << avTotEnergy/num <<"  +/-
      "<<sqrt(erTotEnergy)/(nTotal*num)<<endl;
  out <<"Average Pressure:\t\t " << pressure
       <<"  +/- "<<sqrt(erPress)/nTotal<<endl;
  out <<"Temperature:\t\t\t " << avTemperature
       <<"  +/-
       "<<sqrt(erTemperature)/nTotal<<endl;
  out.close();
}
```

The readInNVE() method is called by the constructor of the MolecularDynamics object.

```
// Method: readInNVE
// Usage:  readInNVE();
// ____________________
// ReadIn reads-in the NVE ensemble settings
// from the NVEfile.dat data file.

void MolecularDynamics::readInNVE
     (int interPotential)
```

Some of the input values depend on the potential. In this case only the LJ has been implemented.

```
{
  int i, j, dimensions = 3;
  int, numComp, numAtom, *comp;
  double numType, *molFract;
  double temperature, density, *mass;
```

References to the Atom class.

```
  AtomMD *newAtom, **atoms;

  //open the NVEfile.dat file
```

Check for successful file opening.

```
  ifstream in;
  in.open("NVEfileMD.dat");
```

```
if(in.fail())
{
  cout << "Cannot open NVEFile.dat!\n";
  return;
}
```

Check that there is at least one component!

```
in >> numComp;
if(numComp <= 0)
{
  cout << "Number of components must
       be > 0!\n";
  return;
 }

comp = new int[numComp];
```

Check that there are at least four atoms to build lattice.

```
in >> numAtom;
if(numAtom < 4)
{
  cout << "At least 4 atoms are
  required!\n";
  return;
}
```

The following code continues to read-in data and performs various bookkeeping tasks.

```
if(!(molFract = new double [numComp]))
{
  cout << "Cannot allocate memory
  to molFract" << endl;
  return;
}

for(i = 0; i < numComp; i++)
    in >> molFract[i];

mass = new double[numComp];

for(i = 0; i < numComp; i++)
    in >> mass[i];

in >> temperature;
in >> density;
in.close();
```

This can be expanded for other potentials.

```
// construct array of atoms
if(potential == 1) // Lennard-Jones
{
  in.open("paramLJ.dat");
  if(in.fail())
  {
    cout<<"Cannot open paramLJ.dat!\n";
    return;
  }

  // assign memory to arrays
  if(!(atoms = new Atom * [numAtom]))
```

```
{
  cout<<"Cannot allocate memory to
       atoms!\n";
  return;
}

epsilon = getMemory(numComp);
sigma = getMemory(numComp);
rCut = getMemory(numComp);

for (i = 0; i < numComp; i++)
{
  for (j = 0; j < numComp; j++)
  {
     in>>epsilon[i][j]>> sigma[i][j]
          >> rCut[i][j];

     //adjust for indistinguishable pairs
     if (j!=i)
     {
       epsilon[j][i] = epsilon[i][j];
       sigma[j][i] = sigma[i][j];
       rCut[j][i] = rCut[i][j];
     }
   }
 }


 // close input file
 in.close();

 int *atnum = new int[numComp + 1];
 atnum[0] = 0;
 //calculate the number of atoms of each
 //type from the molFract

 for(i = 0; i < numComp; i++)
 {
   numType = numAtom * molFract[i];
   int rem = (int) numType;

   if ((numType - rem) >= .5)
     atnum[i+1] = (int) numType + 1;
   else
     atnum[i+1] = (int) numType;
  }
  int sum = 0;
  for (i = 0; i <= numComp; i++)
     sum += atnum[i];

  while (atnum[numComp] < numAtom)
    atnum[numComp]++;

  for(i = 0; i < numComp; i++)
    atnum[i+1] += atnum[i];
```

This section of code creates objects of type Ljatom, which is accessible via the atoms array. The properties of ljatom are passed to the constructor

```
      //loop through the number of components
      for (i = 0; i < numComp; i++)
      {
        //loop through the number of atoms
        //of that type
       for(int j=atnum[i];j<atnum[i+1];j++)
       {
         //create atom(s) and assign mass,
         //type, sigma, epsilon, and rCut
         //(derivatives-2) because
         //acceleration and velocity are
         //stored separately
        newAtom =
            new Ljatom(i, mass[i], epsilon,
            sigma, rCut, dimensions,
            (derivatives-2));

          //store reference to atom in array
          atoms[j] = newAtom;
       }
      }
  } // end of LJ atom array creation
```

A pointer to the NVE ensemble. The properties of the ensemble are passed to the constructor.

```
  ensemble =
   new NVEensemble(atoms, numComp,
                   numAtom, temperature,
                    density, molFract, comp);
  return;
}
```

Get size of block averages.

```
// Method: getnSize()
// Usage:  n = getnSize();
// ----------------------
// Gets the size of the blocks to be averaged.

int MolecularDynamics::getnSize()
{
  return nSize;
}
```

Get number of equilibration steps.

```
// Method: getnEquil()
// Usage:  n = getnequil();
// ----------------------
// Gets the equilibration period.

int MolecularDynamics::getnEquil()
{
  return nEquil;
}
```

Get the total number of steps in the simulation.

```
// Method: getnStep()
// Usage:  n = getnStep();
// ----------------------
// Gets the total number of simulation steps.

int MolecularDynamics::getnStep()
{
  return nStep;
}
```

Get the time step.

```
// Method: gettStep()
// Usage:  n = gettStep();
// ----------------------
// Gets the size of the time step.

int MolecularDynamics::gettStep()
{
  return tStep;
}
```

11.2.3.4 Code for the ensemble class

md_Ver2.0\ensembleMD.cpp

```
// File: ensembleMD.cpp
// ____________________
// File containing functions to implement the
// Abstract EnsembleMD class.

#include "ensembleMD.h"
#include <math.h>

// Method: EnsembleMD
// Usage:  EnsembleMD;
// -------------------
// Constructor for the EnsembleMD class
```

The ensemble constructor is called during from the read-in function during the construction of the md object. The properties of the Ensemble are initialised by the constructor.

```
EnsembleMD::EnsembleMD(Atom **theAtoms,
   int nComp, int nAtoms, double temp,
   double dens, double *mol, int *cmp)
{
  atoms = theAtoms;
  numComp = nComp;
  numAtom = nAtoms;
  temperature = temp;
  density = dens;
  molFract = mol;
  comp = cmp;
  potEnergy = 0.0;
}

// Destructor for EnsembleMD class
```

```
EnsembleMD::~EnsembleMD(){}

// setMethods
// ----------
// These methods assign values to the volume,
// length, and component number for the
// ensemble.  They do not take any parameters,
// and are called during the construction
// phase of the ensemble
```

Calculate volume of simulation box

```
// Method: setVolume
// Usage:  setVolume();
// ----------------_
// Determines the volume of the simulation
// box.

void EnsembleMD::setVolume()
{
  boxVol = numAtom/density;
}
```

Length of simulation box.

```
// Method: setLength
// Usage:  setLength();
// -------------------
// Determines the length of the simulation
// box.

void EnsembleMD::setLength()
{
  boxLen = pow(boxVol, 1.0/3.0);
}
```

Number of components of each atom.

```
// Method: setComp
// Usage:  setComp();
// -----------------
// Determines the number of atoms of each
// type.

void EnsembleMD::setComp()
{
  int i, sum = 0;

  for(i =0; i < numComp - 1; i++)
  {
    comp[i] = (int)(molFract[i] * numAtom);
    sum += comp[i];
  }
  comp[numComp - 1] = numAtom - sum;
}

// getMethods
// ----------
// These methods return the number of atoms,
// an array describing the number of each
```

```
// atom of each type, the number of
// components, the temperature of the
// ensemble, the volume of the ensemble box,
// the length of the ensemble box, the
// potential energy and the kinetic energy.
```

Get the number of atoms in the ensemble.

```
// Method: getNumAtom
// Usage:  n = getNumAtom();
// ------------------------
// Gets the total number of molecules.

int EnsembleMD::getNumAtom()
{
  return numAtom;
}
```

Get the temperature.

```
// Method: getTemp
// Usage:  getTemp();
// ------------------
// Gets the temperature of the ensemble.

double EnsembleMD::getTemp()
{
  return temperature;
}
```

Get the number of components.

```
// Method: getNumComp
// Usage:  n = getNumComp();
// -------------------------
// Gets the number of components.

int EnsembleMD::getNumComp()
{
  return numComp;
}
```

Get the number of each component.

```
// Method: getComp
// Usage:  p = getComp();
// ----------------------
// Gets the number of each individual comp.

int * EnsembleMD::getComp()
{
  return comp;
}
```

Get the volume.

```
// Method: getVolume
// Usage:  getVolume();
// --------------------
// Gets the volume of the simulation box.

double EnsembleMD::getVolume()
{
  return boxVol;
}
```

Get the box length.

```
// Method: getLength
// Usage:  getLength();
// --------------------
// Gets the length of the simulation box.

double EnsembleMD::getLength()
{
  return boxLen;
}
```

Get the potential energy.

```
// Method: getPotEnergy
// Usage: getPotEnergy();
// ----------------------
// Gets potential energy following force cal.
double EnsembleMD::getPotEnergy()
{
  return potEnergy;
}
```

Get the virail

```
// Method: getVirial
// Usage: getVirial();
// -------------------
// Gets virial following force calculation.

double Ensemble::getVirial()
{
  return virial;
}
```

Get the kinetic energy.

```
// Method: getKineticE
// Usage   n = getKineticE();
// --------------------------
// Gets the kinetic energy.

double EnsembleMD::getKineticE()
{
  return kineticE;
}
```

Get the array of Atom objects.

```
// Method: getVirial
// Usage: getVirial();
// -------------------
// Gets virial following force calculation.

double Ensemble::getVirial()
{
  return virial;
}
```

Get atoms

```
// Method: getAtoms
// Usage:  n = getAtoms();
// -----------------------
// Return array of atoms
```

```
AtomMD **EnsembleMD::getAtoms()
{
  return atoms;
}
```

Set the initial accelation.

```
// Method:  initAcceleration
// Usage:   initAcceleration();
// ---------------------------
// Performs initialisation on the
// accelerations of all the atoms
// in the ensemble prior to starting the
// simulation. It requires the force values to
// have been initialised.

void  EnsembleMD::initAcceleration()
{
  double *accel, *force, mass;
  accel = new double [3];

  for(int i = 0; i < numAtom; i++)
  {
    force = atoms[i]->getForce();
    mass = atoms[i]->getMass();
    for (int j = 0; j < 3; j++)
     atoms[i]->setAcceleration(&accel[0]);
  }
}
```

11.2.3.5 Code for NVE_Ensemble class

md_Ver2.0\nveMD.cpp

```
// File: nveMD.cpp
// ---------------
// File containing methods to implement the
// NVEensembleMD class.

#include "nveMD.h"
#include "auxfunc.h"
#include "ljforceMD.h"
#include <iostream>
#include <math.h>

using namespace std;
```

Create NVEensemble.

```
// Method: NVEensembleMD
// Usage:  NVEensembleMD;
// ----------------------
// Instantiate the NVEensembleMD class

NVEensemble::NVEensemble(Atom **atoms,
 int numComp, int numAtom, double temperature,
 double density, double *molFract, int *comp)
: Ensemble(atoms, numComp, numAtom,
temperature, density, molFract, comp)
{
  setVolume();
  setLength();
  setComp();
  initialCoord();
  theForce = new LJforceMD(atoms);
}
```

Set the kinetic energy.

```
// Method: setKineticE
// Usage:  setKineticE();
// ----------------------
// Determines the kinetic energy.

void NVEensembleMD::setKineticE()
{
  int i;
  double sum = 0.0, *velocity;
  for(i = 0; i < numAtom; i++)
  {
```

The sum of all components of velocity.

```
    velocity = atoms[i]->getVelocity();
    sum += atoms[i]->getMass() *
          (velocity[0] * velocity[0] +
           velocity[1] * velocity[1] +
           velocity[2] * velocity[2]);
  }
  kineticE = sum/2;
}
```

Scale the velocities.

```
// Method: scaleVelocity
// Usage:  scaleVelocity();
// ---------------------------
// Scales the velocities during the
// equilibration phase of the simulation. The
// scaling ensures that the simulation
// is performed at the specified energy.

void NVEensemble::scaleVelocity()
{
  int i;
  double tScale, mass, kinE, temp,
  sumVSq = 0.0, *velocity;

  kinE = getKineticE();
  temp = 2*kinE/(3*numAtom - 3);

  tScale = sqrt(getTemp()/temp);

  // scale velocities to reproduce the desired
  // temperature (temp) and acumulate sum
  // for momentum adjustment

  for(i = 0; i < numAtom; i++)
  {
    velocity = atoms[i]->getVelocity();

    velocity[0] *= tScale;
    velocity[1] *= tScale;
    velocity[2] *= tScale;

    atoms[i]->setVelocity(velocity);
  }
}
```

The array indexes, 0, 1,2 signify the x,y and z components of velocity, respectively.

Set the initial velocities.

```
// Method: initialVelocity
// Usage:  initialVelocity();
// --------------------------
// Invokes a random number generator to assign
// initial velocities to molecules at the
// start of a molecular dynamics simulation.

void NVEensemble::initialVelocity()
 {
  int seed; //seed for random number generator
  int i;
  double vX, vY, vZ, vSq, rValue;
  double sumX = 0.0, sumY = 0.0;
  double sumZ = 0.0;
  double *velocity;

 // assign random velocities to molecules in
 // the range of 1 to - 1

  seed = -29;
```

```
  for(i = 0; i < numAtom; i++)
  {
    vX = random(&seed);
    vY = random(&seed);
    vZ = random(&seed);

    vSq = vX * vX + vY * vY + vZ * vZ;

    rValue = sqrt(vSq);

    vX /= rValue;
    vY /= rValue;
    vZ /= rValue;

    sumX += vX;
    sumY += vY;
    sumZ += vZ;

    velocity = new double[3];
    //create new array with 3 elements
    velocity[0] = vX;
    velocity[1] = vY;
    velocity[2] = vZ;
    atoms[i]->setVelocity(velocity);
    //adjust velocity of atom i
  }

  //adjust velocities so that the total linear
  //momentum is zero
  for(i = 0; i < numAtom; i++)

  for(i = 0; i < numAtom; i++)
  {
    velocity = atoms[i]->getVelocity();
    velocity[0] -= sumX/numAtom;
    velocity[1] -= sumY/numAtom;
    velocity[2] -= sumZ/numAtom;
    atoms[i]->setVelocity(velocity);
  }
}
```

Get the long range corrections to the potential energy.

```
// Method: getEnergyLRC
// Usage:  n = getEnergyLRC();
// ---------------------------
// Gets the long range energy correction.

double NVEensembleMD::getEnergyLRC()
{
  return energyLRC;
}
```

Get virial LRC

```
// Method: getVirialLRC
// Usage: n = getVirialLRC();
// -------------------------
// Gets the long range virial correction.

double NVEensemble::getVirialLRC()
{
  return virialLRC;
}
```

Set the initial coordinates by placing the atoms on a face-centred cubic lattice.

```
void NVEensemble::initialCoord()
{
  int i, j, x, y, z, offset;
  double cells, dcells, cellL, halfCellL,
  *tempPos, *position;
  double (*holdPos)[3] = new double
         [numAtom][3];
  tempPos= new double [numAtom];
  position = new double [numAtom];

  //Determine the number of unit cells in
  //each coordinate direction

  dcells = pow(0.25*numAtom, 1.0/3);
  cells =  (int) nearestInt(dcells, 1.0);

  //check if numAtom is an non-fcc number
  //and increase the number of cells
  while((4 * cells * cells * cells) < numAtom)
    cells = cells + 1;

  //Determine length of the unit cell
  cellL = boxLen/ (double) cells;
  halfCellL = cellL/2;

  //Construct the unit cell
  //point to atoms position
```

Assign x,y and z coordinates for the four atoms in the unit cell.

```
  position[0] = 0.0;
  position[1] = 0.0;
  position[2]= 0.0;
  atoms[0]->setPosition(position);

  position[0] = halfCellL;
  position[1] = halfCellL;
  position[2]= 0.0;
  atoms[1]->setPosition(position);

  position[0] = 0.0;
  position[1] = halfCellL;
  position[2]= halfCellL;
  atoms[2]->setPosition(position);

  position[0] = halfCellL;
  position[1] = 0.0;
  position[2]= halfCellL;
```

Assign the remaining positions on the lattice.

```
  atoms[3]->setPosition(position);

  for(i = 4; i < numAtom; i++)
  {
    position = atoms[i]->getPosition();
    position[0] = 0.0;
    position[1] = 0.0;
    position[2] = 0.0;
    atoms[i]->setPosition(position);

  } //init all other atoms to 0

  //Build the lattice from the unit cell by
  //repeatly translating the four vectors of
  //the unit cell through a distance cellL in
  //the x, y and z directions
  offset = 0;
  for (z = 1; z <= cells; z++)
    for (y = 1; y <= cells; y++)
      for (x = 1; x <= cells; x++)
      {
        for (i = 0; i < 4; i++)
        {
          j = i + offset;
          if(j < numAtom)
          {
            position =
              atoms[i]->getPosition();
            holdPos[j][0] = position[0]
               + cellL * (x-1);
            holdPos[j][1] = position[1]
               + cellL * (y-1);
            holdPos[j][2] = position[2]
               + cellL * (z-1);
          }
        }

        offset = offset + 4;
      }

  //Shift centre of box to the origin.
  for (i = 0; i < numAtom; i++)
  {
    tempPos = atoms[i]->getPosition();
    tempPos[0] = holdPos[i][0] - halfCellL;
    tempPos[1] = holdPos[i][1] - halfCellL;
    tempPos[2] = holdPos[i][2] - halfCellL;
    atoms[i]->setPosition(tempPos);
  }

}

// Method: setforce
// Usage:  setForce();
//  ------------------
```

Set the force.

```
// Set the force calculations using Ljforce

void NVEensemble::setForce()
{
  theForce->setForce(numAtom, boxLen,
                     &potEnergy, &virial);
}
```

Get the long range corrections.

```
// Method: lrc
// Usage:  lrc();
// --------------
// Calculations for long range corrections

void NVEensemble::lrc()
{
  theForce->lrc(numComp, comp, boxVol,
                &energyLRC, &virialLRC);
}
```

11.2.3.6 Code for the Atom class

md_Ver2.0\atomMD.cpp

```
// File: atomMD.cpp
// --------------
//  File containing the methods to implement
//  the atom class.

#include "atomMD.h"
#include "auxfunc.h"
#include <iostream>
#include <fstream>
#include <math.h>
```

This constructor is used to build the atom class using the parameters described in the parameter list.

```
// Constructor:Atom
// Usage: Atom atom;
// -------------------
// Builds the Atom class

Atom::Atom(int theType, double theMass, int
dimensions, int derivatives)
{
  int i;
  mass = theMass;
  type = theType;

  //allocate memory for the arrays
  position = new double[dimensions];
  velocity = new double[dimensions];
```

```
  acceleration = new double[dimensions];
  force = new double[dimensions];
  highTimeDerivs =
          getMemory(derivatives, dimensions);

  //initialise values for acceleration
  //and higher time derivatives
  for(i = 0; i < dimensions; i++)
    acceleration[i] = 0.0;

  for(i = 0; i < 3; i++)
    for(int j = 0; j < dimensions; j++)
      highTimeDerivs[i][j] = 0.0;

}
```

This initialization of acceleration is a safeguard only because acceleration will be re-assigned in the initialisation phase of the simulation.
i = 0,1,2 represent the 3rd,4th and 5th time derivatives

```
//2nd constructor for array reference
//construction
AtomMD::AtomMD(){}


// Method:  setType
// Usage:   setType(atomType);
// --------------------------
// Used to identify the atom as belonging to
// a certain type of component within the
// ensemble.

void Atom::setType(int t)
{
  type = t;
}

// Method: setrCutOff
// Usage:  setrCutOff(atomic_rCutOffValue);
// ---------------------------------------
// Used to assign the values for the rCutOffs
// for the various atom types that make up the
// ensemble

void Atom::setrCutOff(double **newrCut)
{
  rCutOff = newrCut;
}

// Method:  setMass
// Usage:   setMass(double atomicMass);
// -----------------------------------
// Assign the mass of the atom

void Atom::setMass(double newMass)
{
  mass = newMass;
}
```

This constructor is used to create an array reference for the objects created by the first constructor.

The purpose and behavior of the remaining methods for this class are explained in detail in the comments.

Set the cut-off.

Set the masses.

Set the acceleration.

```
// Method:  setAcceleration
// Usage:   setAcceleration(acceleration);
//  ------------------------------------
// Sets the pointer within the atom object to
// point to a new 1D array of doubles,
// representing the acceleration of the atom
// in its various dimensions.  Assuming a 3D
// simulation, the required array would have
// three elements, where:
// atomicAcceleration[0] is the x component
// atomicAcceleration[1] is the y component
// atomicAcceleration[2] is the z component
// Note:  this method requires an _array_ to
// be passed to it which has already had
// memory allocated to it.

void Atom::setAcceleration(double *newAccel)
{
  for(int i = 0; i < 3; i++)
    acceleration[i] = newAccel[i];
}
```

Set the positions.

```
// Method:  setPosition
// Usage:   setPosition(position);
// ------------------------------
// Sets the pointer within the atom object to
// point to a new 1D array of doubles,
// representing the position of the atom in
// its various dimensions.  Assuming a 3D
// simulation, the required array would have
// three elements, where:
// atomicPosition[0] = x component of position
// atomicPosition[1] = y component of position
// atomicPosition[2] = z component of position
// Note:  this method requires an _array_ to
// be passed to it which has already had
// memory allocated to it.


void Atom::setPosition(double *newPos)
{
  for(int i = 0; i < 3; i++)
    position[i] = newPos[i];
}
```

Set the velocities.

```
// Method:  setVelocity
// Usage:   setVelocity(velocity);
// ------------------------------
// Sets the pointer within the atom object to
// point to a new 1D array of doubles,
// representing the velocity of the atom in
// its various dimensions.  Assuming a 3D
// simulation, the required array would have
// three elements, where:
// atomicVelocity[0] = x component of velocity
```

```
// atomicVelocity[1] = y component of velocity
// atomicVelocity[2] = z component of velocity
// Note:  this method requires an _array_ to
// be passed to it which has already had
// memory allocated to it.

void Atom::setVelocity(double *newV)
{
 for(int i = 0; i < 3; i++)
    velocity[i] = newV[i];
}
```

Set higher derivatives of distance w.r.t. time.

```
// Method:  setHighTimeDerivs
// Usage:   setHighTimeDerivs(timeDerivative);
// ----------------------------------------
// Sets the pointer within the atom object to
// point to a new 2D array of doubles,
// representing the derivatives of time after
// acceleration and velocity fro that atom,
// in its various dimensions.  Assuming a 3D
// simulation, the required array would be 2D
// where: the first dimension represents the
// derivative of time, and the second
// dimension represents the dimension in
// space.
// For example, timeDerivative[a][b] refers to
// the derivative of time for:
// - derivative a, where a is 0 for the third
// derivative of time, 1 is the fourth
// derivative of time, and 2 is the fifth, and
//  - dimension b, where b is 0 for the x
// component, 1 for the y component, and 2 for
// the z component
// Note:  this method requires an _array_ to
// be passed to it which has already had
// memory allocated to it.

void Atom::setHighTimeDerivs(double
          **newHighTimeDerivs)
{
 for(int i = 0; i < 3; i++)
   for(int j = 0; j < 3; j++)
     highTimeDerivs[i][j] =
               newHighTimeDerivs[i][j];
}
```

Set the force.

```
// Method:  setForce
// Usage:   setForce(atomicForce);
// ------------------------------
// Sets the pointer within the atom object to
// point to a new 1D array of doubles,
// representing the force acting on the atom
// in its various dimensions.  Assuming a 3D
// simulation, the required array would have
// three elements, where:
```

```
// atomicForce[0] = x component of force
// atomicForce[1] = y component of force
// atomicForce[2] = z component of force
// Note:  this method requires an _array_ to
// be passed to it which has already had
// memory allocated to it.

void Atom::setForce(double *newForce)
{
    for(int i = 0; i < 3; i++)
     force[i] = newForce[i];
}

// Alternate version as required

void Atom::setForce(double fX, double fY,
   double fZ)
{
  force[0] = fX;
  force[1] = fY;
  force[2] = fZ;
}
```

Get the type of atom.

```
// Access (Get) Methods
// Usage:  n = getXxxx();
// ----------------------
// These methods return the _value_ stored for
// the mass or type of the atom.

// Method: getType
// Usage:  p = getType();
// ----------------------
// Gets the type associated with each atom.

int Atom::getType()
{
  return type;
}
```

Get atomic masses.

```
//  Method: getMass
//  Usage: n = getMass();
//  ---------------------
//  Get values of the atomic masses.

double Atom::getMass()
{
  return mass;
}
/


/ Access (Get) Methods
// Usage:  n = getXxxxx();
// ----------------------
// These methods return a reference to the
```

```
// arrays which hold the values for rCutOff,
// acceleration, position, velocity, time
// step, and force, as described above their
// respective set method.
```

Get cut-off values.

```
// Method: getrCutOff
// Usage: n = getrCutOff();
// --------------------
// Returns the cut off distances for atom
// pairs

double **Atom::getrCutOff()
{
  return rCutOff;
}
```

Get accelerations.

```
// Method: getAcceleration
// Usage: n = getAcceleration();
// ----------------------------
// return acceleration of atom

double *Atom::getAcceleration()
{
  return acceleration;
}
```

Get positions.

```
// Method: getPosition
// Usage:  n = getPosition();
// -------------------------
// Return position of atom

double *Atom::getPosition()
{
  return position;
}
```

Get velocities.

```
// Method: getVelocity
// Usage:  n = getVelosity();
// -------------------------
// Return the velocity of the atom

double *Atom::getVelocity()
{
  return velocity;
}
```

Get higher derivatives of distance w.r.t. time.

```
// Method: getHighTimeDerivs
// Usage:  n = getHighTimeDerivs();
// -------------------------------
// Return the derivatives of time

double **Atom::getHighTimeDerivs()
{
  return highTimeDerivs;
}
```

Get forces.

```
// Method: getForce
// Usage:  n = getForce();
// ----------------------
// Returns reference to the force array in the
// atom object

double *AtomM::getForce()
{
  return force;
 }
```

11.2.3.7 Code for LJ_Atom class

md_Ver2.0\ljatomMD.cpp

```
// File: ljatom.cpp
// ---------------
// File containing the functions to implement
// the LJatomMD class.

#include "ljatomMD.h"

// Constructor: LJatom
// Usage: LJatom atom;
// -------------------
// Builds the LJatomMD class
```

This constructor is used to build the LJatom object.

