Monte Carlo simulations and molecular dynamics *

Daan Frenkel

January 4, 2007

1 Molecular Dynamics Simulations

Molecular Dynamics simulation is a technique to compute the equilibrium and transport properties of a classical many-body system. In this context, the word *classical* means that the nuclear motion of the constituent particles obeys the laws of classical mechanics. This is an excellent approximation for a wide range of materials. Only when we consider the translational or rotational motion of light atoms or molecules (He, H_2, D_2) or vibrational motion with a frequency ν such that $h\nu > k_B T$, should we worry about quantum effects.

Of course, our discussion of this vast subject is necessarily incomplete. Other aspects of the Molecular Dynamics techniques can be found in [1–4].

1.1 Molecular Dynamics: The Idea

Molecular Dynamics simulations are in many respects very similar to real experiments. When we perform a real experiment, we proceed as follows. We prepare a sample of the material that we wish to study. We connect this sample to a measuring instrument (e.g., a thermometer, manometer, or viscosimeter), and we measure the property of interest during a certain time interval. If our measurements are subject to statistical noise (as most measurements are), then the longer we average, the more accurate our measurement becomes. In a Molecular Dynamics simulation, we follow exactly the same approach. First, we prepare a sample: we select a model system consisting of N particles and we solve Newton's equations of motion for this system until the properties of the system no longer change with time (we equilibrate the system). After equilibration, we perform the actual measurement. In fact, some of the most common mistakes that can be made when performing a computer experiment are very similar to the mistakes that can be made in real experiments (e.g., the sample is not prepared correctly, the measurement is too short, the system undergoes an irreversible change during the experiment, or we do not measure what we think).

To measure an observable quantity in a Molecular Dynamics simulation, we must first of all be able to express this observable as a function of the positions and momenta of the particles in the system. For instance, a convenient definition of the temperature in a (classical) many-body system makes use of

*This text is largely, but not exclusively, based on the book Understanding Molecular Simulation by Daan Frenkel and Berend Smit

the equipartition of energy over all degrees of freedom that enter quadratically in the Hamiltonian of the system. In particular for the average kinetic energy per degree of freedom, we have

$$\left\langle \frac{1}{2} m v_\alpha^2 \right\rangle = \frac{1}{2} k_B T. \tag{1.1}$$

In a simulation, we use this equation as an operational definition of the temperature. In practice, we would measure the total kinetic energy of the system and divide this by the number of degrees of freedom $N_f (= 3N - 3$ for a system of N particles with fixed total momentum[1]). As the total kinetic energy of a system fluctuates, so does the instantaneous temperature:

$$T(t) = \sum_{i=1}^{N} \frac{m_i v_i^2(t)}{k_B N_f}. \tag{1.2}$$

The relative fluctuations in the temperature will be of order $1/\sqrt{N_f}$. As N_f is typically of the order of 10^2–10^3, the statistical fluctuations in the temperature are of the order of 5–10 %. To get an accurate estimate of the temperature, one should average over many fluctuations.

1.2 Molecular Dynamics: A Program

The best introduction to Molecular Dynamics simulations is to consider a simple program. The program we consider is kept as simple as possible to illustrate a number of important features of Molecular Dynamics simulations.

The program is constructed as follows:

1. We read in the parameters that specify the conditions of the run (e.g., initial temperature, number of particles, density, time step).
2. We initialize the system (i.e., we select initial positions and velocities).
3. We compute the forces on all particles.
4. We integrate Newton's equations of motion. This step and the previous one make up the core of the simulation. They are repeated until we have computed the time evolution of the system for the desired length of time.
5. After completion of the central loop, we compute and print the averages of measured quantities, and stop.

Algorithm 1 is a short pseudo-algorithm that carries out a Molecular Dynamics simulation for a simple atomic system. We discuss the different operations in the program in more detail.

1.2.1 Initialization

To start the simulation, we should assign initial positions and velocities to all particles in the system. The particle positions should be chosen compatible with the structure that we are aiming to simulate. In any event, the particles should not be positioned at positions that result in an appreciable overlap of

[1] Actually, if we define the temperature of a microcanonical ensemble through $(k_B T)^{-1} = (\partial \ln \Omega / \partial E)$, then we find that, for a d-dimensional system of N atoms with fixed total momentum, $k_B T$ is equal to $2E/(d(N-1)-2)$.

Algorithm 1 (A Simple Molecular Dynamics Program)

```
program md                      simple MD program

call init                       initialization
t=0
do while (t.lt.tmax)            MD loop
   call force(f,en)             determine the forces
   call integrate(f,en)         integrate equations of motion
   t=t+delt
   call sample                  sample averages
enddo
stop
end
```

Comment to this algorithm:

1. *Subroutines* `init`, `force`, `integrate`, *and* `sample` *will be described in Algorithms 2, 3, and 4, respectively. Subroutine* `sample` *is used to calculate averages like pressure or temperature.*

the atomic or molecular cores. Often this is achieved by initially placing the particles on a cubic lattice.