```
LJatom::LJatom
(int theType, double theMass, double **ep,
 double **sig, double **rC, int dimensions,
 int derivatives)
 :Atom(theType, theMass, dimensions,
       derivatives)
{
  epsilon = ep;
  sigma = sig;
  rCutOff = rC;
}
```

This constructor is used to reference the objects built by the first constructor.

```
// Second constructor
LJatom::LJatom(){}
```

Set methods

```
// Methods:  setSigma and setEpsilon
// Usage:    setSigma(sigma);
//           setEpsilon(epsilon);
// ---------------------------------------
// These methods accept a reference to a 2D
// array of doubles, representing the sigma
// and epsilon values for the interactions
// between each atomic pair type.
// Each dimension will be exactly the number
```

```
// of atom types in the ensemble, where
// sigma[0][0] is the sigma value for
// interactions between an atom of type 0 and
// an atom of type 0, sigma[0][1] is the sigma
// value for interactions between an atom of
// type 0 and an atom of type 1, etc and
// likewise for epsilon.
```

Set LJ epsilon value.

```
// Method: setEpsilon
// Usage:  setEpsilon(epsilon);
// ----------------------------
// Assign values of LJ epsilon parameter

void LJatom::setEpsilon(double **newEpsilon)
{
  epsilon = newEpsilon;
}
```

Set LJ sigma value.

```
// Method: setSigma
// Usage:  setSigma(sigma);
// ------------------------
// Set the LJ parameter Sigma

void LJatom::setSigma(double **newSigma)
{
  sigma = newSigma;
}
```

Get methods

```
// Methods:  getEpsilon and getSigma
// Usage:    n = getEpsilon(); n = getSigma();
// ------------------------------------------
// These methods return references to the
// arrays which store the values for sigma and
// epsilon for that atom.
```

Get LJ epsilon.

```
// Method: getEpsilon
// Usage:  n = getEpsilon();
// -------------------------
// Get values of the Lennard-Jones epsilons.

double **Ljatom::getEpsilon()
{
  return epsilon;
}
```

Get LJ sigma.

```
// Method: getSigma
// Usage:  n = getSigma();
// -----------------------
// Get values of the Lennard-Jones sigmas.

double **LJatom::getSigma()
{
  return sigma;
}
```

11.2.3.8 Code for the Force class

md_Ver2.0\forceMD.cpp

The code for the abstract Force class is very simple because the abstract methods are defined in the class (in this case LJforce) which inherits from it.

```
// File: forceMD.cpp
// -----------------
// File containing functions to implement
// the abstract ForceMD class.

#include "forceMD.h"

//  Constructor
//  Method:  ForceMD
//  Usage:   n = ForceMD(atoms);
//  -------------------------
//  Used to construct the force component of
//  any derived force classes

Force::Force(Atom **theAtoms)
{
  atoms = theAtoms;
}

// destructor
ForceMD::~ForceMD(){}
```

11.2.3.9 Code for the LJ_Force class

md_Ver2.0\ljforceMD.cpp

```
// File: ljforceMD.cpp
// -----------------
// File containing functions to implement the
// LJforceMD class.

#include "ljforceMD.h"
#include "auxfuncMD.h"
#include <iostream>
#include <fstream>

using namespace std;

// constructor
 LJforce::LJforce(Atom **theAtoms)
          :ForceMD(theAtoms)
{
  epsilon = atoms[0]->getEpsilon();
  sigma   = atoms[0]->getSigma();
  rCut    = atoms[0]->getrCutOff();
}
```

The setForce method is the heart of the MD simulation and it determines largely the overall speed of the calculation.

```
// Method: setForce
// Usage:  setForce();
// -------------------
// The ljForce function calculates the force
// experienced by all num atoms due to
// pairwise interatomic interaction.
// The forces are calculated using the
// Lennard-Jones (6-12) potential. The
// potential is truncated at a distance
// rCut and long range corrections must be
// applied outside the method to obtain the
// full contributions to the energy.

void LJforce::setForce(int num, double length,
          double *potEnergy, double *virial)
{
  int i, j, kindi, kindj;
  double rXi, rYi, rZi;
  // position vectors for atom i
  double rXj, rYj, rZj;
  // position vectors for atom j
  double rXij, rYij, rZij;
  // pair seperation vectors
  double rijSq;
  // interatomic seperation squared
  double fXi, fYi, fZi;
  // force vectors for atom i
  double fij, fXij, fYij, fZij;
  // force between atoms i and j
```

```
double rCutSq, sigmaSq;
// squared cut and sigma values
double sigma2, sigma6, sigma12;
// multiples of LJ sigma
double pot;  // Potential between 2 atoms
double *tempPos;       // postion vectors
double *tempPosj;      // postion vector
double potECal, virialCal;
double *fX = new double [num];
double *fY = new double [num];
double *fZ = new double [num];

// Zero the force  arrays.
for(i = 0; i < num; i++)
 {
   fX[i] = 0.0;
   fY[i] = 0.0;
   fZ[i] = 0.0;
 }

potECal = 0.0;
virialCal = 0.0;


//Calculate forces experienced by atoms due
//interatomic pair interactions.

for(i = 0; i < num; i++)
{
  //Select molecue i
  kindi = atoms[i]->getType();
  tempPos = atoms[i]->getPosition();

  rXi   = tempPos[0];
  rYi   = tempPos[1];
  rZi   = tempPos[2];

  fXi = fX[i];
  fYi = fY[i];
  fZi = fZ[i];

  for(j = i + 1; j < num; j++)
  {
     kindj = atoms[j]->getType();
     tempPosj = atoms[j]->getPosition();

     rXj   = tempPosj[0];
     rYj   = tempPosj[1];
     rZj   = tempPosj[2];

     //Calculate pair seperation
     rXij = rXi - rXj;
     rYij = rYi - rYj;
     rZij = rZi - rZj;
```

These method calls are simple and should be quick. The speed may be enhanced by explicit inlining.

Pair separation.

Minimum image convention.

It may be advantageous to inline the nearestInt method.

Only calculate forces below the cut-off distance.

Energy is calculated during the force evaluation at very little addition computational cost.

Application of Newton's third law saves computing time.

```
      //Apply periodic boundary
      rXij -= length * nearestInt(rXij,
           length);
      rYij -= length * nearestInt(rYij,
           length);
      rZij -= length * nearestInt(rZij,
           length);

      rijSq = rXij * rXij + rYij * rYij
            + rZij * rZij;
      rCutSq = rCut[kindi][kindj] *
               rCut[kindi][kindj];

      //Calculate forces for seprations
      //below the cutoff

      if (rijSq <= rCutSq)
      {
        //Determine i-j interactions
        sigmaSq = sigma[kindi][kindj]
                * sigma[kindi][kindj];
        sigma2  = sigmaSq/rijSq;
        sigma6  = sigma2 * sigma2 * sigma2;
        sigma12 = sigma6 * sigma6;
        pot     = sigma12 - sigma6;
        potECal   += 4*epsilon[kindi][kindj]
                     *pot;
        virialCal += 24*epsilon[kindi][kindj]
                     *(pot + sigma12)/3;

        //Calculate forces between atoms
        fij  = 24*epsilon[kindi][kindj] *
               (pot + sigma12)/rijSq;
        fXij = fij * rXij;
        fYij = fij * rYij;
        fZij = fij * rZij;
        fXi += fXij;
        fYi += fYij;
        fZi += fZij;

        //Apply Newton's third law
        fX[j] -= fXij;
        fY[j] -= fYij;
        fZ[j] -= fZij;
      }
    }

   fX[i]  = fXi;
   fY[i]  = fYi;
   fZ[i]  = fZi;

}
```

```
  // Store forces

  for(i = 0; i < num; i++)
    atoms[i]->setForce(fX[i], fY[i], fZ[i]);

  *potEnergy = potECal;
  *virial    = virialCal;

  delete [] fX;
  delete [] fY;
  delete [] fZ;

}
```

Currently the long range correction is specific to the Lennard-Jones (12-6) potential.

```
// Method: lrc
// Usage:  n = lrc(num, nComp, volume,
//              energyLRC);
// ----------------------------------
// ljLRC calculates the long range correction
// terms to both the energy and virial for an
// ensemble interacting via the Lennard-Jones
// (12-6) potential.

void LJforce::lrc(int num, int *nComp,
     double boxV, double *energyLRC,
     double *virialLRC)
{
  int i, j;
  const double PI = 3.141592654;
  double sigmaSq, sig3, rCutSq, rCut3,
   sig3R, sig9R, den;

  *energyLRC = 0.0;
  *virialLRC = 0.0;

  for(i = 0; i < num; i++)
    for(j = 0; j < num; j++)
    {
      sigmaSq   = sigma[i][j] * sigma[i][j];
      rCutSq    = rCut[i][j] * rCut[i][j];
      sig3      = sigma[i][j] * sigmaSq;
      rCut3     = rCut[i][j] * rCutSq;
      sig3R     = sig3/rCut3;
      sig9R     = sig3R * sig3R * sig3R;
      den       = nComp[i] * nComp[j]
                  * sig3/boxV;
      *energyLRC += (8/9.0) * den * PI *
                  epsilon[i][j]
                    * (sig9R - 3 * sig3R);
      *virialLRC += (16/9.0) * den * PI *
                     epsilon[i][j]
                    * (2 * sig9R - 3 * sig3R);
   }
}
```

11.2.3.10 Code for the Integrator class

md_Ver2.0\integrtMD.cpp

The Integrator class is a container class for all derived integrator methods

```
//  File:  integrtMD.cpp
//  Class: Integrator

#include "integrtMD.h"

// Constructor
// -----------
IntegratorMD::Integrator(Atom **theAtoms)
{
  atoms = theAtoms;
 }

// Destructor
// ----------
Integrator::~Integrator(){}
```

11.2.3.11 Code for the Gear_Predictor–corrector class

md_Ver2.0\gpcMD.cpp

```
// File: gpc.cpp
// Class: GearPC

#include "gpcMD.h"
#include "gpcConstantsMD.h"
#include "auxfunc.h"
#include <iostream>

using namespace std;

// Constructor
// ------------

GearPC::GearPC(Atom **theAtoms)
      :Integrator(theAtoms)
{
 //construct superclass
}
```

5-value Gear predictor.

```
// Method: gearPredict
// Usage: gearPredict(num, length);
// -------------------------------
// The Method gearPredict uses Gear's 5-value
// algorithm to predict new coordinates,
// accelerations, velocities, third and fourth
// time derivatives of position.
```

```
void GearPC::gearPredict(int num, double
   length, double delT)
{
  int i, j;
  double *position, *velocity, *acceleration,
     **highTimeDerivs;
  double c1, c2, c3, c4, c5;

  c1 = delT;
  c2 = c1*delT/2;
  c3 = c2*delT/3;
  c4 = c3*delT/4;
  c5 = c4*delT/5;

  for(i = 0; i < num; i++)
  {
    position = atoms[i]->getPosition();
    velocity = atoms[i]->getVelocity();
    acceleration =
         atoms[i]->getAcceleration();
    highTimeDerivs =
         atoms[i]->getHighTimeDerivs();

    for(j = 0; j < 3; j++)
    {
     //components in the x, y and z directions
     position[j] += c1*velocity[j]
                  + c2*acceleration[j]
                  + c3*highTimeDerivs[0][j]
                  + c4*highTimeDerivs[1][j]
                  + c5*highTimeDerivs[2][j];

     position[j] -= length *
          nearestInt(position[j], length);
     velocity[j] += c1*acceleration[j]
                  +  c2*highTimeDerivs[0][j]
                  +  c3*highTimeDerivs[1][j]
                  +  c4*highTimeDerivs[2][j];
     acceleration[j]
                  += c1*highTimeDerivs[0][j]
                   +  c2*highTimeDerivs[1][j]
                   +  c3*highTimeDerivs[2][j];
     highTimeDerivs[0][j]
                   += c1*highTimeDerivs[1][j]
                   +  c2*highTimeDerivs[2][j];
     highTimeDerivs[1][j]
                   += c1*highTimeDerivs[2][j];
    }

   atoms[i]->setAcceleration(acceleration);
   atoms[i]->setVelocity(velocity);
   atoms[i]->setHighTimeDerivs(highTimeDerivs);
   atoms[i]->setPosition(position);
  }
}
```

Predict positions.

highTimeDerivs[a][b]: a is the 3rd, 4th and 5th and b denotes the x, y and z components,.

Apply PBC
Predict velocities.

Predict accelerations.

Predict 3rd derivatives.

Predict 4th derivatives.

Store new values.

Gear corrector method.

```
// Method: gearCorrect
// Usage:  gearCorrect
//         (num, type, length, time, mass);
// ---------------------------------------
// gearCorrect corrects the values of
// position, acceleration, velocity, third
// and forth time derivatives predicted
// by the 5-value Gear method.
// It is called following calls to
// gearPredict and getForce.

double GearPC::gearCorrect(int num, double
    length, double delT)
{

  int i, j;
  double aNew;      // new accelerations
  double adjust;    // corrections to apply
  double mass, sumVSq;
  double c1, c2, c3, c4, c5;
  double cr, cv, cb, cc, cd;
  double *force, *acceleration, *position;
  double *velocity, **highTimeDerivs;
```

Make use of the 'time' relationships

```
  c1 = delT;
  c2 = c1*delT/2;
  c3 = c2*delT/3;
  c4 = c3*delT/4;
  c5 = c4*delT/5;
```

Correction terms involving Gear's constants

```
  // Terms involving Gear's constants
  cr = G0*c2;
  cv = G1*c2/c1;
  cb = G3*c2/c3;
  cc = G4*c2/c4;
  cd = G5*c2/c5;

  sumVSq = 0.0;
```

Retrive values predicted in the first part of Gear's algorithm.

```
  for(i = 0; i < num; i++)
  {
    mass = atoms[i]->getMass();
    force = atoms[i]->getForce();
    acceleration =
      atoms[i]->getAcceleration();
    position = atoms[i]->getPosition();
    velocity = atoms[i]->getVelocity();
    highTimeDerivs =
      atoms[i]->getHighTimeDerivs();
```

Make the corrrections

Calculate adjustments.
Correct positions
Apply PBC

Correct velocity
Update acceleration
Correct 3rd derivative
Correct 4th derivatrive
Correct 5th derivative

Store corrections for future use

Calculate kinetic energy.

Apply corrections to positions.

Return kinetic energy

```
   //determine corrections
   for(j = 0; j < 3; j++)
   {
     aNew = force[j]/mass;
     adjust = aNew - acceleration[j];
     position[j] += cr*adjust;
     position[j] -= length *
        nearestInt(position[j], length);
     velocity[j] += cv*adjust;
     acceleration[j] = aNew;
     highTimeDerivs[0][j] += cb*adjust;
     highTimeDerivs[1][j] += cc*adjust;
     highTimeDerivs[2][j] += cd*adjust;
   }

   atoms[i]->setAcceleration(acceleration);
   atoms[i]->setHighTimeDerivs(highTimeDerivs);
   atoms[i]->setPosition(position);

   // It is computationally convient to
   // calculatre the kinetic energy too:

   sumVSq += mass*(velocity[0]*velocity[0]
           + velocity[1]*velocity[1]
           + velocity[2]*velocity[2]);
   atoms[i]->setVelocity(velocity);
 }

 return sumVSq/2;
}
```

11.2.3.12 Code for auxiliary methods

md_Ver2.0\auxfunc.cpp

```
#include "auxfunc.h"
#include <math.h>
#include <iostream>

using namespace std;

// Method: getMemory
// Usage: p = getMemory(num);
// --------------------------
//  Assigns memory to 2D arrays.

double **getMemory(int num)
{
  int i;
  double **mat;

  mat = new double *[num];

  for(i = 0; i < num; i++)
      mat[i] = new double [num];

  return mat;
}

// Method: getMemory
// Usage: p = getMemory(xNum, yNum);
// ---------------------------------
// Assigns memory to 2D arrays.

double **getMemory(int row, int col)
{
  int i;
  double **mat;

  mat = new double *[row];

  for(i = 0; i < row; i++)
     mat[i] = new double [col];

  return mat;
}
```

There are many instances where it is convenient to use 2-D arrays. The getMemory method allows us to allocate memory to 2-D arrays dynamically rather than pre-determining the array size.

A general method to assign memory to 2-D arrays with different row (xNum) and column (yNum) dimensions. In this case it is used specifically for the higher derivatives.

Methods to deallocate 2D array memory

```
// Method: freeMemory
// Usage: freeMemory(array);
// --------------------------------
// Free memory of 2-dimensional arrays;

void freeMemory(double **array)
{
 free(*array);
 free(array);
}
// Method: freeMemory
// Usage: freeMemory(array);
// --------------------------------
// Free memory of 2-dimensional arrays;

void freeMem(double **array, int row)
{
  int i;

  for(i = 0; i < row; i++)
    delete[] array[i];

      delete array;

}
```

The nearestInt method is useful in applying the minimum image convention to vectors. Alternatively, the round() function could be used as noted in Chapter 5.

```
// Method: nearestInt
// Usage: n = nearestInt(vector, boxL);
// ------------------------------------
// Return the next nearest integer
// resulting from the divison of vector
// by boxL.

int nearestInt(double vector, double
boxL)
{
  int nInt;
  double div;

  div = vector/boxL;

  if(vector >= 0)
    nInt = (int)(div + 0.5);
  else
    nInt = -(int)(0.5 - div);

  return nInt;
}
```

The random number generator is used to generate random numbers for initial velocities etc. It is also useful in MC applications.

```
// Method: random
// Usage: n = random(idum);
// ------------------------
// Random initally takes any negative
// number (idum) and returns a random
// on the interval 0 to 1. The
// algorithm is adapted from
// Press et al.,Numerical Receipies in C:
// The Art of Scientific Computing,
// Cambridge University Press,
// Cambridge, 1988.

double random(int *idum)
{

 // constants for random number generator

  const long int M1 = 259200;
  const int IA1 = 7141;
  const long int IC1 = 54773;
  const long int M2 = 134456;
  const int IA2 = 8121;
  const int IC2 = 28411;
  const long int M3 = 243000;
  const int IA3 = 4561;
  const long int IC3 = 51349;

  int j;
  static int iff = 0;
  static long int ix1, ix2, ix3;
  double RM1, RM2, temp;
  static double r[97];

  RM1 = 1.0/M1;
  RM2 = 1.0/M2;

  if(*idum < 0 || iff == 0){
    // initialise on first call
    iff = 1;
    ix1 = (IC1 - (*idum)) % M1;
    // seeding routines
    ix1 = (IA1 *ix1 + IC1) % M1;
    ix2 = ix1 % M2;
    ix1 = (IA1 * ix1 + IC1) % M1;
    ix3 = ix1 % M3;

    for (j = 0; j < 97; j++) {
        ix1 = (IA1 * ix1 + IC1) % M1;
        ix2 = (IA2 * ix2 + IC2) % M2;
        r[j] = (ix1 + ix2 * RM2) * RM1;
```

```
    }
    *idum = 1;
  }

  // generate next number for
  each sequence

  ix1 = (IA1 * ix1 + IC1) % M1;
  ix2 = (IA2 * ix2 + IC2) % M2;
  ix3 = (IA3 * ix3 + IC3) % M3;

  // randomly sample r vector

  j = 0 + ((96 * ix3) / M3);
  if (j > 96 || j < 0)
     cout <<"Error in generator\n";

   temp = r[j];
  // replace r[j] with next value
  // in the sequence

  r[j] = (ix1 + ix2 * RM2) * RM1;

  return temp;
}
```

11.3 Case study: application of object orientation to the microcanonical MC simulation of Lennard-Jones atoms

The next example is the application of object-oriented analysis, design, and programming to the MC simulation of Lennard-Jones atoms in the *NVE* ensemble. The analysis, design, and programming issues involved in a MC simulation have many similarities with MD. OO enables us to exploit these similarities to reuse existing code/design with either no modifications or relatively minor modifications. Consequently, our approach would be to exploit these similarities rather than starting entirely from scratch.

11.3.1 OOA and OOD

In common with the treatment of MD, our starting basis is a description of the problem underlining the nouns as potential classes.

> Monte Carlo simulation can be performed in a NVE ensemble in which the number of atoms, volume, and energy are held constant. The nature of the atoms and initial values of density and energy are specified. At the outset of the simulation, the atoms are placed randomly on a face-centered cubic lattice and the energy of interaction between pairs of atoms in the ensemble is calculated using the Lennard-Jones potential. The simulation proceeds by generating a Markov chain for a predetermined number of cycles (M_{cycle}). Each cycle

consists of an attempt to displace all the atoms randomly. The change in energy for the trial move is obtained by calculating the energy of the displaced atom via pair interaction with the atoms. The change in energy is used in conjunction with Ray's modified Metropolis criteria to either accept or reject the move. If the move is accepted, the energy of the ensemble and the position of the atom are updated. Both successful and unsuccessful moves contribute to the Markov chain. After the equilibrium period, ensemble averages of energy and density are accumulated and statistics regarding these properties are kept. The statistical data is given at the end.

The candidate classes identified from the earlier problem description are listed in Table 11.3. Comparison with Table 11.1 indicates a high degree of overlap as well as some differences. Some of the potential classes can be easily eliminated because they are too vague and our experience with the MD case can be used to eliminate fcc lattice immediately. One important distinction between the MD and MC approaches is the treatment of energy. In the MD case, energy was treated as an attribute, whereas in Table 11.3 it is designated as a class. This distinction arises because of the different role

TABLE 11.3 Distinction between classes and attributes for the MC simulation.

Potential class	Does the potential class have attributes/classes itself?	Class or attribute
atom	Yes, type, mass, position, etc.	Class
cycle	No	Attribute
density	No	Attribute
energy	Yes	Class
equilibrium period	No	Attribute
fcc lattice	Yes, positions. But a method	
Intermolecular potential Lennard-Jones potential	Yes, potential parameters but these are also part of atom	Attribute
Markov chain	No	Vague
Metropolis criteria	No	Vague
Monte Carlo	Yes, all of the above	Class
NVE ensemble ensemble	Yes, atoms, temperature, etc.	Class
position	No, but vector components	Attribute
simulation	Yes, for example, MC	Class
statistical data, statistics	Yes, many!	Class
temperature	No	Attribute
volume	No	Attribute

of energy in the two simulation techniques. In MD, the simulation is driven by the calculation of forces and the energy is calculated as a by-product of the force evaluation. In contrast, a MC simulation depends critically on the calculation of energy and forces are not determined Fig. 11.13.

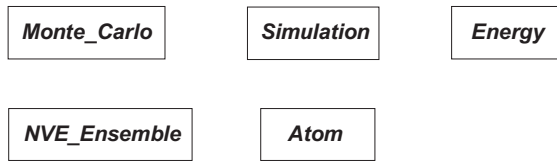


FIGURE 11.13 Short-hand UML notation for the preliminary major classes identified from the OOA of the MC description.

The major classes identified from Table 11.3 are illustrated in UML notation in Fig. 11.13. Comparison with Fig. 11.8 indicates that fewer classes have been identified. The Simulation, NVE_Ensemble, and Atom classes are common to both problems, whereas the Monte_Carlo and Energy classes can be identified as MC analogues of the Molecular_Dynamics and Force classes, respectively, in the MD case study. Consequently, the associations between the classes can be identified by analogy with the MD case. Monte_Carlo is a type of Simulation, which can be performed in an NVE_Ensemble. The NVE_Ensemble is composed of Atoms, and it has an Energy arising from atomic interactions. These associations are illustrated by the UML diagram in Fig. 11.14. Comparison of Fig. 11.14 with Fig. 11.10 highlights the similarity in the associations between classes for the MD and MC techniques. The

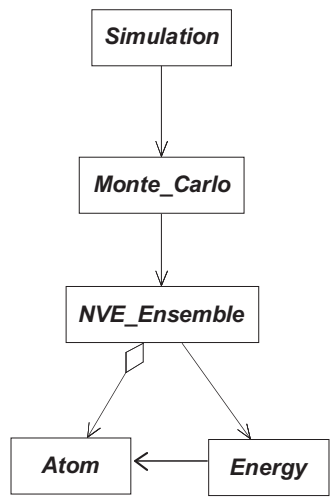


FIGURE 11.14 Preliminary UML diagram for Monte Carlo simulations in the *NVE* ensemble.

UML diagram for the MD case is slightly more complicated because of the additional requirement to have an integrator.

In the MD case, it was argued that there was benefit in having an abstract Ensemble class. The same arguments are clearly valid in this case. Similarly, the Atom and Energy classes can be implemented as abstract classes serving placeholders for potential specific classes. In the current case, we are specifically dealing with Lennard-Jones atoms. These refinements are illustrated in Fig. 11.15. Comparison with Fig. 11.11 shows clearly the similarities and differences between the two simulation techniques.

In the MD simulation, the Molecular_Dynamics class controlled the process of the simulation. In Fig. 11.15, the Monte_Carlo class performs a similar function. However, there is a fundamental difference in the way that this control function is performed. In a MD simulation every molecular displacement corresponds to a valid move because it occurs as a consequence of Newton's

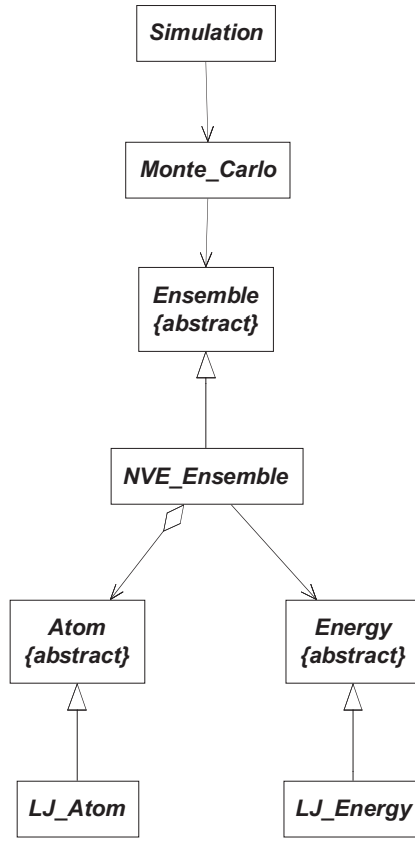


FIGURE 11.15 Further revision of UML diagram for MC simulations showing the use of abstract classes.

equations of motion. Therefore, the Molecular_Dynamics class does not need to interact directly with the *Atom* class to modify atomic coordinates. In contrast, in a MC simulation a decision must be made to either accept or reject a move depending on the outcome of using the acceptance criterion. To implement this decision-making role effectively, the Monte_Carlo class must be able to access atomic coordinates directly. This can be achieved by associating the Monte_Carlo class with the Atom class as illustrated by Fig. 11.16.

A brief description of these classes is summarized in the preliminary data dictionary (Table 11.4). The role of such a dictionary is to have a point of reference during the software design. In the case of a complicated design, it can be expected that the dictionary will evolve considerably before the software is finally coded.

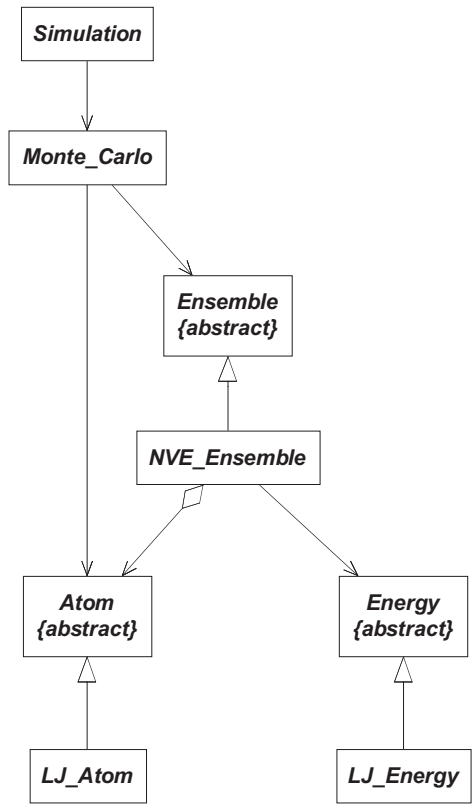


FIGURE 11.16 Final UML diagram for MC simulations showing the addition of a direct association between the Monte_Carlo and Atom classes.

TABLE 11.4 Preliminary data dictionary describing classes in a *NVE* ensemble MC simulation.

Class	Description
Atom	Abstract class serving as a placeholder for the *LJ_Atom* classes. Created by *NVE_Ensemble*.
NVE_Ensemble	An example of an ensemble used to generate all the *Atoms* in the *Ensemble* and the *Energy* classes. Created by *Ensemble*.
Ensemble	Abstract class created by *Monte_Carlo*. Generates *NVE_Ensemble*.
Energy	Abstract class serving as a placeholder for *LJ_Atom* class. Created by *NVE_Ensemble*.
LJ_Atom	Contains details of the Lennard-Jones atom. Created by *Atom*.
LJ_Energy	Performs the energy calculation. Created by *Energy*.
Monte_Carlo	Creates the *Ensemble* and drives the simulation.
Simulation	Creates the *Monte_Carlo* class and handles input, output, and statistical data.

11.3.2 Detailed class description

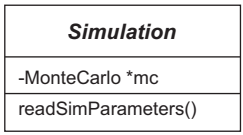
As noted previously in the development of the class description for MD simulation, determining the attributes (both variables and methods) of class is very much an iterative process. However, as will become apparent subsequently, the class descriptions for our MC simulation are very similar to the descriptions for corresponding classes in MD. In most cases, determining MC classes simply involves either the deletion or removal of a few attributes or methods. Indeed, strictly speaking it is not necessary to remove redundant molecular dynamic variables and methods because the encapsulation feature of classes means that these variables will remain dormant unless specifically called upon in the program. However, for the benefit of clarity, redundant MD attributes and methods have been removed in the design presented subsequently. The similarity of the MD and MC classes means that a high percentage of the code developed for the MD program can be reused in the MC program. The reuse of code is an important advantage of OO compared with structural programming.

11.3.2.1 Simulation

The *Simulation* class is the driver class for the whole program. It has one method readSimParameters() responsible for determining the choice of simulation. After the *MonteCarlo* object is created the *run()* method of that class is

called. The class is very similar to its implementation in MD. The major change is that a reference is made to the *Monte_Carlo* class rather than the *Molecular_Dynamics* class.

UML Diagram



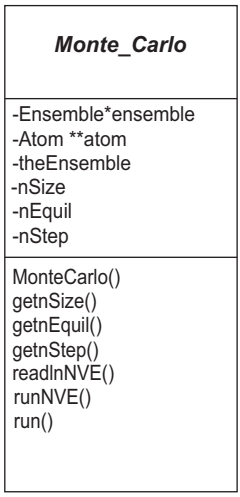
mc_Ver2.0\simMC.h

```
// File:  simMC.h
// -----------------
// This file contains the definition of
// the Simulation class, which defines
// all methods and data for a MC or MD
// simulation, using either.

#ifndef _simMC_h_
#define _simMC_h_

#include "mc.h"

class Simulation
{
  private:
    MonteCarlo *mc;
  public:
    void readSimParameters();
};

#endif
```

11.3.2.2 Monte_Carlo

The *Monte_Carlo* class controls the simulation process via the *runNVE()* method, which is called by the *run()* method. It contains a reference to the *Ensemble* and *Atom* classes. There is a high degree of similarity between the *Monte_Carlo* and *Molecular_Dynamics* classes. The main change has been the removal of MD attributes and apart from the reference to the *Atom* class and the addition of the MonteCarlo() constructor, no new attributes or methods have been added.

UML Diagram



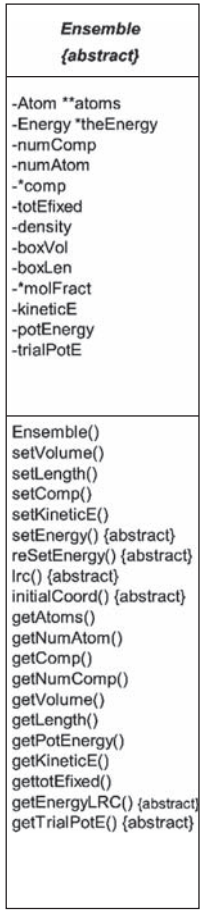
mc_Ver2.0\mc.h

```
//  File:  mc.h
// -----------------
// This file contains the definition of
// the MonteCarlo class, which
// defines all methods and data for a
// MonteCarlo simulation.

#ifndef _mc_h_
#define _mc_h_

#include "ensembleMC.h"
#include "ljatomMC.h"

class MonteCarlo
{
  private:
    Atom **atom;
    Ensemble *ensemble;
    int theEnsemble;
  protected:
    int nSize, nEquil, nStep;
  public:
    MonteCarlo();
    int getnSize();
    int getnEquil();
    int getnStep();
    double gettStep();
    void readInNVE(int );
    void runNVE();
    void run();
};

#endif
```

11.3.2.3 Ensemble

The *Ensemble* class contains most of the required variables and methods of the simulation. It has references to the *Atom* object, an array of *Atoms*, and the simulation parameters. *Ensemble* also has abstract methods that are defined in the derived class *NVE_Ensemble*. The *Force* and *Energy* classes share many common methods and variables.

UML Diagram



mc_Ver2.0\ensembleMC.h

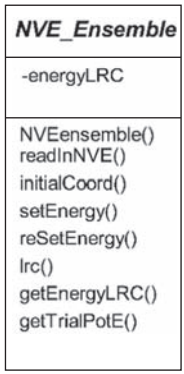
```
// File:  ensembleMC.h
// -----------------
// This file contains the definition of the
// abstract class Ensemble that defines all
// methods and data common to different
// ensemble types (eg derived classes), as
// well as foreshadowing methods of those
// derived classes through the use of abstract
// methods.

#ifndef _ensemble_h
#define _ensemble_h

#include "atomMC.h"
#include "energyMC.h"

// Class: Ensemble
// ---------------
// The EnsembleMC class contains the physical
// attributes of the abstract ensemble, from
// which instantiable ensemble classes are
// derived, such as the NVE ensemble.

class Ensemble
{
 protected:
   Atom  **atoms;       //array of atoms
   Energy *theEnergy;   //the energy
   int numComp;         //no. of components
   int numAtom;         //no. of atoms
   int *comp;           //no. comp. each type
   double totEfixed;    //total energy
   double density;      //density
   double boxVol;       //box volume
   double boxLen;       //box length
   double *molFract;    //mole fractions
   double kineticE;     //kinetic energy
   double potEnergy;    //potential energy
   double trialPotE;    //trial potential E.
 public:
   EnsembleMC(Atom **, int, int,
     double, double, double *, int *);
```

```
   virtual ~Ensemble();
   void setVolume();     //set box volume
   void setLength();     //set box length
   void setComp();       //set atom type
   void setKineticE(double);//set Kinetic E.
   int getNumAtom();     //get total no.atoms
   int *getComp();       //get no.components
   int getNumComp();     //get total no.comp.
   double getVolume();   //get box volume
   double getLength();   //get box length
   double getPotEnergy();//get pot.energy
   double getTotalEfixed();//get total energy
   double getKineticE();  //get kinetic E.

   // Abstract Methods
   // ----------------
   // The following methods are defined in
   // the NVEensemble class (nve.h,nve.cpp).

   virtual void initialCoord()=0;
   //place atoms on lattice

   virtual void setEnergy()=0;
   //Orfer N*N energy calculations

   virtual void reSetEnergy(double)=0;
   //re-set the energy after move is accepted

   virtual void lrc()=0;
   //trigger the long range corrections

   virtual double getTrialPotE(int,Atom **)=0;
   //get potential energy of trial

   virtual double getEnergyLRC()=0;
   //get energy long range correction

};

#endif
```

11.3.2.4 NVE_Ensemble

The *NVE_ensemble* class contains attributes and methods that are specific to the microcanonical ensemble and other methods that are required for the simulation.

UML Diagram



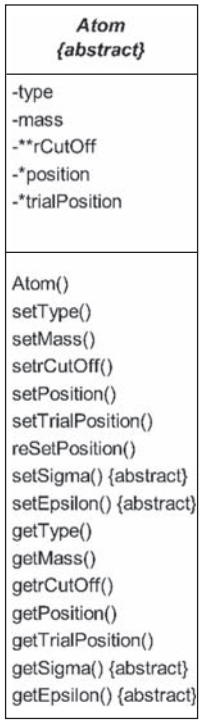
mc_Ver2.0\nveMC.h

```
// File:  nveMC.h
// -----------------
// This file contains the definition of the
// NVEensembleMC class, which defines all
// methods and data for the microcanonical
// ensemble.

#ifndef _nveMC_h
#define _nveMC_h

#include "ensembleMC.h"

class NVEensemble : public Ensemble
{
 private:
   double energyLRC;      //long range E.corr.
 Public:
   NVEensemble(Atom **, int, int, double,
               double, double *, int *);
   void readInNVE();
   //read in ensemble data
   virtual void initialCoord();
   //place atoms on lattice
   virtual void setEnergy();
   //instigate energy calculations
   virtual void reSetEnergy(double);
   //re-set energy after successful move
   virtual void lrc();
   //trigger the long range corrections
   virtual double getEnergyLRC();
   //get energy long range correction
   virtual double getTrialPotE(int, Atom **);
   //get trial potential energy

};
#endif
```

11.3.2.5 Atom

The class *Atom* is an abstract container class for all data and methods common to all atom types such as position and mass. The *Atom* class for the MC simulation is somewhat simpler than the corresponding class for MD because the time derivatives of position are not required. The new additions are variables and methods relating to the trial position.

UML Diagram



mc_Ver2.0\atomMC.h

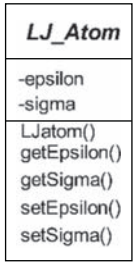
```
// File:  atomMC.h
// -------------
// This file contains the definition of
// abstract class AtomMC, which defines all
// methods and data common to different
// atom types (eg derived classes), as well as
// foreshadowing methods of those derived
// classes through the use of abstract
// methods.

#ifndef _atomMC_h_
#define _atomMC_h_

class Atom
{
 protected:
   int type;                //type of atom
   double mass;             //mass of the atom
   double **rCutOff;        //cut off distance
   double *position;        //position
   double *trialPosition;   //trial position
 public:
   Atom(int, double, int);
   Atom();
   void setType(int);
   //assign type to atom
   void setrCutOff(double **);
   //assign cut off point for atom
   void setMass(double);
   //assign mass to atom
   void setPosition(double *);
   //assign position of atom
   void reSetPosition()
   //re-assign position of atom
   void setTrialPosition(double *);
   //assign atom trial position of atom
  int getType();
  //get type of atom
  double getMass();
  //get mass of atom
  double **getrCutOff();
  //return cut off point for atom
```

```
  double *getPosition();
  //get position of atom
  double *getTrialPosition();
  //get trial position of atom

  // Abstract Virtual Methods
  // ------------------------
  // LJatomMD derived class methods
  // ----------------------------

  virtual void setSigma(double **)=0;
  //set sigma for atom pairs
  virtual void setEpsilon(double **)=0;
  //set epsilon for atom pairs
  virtual double **getEpsilon()=0;
  //get epsilon for atom pairs
  virtual double **getSigma()=0;
  //get sigma for atom pairs
};

#endif
```

11.3.2.6 LJ_Atom

The *LJ_Atom* class inherits from *Atom* and defines those attributes and methods that are specific to atoms interacting via the Lennard-Jones potential. In particular, it contains the ε and σ parameters. The *LJatom* class used in MC simulations is almost identical to that used for MD. The only difference between the two classes is a subtle difference in the second constructor.

UML Diagram



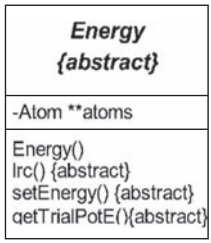
mc_Ver2.0\ljatomMC.h

```
// File:  ljatomMC.h
// -----------------
// This file contains the definition of the
// Ljatom class, which defines all methods and
// data for an LJatomMC object interacting via
// the Lennard-Jones 12-6 intermolecular
// potential.

#ifndef _ljatomMC_h
#define _ljatomMC_h

#include "atomMC.h"

class LJatom : public Atom {
 private:
   double **epsilon;             //LJ epsilon
   double **sigma;               //LJ sigma
 public:
   LJatom(int , double , double **,
     double **, double **, int);
   LJatom();
   virtual void setSigma(double **);
   //set sigma for atom pairs
   virtual void setEpsilon(double **);
   //set epsilon for atom pairs
   virtual double **getEpsilon();
   //get epsilon for atom pairs
   virtual double **getSigma();
   // get sigma for atom pairs
};

#endif
```

11.3.2.7 Energy

Energy is an abstract class that maintains a reference to the array of atoms. It is a container class for the methods in its derived classes that are responsible for performing the *setEnergy* calculations and long-range corrections. The *Energy* class performs the analogous role to the *Force* class in MD.

UML Diagram



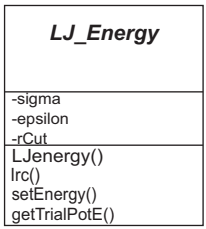
mc_Ver2.0\energyMC.h

```
// File:  energyMC.h
//-----------------
// This file contains the definition of
// abstract class Energy, which defines all
// methods and data common to different
// force types (eg derived classes), as well
// as foreshadowing methods of those derived
// classes through the use of abstract
// methods.

#ifndef _energyMC_h
#define _energyMC_h

#include "atomMC.h"

class Energy
{
 protected:
   Atom **atoms;   //reference to atom array
 public:
   Energy(Atom **);
   virtual ~Energy();

   // Abstract Methods for subclasses
   // -------------------------------

   // For LJEnergy:
   virtual void setEnergy(int, double,
              double *)=0;
   //set energy calculations
   virtual void lrc(int, int *, double,
              double *)=0;
   //long range corrections
   virtual double getTrialPotE
              (int, double, int, Atom **)=0;
   //get trial potential energy
};

#endif
```

11.3.2.8 LJ_Energy

LJ_Energy is the subclass of *Energy* that is responsible for performing the long-range corrections and energy calculations using the Lennard-Jones potential. It plays the analogous role to *LJforce* in a MD simulation.

UML Diagram



mc_Ver2.0\ljenergyMC.h

```
// File:  ljenergyMC.h
// -------------
// This file contains the definition of the
// Ljenergy class, which defines all methods
// and data for an EnergyMC object which
// interacts according to the Lennard-Jones
// intermolecular potential.

#ifndef _ljenergyMC_h_
#define _ljenergyMC_h_

#include "energyMC.h"

class Ljenergy : public Energy
{
 private:
   double **epsilon, **sigma, **rCut;
   //reference to epsilon, sigma and rCut
 public:
   Energy(Atom **);
   virtual void setEnergy(int, double,
            double *);
   //set energy calculations
   virtual void lrc(int, int *, double ,
             double *);
   //long range corrections
   virtual double getTrialPotE(int, double,
            int, Atom **);
   //get trial potential energy
};

#endif
```

11.3.2.9 Auxiliary methods

The auxiliary methods required for the MC simulation are identical to the auxiliary methods used in MD.

11.3.3 OOP in C++

After defining the class diagrams and prototyping the methods and variables required for the various classes, we enter the programming phase. The variables and methods for each class are written in a source file with the same name as the header file. The code for many of the methods is identical to the MD program. This illustrates the reusability feature of the object-oriented approach.

11.3.3.1 Code for the main() function

As discussed previously, the main() exists effectively outside of the analysis. It has a minimal role involving the instantiation of an object of type

Simulation and the execution of one simple method. Thereafter, the program proceeds in accordance with the instructions in the various classes, which control the creation of objects and message passing between objects via method calls. The code for main() presented subsequently is identical to the code for main() in the MD simulation.

mc_Ver2.0\mainMC.cpp

```
// File mainMC.cpp
// -----------
// This file contains the main() function
// responsible for starting the simulation
/  process.

#include "simMC.h"

main()
{
  Simulation sim;
  sim.readSimParameters();
  return 0;
}
```

An object of type Simulation is instantiated.

Call to readSimParameters() invokes the simulation process.

11.3.3.2 Code for the simulation class

The code for the *Simulation* class is identical to the corresponding class for MD except that an object of type *Monte_Carlo* is created rather than an object of type *Molecular_Dynamics*.

mc_Ver2.0\simMC.cpp

The simulation class is responsible for reading in the simulation parameters.

In the main() an object of type Simulation is instantiated and the readSimparameters() method is called.

The simulation settings are read from the "sim.dat" file.

If the MC option is chosen a reference (mc) to an object of MonteCarlo is created. The run() method of MonteCarlo is called which starts the simulation process.

```
// File:  simMC.cpp
// -----------------
// This file contains the implementation of
// the Simulation class, which defines all
// methods and data for a molecular
// simulation, using either molecular
// dynamics or Monte Carlo etc.

#include "simMC.h"
#include <fstream>
#include <iostream>
using namespace std;

// Method:  readSimParameters
// Usage:   readSimParameters();
// ---------------------
// Reads in parameters for NVE ensemble

void Simulation::readSimParameters()
{
  int simulation;
  //open the simMC.dat file
  ifstream in;
  in.open("simMC.dat");
  if(in.fail()){
       cout << "Cannot open simMC.dat!\n";
       return;
  }

  in >> simulation;
  in.close();
  switch(simulation)
  {
    case 2:  //Monte Carlo
       mc = new MonteCarlo();
       mc->run();
       break;
    default:
       cout << "Invalid! Aborting!" << endl;
       break;
  }
  return;
}
```

11.3.3.3 Code for the Monte_Carlo class

There is considerable similarity between the *Monte_Carlo* and *Molecular_Dynamics* classes. The code for the getnSize(), getnEquil(), and getnStep () methods are identical and only minor changes are required in the readInNVE () method. The most significant changes are confined to the runNVE() method, which is an implementation of Algorithm 9.4.

mc_Ver2.0\mc.cpp

```
#include "mc.h"
#include "nveMC.h"
#include "auxfunc.h"
#include <fstream>
#include <iostream>
#include <math.h>
using namespace std;

// File:  mc.cpp
// Class: MonteCarlo
// -----------------------
// This file contains the implementation
// details for the MonteCarlo class

// Constructor
// -----------
// Accesses parameter file mc.dat, which
// identifies the choice of intermolecular
// potential, and ensemble, and it
// constructs the MonteCarlo object.

MonteCarlo::MonteCarlo()
{
  int potential;
  ifstream in;

  in.open("mc.dat");

  if(in.fail())
  {
    cout << "Unable to open mc.dat
          for MC parameters" << endl;
    return;
   }
  in >> nStep;
  in >> nEquil;
  in >> nSize;
  in >> theEnsemble;
  in >> potential;
  in.close();
  switch(theEnsemble)
  {
```

The constructor is called when a Monte_Carlo object (mc) is created in the Simulation object (sim).

Read-in the simulation settings.

The appropriate read-in method is called

depending on the choice of ensemble..

```
    case 1:          //NVE ensemble
      readInNVE(potential);
      break;
      //other ensembles here
  }
}
```

The run method is invoked by mc->run() from the sim object.

```
// Method:  run
// Usage:   run();
// ---------------
// Runs the appropriate simulation run method
// determined by the choice of ensemble.
```

Different ensembles will require different run() methods.This method controls the execution of the simulation.

```
void MonteCarlo::run()
{
  switch(theEnsemble)
  {
    case 1: // NVE ensemble
    runNVE();
    break;
    //run methods for other ensembles can be
    //inserted here
   }
}
```

The runNVE() method controls the execution of the simulation

```
void MonteCarlo::runNVE()
{
  ofstream out;
  out.open("mc_nve.out", ios::out);

  if(out.fail()) {
     cout << "could not open output file!"
          << endl;
     return;
  }
  int i, index, j = 0, k, n, nTotal,
      seed = -50, counter = 0, accept = 0;
```

Simulation settings.

```
  int nStep  = getnStep();
  int nEquil = getnEquil();
  int nSize  = getnSize();
  int num    = ensemble->getNumAtom();
```

Initialising averages

```
  double length = ensemble->getLength();
  double avPotEnergy = 0.0;
  double avKinEnergy = 0.0;
  double avTotEnergy = 0.0;
  double avTemperature = 0.0;
  double erKinEnergy = 0.0;
  double erPotEnergy = 0.0;
  double erTotEnergy = 0.0;
  double erTemperature = 0.0;
  double *accumPotE, *accumKinE;
  double *accumTotE, *accumTemp;
  double *trial, *position, f;
  double eOld, eNew, eTrial, eDelta,
        trialPotE, potE, kinE, totE, temp;
```

Determine maximum displacement
Determine size of simulation blocks. Properties to be averaged are accumulated in blocks of size n.

Initialisation of the properties of ensemble prior to the simulation.

Check for a valid initial configuration

Start of Markov chain

Each cycle in num trial displacements long.

Randomly select atom.

```
double rMax, eFixed, kNew, kOld, wFactor;
double tally = 0.0, lrc;
double const MAX = 0.1;
rMax = MAX*length;
nTotal = nStep - nEquil;
n = nTotal/nSize;
f = 3.0*num/2.0 - 1.0;
accumPotE = new double [n];
accumKinE = new double [n];
accumTotE = new double [n];
accumTemp = new double [n];
position = new double [3];
trial = new double [3];

for (i = 0; i < n; i++)
{
  accumPotE[i] = 0.0;
  accumKinE[i] = 0.0;
  accumTotE[i] = 0.0;
  accumTemp[i] = 0.0;
}

atom = ensemble->getAtoms();
ensemble->lrc();
//perform long range corrections
lrc = ensemble->getEnergyLRC();
ensemble->setEnergy();
//set initial energy
potE = ensemble->getPotEnergy();
eFixed = ensemble->gettotEfixed();
kNew   = eFixed - (potE + lrc);
//determine inital kinetic energy

if(kNew > 0)
    ensemble->setKineticE(kNew);
else
{
    cout << "Initial configutaion has
         negative kinetic energy. Aborting."
         << endl;
    exit(0);
}

cout << "Results will be directed to
     the file \"mc_nve.out\"" << endl;

for(i = 0; i < nStep; i++)
{
   for (k = 0; k < num; ++k)
   {
     ++counter;
     potE = ensemble->getPotEnergy();
     kOld = ensemble->getKineticE();

     //Select atom randomly
```

Determine a trial position for each atom of the ensemble.

```
index = (int) num*random(&seed);
position  = atom[index]->getPosition();
atom[index]
       ->setTrialPosition(position);

// determine energy at current position
eOld = ensemble
       ->getTrialPotE(index,atom);
```

Set trial position.

```
// find trial position
trial[0] =  position[0]
        + rMax*(2.0*random(&seed)-1.0);
trial[1] =  position[1]
        + rMax*(2.0*random(&seed)-1.0);
trial[2] =  position[2]
        + rMax*(2.0*random(&seed)-1.0);
```

Minimum image convention.

```
// apply periodic boundary conditions
trial[0] -= length
         * nearestInt(trial[0],length);
trial[1] -= length
         * nearestInt(trial[1],length);
trial[2] -= length
         * nearestInt(trial[2],length);
atom[index]->setTrialPosition(trial);
```

Determine the energy of the ensemble at the trial positions

```
// calculate energy at trial position
eNew = ensemble
        ->getTrialPotE(index,atom);
eDelta = eNew - eOld;

trialPotE = potE + eDelta;
kNew = eFixed - (trialPotE +lrc);
```

Check for valid kinetic energy.
Apply Ray's modified Metropolis acceptance criterion.

Re-set atomic coordinates and ensemble energy.

```
if (kNew > 0)
{
  wFactor = pow(kNew/kOld, f);
  if (wFactor > 1 || wFactor >
      random(&seed))  //accept move
  {
    atom[index]->reSetPosition();
    ensemble->setKineticE(kNew);
    ensemble->reSetEnergy(trialPotE);
    accept++;
  }
 }
}
```

The maximum displacement is adjusted to obtain the desired acceptance rate.

```
tally = accept/(double) counter;
if (i < nEquil)
{
  if (tally < 0.5)
    rMax *=0.95;
  else
    rMax *= 1.05;
```

Accumulation of averages after equilibration period.

Accumulation of block statistics.

Analysis of block averages and statistical uncertainties

```
      }

    if(i >= nEquil)
    {
      potE = ensemble->getPotEnergy() + lrc;
      kinE = ensemble->getKineticE();
      totE = potE + kinE;
      temp = 2.0*kinE/(3.0*num);
      avPotEnergy += potE;
      avKinEnergy += kinE;
      avTotEnergy += totE;
      avTemperature += temp;

      if((i != nEquil) &&
              ((i % nSize) == 0))
      {
        accumPotE[j] /= nSize;
       accumKinE[j] /= nSize;
        accumTotE[j] /= nSize;
        accumTemp[j] /= nSize;
       j++;
      }

      accumPotE[j] += potE;
      accumKinE[j] += kinE;
      accumTotE[j] += totE;
      accumTemp[j] += temp;
    }
}

avPotEnergy /= nTotal;
avKinEnergy /= nTotal;
avTotEnergy /= nTotal;
avTemperature /= nTotal;

for(i = 0; i < n; i++)
{
  erPotEnergy += pow((accumPotE[i]
     - avPotEnergy),2);
  erKinEnergy += pow((accumKinE[i]
     - avKinEnergy),2);
  erTotEnergy += pow((accumTotE[i]
      - avTotEnergy),2);
  erTemperature += pow((accumTemp[i]
      - avTemperature),2);
}

out <<"Average Potential Energy/N:\t"
 << avPotEnergy/num <<" +/"<<
 sqrt(erPotEnergy)/(num*nTotal)<<endl;
out <<"Average Kinetic Energy/N:\t "
 << avKinEnergy/num <<" +/"<<
  sqrt(erKinEnergy)/(num*nTotal)<<endl;
out <<"Average Total Energy/N:\t\t" <<
 avTotEnergy/num <<"  +/-
```

```
   "<<sqrt(erTotEnergy)/(num*nTotal)<<endl;
  out <<"Average Temperature:\t\t "
    << avTemperature <<"  +/-
    "<<sqrt(erTemperature)/nTotal<<endl;
  out <<"Acceptance Rate:\t\t "<<100*tally
    <<" %"<<endl;
  out.close();
}

void MonteCarlo::readInNVE(int interPotential)
{
  int i, j;
  double numType, *molFract, **epsilon,
    **sigma, **rCut,  potE, density;
  int dimensions = 3, numComp, numAtom, *comp;
  Atom *newAtom, **atoms;

  // open the NVEfileMC.dat file
  ifstream in;
  in.open("NVEfileMC.dat");

  if(in.fail()){
    cout << "Cannot open
        NVEFileMC.dat!\n";
    return;
  }

  in >> numComp;

  if(numComp <= 0){
     cout << "Number of components
     must be > 0!\n";
     return;
  }

  comp = new int[numComp];
  in >> numAtom;

  if(numAtom < 4){
    cout << "At least 4 atoms
     are required!\n";
    return;
  }

  if(!(molFract = new double [numComp]))
  {
    cout << "Cannot allocate memory
      to molFract" << endl;
    return;
  }
  for(i = 0; i < numComp; i++)
      in >> molFract[i];

  double *mass = new double[numComp];
```

The readInNVE() method is called by the constructor of the Monte_Carlo object.

References to the Atom class.

Some of the input values depend on the potential. In this case only the LJ has been implemented.

Check for successful file opening.

Check that there is at least one component!

Check that there are at least four atoms to build lattice.

The following code performs continues to read-in data and performs various bookkeeping tasks.

```
for(i = 0; i < numComp; i++)
      in >> mass[i];

in >> potE;
in >> density;

// Convert potE per particle to the
// total ensemble value
potE *= numAtom;

in.close();

// construct array of atoms
if(interPotential == 1) // Lennard-Jones
{
     in.open("paramLJ.dat");
     if(in.fail()){
       cout << "Cannot open paramLJ.dat!\n";
       return;
     }

     // assign memory to arrays
     if(!(atoms = new Atom * [numAtom]))
     {
      cout << "Cannot allocate memory
         to atoms!\n";
      return;
     }

     epsilon = getMemory(numComp);
     sigma = getMemory(numComp);
     rCut = getMemory(numComp);

     for (i = 0; i < numComp; i++)
     {
       for (j = 0; j < numComp; j++)
       {
         in >> epsilon[i][j] >>
             sigma[i][j] >> rCut[i][j];

         //adjust for indistinguishable
        // pairs
         if (j!=i)
         {
          epsilon[j][i] = epsilon[i][j];
          sigma[j][i] = sigma[i][j];
          rCut[j][i] = rCut[i][j];
         }
       }
     }

     // close input file
     in.close();

     int *atnum = new int[numComp + 1];
```

This can be expanded for other potentials.

In a mixture the i-j and j-i potential parameters are the same

```
        atnum[0] = 0;

        // calculate the numebr of atoms
        // of each type from the molFract
        for(i = 0; i < numComp; i++)
        {
          numType = numAtom * molFract[i];
          int rem = (int) numType;

          if ((numType - rem) >= .5)
             atnum[i+1] = (int) numType + 1;
          else
             atnum[i+1] = (int) numType;
        }

        int sum = 0;
        for (i = 0; i <= numComp; i++)
        sum += atnum[i];

        while (atnum[numComp] < numAtom)
               atnum[numComp]++;

        for(i = 0; i < numComp; i++)
               atnum[i+1] += atnum[i];

        // loop through the  components
        for (i = 0; i < numComp; i++)
        {
          //loop through the atoms of that type
          for (int j = atnum[i]; j <
             atnum[i+1]; j++)
          {
            //create atom(s) and assign
            // properties,
            newAtom =
            new  LJatom(i, mass[i],
              epsilon, sigma, rCut, dimensions);

            //store reference to atom in array
            atoms[j] = newAtom;
          }
        }
    } // end of LJ atom array creation

    ensemble =
      new NVEensemble
      (atoms, numComp, numAtom, potE,
                      density, molFract, comp);

    return;
}

// Method: getnSize()
// Usage:  n = getnSize();
// -----------------------
```

Create objects of type Ljatom which is accessible via the atoms array. The properties of Ljatom are passed to the constructor

A pointer to the *NVE* ensemble. The properties of the ensemble are passed to the constructor.

Get block size.

```
// Gets the size of the blocks to be averaged.

int MonteCarlo::getnSize()
{
  return nSize;
}
```

Get equilibration period.

```
// Method: getnEquil()
// Usage:  n = getnequil();
// ------------------------
// Gets the equilibration period.

int MonteCarlo::getnEquil()
{
  return nEquil;
}
```

Get number of simulation cycles.

```
// Method: getnStep()
// Usage:  n = getnStep();
// ----------------------
// Gets the total number of simulation steps.

int Montecarlo::getnStep()
{
  return nStep;
}
```

11.3.3.4 Code for the Ensemble class

The code for the *Ensemble* class is similar to the corresponding class for MD. The code for the setVolume(), setLength(), setComp(), getNumAtom(), getComp(), getVolume(), getAtoms(), and getKineticE() methods that were developed for MD can be reused for MC simulations without any modification. New code is required only for the setKineticE()andgetTotalEfixed() methods. In addition, a slight modification is required in the constructor method to initialize totalEfixed and trialPotE and remove the temperature variable.

mc_ver2.0\ensembleMC.cpp

```
// File: ensembleMC.cpp
// --------------------
// File containing functions to implement the
// Abstract EnsembleMC class.

#include "ensembleMC.h"
#include <math.h>

// Method: EnsembleMC
// Usage: EnsembleMC;
// ------------------
// Constructor for the EnsembleMC class

Ensemble::Ensemble(AtomMC **theAtoms,
   int nComp, int nAtoms, double totE,
   double dens, double *mol, int *cmp)
{
   atoms = theAtoms;
   numComp = nComp;
   numAtom = nAtoms;
   totEfixed = totE;
   density = dens;
   molFract = mol;
   comp = cmp;
   potEnergy = 0.0;
   trialPotE = 0.0;
}

// Destructor for EnsembleMC class
EnsembleMC::~EnsembleMC(){}

// setMethods
// ----------
// These methods assign values to the volume,
// length, and component number for the
```

The ensemble constructor is called from the read-in function during the construction of the mc object. The properties of the Ensemble are initialized by the constructor.

```
// ensemble.  They do not take any parameters,
// and are called during the construction
// phase of the ensemble
```

Volume of the ensemble.

```
// Method: setVolume
// Usage:  setVolume();
// --------------------
// Determines the volume of the simulation
// box.

void EnsembleMC::setVolume()
{
  boxVol = numAtom/density;
}
```

Length of simulation box.

```
// Method: setLength
// Usage:  setLength();
// --------------------
// Determines the length of the simulation
// box.

void EnsembleMC::setLength()
{
  boxLen = pow(boxVol, 1.0/3.0);
}
```

Number of components of each atom.

```
// Method: setComp
// Usage:  setComp();
// ------------------
// Determines the number of atoms of each
// type.

void EnsembleMC::setComp()
{
  int i, sum = 0;

  for(i =0; i < numComp - 1; i++)
  {
    comp[i] = (int)(molFract[i] * numAtom);
    sum += comp[i];
  }
  comp[numComp - 1] = numAtom - sum;
}
```

Set the kinetic energy.

```
// Method: setKineticE
// Usage:  setKineticE(kNew);
// --------------------------
// Sets the kinetic energy of the ensemble.

double EnsembleMC::setKinetic(double kNew)
{
  kineticE = knew;
}
```

```
// getMethods
// ----------
// These methods return the number of atoms,
// an array describing the number of each
// atom of each type, the number of
// components, the temperature of the
// ensemble, the volume of the ensemble box,
// the length of the ensemble box, the
// potential energy, the total energy, and the
// kinetic energy of the ensemble.
```

Get the number of atoms in the ensemble.

```
// Method: getNumAtom
// Usage:  n = getNumAtom();
// -------------------------
// Gets the total number of molecules in the
// ensemble.

int Ensemble::getNumAtom()
{
  return numAtom;
}
```

Get the number of components.

```
// Method: getNumComp
// Usage:  n = getNumComp();
// -------------------------
// Gets the number of components.

Int EnsembleMC::getNumComp()
{
  return numComp;
}
```

Get the number of each component.

```
// Method: getComp
// Usage:  p = getComp();
// ----------------------
// Gets the number of each individual comp.

int * Ensemble::getComp()
{
  return comp;
}
```

Get the volume.

```
// Method: getVolume
// Usage:  getVolume();
// --------------------
// Gets the volume of the simulation box.

double Ensemble::getVolume()
{
  return boxVol;
}
```

Get the box length.

```
// Method: getLength
// Usage:  getLength();
// --------------------
// Gets the length of the simulation box.
```

```
double EnsembleMC::getLength()
{
  return boxLen;
}
```

Get the potential energy.

```
// Method: getPotEnergy
// Usage:  getPotEnergy();
// ----------------------
// Gets potential energy following force cal.

double EnsembleMC::getPotEnergy()
{
  return potEnergy;
}
```

Get the total energy

```
// Method: gettotEfixed
// Usage:  n = gettotEfixed();
// --------------------------
// Gets virial following force calculation.

double EnsembleMC::gettotEfixed()
{
  return totEfixed;
}
```

Get the kinetic energy.

```
// Method: getKineticE
// Usage n = getKineticE();
// -----------------------
// Gets the kinetic energy.

double EnsembleMC::getKineticE()
{
  return kineticE;
}
```

Get the array of Atom objects.

```
// Method: getAtoms
// Usage:  n = getAtoms();
// ----------------------
// Return array of atoms

AtomMC **EnsembleMC::getAtoms()
{
  return atoms;
}
```

11.3.3.5 Code for NVE_Ensemble class

The code for the *NVE_Ensemble* class reuses the code for the initialCoord() and getEnergyLRC() methods in the corresponding class in MD simulation. The code force the setEngergy() method analogous to the setForce() method and only minor changes are required to the NVEensemble() and lrc() methods. The only entirely new code required is for getTrialPotE() method.

mc_Ver2.0\nveMC.cpp

```
// File: nveMC.cpp
// -------------------
// File containing functions to implement the
// NVEensembleMC class.

#include <math.h>
#include <string.h>
#include "nveMC.h"
#include "auxfunc.h"
#include "ljenergyMC.h"
```

Create NVEensemble

```
// Method: NVEensemble
// Usage:  NVEensemble;
// ----------------------
// Instantiate the NVEensembleMC class

NVEensemble::NVEensemble(Atom **atoms, int
numComp, int numAtom, double totEfixed, double
density, double *molFract, int *comp)
: Ensemble(atoms, numComp, numAtom, totEfixed,
density, molFract, comp)
{
  setVolume();
  setLength();
  setComp();
  initialCoord();
  theEnergy = new LJenergy(atoms);
}
```

Set the initial coordinates by placing the atoms on a face-centred cubic lattice.

```
// Method: initialCoord
// Usage:  initialCoord();
// ----------------------
// Places atoms upon a lattice to get initial
// coordinates

void NVEensembleMC::initialCoord()
{
  int i, j, x, y, z, offset;
  double cells, dcells, cellL, halfCellL,
         *tempPos, *position;
```

```
double (*holdPos) [3] =
     new double [numAtom][3];
tempPos= new double [numAtom];
position = new double [numAtom];

//Determine the number of unit cells in each
//coordinate direction

dcells = pow(0.25 * (double)
             numAtom, 1.0/3.0);
cells =  (int) nearestInt(dcells, 1.0);

//check if numAtom is an non-fcc number of
//atoms and increase the number of cells if
//necessary

while((4 * cells * cells * cells) < numAtom)
  cells = cells + 1;

//Determine length of the unit cell
cellL = boxLen/ (double) cells;
halfCellL = 0.5 * cellL;
```

Assign x, y and z coordinates for the four atoms in the unit cell.

```
//Construct the unit cell
//point to atoms position
position[0] = 0.0;
position[1] = 0.0;
position[2]= 0.0;
position = atoms[0]->setPosition(position);
position[0] = halfCellL;
position[1] = halfCellL;
position[2]= 0.0;
position = atoms[1]->setPosition(position);
position[0] = 0.0;
position[1] = halfCellL;
position[2]= halfCellL;
position = atoms[2]->setPosition(position);
position[0] = halfCellL;
position[1] = 0.0;
position[2]= halfCellL;
position = atoms[3]->setPosition(position);
```

Assign the remaining positions on the lattice.

```
for(i = 4; i < numAtom; i++)
{
  position[0] = 0.0;
  position[1] = 0.0;
  position[2] = 0.0;
  atoms[i]->setPosition(position);
} //init all other atoms to 0

// Build the lattice from the unit cell by
// repeatedly translating the four vectors
// of the unit cell through a distance cellL
// in the x, y and z directions
```

```
  offset = 0;
  for (z = 1; z <= cells; z++)
    for (y = 1; y <= cells; y++)
      for (x = 1; x <= cells; x++){
        for (i = 0; i < 4; i++){
          j = i + offset;
          if(j < numAtom){
              position =
                  atoms[i]->getPosition();
             holdPos[j][0] = position[0]
                        + cellL * (x-1);
             holdPos[j][1] = position[1]
                        + cellL * (y-1);
             holdPos[j][2] = position[2]
                        + cellL * (z-1);
          }
        }
        offset = offset + 4;
      }

  // Shift centre of box to the origin.
  for (i = 0; i < numAtom; i++)
  {
        tempPos = atoms[i]->getPosition();
        tempPos[0]= holdPos[i][0}- halfCellL;
        tempPos[1]= holdPos[i][1]- halfCellL;
        tempPos[2]= holdPos[i][2]- halfCellL;
        atoms[i]->setPosition(temPos)
  }
}
```

Set the potential energy at the start of the simulation.

```
// Method: setEnergy
// Usage:  setEnergy();
// --------------------
// Set the energy calculations using
// LJenergy. This is an N*N calculation to
// set the energy at the start of the
// simulation.

void NVEensemble::setEnergy()
{
  theEnergy->setEnergy(numAtom, boxLen,
              &potEnergy);
}
```

Re-set the energy after an accepted move.

```
// Method: reSetEnergy
// Usage:  reSetEnergy(energy);
// ----------------------------
// Re-sets the energy of the ensemble
// following an accepted move

void NVEensembleMC::reSetEnergy(double eng)
{
 potEnergy = eng;
```

```
}


// Method: getTrialPotE
// Usage: n = getTrialPotE(atoms)
// --------------------------------
// Performs N*N calculations to update energy
// after all atoms are moved.

double NVEensemble::getTrialPotE
      (int index,  Atom **a)
{
  return theEnergy->getTrialPotE
         (numAtom, boxLen, index, a);
}

// Method: lrc
// Usage: lrc();
//  ------------
// initiate the force calculations for long
// range corrections

void NVEensemble::lrc()
{
  theEnergy->lrc(numComp, comp, boxVol,
                 &energyLRC);
}

// Method: getEnergyLRC
// Usage: n = getEnergyLRC();
// --------------------------
// Gets the long range energy correction.

double NVEensemble::getEnergyLRC()
{
  return energyLRC;
}
```

Determines the energy of the trial configuration.

Determine l.r.c to energy

Get energy long range correction

11.3.3.6 Code for the Atom class

The *Atom* class reuses the unaltered code for setType(), setCutOff(), setMass(), setPosition(), getType(), getMass() and getrCutOff(), and getPosition() methods of the corresponding class in the MD program. Very little coding is required for the new setTrialPosition(), reSetPosition(), and getTrialPosition() methods.

mc_Ver2.0\atomMC.cpp

```
// File: atomMC.cpp
// --------------
// File containing the method bodies
// to implement the atom class.

#include "atomMC.h"

// Constructor:Atom
// Usage: Atom atom;
// -------------------
// Builds the Atom class

Atom::Atom(int theType, double theMass,
      int dimensions)
{
  int i;
  mass = theMass;
  type = theType;

  // allocate memory for the arrays
  position = new double[dimensions];
  trialPosition = new double[dimensions];
}

// 2nd constructor for array reference
// construction

Atom::AtomMC(){}

// Method: setType
// Usage:  setType(atomType);
// --------------------------
// Used to identify the atom as belonging to
// a certain type of component within the
// ensemble.

void Atom::setType(int t)
{
  type = t;
}

// Method: setrCutOff
// Usage:  setrCutOff(atomic_rCutOffValue);
// ----------------------------------------
// Used to assign the values for the rCutOffs
// for the various atom types that make up the
// ensemble

void Atom::setrCutOff(double **newrCut)
{
  rCutOff = newrCut;
```

This constructor is used to build the atom class using the parameters described in the parameter list

Set the atom type.

Set the cut-off distance.

```
}

// Method:  setMass
// Usage:   setMass(double atomicMass);
//  ---------------------------------
// Assign the mass of the atom

void Atom::setMass(double newMass)
{
  mass = newMass;
}

// Method:  setPosition
// Usage:   setPosition(position);
// ------------------------------
// Sets the pointer within the atom object to
// point to a new 1D array of doubles,
// representing the position of the atom in
// its various dimensions.  Assuming a 3D
// simulation, the required array would have
// three elements, where:
// atomicPosition[0] = x component of position
// atomicPosition[1] = y component of position
// atomicPosition[2] = z component of position
// Note:  this method requires an _array_ to
// be passed to it which has already had
// memory allocated to it.  It does not
// take on the values in the array, but the
// address of the array itself.

void Atom::setPosition(double *newPos)
{
  position = newPos;
}

// Method: setTrialPosition
// Usage:  setTrialPosition(newPos);
// --------------------------------
// Assigns trial positions

void Atom::setTrialPosition(double *newPos)
{
  trialPosition[0] = newPos[0];
  trialPosition[1] = newPos[1];
  trialPosition[2] = newPos[2];
}

// Method: reSetPosition()
// Usage:  reSetPosition();
// -----------------------
// Re-sets the positions if a move
// is accepted.

void Atom::reSetPosition()
{
```

Set the atomic masses

Set the position.

Set the trial position.

Re-set the position when move is accepted.

```
  position[0] = trialPosition[0];
  position[1] = trialPosition[1];
  position[2] = trialPosition[2];
}
```

Get the type of atom.

```
// Method: getType
// Usage:  p = getType();
// ----------------------
// Gets the type associated with each atom.

int Atom::getType()
{
  return type;
}
```

Get the atomic mass.

```
// Method: getMass
// Usage:  n = getMass();
// ----------------------
// Get values of the atomic masses.

double Atom::getMass()
{
  return mass;
}

// Access (Get) Methods
// Usage:  n = getXxxxx();
// ----------------------
// These methods return a reference to the
// arrays which hold the values for rCutOff,
// acceleration, position, velocity, time
// step, and force.
```

Get the cut-off value.

```
// Method: getrCutOff
// Usage:  n = getrCutOff();
// -------------------------
// Returns the cut-off distances for atoms.

double **Atom::getrCutOff()
{
  return rCutOff;
}
```

Get the position.

```
// Method: getPosition
// Usage:  n = getPosition();
// --------------------------
// Return position of atom

double *Atom::getPosition()
{
  return position;
}
```

Get the trial position.

```
// Method: getTrialPosition
// Usage: n = getTrialPosition();
```

```
// ------------------------------
// Return the trial position

double *Atom::getTrialPosition()
{
  return trialPosition;
}
```

11.3.3.7 Code for the LJ_Atom class

The code for the *LJ_Atom* class (mc_Ver2.0\ljatomMC.cpp) is identical to the corresponding code for the MD program, and it is therefore not listed here. This is not surprising because the *LJatom* class deals with the simulation parameters of the Lennard-Jones potential, which are totally independent of the nature of the simulation.

11.3.3.8 Code for the Energy class

The *Energy* class plays the analogous role to the *Force* class in the MD simulation. The code for the class is a trivial adaptation of the *Force* class.

mc_Ver2.0\energyMC.cpp

```
// File: energyMC.cpp
// ----------------
// File containing functions to implement
// the abstract EnergyMC class.

#include "energyMC.h"

// constructor
// Method: Energy
// Usage:  n = Energy(atoms);
// -------------------------
// Used to construct the force component of
// any derived energy classes.

Energy::Energy(Atom **theAtoms)
{
  atoms = theAtoms;
}

// destructor
EnergyMC::~EnergyMC(){}
```

11.3.3.9 Code for the LJ_Energy class

The constructor for the *LJ_Energy* class is identical to the constructor of the *LJ_Force* class, and the setEnergy() method is analogous to the setForce() method with the exception that the force calculations have been omitted. The getTrialPotE() method reuses almost all of the code of the setEnergy() method. The lrc() method reuses the code for the corresponding method in the *LJ_Force* class.

mc_Ver2.0\ljenergyMC.cpp

```
// File: ljenergyMC.cpp
// --------------------
// File containing methods to implement
// the Ljenergy class.

#include "ljenergyMC.h"
#include "auxfunc.h"

// constructor
LJenergy::LJenergy(AtomMC **theAtoms)
:Energy(theAtoms)
{
  epsilon = atoms[0]->getEpsilon();
  sigma   = atoms[0]->getSigma();
  rCut    = atoms[0]->getrCutOff();
}
```

Determine the potential energy of the ensemble at the start of the simulation.

```
// Method: setEnergy
// Usage:  setEnergy();
// --------------------
// The ljenergy method calculates the energy
// experienced by all num atoms due to pair
// interatomic interaction. The energy is
// calculated using the Lennard-Jones (6-12)
// potential. The potential is truncated at a
// distance rCut and long range corrections
// must be applied outside the method to
// obtain the full contributions to energy.

void Ljenergy::setEnergy(int num,
      double length, double *potEnergy)
{
  int i, j, kindi, kindj;
  double rXi, rYi, rZi;  //position vect. of i
  double rXj, rYj, rZj;  //position vect. of j
  double rXij, rYij, rZij;  //separation vect.
  double rijSq;          //separation squared
  double rCutSq, sigmaSq;//squared rCut etc
  double sigma2, sigma6; //sigma multiples
  double sigma12;
```

```
double pot , eps;
double *tempPos, *tempPosj;

*potEnergy = 0.0;

// perform full N^2 calculation at the
// beginning of the simulation.

// Calculate energy experienced by atoms due
// interatomic pair interactions.
```

Outer energy loop.

```
for(i = 0; i < num - 1; i++)
{
  // Select molecule i
  kindi = atoms[i]->getType();
  tempPos = atoms[i]->getPosition();

  rXi   = tempPos[0];
  rYi   = tempPos[1];
  rZi   = tempPos[2];
```

Inner energy loop.

```
  for(j = i + 1; j < num; j++)
  {
    kindj = atoms[j]->getType();
    tempPosj = atoms[j]->getPosition();

    rXj   = tempPosj[0];
    rYj   = tempPosj[1];
    rZj   = tempPosj[2];
```

Determine pair separations.

```
    // Calculate pair separation
    rXij = rXi - rXj;
    rYij = rYi - rYj;
    rZij = rZi - rZj;
```

Periodic boundary calculation.

```
    // Apply periodic boundary
    rXij -= length *
                  nearestInt(rXij, length);
    rYij -= length *
                  nearestInt(rYij, length);
    rZij -= length *
                  nearestInt(rZij, length);
    rijSq = rXij*rXij +
                     rYij*rYij + rZij*rZij;
```

Separation must be less than the cut-off distance for calculation to proceed.

```
    // Calculate energy for separations
    // below the cutoff
    rCutSq = rCut[kindi][kindj] *
                       rCut[kindi][kindj];

    if (rijSq < rCutSq){
      // Determine potential between i and j
      sigmaSq = sigma[kindi][kindj] *
                        sigma[kindi][kindj];
      sigma2  = sigmaSq/rijSq;
```

```
        sigma6  = sigma2 * sigma2 * sigma2;
        sigma12 = sigma6 * sigma6;
        pot     = sigma12 - sigma6;
        eps = epsilon[kindi][kindj];
        *potEnergy   += eps * pot;
      }
    }
  }

  *potEnergy *= 4.0;
}
```

Calculate the energy of a trial ensemble obtained by displacing one atom a tim.

```
// Method: getTrialPotE
// Usage:  getTrialPotE();
// ----------------------
// The ljenergy method calculates the energy
// experienced by a given atom (index) due to
// pair interatomic interaction. The energy is
// calculated using the  Lennard-Jones (6-12)
// potential. The potential is truncated at a
// distance rCut and long range corrections
// must be applied outside the method to
// obtain the full contributions to energy.

void LJenergy::getTrialPotE(int num,
      double length, int index, Atom **atom)
{
  int i, j, kindi, kindj;
  double rXi, rYi, rZi;  //position vect. of i
  double rXj, rYj, rZj; // position vect. of j
  double rXij, rYij, rZij;  //separation vect.
  double rijSq;           //separation squared
  double rCutSq, sigmaSq; //squared rCut etc
  double sigma2, sigma6;  //sigma multiples
  double sigma12;
  double pot, potE, eps;
  double *tempPos, *tempPosj;

  potE = 0.0;

  // Calculate energy experienced by the
  // sellected atom (index)
  // with all other distinct pairs
```

Properties of trial atom.

```
  // Properties of index atom.
  kindi = atom[index]->getType();
  tempPos = atom[index]->getTrialPosition();

  rXi   = tempPos[0];
  rYi   = tempPos[1];
  rZi   = tempPos[2];
```

Energy loop.

Avoid self-interaction with the trial atom.

Determine pair separations.

Periodic boundary calculation.

Separation must be less than the cut-off distance for calculation to proceed.

```
// Loop over all the other atoms
  for(j = 0; j < num; j++)
  {

    //Exclude self-interaction
    if(j = != index)
    {
      kindj = atom[j]->getType();
      tempPosj = atom[j]->getTrialPosition();

      rXj   = tempPosj[0];
      rYj   = tempPosj[1];
      rZj   = tempPosj[2];

      // Calculate pair separation
      rXij = rXi - rXj;
      rYij = rYi - rYj;
      rZij = rZi - rZj;

      // Apply periodic boundary
      rXij -= length *
                    nearestInt(rXij, length);
      rYij -= length *
                    nearestInt(rYij, length);
      rZij -= length *
                    nearestInt(rZij, length);

      rijSq = rXij*rXij
                     + rYij*rYij + rZij*rZij;

      // Calculate energy for separations
      // below the cutoff

      rCutSq = rCut[kindi][kindj] *
                          rCut[kindi][kindj];

      if (rijSq < rCutSq){
        // Determine potential between i and j
        sigmaSq = sigma[kindi][kindj] *
                  sigma[kindi][kindj];
        sigma2  = sigmaSq/rijSq;
        sigma6  = sigma2 * sigma2 * sigma2;
        sigma12 = sigma6 * sigma6;
        pot     = sigma12 - sigma6;
        eps = epsilon[kindi][kindj];
        potE  += eps * pot;
      }
    }
  }
  return 4*potE;
}
```

Determine long range corrections.

```
// Method: lrc
// Usage:  n = lrc(num, nComp,
//                 volume, energyLRC);
// ------------------------------------
// ljLRC calculates the long range correction
// term to the energy for an ensemble
// interacting via the Lennard-Jones (12-6)
// potential.

void Ljenergy::lrc(int num, int *nComp, double
     boxV, double *energyLRC)
{
  int i, j;
  const double PI = 3.141592654;
  double eps, sigmaSq, sig3, rCutSq, rCut3,
         sig3R, sig9R, den;

  *energyLRC = 0.0;

  for(i = 0; i < num; i++)
    for(j = 0; j < num; j++)
    {
      eps       = epsilon[i][j];
      sigmaSq   = sigma[i][j] * sigma[i][j];
      rCutSq    = rCut[i][j] * rCut[i][j];
      sig3      = sigma[i][j] * sigmaSq;
      rCut3     = rCut[i][j] * rCutSq;
      sig3R     = sig3/rCut3;
      sig9R     = sig3R * sig3R * sig3R;
      den       = nComp[i] * nComp[j] *
                  sig3/boxV;
      *energyLRC += (8.0/9.0) * den *
                 PI * eps* (sig9R - 3*sig3R);
      }
}
```

11.3.3.10 Code for the auxiliary methods

The code for the auxiliary methods (mc_ver2.0\auxfunc.cpp) is identical to that used for the MD program and therefore it is not listed here.

11.4 Case study: combined MD and MC program for Lennard-Jones atoms in the microcanonical ensemble

11.4.1 OOA

It is evident that the UML diagrams for the MD (Fig. 11.12) and MC simulation (Fig. 11.16) have many features in common. This commonality is also reflected in the high degree of code reuse between the two different simulation programs. The MD and MC UML diagrams can be combined easily to produce a combined analysis for simulation in general. The combined analysis is illustrated in Fig. 11.17.

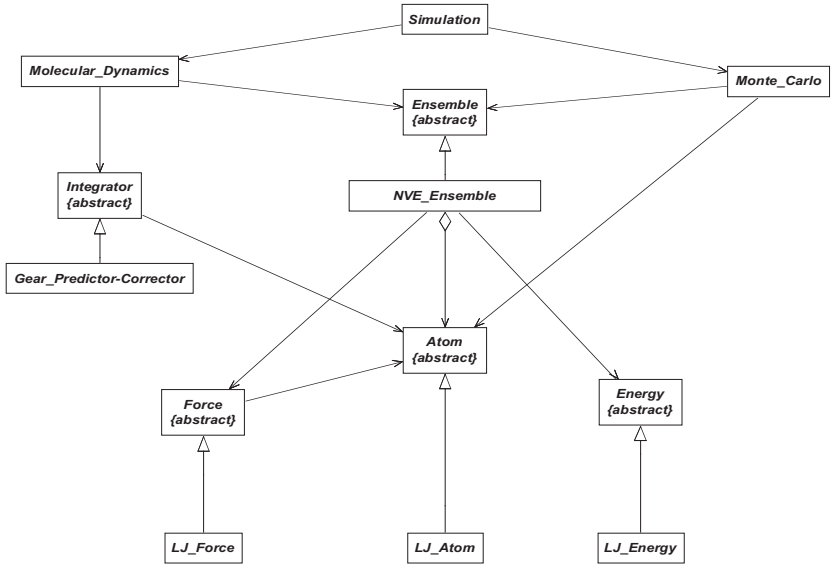


FIGURE 11.17 UML diagram unifying the MD and MC techniques.

11.4.2 OOP in C++

The design presented in Fig. 11.17 did not require any new analysis to be performed. Similarly, the implementation of the design can be achieved almost without the addition of any new code! Two different approaches can be taken. The code for the classes of the combined simulation program can be obtained simply by merging the code for the MC and MD programs. For example, to obtain the *Atom* class, we can combine the attributes and methods of the *Atom* classes from the MC and MD programs. Alternately, we can obtain a combined MC/MD program by simply compiling the files containing MC and MD classes together. This latter approach is adopted here. Therefore, the only changes required to the code is a trivial modification to the simulation class and the main(). The code for main() presented subsequently is identical to the corresponding code for either simulation techniques with the exception that the header file for the combined simulation class (simMCMD.h) must be used.

mcmd_Ver2.0\mainMCMD.cpp

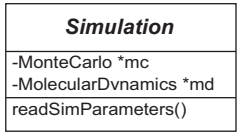
```
// File mainMCMD.cpp
// -----------
// This file contains the main() function
// responsible for starting the simulation
/  process.

#include "simMC.h"

main()
{
  Simulation sim;
  sim.readSimParameters();
  return 0;
}
```

An object of type Simulation is instantiated.

Call to readSimParameters() invokes the simulation process.

The description of the simulation class must contain a reference to both the *Monte_Carlo* and *Molecular_Dynamics* object. Only one of these objects will be instantiated depending on the choice made by the user.

UML Diagram



mcmd_Ver2.0\simMCMD.h

```
// File:  simMCMD.h
// -----------------
// This file contains the definition of
// the Simulation class, which defines
// all methods and data for a
// Molecular Simulation, using either
// molecular dynamics or Monte Carlo.


#ifndef _simMCMD_h_
#define _simMCMD_h_

#include "mc.h"
#include "md.h"

class Simulation
{
  private:
    MonteCarlo *mc;
    MolecularDynamics *md;
  public:
    void readSimParameters();
};

#endif
```

mcmd_Ver2.0\simMCMD.cpp

The simulation class is responsible for reading in the simulation parameters.

```
// File:  simMCMD.cpp
// -----------------
// This file contains the implementation of
// the Simulation class, which defines all
// methods and data for a molecular
// simulation, using either molecular
// dynamics or Monte Carlo etc.

#include "simMCMD.h"
#include <fstream>
#include <iostream>
using namespace std;
```

In the main() an object of type Simulation is instantiated and the readSimparameters() method is called.

```
// Method:  readSimParameters
// Usage:   readSimParameters();
// ---------------------
// Reads in parameters for NVE ensemble

void Simulation::readSimParameters()
{
  int simulation;
  //open the sim.dat file
```

The simulation settings are read from the "sim.dat" file.

```
  ifstream in;
  in.open("simMCMD.dat");

  if(in.fail()){
       cout << "Cannot open sim.dat!\n";
       return;
  }

  in >> simulation;
  in.close();
```

The object created depends on whether the MC or MD option is chosen. The corresponding run() method is called which starts the simulation process.

```
  switch(simulation)
  {
    case 1:  //Molecular dynamics
      md = new MolecularDynamics();
      md->run();
      break;
    case 2:  //Monte Carlo
      mc = new MonteCarlo();
      mc->run();
      break;
    default:
      cout << "Invalid! Aborting!" << endl;
      break;
  }
  return;
}
```

Apart from these trivial changes, no other changes are required to the remaining code (mcmd_Ver2.0). This clearly illustrates the advantage of the object-oriented approach in promoting maximum code reuse with minimal effort. To achieve the same effect using a procedural programming approach would almost invariably require additional code development and algorithm changes.

11.5 Case study: extensions

An advantage of OO is that the design and code can be extended to other situations relatively easily. For the benefit of simplicity we illustrate extensions to other situations using the MC analysis as the basic building block. Extension of the MD simulation is exactly analogous.

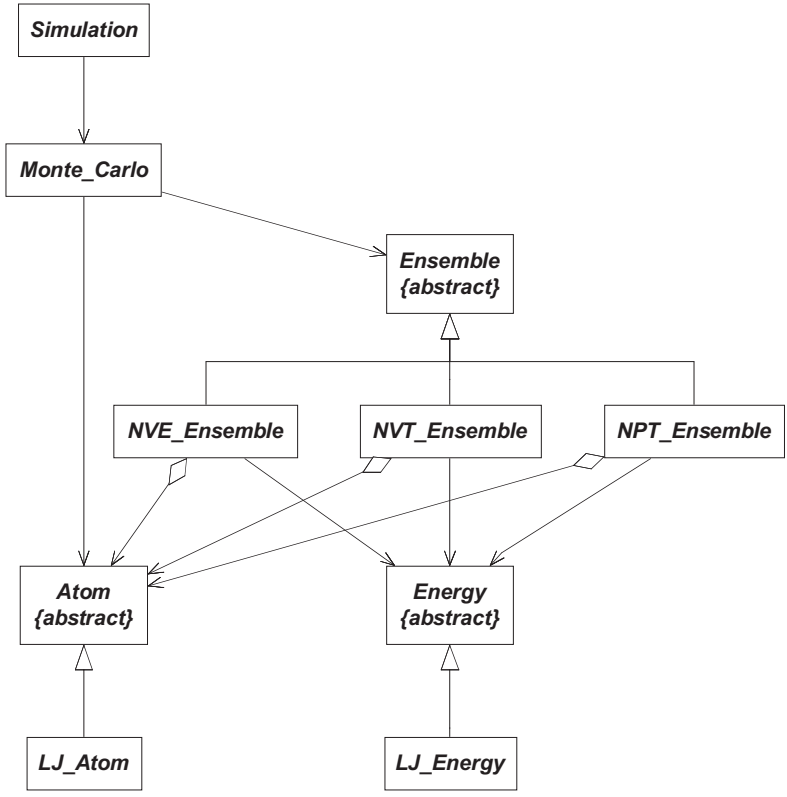


FIGURE 11.18 UML diagram showing the extension of the MC object-oriented analysis for new ensembles.

11.5.1 Other ensembles

Our analysis has been limited to the microcanonical ensemble; however the analysis can be extended easily to any desired ensemble as illustrated by Fig. 11.18.

The additional coding implications of Fig. 11.18 are minimal. For each new ensemble we create a new *Ensemble* class along the same lines as the microcanonical ensemble. Different read-in methods (*readInNVE()*, *readInNVE()*, and *readInNPT()*) will be required in the *Ensemble* class because each ensemble is characterized by different properties. Similarly, because the simulation algorithm is specific to the type of ensemble, specific run methods (*runNVE()*, *runNVE()*, and *runNPT()*) are required in the *Monte_Carlo* class.

11.5.2 Other ensembles and other potentials

The analysis can be extended simply to include any desired potential as shown in Fig. 11.19.

The incorporation of additional potentials such as the exp-6 (*Exp6*) or hard-sphere intermolecular potential (*HS*) involves the creation of new

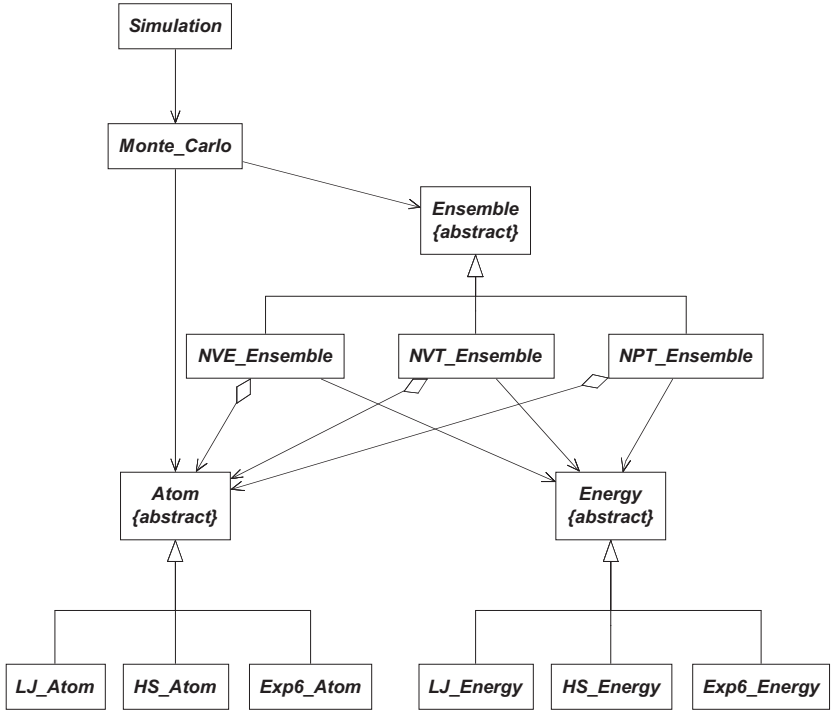


FIGURE 11.19 UML diagram showing the addition of various intermolecular potentials.

classes that inherit from the Atom class. The potential classes contain the characteristic potential parameters. The read-in method must also be altered to account for the different potential. However, more significantly, the choice of potential changes the nature of the energy calculation. The creation of potential specific energy classes such as *HS_Energy* and *Exp6_Energy* involves new methods to account for the potential specific interactions.

11.5.3 Other ensembles, other potentials, and molecules

Our analysis has been confined to atomic systems. A strength of the OO is that the analysis can be extended readily to molecules. By definition, a molecule is composed of two or more atoms. Consequently, we can devise a *Molecule* class, which is an aggregation of *Atom* classes. This is illustrated in Fig. 11.20.

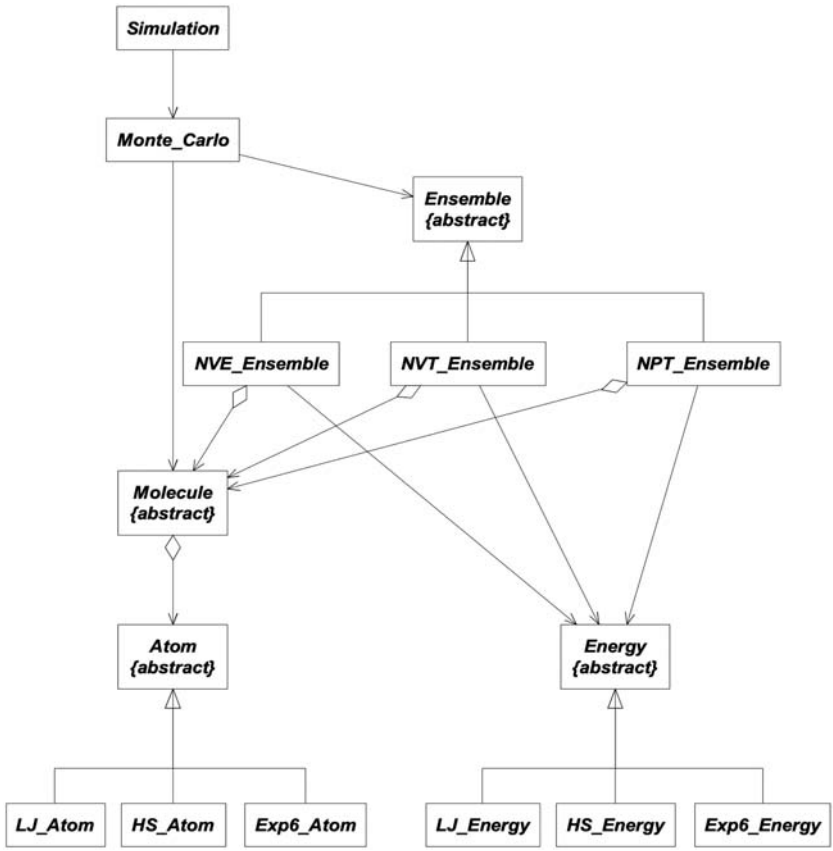


FIGURE 11.20 UML diagram showing the extension of the object-oriented analysis to molecules.

In Fig. 11.20 the Molecule class acts as a container for Atoms that compose the molecule and as such it contains information regarding the molecule such as number of atoms, bond separation, etc.

11.6 Summary

OO has a valuable role in the development of simulation software, particularly for complicated molecules and processes. It requires a shift of focus from algorithm-centered to object-centered design. The benefit of an object-oriented approach is that it permits a high degree of software reuse. The reuse of code is illustrated by the high degree of commonality between the MD and MC programs. OO allows us to manage future increases in complexity much more effectively than can be achieved using procedural programming. However, to derive full benefit of OO approach it is important that the analysis and design aspects are carefully considered before coding commences. Of course, redesign after coding is natural part of iterative improvement during the software life cycle but a well-thought out initial OOA and OOD should greatly help minimize the effort that is subsequently required. The extent of reuse will be tested in Chapter 12 when the codes are modified for parallel execution.

References

Abraham, M.J., Murtolad, T., Schulz, R., Szilárd, P., Smith, J.C., Hess, B., Lindahl, E., 2015. GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1-2, 19–25.

Amorsy, M., Grundy, J., Sadus, R.J., van Straten, W., Barnes, D.G., Kaluza, O., 2013. A suite of domain-specific visual languages for scientific software application modeling. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing.

Baekdal, L., Joosen, W., Larsen, T., Kolafa, J., Ovesen, J.H., Perram, J.W., Petersen, H.G., Bywater, R., Ratner, M., 1996. The object-oriented development of a parallel application in protein dynamics: why we need software tools for HPCN applications. Comp. Phys. Commun. 97, 124–135.

Deitel, H.M., Deitel, P.J., 2002. C++ How to Program, fourth ed. Prentice-Hall, Englewood Cliffs, New Jersey.

Eriksson, H.-E., Penker, M., Lyons, B., Fado, D., 2003. UML 2 Toolkit. John Wiley & Sons, New York.

Haney, S.W., 1994. Is C++ fast enough for scientific computing? Comp. Phys. 8, 690–694.

Khan, E.H., Al-A'ali, M., Girgis, M.R., 1995. Object-oriented programming for structured procedural programmers. Computer 28, 48–57.

Nelson, M.T., Humphrey, W., Gursoy, A., Dalke, A., Kalé, L.V., Skeel, R.D., Schulten, K., 1996. NAMD: A parallel, object-oriented molecular dynamics program. Int. J. Supercompter Appl. High. Perform. Comput. 10, 251–268.

Parsons, D., 1997. Object-Oriented Programming with C++, second ed. DP Publications, London.

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernades, J.P., Saraiva, J., 2021. Ranking programming languagues by energy efficiency. Sci. Compt. Program. 205, 102609.

Phillips, J.C., Hardy, D.J., Maia, J.D.C., Stone, J.E., Ribeiro, J.V., Bernardi, R.C., Buch, R., Fiorin, G., Hénin, J., Jiang, W., McGreevy, R., Melo, M.C.R., Radak, B.K., Skeel, R.D., Singharoy, A., Wang, Y., Roux, B., Aksimentiev, A., Luthey-Schulten, Z., Kalé, L.V., Schulten, K., Chipot, C., Tajkhorshid, E., 2020. Scalable molecular dynamics on CPU and GPU architectures with NAMD. J. Chem. Phys. 153, 044130.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., 1991. Object-Oriented Modelling and Design. Prentice Hall, Englewood Cliffs.

Stroustrup, B., 2013. The C++ Programming Language, fourth ed. Addison-Wesley, Reading, Massachusetts. See also: https://www.stroustrup.com/4th.html.

Stroustrup, B. 2023. A Tour of C++ , third ed. Addison-Wesley, Boston.

Thompson, A.P., Aktulga, H.M., Berger, R., Bolintineanu, D.S., Brown, W.M., Crozier, P.S., Veld, P.J., Kohlmeyer, A., Moore, S.G., Nguyen, T.C., Shan, R., Stevens, M.J., Tranchida, J., Trott, C., Plimpton, S.J., 2022. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Comp. Phys. Commun. 271, 108171.

This page intentionally left blank

Chapter 12

GPU and CPU parallel molecular simulation with CUDA and MPI

The previous chapters have implicitly assumed that molecular simulations are executed in serial on a single processor. In practice, many useful molecular simulation codes leverage the power of advances in parallel computation (Sanders and Kandrot, 2011) to provide results that have immensely expanded both the value and the scientific reach (Abraham et al., 2015; Kirk and Hwu, 2010; Couturier, 2014; Heinecke et al., 2015) of molecular simulation methods. The general approach for parallel computing (Heermann and Burkitt, 1991; Mattson et al., 2005) is to distribute the numerically intensive aspects over multiple processors or cores, which are executed simultaneously. This inevitably requires the use of additional instructions in the software to take advantage of the parallel hardware architecture. The use of message passing (MP) directives, such as in OpenMP[1] (Chandra et al., 2001; Quinn, 2004; Mattson et al., 2019), is a simple way to achieve this, whereas the use of a message passing interface (MPI)[2] (Pacheco, 1997; Karniadakis and Kerby, 2003) provides much greater control and customization, and as such it is arguably the most commonly used approach for multiprocessor systems.

Hardware architectures (Owens et al., 2008; Hwu, 2011; Sanders and Kandrot, 2011; Barlas, 2015) have been developed by combining a central processing unit (CPU) with a graphics processing unit (GPU). Although originally used in gaming and to enhance image rendering (Scarpino, 2012), the use of GPUs provides an enormous speed-up for numerical computing (Heinecke et al., 2015). Similar to conventional parallel CPU programming, directives in the software are required to take advantage of the GPU architecture. The most widely used GPUs are provided by Nvidia,[3] which uses the compute unified device architecture (CUDA) language (Ansorge, 2022; Farber, 2011; Ploskas and Samaras, 2016; Storti and Yurtoglu, 2016) to instruct the GPUs. We will focus on CUDA, but OpenCL (Gaster et al., 2012; Munshi et al., 2012) also provides an alternative means of GPU instruction.

1. Details of OpenMP are maintained at https://www.openmp.org
2. Details of Open-MPI are maintained at https://www.open-mpi.org
3. Details of CUDA are maintained by Nvidia at https://developer.nvidia.com/cuda-toolkit

Molecular Simulation of Fluids. DOI: https://doi.org/10.1016/B978-0-323-85398-9.00011-3

This chapter aims to illustrate how the principles of parallel computation can be applied to molecular dynamics (MD) and Monte Carlo (MC) molecular simulation code. To achieve this goal, case studies are provided using the serial MC and MD simulation C++ code outlined in Chapter 11. This approach aims to maximize the clarity of explanation of the parallel implementation by making the simplest possible CUDA and MPI changes to the serial code. The emphasis on simplicity means that no attempt has been made to optimize the computational efficiency of the code. Details of much more sophisticated approaches can be obtained elsewhere (Pacheco, 1997; Pacheco, 2011; Couturier, 2014; Heinecke et al., 2015). The parallel implementation inevitably means that some explanations of CUDA and MPI directives are required. Again, this has been kept to a minimum and it is not a substitute for detailed sources of information that are available elsewhere (Ansorge, 2022; Barlas, 2015; Karniadakis and Kerby, 2003; Quinn, 2004).

Code availability

The CUDA and MPI source code for programs developed in the case studies are available for download form the book's website, which is maintained by the publisher: https://www.elsevier.com/books-and-journals/book-companion/9780323853989. A software user's guide is given in the Appendix.

12.1 Basic GPU implementations with CUDA

Traditional computing involves executing code exclusively on either one CPU or distributing the computational load on several CPUs. In contrast, GPU programming involves coordinating the calculations between a *host* CPU and a GPU, which is commonly referred to as the *device*. Modern high-performance computers (Couturier, 2014; Hou et al., 2013) involve both multiple CPUs and GPUs. However, for the benefit of simplicity, our discussion will be limited to a prototype system comprising one CPU and one GPU. Hereafter, the terms "host" and "device" will be used almost exclusively. The host and the device have their own separate areas of memory, and a lot of GPU programming involves the allocation/deallocation of memory and copying the stored memory of variables between the host and the device as required.

The CUDA language possess many functions, keywords, and other syntax to handle interactions between the host and the device, which are being continually expanded and modified. In the case studies detailed below, we will restrict our attention to only a small subset of the available options that are necessary to implement the serial codes on the GPUs. These are summarized in Table 12.1. Ansorge (2022) and Storti and Yurtoglu (2016) have provided comprehensive descriptions of CUDA for scientific applications. Depending on how much time has elapsed since publication, it is likely that alternative functions will be available but, assuming backward compatibility is maintained, the implementations given here should still work.

TABLE 12.1 Summary of key CUDA functions or syntax used in the case studies.

CUDA code	Description
__global__	Specifier used for kernel functions
__device__	Specifier used for functions that are accessed in a kernel
__shared__	Specifier to indicate shared memory
__syncthreads()	Function to synchronize threads on the device
cudaMalloc()	Function to allocate memory on the device
cudaMemcpy()	Function to copy memory to or from the device
cudaMemcpyHostToDevice	Keyword used in **cudaMemcpy()** to indicate memory is copied from host to device
cudaMemcpyDeviceToHost	Keyword used in **cudaMemcpy()** to indicate memory is copied from device to host
cudaFree()	Function to free memory on the device
<<< x, y >>>	Syntax used when kernel is launched, providing details of the number of blocks (x) and the size of the blocks (y)
threadIndex.x	Keyword for the index of threads in a block
blockIndex.x	Keyword for the index of the block
blockDim.x	Keyword for the number of threads in the block
gridDim.x	Keyword for the number of threads in a grid composed of all the blocks

12.1.1 Defining a Kernel

The first step in creating a GPU program is to determine the computational intensive parts and create the so-called *kernel* functions that are called from the host but executed of the device. The syntax of a kernel function is the same as a C/C++ function but with the addition of the **__global__** specifier, that is,

```
__global__
void nameOfKernel(parameter list …)
{
  // body of kernel
}
```

The other distinguishing feature of the kernel is that **void** is the only possible return type. The code within the kernel is largely indistinguishable from C/C++ code expect for a few additional CUDA statements required by

the device. Other functions can be accessed within the kernel but they must have the __**device**__ specifier, that is,

```
__device__
 return type nameOfDeviceFunction
{
   // body of device function
}
```

where type is the data type of the variable being retuned, that is, **int**, **float**, **double**, etc. Unlike the kernel itself, the device function can have any possible return type.

12.1.2 Allocating and copying memory

Memory for pointer variables is allocated on both the host and the device, which is handled via various memory functions. Memory on the host is allocated in the normal way using **malloc()**:

```
pointer = (type *) malloc (sizeof(type));
```

The **cudaMalloc()** function is used to allocate memory on the device, that is,

```
cudaMalloc((void **) &pointer_dev, sizeof(type));
```

Notice that, in this example, the _dev suffix has been appended to the name of the device variable. This is a handy convention to distinguish between host (no suffix) and device variables.

The **cudaMemcpy()** function handles the copying of pointers to and from the device. This is necessary to allow the device access to pointers allocated on the host. To copy a pointer to the device, we make use of the keyword **cudaMemcpyHostToDevice**, that is,

```
cudaMemcpy(pointer_dev, pointer, sizeof(type), cudaMemcpyHostToDevice);
```

After the kernel is invoked, any pointer on the device can be copied to the host by using the reverse of this function, that is,

```
cudaMemcpy(pointer, pointer_dev, sizeof(type), cudaMemcpyDeviceToHost);
```

This is necessary for the host to access any pointers modified by the device. It is also advisable to free memory when no longer required. This is done on the host side via:

```
free(pointer);
cudaFree(pointer_dev);
```

12.1.3 Launching the kernel

It is common GPU parlance to use the word "launch" for the invocation of a kernel. The kernel is launched (or invoked) by the statement:

```
nameOfKernel <<< numOfBlocks, blockSize >>> (parmeter list …);
```

The kernel is launched in the same way as any other void function except for the terms contained within the <<< >>> bracket, which is responsible for the parallel execution of the code. The variable blockSize is the number of threads per block. GPUs execute kernels using blocks of threads that are in multiples of 32, and the programmer has discretion to choose a suitable value depending on the size (N) of the computation. In a molecular simulation, N will be the number of particles. To distribute the calculation among parallel threads, numOfBlocks is determined via the relationship,

```
numOfBlocks =  (N + blockSize – 1)/blockSize
```

where the use of −1 and integer division ensures the correct number of blocks to obtain at least N threads.

Collectively, the blocks of parallel threads constitute a grid. In one dimension, CUDA makes use of keywords, **threadIndex.x** (the index of the threads within the block), **blockIndex.x** (the index of the block), **blockDim.x** (the number of threads in each block), and **gridDim.x** (the number of threads in the grid composed of all blocks) to appropriately distribute the calculation within the kernel. In all cases the indices commence from 0. We will examine this in more detail when developing the appropriate simulation kernels. There are similar keywords for the **y** and **z** dimensions, but we will not make use of these here.

12.1.4 Reduction

When GPUs are used, the properties evaluated in a molecular simulation, such as energy and pressure, are evaluated in a GPU application across multiple threads in different blocks. These partial results need to be summed in a process known as reduction. In contrast to MPI (see Section 12.5), there is no CUDA function that performs this operation. Instead, a suitable user-specified code is required. There are many considerations that are required to optimize this process, involving factors such as interleaved or sequential addressing. Here, we will adapt a simple approach reported by Harris,[4] which is suitable for molecular simulation. The general approach is to perform the reduction process for all the threads in each block during the execution of the kernel and sum the contributions on the host. The reduction process is common to both MD and MC simulations.

4. Harris, M., https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf (last accessed in January 2023).

The CUDA code for the reduction operation is given in Listing 12.1.

LISTING 12.1 CUDA code for the reduction operation.

1. Code to be executed on the device within the kernel

```
// Shared memory is required to sum the energy (and any other of property) from
// the diiferent threads.

__shared__ double kernelECache[numThreadsInBlock];
// Add similar statements for other properties here

// Insert code for force/energy calculations here

int cacheIndex = threadIdx.x;
//Set the kernel cache values after force/energy calculation
kernelECache[cacheIndex] = potE;
// Set other cache values here as required for other properties

// Synchronize thread in this block
__syncthreads();

// Perform reductions, assuming numThreadsInBox is a power of 2.
int k = blockDim.x/2;
while(k != 0)
{
  if(cacheIndex < k)
  {
    kernelECache[cacheIndex] += kernelECache[cacheIndex + k];
    // Add other cache values here as required for other properties
  }

  __syncthreads();

  k /= 2;
}

if(cacheIndex == 0)
{
  potEnergy[blockIdx.x] = kernelECache[0];
  // Add other array entries here as required for other properties
}
```

2. Code to be executed on the host following the completion of the kernel

```
// Sum the partial contributions of the CPU side
potE = 0;
//Initialize other sums here as required for other properties

for(int k = 0; k < numBlocks; k++)
{
  potE += potEnergy[k];
  // Accumulate other properties here as required
}
```

In Listing 12.1, codes are provided that must be separately executed on the device and the host. The device code forms part of the kernel that evaluated either the force (MD) or the energy (MC and MD). To perform the required reduction, a shared memory is required, which explains the following array definition:

```
__shared__ double kernelECache[numThreadsInBlock];
```

Prior to the reduction operation, the threads in the block are synchronized:

```
__syncthreads();
```

The code assumes that numThreadsInBox is a power of 2, which allows the initial value of k of the **while** loop to be determined by

```
int k = blockDim.x/2;
```

The **while** loop continues to sum the required properties until the 0 index is encountered, with k being halved for each new iteration. When cacheIndex equals 0, the complete sum of the property of the block is stored. It is only after the execution of the kernel is completed that the complete value to the property is obtained by adding together the individual block contributions on the host.

12.1.5 General GPU algorithm

As will be illustrated in the case studies discussed subsequently, some general steps are required both before and after the launch of the kernel. These steps, which are not restricted to molecular simulation, are given in Algorithm 12.1.

ALGORITHM 12.1 General procedure for GPU implementation via a kernel.

Part 1.	Identify the computationally intensive code to be executed on the GPU using a kernel (nameOfKernel(parameters)).
Part 2.	Specify the number of threads (blockSize) and use this with the size of the calculation (N) to determine numOfBlocks: numOfBlocks ← (N + blockSize – 1)/blockSize
Part 3.	Devise a way to distribute the calculation amongst the threads using code inside of the kernel, e.g.,

```
Part 3.1    tID ← threadIdx.x + blockIdx.x * blockDim.x
            step ← blockDim.x * gridDim.x
Part 3.2    loop i ← tID ... N
              .
              .
            i ← i + step
            end i loop
Part 3.3    Perform any necessary reductions (Listing 12.1).

Part 4.     Allocate array/pointer memory required by the host and
            device with malloc() and cudaMalloc().

Part 5.     Copy arrays/pointers from the host to the device
            with cudaMemcpy(….. cudaMemcpyHostToDevice ).

Part 6.     Launch the kernel:
            nameOfKernel <<<numBlocks, blockSize>>> (parameters)

Part 7.     Copy required arrays/pointers from the device to the host
            with cudaMemcpy(….cudaMemcpyDeviceToHost).

Part 8.     Perform any required post-kernel calculations on the host.

Part 9.     Free array/pointer memory on the device and host
            with cudaFree() and free().
```

12.1.6 Compiling molecular simulation codes using CUDA

The details for compiling the CUDA codes are summarized in the software user's guide given in the appendix. However, there is an important issue that affects the accuracy of molecular simulations, which warrants particular attention. GPUs and CUDA were not specifically designed for numerical computation but primarily to incorporate the requirements of both graphics and gaming applications. To optimize these latter applications, the floating-point multiply-add (FMAD) instruction is widely used and, at the time of writing, the default setting for the CUDA compiler (nvcc) is to replace as much as possible of the floating-point computations with FMAD instructions. The effect of the FMAD instructions is to reduce the precision of the calculations. As maintaining precision is very important for the accuracy of molecular simulations, the code should be compiled without FMAD instructions, which is achieved by using the --fmad = false compiler directive.

12.2 Case study: CUDA implementation of MD code (/mdCU_Ver2.0)

The starting basis for the GPU code development is the serial MD code as detailed in Chapter 11 (found in the directory /md_Ver2.0). The advantage of using this code

is that it has been previously tested, and it is known to work correctly in serial. This is important because the results obtained from the parallel code must exactly match the serial case. Even small deviations are a sign that errors have been made and should never be dismissed as unimportant. Retrofitting parallel concepts to serial code also has the learning advantage of focusing on the key elements of CUDA, providing lessons that can be easily applied to other situations. The goal of this case study is to transform the serial MD code to work on GPUs as simply as possible. This almost invariably means that optimal performance cannot be realistically expected because parallel execution was not incorporated into its initial design.

The most straightforward way (Heermann and Burkett, 1991) to introduce parallelization in a molecular simulation is via force/energy decomposition, which involves assigning the force/energy calculation among different threads or CPUs. This is the approach that is used here. An alternative approach is spatial or domain decomposition (Fincham, 1987), which splits the computational load into different regions. The merits of different approaches is discussed elsewhere (Li et al., 2006, 2008). It should also be noted that our case studies involve simple pairwise interactions using the Lennard-Jones (LJ) potential (Chapter 3) and as such they do not address the complexities involved in electrostatic interactions. Issues involved in parallel implementation for induction and three-body interactions are detailed elsewhere (Li et al., 2006, 2008). Considerable progress has been reported in using GPUs in large-scale simulations involving fast multiple methods (Chapter 5) (Kohnke et al., 2020) and biological systems (Phillips et al., 2020).

Our implementation of the MD program in CUDA code (found in the directory /mdCU_Ver2.0) maintains all but one of the existing .cpp and .h files: ljforceMD.cpp becomes ljforceMD.cu. There is only one additional .cu file (cudaSimKernels.cu) plus a corresponding header file (cudaSimKernels.h). The compiler requires the.cu suffix for the two source files. The only changes to the class definitions are confined to atom. The key changes to the files are summarized in Table 12.2.

12.2.1 Key variable additions

In the atom class (atomMD.h), the following public variables have been added:

```
int *kind;       // atom type
double *r;       // position vector
double *f;       // force array
double *rCutii   // cut-off distance
double *epsii;   // LJ parameter
double *sigii;   // LJ parameter
```

The addition of these new public variables is entirely for the pragmatic reason of avoiding the need to pass objects to kernels. The *r and *f pointers are the key changes, whereas the other variables are included to maintain the generality of the code to handle mixtures and as such are not required for pure systems. Apart from these simple additions, no other changes are required in the class definitions.

Values for these additional variables are initialized in the md.cpp file. The required additional code is given in Listing 12.2.

TABLE 12.2 Summary of key changes and additions to the MD files.

Existing file	New file	Description of changes and/or additions
atomMD.h		New public variables added to the atom class
md.cpp		New code to initialize additional variables in the atom class
gearPC.cpp		Minor code changes to reflect the use of the variables added to the atom class
	cudaSimKernels.h	Definition of the kernel setLJForceKernel
	cudaSimKernels.cu	Code to implement the kernel setLJForceKernel
ljforceMD.cpp	ljforceMD.cu	The .cpp suffix is replaced with .cu. Instead of performing the calculations, the kernel setLJForceKernel is launched, which requires the addition of memory management code

LISTING 12.2 Additional code required in md.cpp **to initialize the new variables in the** atom **class.**

```
atom = ensemble->getAtoms();

for(i = 0; i < num; i++)
{
  position = atom[i]->getPosition();
  atom[0]->r[3*i]     = position[0];
  atom[0]->r[3*i + 1] = position[1];
  atom[0]->r[3*i + 2] = position[2];

  l = atom[0]->kind[i] = atom[i]->getType();

  eps = atom[0]->getEpsilon();
  sig = atom[0]->getSigma();
  cut = atom[0]->getrCutOff();

  atom[0]->epsii[i]  = eps[l][l];
  atom[0]->sigii[i]  = sig[l][l];
  atom[0]->rCutii[i] = cut[l][l];
}
```

As it is apparent from the above initializations, the atom[0] array becomes the master array for the GPU implementation containing the properties of the ensemble. Small changes are required in gpcMD.cpp to reflect the use of the atom[0] array.

12.2.2 The force kernel

The most time-consuming operation in the MD code is the evaluation of the forces (Algorithm 12.1, Part 1), which are evaluated in the ljforceMD.cpp file. The heart of this code becomes the kernel setLJForceKernel, which is defined in cudaSimKernels.h as:

```
__global__
void setLJForceKernel(int num, int *type, double length, double *rCut,
    double *sigma, double *epsilon, double *r, double *force,
    double *potEnergy, double *virial);
```

The complete implementation is found in the cudaSimKernels.cu file, and here we focus on the key CUDA features of the code. A partial listing, with the salient features, is given in Listing 12.3.

In the force code, the calculations of potE and virE are spread among the various threads in the different blocks and their values need to be reduced. This requires a few memory operations and a purpose-built reduction code, such as given in Listing 12.1. First, the shared memory is needed to sum potE and virE of the different threads. This requirement explains the following array declarations:

```
__shared__ double kernelECache[numThreadsInBlock];
__shared__ double kernelVCache[numThreadsInBlock];
```

When the kernel is launched, multiple blocks of multiple threads are executed in parallel. This means that we must account for the thread identity (tID), which is conveniently determined by

```
int tID = threadIdx.x + blockIdx.x * blockDim.x;
```

That is, the identity of the thread is determined by calculating the offset (block index multiplied by block size) to the beginning of its block. This is a standard code for GPU applications and not specific to our molecular simulations (Algorithm 12.1, Part 3.1)

In a serial MD code, the forces are evaluated using a step size of one. In contrast, the key to obtain parallelism is to use a very large step size (Algorithm 12.1, Part 3.1) determined by

```
int step = blockDim.x * gridDim.x;
```

This is a very simple but effective way to divide force calculation among the threads, which is in contrast to other more sophisticated approaches (Li et al., 2006, 2008) used in MPI applications.

LISTING 12.3 Key code details for the kernel setLJForceKernel.

```
// Shared memory is required to sum the energy and virial of
// the diiferent threads.
 __shared__double kernelECache[numThreadsInBlock];
 __shared__double kernelVCache[numThreadsInBlock];

 // Distribute calculations among the threads
 int tID = threadIdx.x + blockIdx.x * blockDim.x;
 int cacheIndex = threadIdx.x;
 int step = blockDim.x * gridDim.x;

 potE = 0.0;
 virE = 0.0;

 // Calculate forces experienced by ALL distinct pairs of atoms
 // without using Newton's third law short cut (fij = -fji).

 for(i = tID; i < num; i += step)
  {
    // Type of atom i
    kindi = type[i];

    rXi  = r[3*i];
    rYi  = r[3*i + 1];
    rZi  = r[3*i + 2];
    fXi  = force[3*i];
    fYi  = force[3*i + 1];
    fZi  = force[3*i + 2];


 for(j = 0; j < num; j++)
 {
      // Exclude self-self interaction
      if(j != i)
      {
        //  Type of atom j
        kindj = type[j];

        // Select position vectors
        rXj   = r[3*j];
        rYj   = r[3*j + 1];
        rZj   = r[3*j + 2];

        // Calculate pair separation
        rXij = rXi - rXj;
        rYij = rYi - rYj;
        rZij = rZi - rZj;

        // Apply periodic boundary conditions
        div = rXij/length;
        if(rXij  >= 0)
          nInt = (int)(div + 0.5);
```

```
      else
        nInt = -(int)(0.5 - div);

      // Add similar code for rYij, and rZij here
      // Account for dissimilar atoms in a binary mixture for rCutSq (code not shown)

      if (rijSq <= rCutSq)
      {
        // Account for dissimilar atoms of binary mixture here (code not shown)

        // Determine potential between i and j
        sigmaSq = sig * sig;
        sigma2  = sigmaSq/rijSq;
        sigma6  = sigma2 * sigma2 * sigma2;
        sigma12 = sigma6 * sigma6;
        pot     = sigma12 - sigma6;
        potE   += 4*eps * pot;
        virE   += 24*eps * (pot + sigma12)/3;

        // Calculate forces between atoms
        fij  =  24 * eps * (pot + sigma12)/rijSq;
        fXij = fij * rXij;
        fYij = fij * rYij;
        fZij = fij * rZij;
        fXi += fXij;
        fYi += fYij;
        fZi += fZij;

      } // end of rijSq check

    } // end  of i != j check

  } // end of j loop

  force[3*i]     = fXi;
  force[3*i + 1] = fYi;
  force[3*i + 2] = fZi;

} // end of i loop

// Values of energy terms are halved because ALL pairs, including
// Indistinguishable pairs (ij and ji), were included in the calculations
potE /= 2.0;
virE /= 2.0;

// Implement reduction (listing 12.1) here before returning to the host
kernelECache[cacheIndex] = potE;
kernelVCache[cacheIndex] = virE;
```

The code for step is common to most GPU applications, and CUDA automatically determines both **blockDim.x** and **gridDim.x** for us.

In most respects, the above code is a minor variation of the standard force evaluation such as that coded in ljforceLJ.cpp. The calculations in the outer force loop (Algorithm 12.1, Part 3.2) are commenced from tID and incremented by step. When num is small, the value of step can be relatively large, which means few executions in a given block. Although this is not optimal, MD simulations with only a few hundred atoms still benefit from a very large speed-up. It should be noted that MD code with millions of atoms (Heinecke et al., 2015) can be very efficiently executed on GPU-enabled code, which means the relative system size computational impediment of serial code is not an issue for GPU applications.

An important difference with the serial code is that the force loop calculates all interaction pairs, that is, it does not use Newton's third law shortcut. The reason that Newton's third law shortcut cannot be safely used is that the assignment of the *j*th contribution on any given thread is in a race with the assignment of the *j*th contribution on other threads.[5] The parallel code without the shortcut will evaluate twice as many interactions than the serial code. This means that the values of potE and virE are halved after the force evaluation to reflect the fact that indistinguishable pairs were included. The large speed-up of the GPU implementation more than compensates for the additional force evaluations. The shortcut can be successfully implemented (Pacheco, 2011), but this requires great care and the gains are at least partially offset by other computational impediments.

Other relatively minor differences in the above code compared with the serial version stem from the introduction and use of the additional public variables in the atom class as detailed previously. A host function cannot be called from within a kernel, which means that rather than handling the periodic boundary conditions (PBC) using the nearestInt() function, the code for the PBC is explicitly given in the kernel. The alternative[6] would be to write the nearestInt() function specifically for the device using the **__device__** specifier. After the force evaluation, a reduction is performed as given in Listing 12.1. The kernelECache[] and kernelVCache[] arrays are subsequently summed on the host to obtain the total energy and virial, respectively.

12.2.3 Launching the force kernel

The creation of the kernel setLJForceKernel means that the role of the ljforceMD method is to launch the kernel (Algorithm 12.1, Part 6) and handle the memory requirements (Algorithm 12.1, Parts 4, 5, and 7) for the interaction between the host and the device. The ljforceMD.cpp file is redesigned as ljforceMD.cu because it now contains the kernel. A partial listing ljforceMD.cu is summarized in Listing 12.4.

5. This also applies to OpenMP and MPI code. Some publicly distributed parallel codes do not appear take this into account, which means that care is required when choosing codes for research applications. This is an insidious problem that could be easily masked by the the calculated statistical uncertainties. As a precaution, parallel code should be initially tested by comparison with results from the equivalent serial code.

6. This complication can be avoided by simply using the round() function (Chapter 5) in the PBC statements.

LISTING 12.4 Key code details for ljforceMD.cu.

```
int blockSize = numThreadsInBlock;
int numBlocks = (num + blockSize - 1)/blockSize;

// Initialize forces
for(int i = 0; i < num; i++)
{
   atoms[0]->f[3*i]     = 0;
   atoms[0]->f[3*i + 1]  = 0;
   atoms[0]->f[3*i + 2]  = 0;
 }

// Allocate memory on the host
energy = (double *) malloc(numBlocks*sizeof(double));
vir  = (double *) malloc(numBlocks*sizeof(double));

// Allocate memory on the device
cudaMalloc((void **) &r_dev, 3*num*sizeof(double));

// Add similar cudaMalloc()  statements for force_dev, energy_dev, vir_dev, kind_dev,
// sigma_dev, epsilon_dev and rCut_dev here

// Copy arrays to the deivice
cudaMemcpy(r_dev,  atoms[0]->r, 3*num*sizeof(double), cudaMemcpyHostToDevice);

// Add similar cudaMemcpy() statements for force_dev, force_dev, kind_dev,
// sigma_dev, epsilon_dev and rCut_dev here

// Launch the kernel on the device
 setLJForceKernel <<<numBlocks, blockSize>>> (num, kind_dev, length,
                  rCut_dev, sigma_dev, epsilon_dev, r_dev, force_dev,
                  energy_dev, vir_dev);

 // Copy the 'energy' 'virial' and 'force' arrays back from the device to the host
cudaMemcpy(energy, energy_dev, numBlocks*sizeof(double), cudaMemcpyDeviceToHost);
cudaMemcpy(vir, vir_dev, numBlocks*sizeof(double), cudaMemcpyDeviceToHost);
cudaMemcpy(atoms[0]->f, force_dev, 3*num*sizeof(double), cudaMemcpyDeviceToHost);

// Determine potE and virE
potE = 0;
virE = 0;
for(int k = 0; k < numBlocks; k++)
{
    potE +=  energy[k];
    virE +=  vir[k];
  }

*potEnergy = potE;
*virial    = virE;

// Free memory on the device
cudaFree(energy_dev);

// Add similar cudFree() statements for r_dev, force_dev, kind_dev, sigma_dev
// epsilon_dev and rCut_dev here

// Free memory on the host
free(energy);
free(vir);
```

At the outset, the calculations are distributed among the threads as outlined above by determining blockSize and numBlocks. These values are calculated using numThreadsInBlock (Algorithm 12.1, Part 2), which is a symbolic constant defined in cudaSimKernels.h. The force array is reinitialized prior to the execution of the kernel.

Memory for pointer variables that are passed to the kernel is allocated on both the host and the device using suitable **malloc()** and **cudaMalloc()** statements, respectively (Algorithm 12.1, Part 4). Memory for the arrays is copied to the device using **cudaMemcpy()** statements (Algorithm 12.1, Part 5), which involve the **cudaMemcpyHostToDevice** keyword.

Having established the appropriate memory requirements, the kernel setLJForceKernel is launched with numBlocks and blockSize specified between <<< and >>> (Algorithm 12.1, Part 6). The remaining parameters are passed to the kernel in the conventional way for a normal function.

After the execution of the kernel is completed, **cudaMemcpy()** statements using the **cudaMemcpyDeviceToHost** keyword are required to copy values from the device to the host (Algorithm 12.1, Part 7). This is restricted to values of *f, energy, and vir arrays, as these are the only quantities that are required for further computations on the host (Algorithm 12.2, Part 8). The remainder of the code involves adding the contributions of the energy and vir arrays to obtain potE and virE and using **free()** and **cudaFree()** statements to release memory on the host and the device, respectively (Algorithm 12.1, Part 9).

12.2.4 Changes to the MD integrator

The integrator code, in this case gearPC.cpp, is also a candidate for transformation into one or more kernels. However, as there is a trade-off among memory management, communication, and kernel execution time, this would not have much (if any) additional computational advantage compared with the speed-up of the force calculation. Therefore the modifications to gearPC.cpp have been restricted to the changes required to evaluate the positions, velocities, acceleration, and higher derivatives involving the new public variables introduced in the atom class. Nonetheless, it would be a useful exercise to apply the above concepts to the gearPC.cpp code and compare the different outcomes.

12.3 Case study: CUDA implementation of MC code (/mcCU_Ver2.0)

The transformation of the serial MC code of Chapter 11 (found in the directory /mc_Ver2.0) for use on GPUs can be achieved in a similar way to the transformation of the MD code in the MC case study. Indeed, much of the CUDA code required for MC is identical to MD, and as such, the discussion of these common aspects will not be repeated here, except to point out the similarities.

A summary of the key changes to the serial MC code is given in Table 12.3. In common with the MD case, the details closely follow Algorithm 12.1.

In common with the transformed MD code, the .cpp files are retained, with the ljenergyMC.cpp file redesignated as ljenergyMC.cu. The additional files are cudaSimKernels.cpp and cudaSimKernels.cu.

12.3.1 Energy kernels

The serial MC code involves two types of energy calculations. At the beginning of the simulation, the energy is determined (setEnergy()) for all indistinguishable pair of atoms. Thereafter, the change in energy (getTrialPotE()) is calculated in response to attempted atom displacements, which involves only $N-1$ calculations. This means that two key kernels for the calculation of energy can be identified (Algorithm 12.1, Part 1). Namely, one for calculating the total energy of all pairs (setLJEnergyKernel) and another for calculating the energy of a simple atom with all other atoms (energyLJKernel). The prototype statements for these kernels are found in cudaSimKernels.h, and the code is given in cudaSimKernels.cu. The prototype statements are:

TABLE 12.3 Summary of key changes and additions to the MC files.

Existing file	New file	Description of changes and/or additions
atomMC.h		New public variables added to the atom class
mc.cpp		New code to initialize additional variables in the atom class
	cudaSimKernels.h	Definition of the kernels setLJEnergyKernel and energyLJKernel
	cudaSimKernels.cu	Code to implement the kernels setLJEnergyKernel and energyLJKernel
ljenergyMC.cpp	ljenergyMC.cu	The .cpp suffix is replaced with .cu. Instead of performing the calculations, the kernels setLJEnergyKernel energyLJKernel are launched from the setEnergy() and getTrialPotE() methods, respectively. This requires the addition of memory management code

```
__global__
void setLJEnergyKernel(int num, int *type,  double length, double *rCut,
      double *sigma, double *epsilon, double *r,  double *energy);

__global__
void energyLJKernel(int num, int index, int *type, double length, double *rCut,
    double *sigma, double *epsilon, double *ri, double *r, double *potEnergy);
```

Although the setLJEnergyKernel has a relatively minor role in either *NVE* or *NVT* MC simulations, it has nonetheless been implemented in our CUDA code for both the benefit of completeness and its utility in future *NpT* code for which it would have an important role. For the benefit of brevity, we will focus on the details of the kernel energyLJKernel, which also reflect most of the aspects of the kernel setLJEnergyKernel. However, before proceeding further, it is important to note that, unlike the equivalent serial code, the setLJEnergyKernel code is not restricted to indistinguishable pairs. This change ensures that no i–j pairs are missed in the process of parallelization. Therefore the value of any property calculated in the all-pair energy calculation must be halved.

A partial listing of the code for the kernel energyLJKernel is given in Listing 12.5. Comparing Listing 12.5 with Listing 12.1, it is apparent that CUDA implementation of the MC code to determine the energy is identical to the steps required for the force evaluation. Apart from the fact that only $N-1$ interactions are evaluated, the difference between the two listings are largely confined to the existing differences in the serial code that are required to evaluate forces instead of energies. The mechanism (Algorithm 12.1, Part 3.1) to distribute the calculation between threads using tID and a loop (Algorithm 12.1, Part 3.2) increment size of step instead of 1 are the same in both cases.

LISTING 12.5 Key code details for the kernel energyLJKernel**.**

```
__shared__double kernelECache[numThreadsInBlock];
int tID = threadIdx.x + blockIdx.x *  blockDim.x;
int cacheIndex = threadIdx.x;
int step = blockDim.x * gridDim.x;

// Calculate energy experienced by the selected atom (index)
// with all other distinct pairs
kindi = type[index];
potE = 0.0;
```

```
// Loop over all the other atoms
for(int j = tID; j < num; j += step)
{
  // Exclude self-interaction
  if(j != index)
  {
    rXij = ri[0] - r[3*j];
    rYij = ri[1] - r[3*j + 1];
    rZij = ri[2] - r[3*j + 2];

    // Apply periodic boundary conditions
    div = rXij/length;
    if(rXij  >= 0)
      nInt = (int)(div + 0.5);
    else
      nInt = -(int)(0.5 - div);
    rXij -= length *  (double) nInt;
    // Add similar code for rYij, and rZij here
    // Add code here to account for dissimilar atoms in the binary mixture

    // Calculate forces for separations below the cutoff
    if (rijSq <= rCutSq)
    {
      // Add code here to account for dissimilar atoms in the binary mixture
      sigmaSq = sig * sig;
      sigma2  = sigmaSq/rijSq;
      sigma6  = sigma2 * sigma2 * sigma2;
      sigma12 = sigma6 * sigma6;
      pot     = sigma12 - sigma6;
      potE    += eps*pot;

    }  // end rijSq if

  }  // end j != index if

} // end for loop
// Implement reduction (listing 12.1) here before returning to the host
kernelECache[cacheIndex] = 4*potE;
```

12.3.2 Launching the energy kernels

The use of energy kernels means that that role of the getTrialPotE() and setEnergy() methods is largely confined to preparing launching the kernels (Algorithm 12.1, Part 6). As the details of the two kernels have a high degree of commonality, we will only detail the steps for getTrialPotE().

LISTING 12.6 Key code details of getTrialPotE.cu.

```
// Distribution among threads
int blockSize = numThreadsInBlock;
int numBlocks = (num + blockSize - 1)/blockSize;

// Assign properties of the index atom
position  = atom[index]->getTrialPosition();
ri[0]     = position[0];
ri[1]     = position[1];
ri[2]     = position[2];

// Allocate memory on host
ri = (double *) malloc(3*sizeof(double));
energy = (double *) malloc(numBlocks*sizeof(double));

// Allocate memory on device
cudaMalloc((void **) &ri_dev,      3*sizeof(double));
// Similar cudaMalloc statements for r_dev, energy_dev, kind_dev,
// sigma_dev, epsilon_dev  and rCut_dev

// Copy memory to device
cudaMemcpy(ri_dev,  ri,  3*sizeof(double),  cudaMemcpyHostToDevice);
// Similar cudaMemcpy statements for r_dev, kind_dev, sigma_dev,
// epsilon_dev and rCut_dev

// Launch kernel
energyLJKernel <<<numBlocks, blockSize>>> (num, index, kind_dev, length, rCut_dev,
                          sigma_dev, epsilon_dev, ri_dev, r_dev, energy_dev);

// Copy memory to host
cudaMemcpy(energy, energy_dev, numBlocks*sizeof(double), cudaMemcpyDeviceToHost);

// Sum partial results from reduction in device
for(int k = 0; k < numBlocks; k++)
   potE +=  energy[k];

// Free memory on device
cudaFree(energy_dev);
// Additional cudaFree() statements for ri_dev, r_dev, kind_dev,
// sigma_dev, epsilon_dev, rCut_dev

// Free memory on host
free(energy);
free(ri);
```

It is evident that Listing 12.6 for MC largely mirrors Listing 12.3 for MD.

12.4 Parallel programming with MPI

MPI provides the functions to allow computations to be distributed over multiple CPUs or cores. There are a large number of MPI functions, which are examined in detail elsewhere (Karniadakis and Kirby, 2003; Quinn, 2004).

We will only focus on a very small subset of these functions that represent the bare minimum to execute the serial MD and MC codes in parallel. These are summarized in Table 12.4. As the aim is to illustrate parallel concepts as simply as possible, it is almost inevitable that the computational efficiency of the resulting code will not be optimal.

12.4.1 General MPI procedure

Our implementation of MPI for both the MD and MC simulations will follow the general procedure in Algorithm 12.2, the details of which also apply to other situations.

TABLE 12.4 Summary of key MPI functions or syntax used in the case studies.

MPI function or syntax	Description
MPI_Init (&argc, &argv)	Initialize the MPI process to setup the use of the MPI library
MPI_Finalize()	Called after MPI functions are no longer needed, used to free memory etc. It is typically the last statement in main
MPI_Datatype	The data type, e.g., **MPI_DOUBLE**, **MPI_INT**, for double, int, respectively
MPI_Op	The type of operation, e.g., **MPI_SUM**, **MPI_PROD**, for addition and multiplication, respectively
MPI_Comm	The type of communication, e.g., **MPI_COMM_WORLD** for all the processor running at the start of execution
Int **MPI_Comm_size**(MPI_Comm, int* size)	Determine the number of processors that can send message, i.e., the size parameter is the number of processors used in the communicator
Int **MPI_Comm_rank**(MPI_Comm, int* rank)	Determine the rank of the process via the rank parameter
Int **MPI_Allreduce**(void * input, void* result, int, size, MPI_Datatype, MPI_Op, MPI_Comm)	Combine all the contents (size) of each processor (input) according to MPI_Op and assign it to the second parameter (result). All processors have the combined value

ALGORITHM 12.2 General procedure for implementing MPI.

Part 1. Place **MPI_Init()** before MPI calls and **MPI_Finalize**() after MPI is finished.

Part 2. Identify computationally intensive code that would benefit from parallel execution (nameOfParallelFunction()).

Part 3. Devise a way to distribute the nameOfParallelFunction() calculation among the processors, e.g.,

Part 3.1
```
MPI_Comm_size (MPI_COMM_WORLD, &totalNodes)
MPI_Comm_rank (MPI_COMM_WORLD, &myNode)
step ← totalNodes
start ← myNode
```

Part 3.2
```
loop i ← start ... N
    .
    .
    i ← i + step
end i loop
```

Part 3.3 **MPI_Allreduce()**

12.5 Case study: MPI implementation of MD code (/mdMPI_Ver2.0)

The aim of this case study is to use MPI to enable execution of the serial MD code (/md_Ver2.0) on multiple CPUs or cores (/mdMPI_Ver2.0). A summary of the required changes to the serial MD files is given in Table 12.5.

TABLE 12.5 Summary of changes required for implementing MPI for MD.

Existing MD file	Description of changes
atomMD.h	New public variables added to the atomMD class
md.cpp	New code to initialize additional variables in the atomMD class
gearMD.cpp	Minor code changes to reflect the use of the variables added to atomMD
mainMD.cpp	Include the mpi.h library and add calls to **MPI_Init** () and **MPI_Finalize()**
ljForceMD.cpp	Include the mpi.h library, distribute the force calculation among the CPUs or cores, and add calls to **MPI_Comm_size(), MPI_Comm_rank**(), and **MPI_Allreduce**()

It is apparent from Table 12.5 that relatively few changes are required to implement MPI on the existing serial code.

12.5.1 Key variable additions

In the atom class (atomMD.h), the following public variables have been added:

```
double *r;      // position vector
double *f;      // force array
```

In common with the CUDA implementation, the need for these additional values is entirely for pragmatic reason to avoid multiple arrays for the x, y, and z components. Unlike the CUDA code, additional variables to handle the LJ potential mixture parameters are not required.

The atom[0] array becomes the master array for handling the position vectors (*r) and forces (*f). The vectors are initialized in md.cpp via the following code, whereas the initialization of the forces occurs later in setForce().

```
// Initialize 'master' root node position array
atom = ensemble->getAtoms();

for(i = 0; i < num; i++)
{
  position = atom[i]->getPosition();
  k = 3*i;
  atom[0]->r[k]     = position[0];
  atom[0]->r[k + 1] = position[1];
  atom[0]->r[k + 2] = position[2];
}
```

12.5.2 Changes to mainMD.cpp

The new mainMD() function is given in full in Listing 12.7. The changes to main() are the additions of the **MPI_Init()** and **MPI_Finalize()** functions (Algorithm 12.2, Part 1) and the inclusion of the mpi.h library. **MPI_Init()** must be called only once, and this must occur before any other MPI functions are called. Although it uses pointers (argc and argv) from the main function, it can be called elsewhere in the program. The **MPI_Finalize()** function is commonly included at the end of main(), but it could be included anywhere after the MPI functions are no longer used.

LISTING 12.7 Code details of main.cpp.

```
#include "simMD.h"
#include <mpi.h>

int main(int argc, char **argv)
{
  // Initialise MPI
  MPI_Init(&argc,&argv);

  Simulation sim;
  sim.readSimParameters();

  / / End MPI
  MPI_Finalize();
  return 0;
}
```

12.5.3 Changes to ljForceMD.cpp

The implementation of MPI to the setForce() method (Algorithm 12.2, Part 2) in ljForceMD.cpp is detailed in Listing 12.8. At the start of the method, some new additional variables (potECalR, virialCalR and *f) are defined, which will be required later for the reduction part of the method. The MPI implementation (Algorithm 12.2, Part 3) also requires the definition of myNode, totalNodes, lastNode, start, end, and step. The variable myNode identifies the rank (or index) of the CPU or core. The initial setup involves using the **MPI_Comm_size()** and **MPI_Comm_rank()** functions (Algorithm 12.2, Part 3.1) to determine totalNodes and myNode, respectively

LISTING 12.8 Key code details of the setForce() **method in** ljForceMD.cpp.

```
// Receiving variables for MPI_Allreduce()
double potECalR, virialCalR;
double *f = new double [3*num];

// Zero the force  arrays.
for(i = 0; i < num; i++)
{
   k = 3*i;
   f[k] = f[k + 1] = f[k + 2] = 0;
 }
// Additional variables  for MPI
int myNode, totalNodes;
int start, end, step;

// Set-up MPI
MPI_Comm_size (MPI_COMM_WORLD, &totalNodes);
MPI_Comm_rank (MPI_COMM_WORLD, &myNode);
```

```
step = totalNodes;
start = myNode;
end = num;

// Calculate forces experienced by atoms due
// interatomic pair interactions.
for(i = start; i < end; i += step)
{
  k = 3*i;
  // Select molecule i
  kindi = atoms[i]->getType();

  rXi   = atoms[0]->r[k];
  rYi   = atoms[0]->r[k + 1];
  rZi   = atoms[0]->r[k + 2];

  fXi = f[k];
  fYi = f[k + 1];
  fZi = f[k + 2];

  for(j = 0; j < num; j++)
  {
   if(j != i)  // Exclude self interaction
   {
     kindj = atoms[j]->getType();

     l = 3*j;
     rXj   = atoms[0]->r[l];
     rYj   = atoms[0]->r[l + 1];
     rZj   = atoms[0]->r[l + 2];

     // Calculate pair separation
     rXij = rXi - rXj;
     rYij = rYi - rYj;
     rZij = rZi - rZj;

     // Apply periodic boundary coinditions
     rXij -= length * nearestInt(rXij, length);
     rYij -= length * nearestInt(rYij, length);
     rZij -= length * nearestInt(rZij, length);

     rijSq = rXij * rXij + rYij * rYij + rZij * rZij;

     // Calculate forces for separationsbelow the cutoff
     rCutSq = rCut[kindi][kindj] * rCut[kindi][kindj];
     if (rijSq <= rCutSq)
     {
       // Determine potential between i and j

       sigmaSq = sigma[kindi][kindj] * sigma[kindi][kindj];
       sigma2  = sigmaSq/rijSq;
       sigma6  = sigma2 * sigma2 * sigma2;
       sigma12 = sigma6 * sigma6;
       pot     = sigma12 - sigma6;
       potECal   += 4*epsilon[kindi][kindj] *pot;
       virialCal += 24*epsilon[kindi][kindj] *(pot + sigma12)/3;
```

```
        // Calculate forces between atoms
        fij  = 24*epsilon[kindi][kindj] * (pot + sigma12)/rijSq;
        fXij = fij * rXij;
        fYij = fij * rYij;
        fZij = fij * rZij;
        fXi += fXij;
        fYi += fYij;
        fZi += fZij;

      } // end rijSq if

    }  // end j != i if

  }  // end j for loop

  f[k]     = fXi;
  f[k + 1]  = fYi;
  f[k + 2]  = fZi;

}  // end i for loop

// MPI Reductions
MPI_Allreduce(&potECal, &potECalR, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
MPI_Allreduce(&virialCal, &virialCalR, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
MPI_Allreduce(f, atoms[0]->f, 3*num, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

delete [] f;

*potEnergy =  potECalR/2;
*virial     = virialCalR/2;
```

The outer i force loop of the serial code starts with the 0 index and increments this to the number of atoms (num) in increments of 1. The simplest way to distribute the force calculation (Algorithm 12.2, Part 3.2) is to increase the increment used (step) to be equal to the total number of CPUs or cores (totalNodes). This is similar to the way the force calculation was handled with GPUs, and more elaborate MPI force decompositions are also possible (Li et al., 2006, 2008). The starting point (start) for the loop corresponds to the rank (myNode) and continues until end, which is the number of atoms (num). If only one CPU is available, the range of the for loop is between 0 and num in increments of 1, which is the case of a conventional serial execution.

An alternative to the above implementation is to distribute the atoms between the processors. This is done using the following code prior to the for loop.

```
int lastNode = totalNodes - 1;    // Node count starts at 0
int numCal = num/totalNodes;   // Distribute calculation evenly between nodes
start = myNode*numCal;         // Different starting point for each node

if (myNode == lastNode)        // Different end point for each node
  end = num;
else
  end   = (myNode + 1)* numCal;

step = 1;
```

In this case, the increment of the for loop is 1. The number of calculations (numCal) is divided equally between the processors, and both the start and end points of the for loop depend on the rank of the processor. This approach requires some additional bookkeeping via the if-else statement, but the difference in computational efficiency between the two approaches is negligible.

The inner *j* for loop increments over all the atoms. This means that the calculations are not confined to indistinguishable *i-j* combinations, that is, both *i-j* and *j-i* contributions are countered. Consequently, Newton's third law shortcut is not implemented and the properties evaluated in the force evaluations are halved. This is done to avoid inaccuracies because of the racing condition as discussed previously for the GPU implementation. A specialized algorithm is required to implement Newton's third law in MPI and is detailed elsewhere (Pacheco, 2011).

At the end of the force evaluations (Algorithm 12.2, Part 3.3), the partially evaluated properties are combined using the **All_Reduce()** MPI function. In contrast, the CUDA implementation required a user-defined function (Listing 12.1) to perform the reduction operation.

12.5.4 Changes to the MD integrator

In common with the CUDA case, the integrator code (gearPC.cpp) is also a candidate for MPI execution because of the loop over all atoms. However, the computational benefit (if any) is likely to be relatively small compared to using MPI for the force evaluations. Therefore the modifications to gearPC.cpp have been restricted to the changes required to accommodate the use of the new variables introduced in the atom class.

12.6 Case study: MPI implementation of MC code (/mcMPI_Ver2.0)

This case study uses MPI to enable execution of the serial MC code (/mc_Ver2.0) on multiple CPUs or cores (/mcMPI_Ver2.0). The required changes to the serial MC files are summarized in Table 12.6. It is apparent from Table 12.6 that the changes are confined to only two files and that unlike the other case studies, the changes can be accomplished without any changes to the membership of any classes.

12.6.1 Changes to mainMC.cpp

The changes required to main.cpp exactly mirror Listing 12.7. Namely, the addition of calls (Algorithm 12.2, Part 1) to both **MPI_Init()** and **MPI_Finalize()**, and the inclusion of the mpi.h library.

TABLE 12.6 Summary of changes required to implement MPI for MC.

Existing MC File	Description of changes
mainMC.cpp	Include the mpi.h library and add calls to **MPI_Init()** and **MPI_Finalize()**
ljEnergyMC.cpp	Include the mpi.h library, distribute the energy calculation for the setEnergy() and getTrialPotE() methods among the CPUs or cores, and add calls to **MPI_Comm_size()**, **MPI_Comm_rank()**, and **MPI_Allreduce()**

12.6.2 Changes to ljEnergyMC.cpp

The changes to the ljEnergyMC.cpp file involve changes to the setEnergy() and getTrialPotE() methods (Algorithm 12.2, Part 2). The former is called only once at the start of the simulation to determine the initial energy, whereas the latter is called repeatedly when individual attempted particle displacements are involved. MPI have been implemented for both methods. As the required MPI changes are similar, we will focus on getTrialPotE(). It should be noted that, as the setEnergy() method involves a double loop, it must be implemented for all pairs, including indistinguishable pairs. The calculated properties are then halved at the end of the energy evaluation. This is necessary to avoid inaccuracies due to the racing condition as described previously in the other case studies.

A partial listing of getTrialPotE() is given in Listing 12.9. In common with MPI MD case study, additional variables required for the MPI implementation are defined and the MPI setup commences (Algorithm 12.2, Part 3.1) by determining the total number of processors (totalNodes) available and the rank of the current processor (myNode) using the **MPI_Comm_Size()** and **MPI_Comm_Rank()** functions. The properties of the index atom are determined in the same way as given in the serial MC code. Thereafter, the energy of interaction of the index atom is obtained by calculating the interactions of all the other atoms.

In the MPI MD case study, the calculation was simply allocated between the processors by using a step size that was equal to totalNodes. This procedure could be applied in the current case, but instead we have illustrated the alternative that was discussed, but not used, in the MPI MD case study.

LISTING 12.9 Key code details of the getTrialPotE() **method in** ljEnergyMC.cpp.

```
// Additonal variables for MPI implentation
int myNode, totalNodes, lastNode;
int start, end, numCal;
double potECalR, virialCalR;   //Receiving variables for MPI_Allreduce

// Set-up MPI
MPI_Comm_size(MPI_COMM_WORLD, &totalNodes);
MPI_Comm_rank(MPI_COMM_WORLD, &myNode);

lastNode = totalNodes - 1;     // Node count starts at 0
numCal = num/totalNodes;    // Distribute calculation evenly between nodes
start = myNode*numCal;       // Different starting point for each node

if(myNode == lastNode)         // Different end point for each node
  end = num;
else
  end   = (myNode + 1)* numCal;

// Calculate energy experienced by the selected atom (index)
// with all other distinct pairs

// Properties of index atom.
kindi = atom[index]->getType();
tempPos = atom[index]->getTrialPosition();
rXi   = tempPos[0];
rYi   = tempPos[1];
rZi   = tempPos[2];

potE = 0.0;

// Loop over all the other atoms
for(j = start; j < end; j++)
{
  // Exclude self-interaction
  if(j != index)
  {
     kindj = atom[j]->getType();
     tempPosj = atom[j]->getPosition();
     rXj   = tempPosj[0];
     rYj   = tempPosj[1];
     rZj   = tempPosj[2];

     // Calculate pair separation
     rXij = rXi - rXj;
     rYij = rYi - rYj;
     rZij = rZi - rZj;

     // Apply periodic boundary conditions
     rXij -= length * nearestInt(rXij, length);
     rYij -= length * nearestInt(rYij, length);
     rZij -= length * nearestInt(rZij, length);
```

```
    rijSq = rXij * rXij + rYij * rYij + rZij * rZij;

    // Calculate forces for separations below the cutoff
    rCutSq = rCut[kindi][kindj] * rCut[kindi][kindj];

    if (rijSq <= rCutSq)
    {
      // Determine potential between i and j
       sigmaSq = sigma[kindi][kindj] * sigma[kindi][kindj];
       sigma2  = sigmaSq/rijSq;
       sigma6  = sigma2 * sigma2 * sigma2;
       sigma12 = sigma6 * sigma6;
       pot     = sigma12 - sigma6;
       eps     = epsilon[kindi][kindj];
       potE    += eps * pot;
    } // end rijSq if

  }  // end j != index if

} // end j for

// MPI reduction
MPI_Allreduce(&potE, &potECalR, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

return 4*potECalR;
```

At the end of the energy evaluation (Algorithm 12.2, Part 3.3), **MPI_Allreduce()** is called to determine potECalR. In contrast, an additional reduction for the forces was required in the MPI MD case study. The absence of this contribution explains why no additional force variables were required in the atom class. The MPI implementation also does not require the additional position vector in the atom class because the positions are updated outside the energy evaluation.

12.7 Case study: relative performance of MD and MC MPI and GPU codes

It is of interest to determine the performance of the parallel codes compared to the serial implementations. To achieve this, we have tested the code against a common benchmark condition for the LJ dense supercritical fluid ($T^* = 1.5$, $\rho^* = 0.6$, $E^* = -1.6088$ using $r_{cut} = 3.4\ \sigma$) for a large range of N. A relatively small run length was used for the benchmark comparisons (8000 overall, with 6000 for equilibration) because of the very long execution times of the serial code for large values of N. Much larger run lengths are used for actual production runs (Chapter 8).

12.7.1 Performance of MD and MC GPU codes

The speed-up is defined as an execution time of the serial code divided by the execution time of the GPU code. The speed-up obtained for the MD GPU code

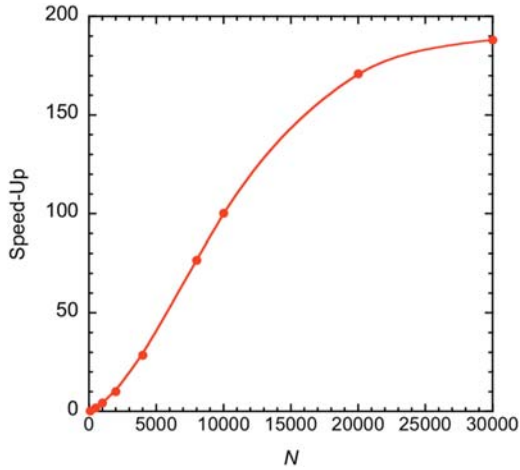


FIGURE 12.1 Speed-up of GPU MD code (32 threads) compared to serial code versus N for the LJ fluid (see text for details). The line through the points is only for guidance.

is illustrated in Fig. 12.1. For $N < 500$, there is no advantage in using the GPU code compared with serial execution. However, thereafter, the speed-up increases rapidly with N, as the larger values of N take full advantage of the simultaneous launch of multiple threads on the GPU. The MD GPU code provides outstanding performance, nearly reaching a 200-fold speed-up compared with the serial code. This would allow simulations with much larger values of N, making million atom simulations feasible. It should be noted that these impressive gains were made without any attempts to optimize the efficiency of the code, and further improvements are possible. Nonetheless, the performance is broadly consistent with earlier results (Hou et al., 2013; Rapaport, 2022).

In contrast, the performance of the MC GPU code is very much less impressive as illustrated in Fig. 12.2, which show the speed-up versus N. It is apparent that the GPU MC code is only beneficial when $N > 10{,}000$, and the speed-up quickly starts to tail-off. This is a disappointing result, particularly in contrast to the enormous computational advantage of GPUs for MD. A partial explanation for the performance difference between the MD and MC GPU codes is that for each MC cycle, N $(N - 1)$ interactions are calculated by launching the x threads $2N$ times, that is, N energy calculations for an atom in its current location plus N energy calculations of the trial displaced location. The equivalent of a MC cycle is the MD time step. In contrast, for each MD time step, the N $(N - 1)$ interactions are calculated by launching the x threads only once. There is scope to improve the performance of the MC code by taking advantage of algorithms such as a neighbor list (Chapter 6).

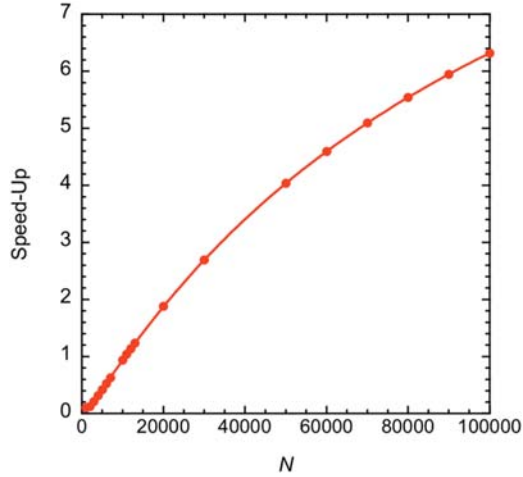


FIGURE 12.2 Speed-up of GPU MC code (32 threads) compared to serial code versus N for the LJ fluid (see text for details). The line through the points is only for guidance.

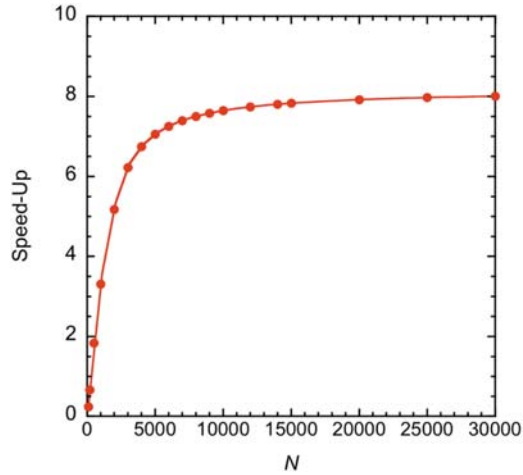


FIGURE 12.3 Speed-up of MC MPI code (32 CPUs) compared to serial code versus N for the LJ fluid (see text for details). The line through the points is only for guidance.

12.7.2 Performance of the MD MPI code

Fig. 12.3 illustrates the speed-up of MD MPI code distributed over 32 processors. In this instance, the maximum theoretical speed-up is 16 because the MPI code performs on average twice as many force evaluations than the serial code because Newton's shortcut is not implemented in the MPI code.

It is apparent from the comparison given in Fig. 12.3 that the MPI implantation is computationally advantageous when $N > 250$. The speed-up increases steadily with N before reaching a plateau value of 8 for $N > 10{,}000$. This is half of the maximum theoretical value, which indicates that there is ample scope to attempt to optimize the code further. However, any gains in efficiency would be dwarfed by much greater speed of the GPU code illustrated in Fig. 12.1.

12.8 Summary

The serial MC and MD codes developed in Chapter 11 can be easily transformed to execute in parallel using either MPI of GPUs. This transformation was achieved by using only a small subset of the available MPI or CUDA functions. When using parallel computation, care must be taken to avoid the racing condition, which is the source of potential inaccuracies. Compared with serial calculations, indicated GPU execution provides a much greater speed-up than is possible with MPI. In contrast, MC simulations do not share the enormous benefits of MD code executed using GPUs. However, these observations come with some important caveats, such as no attempt was made to optimize the codes; the LJ potential is a simple potential that does not include electrostatic interactions; and the calculations were confined to pairwise interactions.

References

Abraham, M.J., Murtolad, T., Schulz, R., Szilárd, P., Smith, J.C., Hess, B., Lindahl, E., 2015. GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1–2, 19–25.

Ansorge, R., 2022. Programming in Parallel with CUDA: A Practical Guide. Cambridge University Press, Cambridge.

Barlas, G., 2015. Multicore and GPU Programming: An Integrated Approach. Morgan Kaufmann, Amsterdam.

Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R., 2001. Parallel Programming in OpenMP. Morgan Kaufmann, San Francisco.

Couturier, R. (Ed.), 2014. Designing Scientific Applications on GPUs. CRC Press, Boca Raton.

Farber, R., 2011. CUDA: Application Design and Development. Morgan Kaufmann, Amsterdam.

Fincham, D., 1987. Parallel computers and molecular simulation. Mol. Sim. 1, 1–45.

Gaster, B.R., Howes, L., Kaeli, D.R., Mistry, P., Schaa, D., 2012. Heterogeneous Computing with OpenCL. Morgan Kaufmann, Amsterdam.

Heermann, D.W., Burkitt, A.N., 1991. Parallel Algorithms in Computational Science. Springer, Berlin.

Heinecke, A., Eckhardt, W., Horsch, M., Bungartz, H.-J., 2015. Supercomputing for Molecular Dynamics Simulations: Handling Multi-Trillion Particles in Nanofluidics. Springer, Heidelberg.

Hou, Q., Li, M., Zhou, Y., Cui, J., Cui, Z., Wang, J., 2013. Molecular dynamics simulations with many-body potentials on multiple GPUs—the implementation, package and performance. Comput. Phys. Commun. 184, 2091–2101.

Hwu, W.-M., 2011. GPU Computing Gems: Emerald Edition. Morgan Kaufmann, Burlington.
Karniadakis, G.E., Kirby II, R.M., 2003. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation. Cambridge University Press, Cambridge.
Kirk, D.B., Hwu, W.-M.W., 2010. Programming Massively Parallel Processor: A Hands-on Approach. Morgan Kaufmann, Amsterdam.
Kohnke, B., Kutzner, C., Grubmüller, J., 2020. A GPU-accelerated fast multiple method for GROMAS: performance and accuracy. J. Chem. Theor. Comput. 16, 6938–6949.
Li, J., Zhou, Z., Sadus, R.J., 2006. Modified force decomposition algorithms for calculating three-body interaction via molecular Dynamics. Comput. Phys. Commun. 175, 384–392.
Li, J., Zhou, Z., Sadus, R.J., 2008. Parallel algorithms for molecular dynamics with induction forces. Comput. Phys. Commun. 178, 384–392.
Mattson, T.G., Sanders, B.A., Massingill, B.L., 2005. Patterns for Parallel Programming. Addison-Wesley, Boston.
Mattson, T.G., He, Y., Koniges, A.E., 2019. The OpenMP Common Core: Making OpenMP Simple Again. MIT Press, Cambridge.
Munshi, A., Gaster, B.R., Mattson, T.G., Fung, J., Ginsburg, D., 2012. OpenCL Programming Guide. Addison-Wesley, Upper Saddle River.
Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C., 2008. GPU computing. Proc. IEEE 96, 879–899.
Pacheco, P.S., 1997. Parallel Programming with MPI. Morgan Kaufmann, San Francisco.
Pacheco, P.S., 2011. An Introduction to Parallel Programming. Morgan Kaufmann, Amsterdam.
Phillips, J.C., Hardy, D.J., Maia, J.D.C., Stone, J.E., Ribeiro, J.V., Bernardi, R.C., Buch, R., Fiorn, G., Hénin, J., Jiang, W., McGreevy, R., Melo, M.C., Radak, B.K., Skeel, R.D., Singharoy, A., Wang, Y., Roux, B., Aksimentiev, A., Luthey-Schulten, Z., Kalé, Schultern, K., Chipot, C., Tajkhorshid, E., 2020. Scalable molecular dynamics on CPU and GPU architectures with NAMD. J. Chem. Phys. 153, 044130.
Ploskas, N., Samaras, M., 2016. GPU Programming in MATLAB. Morgan Kaufmann, Amsterdam.
Quinn, M.J., 2004. Parallel Programming in C with MPI and OpenMP. McGraw-Hill, Boston.
Rapaport, D.C., 2022. GPU molecular dynamics: algorithms and performance. J. Phys. Conf. Ser. 2241, 012007.
Sanders, J., Kandrot, E., 2011. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley, Upper Saddle River.
Scarpino, M., 2012. OpenCL in Action: How to Accelerate Graphics and Computation. Manning, Shelter Island.
Storti, D., Yurtoglu, M., 2016. CUDA for Engineers: An Introduction to High-Performance Parallel Computing. Addison-Wesley, New York.

Appendix 1

Software user's guide

The following is a brief description of the how to use the software that was detailed in Chapters 11 and 12. The software can be downloaded from the book's (https://www.elsevier.com/books-and-journals/book-companion/9780323853989), where any future updates will also be posted as required. The code can be either used or modified for any noncommercial research application. It should be noted that the programs were developed specifically for this book to illustrate the application of object-orientation to molecular simulation. We believe that they are free from significant error. In particular, the serial and parallel codes yield exactly the same results. However, the codes have not had the benefit of extensive testing and their application to mixtures has not been investigated. Therefore the onus is on the user to verify the correctness of the code if it is used in a research application.

A.1 C++ code for the molecular dynamics simulation of Lennard-Jones Atoms in the Microcanonical Ensemble (\md_Ver2.0)

The molecular dynamics code is located in the "md_Ver2.0" directory. The names of both the source files (*.cpp) and header files (*.h) follow closely the names of the class that they represent. Files relevant to molecular dynamics are identified by the "MD" suffix. The source files are: atomMD.cpp, auxfuncMD.cpp, ensembleMD.cpp, forceMD.cpp, gpcMD.cpp, integrtMD.cpp, ljatomMD.cpp, ljforceMD.cpp, mainMD.cpp. md.cpp, nveMD.cpp and simMD.cpp. The header files are atomMD.h, auxfuncMD.h, ensembleMD.h, forceMD.h, gpcMD.h, integrtMD.h, ljatomMD.h, ljforceMD.h, md.h, nveMD.h and simMD.h. Notice that each source file (expect mainMD.cpp) has a corresponding header file. Both the source and header files must be compiled together.

In addition to the source files and header files, there are four data files: md.dat, NVEfileMD.dat, paramLJ.dat and sim.dat. The structure and meaning of the data files are explained in Table A.1. All input data are in reduced units relative to the intermolecular potential parameters.

TABLE A.1 organization of data files for molecular dynamics simulation.

Name of file	Contents	Meaning
sim.dat	1	Simulation choice
md.dat	8000	Total Number of time steps
	6000	Number of equilibration time steps
	10	Size of blocks to average
	0.002	Time step
	1	Integrator method (1 = Gear)
	5	Number of Gear values
	1	Ensemble (1 = NVE ensemble)
	1	Potential (1 = Lennard-Jones)
NVEfileMD.dat	1	Number of components
	256	Number of atoms
	1.0	Mole fraction for each type of atom
	1.0	Mass of each type of atom
	1.5	Temperature
	0.6	Density
paramLJ.dat	1.0 1.0 3.4	Epsilon, sigma and r_{cut}

The data in Table A.1 are for a simulation of a one-component system. However, the program can be executed for a multicomponent mixture without any modification of the source code. To use the program for a multicomponent mixture, changes are required only in the NVEfileMD.dat and paramLJ.dat files. In NVEfileMD.dat, the number of components is specified and the mole fractions and masses of each component must be given. In the paramLJ.dat file, an additional line of data is required for each component. By default, the program will calculate the properties of a three-dimensional fluid, however, calculations for the two-dimensional case can be achieved by simple modification of the source code. The Gear 5-value method is the default integrator. If another Gear integrator is desired, this can be achieved by simply specifying the value in NVEfileMD.dat and changing the constants in the gearCorrect() method.

The output is written to nve_md.out. This file simply contains the average energies and temperature of the ensemble in reduced units. The source code can be modified easily to collect statistics for other thermodynamic properties of the ensemble such as heat capacity, pressure, etc.

A.2 C++ code for the Monte Carlo simulation of Lennard-Jones

Atoms in the Microcanonical Ensemble (\mc_Ver2.0)

The Monte Carlo code is located in the "mc_Ver2.0" directory. The names of both the source files (*.cpp) and header files (*.h) follow closely the names of the class that they represent. Files relevant to Monte Carlo simulation are identified by the "MC" suffix. The source files are: atomMC.cpp, auxfuncMC.cpp, energyMC.cpp, ensembleMC.cpp, ljatomMC.cpp, ljenergyMC.cpp, mainMC.cpp. mc.cpp nveMC.cpp and simMC.cpp. The header files are atomMC.h, auxfuncMC.h, energyMC.h, ensembleMC.h, ljatomMC.h, ljenergyMC.h, mc.h, nveMC.h and simMC.h. Notice that each source file (except mainMC.cpp) has a corresponding header file. Both the source and header files must be compiled together.

In addition to the source files and header files, there are four data files: mc.dat, NVEfileMC.dat, paramLJ.dat and sim.dat. The structure and meaning of the data files are explained in Table A.2. All input data are in reduced units relative to the intermolecular potential parameters.

The data in Table A.2 are for a simulation of a one-component system. However, the program can be executed for a multicomponent mixture without any modification of the source code. To use the program for a multicomponent mixture, changes are required only in the NVEfileMC.dat and paramLJ.dat files. In

TABLE A.2 Organization of data files for Monte Carlo simulation.

Name of file	Contents	Meaning
sim.dat	2	Simulation choice
mc.dat	8000	Total number of cycles
	2000	Number of equilibration cycles
	10	Size of blocks to average
	1	Ensemble (1 = NVE ensemble)
	1	Potential (1 = Lennard-Jones)
NVEfileMC.dat	1	Number of components
	256	Number of atoms
	1.0	Mole fraction for each type of atom
	1.0	Mass of each type of atom
	-1.6088	Total energy of the ensemble
	0.6	Density of the ensemble
paramLJ.dat	1.0 1.0 3.4	Epsilon, sigma and r_{cut}

NVEfileMC.dat, the number of components is specified and the mole fractions and masses of each component must be given. In the paramLJ.dat file, an additional line of data is required for each component. By default, the program will calculate the properties of a three-dimensional fluid, however, calculations for the two-dimensional case can be achieved by simple modification of the source code.

The output is written to nve_mc.out. This file simply contains the average energies and temperature of the ensemble in reduced units. The source code can be modified easily to collect statistics for other thermodynamic properties of the ensemble such as heat capacity, pressure, etc.

A.3 C++ code for the combined Monte Carlo and molecular Dynamics Simulation Program for Lennard-Jones Atoms in the Microcanonical Ensemble (\mcmd_Ver2.0)

The "mcmd_Ver2.0" directory contains the source and header files for both the Monte Carlo and molecular dynamics programs outlined above. The only difference is that that simMD.cpp and simMD.h, or simMC.cpp and simMC.h files are replaced by the files simMCMD.cpp and simMCMD.h, respectively. The data files used are identical to those required for the molecular dynamics and Monte Carlo programs, with the exception that there is a single simMCMD.dat file that replaces either simMC.dat or simMD.dat. This is a name change only and the organisation of the file is unaltered.

A.4 CUDA code for the molecular dynamics simulation of Lennard-Jones Atoms in the Microcanonical Ensemble (\mdCU_Ver2.0)

The MD files for the CUDA implementation are located in the "mdCU_Ver2.0" directory. Details of the .cpp files and associated.h files are mostly identical to the descriptions given in A.1. The exceptions are the presence of the cudaSimKernels,cu, ljForce.cu, cudaSimKernels.h files. In the cudaSimKernels.h, the constant numThreadInBock is used to specify the number of threads used. The organization of the data files is identical to that given in Table A.1.

A.5 CUDA code for the Monte Carlo simulation of Lennard-Jones Atoms in the Microcanonical Ensemble (\mdCU_Ver2.0)

The MC files for CUDA are located in the "mcCU_Ver2.0" directory and are mostly identical to the files used in A.2. The organization of the data files is identical to that given in Table A.2. The additional CUDA files are: cudaSimKernels,cu, ljenergyMC.cu, cudaSimKernels.h files. In the cudaSimKernels.h, the constant numThreadInBock is used to specify the number of threads used.

A.6 MPI code for the molecular dynamics simulation of Lennard-Jones Atoms in the Microcanonical Ensemble (\mdMPI_Ver2.0)

The MPI MD files are located in the directory "mdMPI_Ver2.0." The file structure and the use of the input files are identical to those given in A.1.

A.7 MPI code for the molecular dynamics simulation of Lennard-Jones Atoms in the Microcanonical Ensemble (\mcMPI_Ver2.0)

The MPI MC files are located in the directory "mcMPI_Ver2.0." The file structure and the use of the input files are identical to those given in A.2.

A.8 Compilation

A.8.1 Serial compilation

To generate executable code, all the source files and header files in the respective directories must be compiled together. For example, on a UNIX system, programs for a Monte Carlo (mcprogram.exe), molecular dynamics (mdprogram.exe) and combined (mcmdprogram.exe) simulation can be obtained using the g++ compiler as follows:

```
g++ -o mcprogram.exe mc.cpp *MC.cpp
g++ -o mdprogram.exe md.cpp *MD.cpp
g++ -o mcmdprogram.exe *.cpp
```

Note that, the corresponding header files must be present in the same directory. It is good practice to keep the Monte Carlo and molecular dynamics code in separate directories. However, if they become mixed, the common files can be easily identified by the "MC" and "MD" suffixes.

A.8.2 MPI compilation

The compilation of the MPI files using the g++ compiler requires the addition of the -lmpi compiler option, that is,

```
g++ -o mcprogram.exe mc.cpp *MC.cpp -lmpi
g++ -o mdprogram.exe md.cpp *MD.cpp -lmpi
```

The code, can be executed on, for example, 4 processes by using the command

```
mipirun –np 4./mdprogram.exe
```

A.8.2.1 CUDA compilation

The *.cpp and *.cu files are compiled together using the CUDA (nvcc) compiler. Examples of simple valid compilation statements are:

```
nvcc --fmad=false -o mcprogram.exe mc.cpp *MC.cpp *.cu
nvcc --fmad=false -o mdprogram.exe md.cpp *MD.cpp *.cu
```

For the CUDA compilation it is highly advisable to disable the use of FMAD, which is the default option for the nvcc compiler. This is discussed in Chapter 12.

Appendix 2

List of algorithms

Algorithms described in the text using pseudo code are summarized below. Some additional algorithms are also discussed in the text without using pseudo code.

(*Continued*)

(Continued)

No.	Algorithm description	Page
10.3	Monte Carlo implementation of Gibbs-Duhem integration for a one-component fluid at several different temperatures	377
10.6	ECM algorithm for SLE	393
	Molecular dynamics	
1.2	General outline of a molecular dynamics process	6
5.8	Force calculation using a linked-cell list	177
5.14	Calculation of intermolecular forces for a single time step using the PPPM method	192
5.17	Force interpolation in one-dimension of the particle mesh	195
7.1	General predictor-corrector procedure for solving equations of motion	247
7.2	Example of the implementation of Gear's four-value (3^{rd} order) predictor-corrector algorithm using scaled time derivatives	248
7.3	Original Verlet method for integrating equations of motion	252
7.4	Leap frog Verlet method for integrating equations of motion	253
7.5	Velocity-Verlet method for integrating equations of motion	255
7.6	Beeman-Verlet method for integrating equations of motion	257
7.7	RK4 integration of the equations of motion	259
7.8	*Shake* method for incorporating bond-length constraints	268
7.9	*Rattle* method for incorporating bond-lengthconstraints	274
8.4	A four-value Gear predictor applied independently to both positions and velocities using scaled time derivatives	294
9.5	General molecular dynamics *NVE* ensemble strategy	328
9.6	Molecular dynamics in the *NVE* ensemble using periodic scaling of velocities	329
9.7	Molecular dynamics in the *NVT* ensemble using simple velocity scaling	331
9.8	Molecular dynamics in the *NVT* ensemble using the Andersen thermostat	334
9.9	Molecular dynamics in the *NVT* ensemble using the Nosé-Hoover equations of motion and a predictor-corrector strategy	337
9.10	Molecular dynamics in the *NpH* ensemble using Andersen's method	341
9.11	Molecular dynamics in the *NpT* ensemble using Andersen's method	343
9.12	Molecular dynamics in the *NpT* ensemble using the Nosé-Hoover equations of motion and a predictor-corrector strategy	345
9.13	Molecular dynamics in a μVT ensemble using the Lo-Palmer algorithm	350
9.14	Parallel grand canonical molecular dynamics (PGCMD) algorithm	352
10.2	Gibbs ensemble molecular dynamics using the Lo-Palmer algorithm	369
	Nonequilibrium molecular dynamics	
8.1	General outline of a NEMD process at constant strain rate ($\dot{\gamma}$) using the Gear predictor-corrector integrator	288
8.2	Evaluation of the tensor components to pressure	291
8.3	Calculation of the α thermostatic constant	293
8.5	A four-value Gear corrector applied independently to both positions and velocities implementing *sllod* under constant strain rate, using scaled time derivatives	295

(*Continued*)

(Continued)

This page intentionally left blank

Appendix 3

Notation

In each chapter, all symbols and abbreviations are defined in the text when they are encountered for the first time. The most commonly used notation is summarized below; vectors are distinguished from scalar quantities by the use of bold typeface. Extensive use has been made of pseudo code to describe algorithms and we have endeavored to make the pseudo code non-language specific. The major pseudo code conventions are summarized; We have also made some use of UML notation that is also explained here.

Acronyms and abbreviations

μVT	grand canonical ensemble
a.u.	atomic units
AMBER	assisted model building with energy refinement
AMOEBA	atomic optimized multipole optimized energetics for biomolecular application
ATM	Axilrod-Teller-Mutō
BFW	Barker-Fisher-Watts
CBMC	configurational bias Monte Carlo
CCSD(T)	coupled cluster single and double excitations
CFC	continuous fractional component
CHARMM	chemistry at Harvard macromolecular mechanics
CPMD	Car-Parrinello molecular dynamics
CPU	central processing unit
CPWC	chemical potential weak coupling
CUDA	compute unified device architecture
DCMG	double Gaussian core model
DFT	density functional theory
ECEPP	empirical conformational energy program for peptides
EMD	equilibrium molecular dynamics
ECM	entropy correlation method
erfc	complementary error function
exp-6	exponential-6
FENE	finitely extensible nonlinear elastic
FMAD	floating-point multiply-add

FMM	fast multipole method
FSS	finite size scaling
GCM	Gaussian core model
GDI	Gibbs-Duhem integration
GEMC	Gibbs ensemble Monte Carlo
GPU	graphics processing unit
HF	Hartree-Fock
HFD	Hartree-Fock dispersion
HS	hard sphere
HSA	hard sphere plus attractive
LD	Leonhard-Deiters
LJ	Lennard-Jones
LJ/M	Lennard-Jones/Mie
LR	long-range
LRC	long-range correction
MBF	mean Boltzmann factor
MC	Monte Carlo
MCY	Matsuoka-Clement-Yoshimine
MCYna	MCY nonadditive
MD	molecular dynamics
MM	molecular mechanics
MSS	multistage sampling
MWS	Marcelli-Wang-Sadus
NCC	Niesar-Corongiu-Clementi
NEMD	nonequilibrium molecular dynamics
NpH	isobaric constant-enthalpy ensemble
NpT	isobaric isothermal ensemble
NVE	microcanonical ensemble
NVE**PG**	microcanonical ensemble, with constant **P** and **G**
OMT	object modeling technique
OO	object-orientation
OOA	object-oriented analysis
OOD	object-oriented design
OOP	object-oriented programming
OPLS	optimized potential for liquid simulation
PC	predictor-corrector; personal computer
PGCMD	parallel grand canonical molecular dynamics
rattle	method for handling bond constraints (Algorithm 7.9)
PPPM	particle-particle and particle-mesh
RMSD	root mean squared deviation
RK	Runge-Kutta
RPM	restricted primitive model
SAAP	simplified ab initio atomic potential
SCF	self consistent field
shake	method for handling bond constraints (Algorithm 7.8)
sllod	method for handling equations of motion in NEMD
SLE	solid-liquid equilibria
SPC	simple point charge

SPC/E	SPC/extended
SR	short-range
SS	soft sphere
TIP*x*P	transferable intermolecular potential *x* point
UFF	universal force field
UML	unified modeling language
VLE	vapor-liquid equilibria
WCA	Weeks-Chandler-Anderson

Latin alphabet

$\mathbf{a}$	acceleration vector
A	Helmholtz function; dynamic property; empirical constant
B	empirical constant in intermolecular potentials
b	empirical constant in intermolecular potentials
C	heat capacity; dispersion coefficient; potential constant
d	diameter of an impenetrable hard core; dimensions
D	diffusion coefficient; damping parameters; bond dissociation energy; dipole
E	total energy (Hamiltonian)
$\mathbf{F}$	force vector
F	total number of degrees of freedom
f	force; damping function; degrees of freedom
G	Gibbs function; general variable
$\mathbf{G}$	quantity for the initial position of the center of mass
H	Hamiltonian; enthalpy; hexadecapole
h	Planck's constant
k	Boltzmann's constant
K	kinetic energy; force constant
k	reciprocal vector in Ewald sum
l	bond separation
L	Lagrangian; box length
m	mass
N	number of particles
$\boldsymbol{O}$	octupole
$\mathbf{P}$	linear momentum; pressure tensor
$\mathbb{P}_i$	probability of observing the *i*th state
p	probability; instantaneous pressure
q	charge; position
Q	generalized force; quadrupole
R	maximum level of refinement
$\mathbf{r}$	position vector
r	separation; refinement
S	entropy
t	instantaneous temperature
T	temperature
U	potential energy (internal energy)
V	volume; potential function; rotational barrier
$\mathbf{v}$	velocity vector

w	instantaneous virial
W	internal virial
X	general property
z	adjustable parameter
Z	partition function

Greek alphabet

α	thermal expansion coefficient; constant of motion; repulsive steepness parameter; molecule identifier; atom constant; polarizability; adjustable parameter
β	inverse temperature (1/kT); isothermal compressibility molecular identifier
δ	instantaneous change; Dirac's delta
Γ	oscillator coefficient; gamma function
γ	ratio of polarizabilities; atomic interaction constant; thermal pressure coefficient; general parameter
ε	intermolecular potential energy parameter; relative permittivity
η	viscosity
θ	bond angle
λ	thermal conductivity coefficient; multiple of hard sphere diameter; Lagrange multiplier
μ	chemical potential; dipole moment
ρ	density; distribution function; source distribution
σ	intermolecular potential distance parameter; root mean square deviation
τ	dihedral angle
ν	three-body non-additive coefficient
φ	electrostatic potential
χ	out-of-plane bond angle
ψ	wave function; grand canonical partition function
Ω	phase-space volume
ω	phase-space density

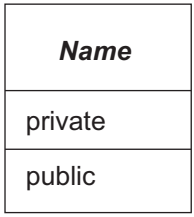
Subscripts and superscripts

0	reference property
a	applied
a,b,i,j	molecule identifiers
ave	average
b	bond
c	constraint; correction
corr	correction
disp	dispersion
e	external
ens	ensemble
m	minimum position
pot	potential
r	residual
rep	repulsive
r	residual property
test	test property

Pseudo code conventions

Code	Explanation
if (*logical test*) *statement* 1 **else**(*optional test*) *statement* 2 **end if**	If some *logical test* is satisfied, *statement* 1 is executed otherwise, another (optional) test is performed. If the second test is satisfied, *statement* 2 is executed.
←	The arrow signifies assignment, e.g., a ← 5 means a is assigned the value of 5.
=	The equals sign is used to signify equivalence in logical tests (compare with assignment above).
//	Start of a comment.
loop i ← 1... **N** *statement* 1 **end loop**	Execute *statement* 1 N times, that is, until *i* is equal to N. The value of *i* is incremented by 1 each time through the loop. In nested loops, the end of a particular loop is specified by incorporating the loop index into the **end loop** statement.
loop *statement* 1 **while** (*logical test*)	The code represented by *statement* 1 is executed until a termination condition is reached. The termination condition is determined at the end of the loop.
loop while (*logical test*) *statement* 1 **end loop**	The code represented by *statement* 1 is executed until a termination condition is reached. The termination condition is determined at the start of the loop.
erfc(*x*)	The complementary error function of *x*.
exp(*x*)	The exponential of *x*.
int(*x*)	The integer component of *x*.
mod (*x,y*)	The remainder obtained when *x* is divided by *y*.
ln(*x*)	The natural logarithm of *x*.
nint(*x*)	The nearest integer to *x*.
pbc(*distance*)	Apply periodic boundary conditions and adjust the *distance* accordingly.
rand()	Generate a random number on the interval of 0 to 1.
real(*x*)	The real component of *x*.

UML conventions

UML notation	Explanation
	UML diagram for a class. The name of the class is given at the top followed by private and public attributes and/or methods.

(*Continued*)

(Continued)

UML notation	Explanation
	Short-hand notation for a class, containing only the name of the class.
	This arrow is used to signify association (needs-a, requires-a, uses-a relationship). The arrow signifies the direction of the relationship
	This arrow is used to signify aggregation (part-of relationship). The diamond is placed on the container class and the arrow points to the parts. For example, an atom is part of a molecule.
	This arrow symbol is used to denote inheritance (is-a relationship) between classes. The parent class is positioned at the head of the arrow and the child class is placed at the bottom. The child class inherits properties from the parent class.

Index

Note: Page numbers followed by "*b*," "*f*," and "*t*" refer to boxes, figures, and tables, respectively.

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

SECOND EDITION

Molecular Simulation of Fluids

Theory, Algorithms, Object-Orientation, and Parallel Computing

RICHARD J. SADUS

A fully updated, practical guide to key knowledge, algorithms, and methods for successfully understanding and applying molecular simulation to fluids

Molecular simulation provides researchers with unique insights into interactions occurring in fluids that are ultimately responsible for their macroscopic behavior. Since the publication of the first edition of *Molecular Simulation of Fluids*, advances in theory, algorithms, and computer hardware have greatly expanded the capabilities of molecular simulation. This second edition has been thoroughly updated and expanded to encompass the latest progress in the field. It provides comprehensive coverage of theory, algorithms, and both serial and parallel implementations.

The book begins with a clear introduction to the field and a review of theoretical foundations, followed by an exploration of both empirical and ab initio intermolecular potentials. Monte Carlo simulation and integrators for molecular dynamics are then discussed in detail, followed by nonequilibrium molecular dynamics and molecular simulation of ensembles and phase equilibria. The book also delves into the use of object-orientation, providing working examples coded in C++. Furthermore, it examines practical parallel molecular simulation algorithms using MPI and GPUs, with a specific focus on CUDA.

Drawing on the extensive experience of its expert author, this book is a practical, accessible guide to this increasingly important topic for all those currently using, or interested in using, molecular simulation to study fluids.

Key Features

- Fully updated and revised to reflect advances in the field, including new chapters on intermolecular potentials and parallel algorithms
- Includes a detailed description of the use of object-orientation in molecular simulation
- Covers the application of the message passing interface (MPI) and graphics processing unit (GPU) programming using the compute unified device architecture (CUDA) to both Monte Carlo and molecular dynamics simulation codes
- Covers a wide range of simulation topics using both Monte Carlo and molecular dynamics approaches
- Provides access to downloadable simulation code, including MPI and GPU/CUDA codes, to encourage practice and support further learning

About the Author

Richard J. Sadus is a Professor at the Department of Computing Technologies, Swinburne University of Technology, Australia. He uses molecular simulation with the aim of accurately predicting the thermodynamic properties and phase behavior of materials. This is greatly facilitated by the implementation of object-oriented approaches and high-performance computing environments. After his Alexander von Humboldt Fellowship in Germany, he joined Swinburne University of Technology. There, he was promoted to a professorship and later held various other leadership roles within the university. Prof. Sadus is the author of an earlier research monograph: *High Pressure Phase Behaviour of Multicomponent Fluid Mixtures*, which is also published by Elsevier. He is a fellow of the American Institute of Chemical Engineers.

ISBN 978-0-323-85398-9

9 780323 853989

ELSEVIER elsevier.com/books-and-journals