In the present case (Algorithm 2), we have chosen to start our run from a simple cubic lattice. Assume that the values of the density and initial temperature are chosen such that the simple cubic lattice is mechanically unstable and melts rapidly. First, we put each particle on its lattice site and then we attribute to each velocity component of every particle a value that is drawn from a uniform distribution in the interval $[-0.5, 0.5]$. This initial velocity distribution is Maxwellian neither in shape nor even in width. Subsequently, we shift all velocities, such that the total momentum is zero and we scale the resulting velocities to adjust the mean kinetic energy to the desired value. We know that, in thermal equilibrium, the following relation should hold:

$$\left\langle v_\alpha^2 \right\rangle = k_B T/m, \tag{1.3}$$

where v_α is the α component of the velocity of a given particle. We can use this relation to define an instantaneous temperature at time t $T(t)$:

$$k_B T(t) \equiv \sum_{i=1}^{N} \frac{m v_{\alpha,i}^2(t)}{N_f}. \tag{1.4}$$

Clearly, we can adjust the instantaneous temperature $T(t)$ to match the desired temperature T by scaling all velocities with a factor $(T/T(t))^{1/2}$. This initial setting of the temperature is not particularly critical, as the temperature will change anyway during equilibration.

As will appear later, we do not really use the velocities themselves in our algorithm to solve Newton's equations of motion. Rather, we use the positions of all particles at the present (`x`) and previous (`xm`) time steps, combined with our knowledge of the force (`f`) acting on the particles, to predict the positions at the next time step. When we start the simulation, we must bootstrap this pro-

Algorithm 2 (Initialization of a Molecular Dynamics Program)

```
subroutine init              initialization of MD program
sumv=0
sumv2=0
do i=1,npart
  x(i)=lattice_pos(i)        place the particles on a lattice
  v(i)=(ranf()-0.5)          give random velocities
  sumv=sumv+v(i)             velocity center of mass
  sumv2=sumv2+v(i)**2        kinetic energy
enddo
sumv=sumv/npart              velocity center of mass
sumv2=sumv2/npart            mean squared velocity
fs=sqrt(3*temp/sumv2)        scale factor of the velocities
do i=1,npart                 set desired kinetic energy and set
  v(i)=(v(i)-sumv)*fs        velocity center of mass to zero
  xm(i)=x(i)-v(i)*dt         position previous time step
enddo
return
end
```

Comments to this algorithm:

1. *Function* `lattice_pos` *gives the coordinates of lattice position* `i` *and* `ranf()` *gives a uniformly distributed random number. We do not use a Maxwell-Boltzmann distribution for the velocities, on equilibration it will become a Maxwell-Boltzmann distribution.*

2. *In computing the number of degrees of freedom, we assume a three-dimensional system (in fact, we approximate* N_f *by* $3N$*).*

cedure by generating approximate previous positions. Without much consideration for any law of mechanics but the conservation of linear momentum, we approximate `x` for a particle in a direction by `xm(i) = x(i) - v(i)*dt`. Of course, we could make a better estimate of the true previous position of each particle. But as we are only bootstrapping the simulation, we do not worry about such subtleties.

1.2.2 The Force Calculation

What comes next is the most time-consuming part of almost all Molecular Dynamics simulations: the calculation of the force acting on every particle. If we consider a model system with pairwise additive interactions (as we do in the present case), we have to consider the contribution to the force on particle `i` due to all its neighbors. If we consider only the interaction between a particle and the nearest image of another particle, this implies that, for a system of N particles, we must evaluate $N \times (N-1)/2$ pair distances.

This implies that, if we use no tricks, the time needed for the evaluation of the forces scales as N^2. There exist efficient techniques to speed up the evaluation of both short-range and long-range forces in such a way that the computing time scales as N, rather than N^2. In Appendix C of ref. [5], we describe some of the more common techniques to speed up the simulations.

Algorithm 3 (Calculation of the Forces)

```
subroutine force(f,en)                 determine the force
en=0                                   and energy
do i=1,npart
   f(i)=0                              set forces to zero
enddo
do i=1,npart-1
  do j=i+1,npart                       loop over all pairs
     xr=x(i)-x(j)
     xr=xr-box*nint(xr/box)            periodic boundary conditions
     r2=xr**2
     if (r2.lt.rc2) then               test cut-off
       r2i=1/r2
       r6i=r2i**3
       ff=48*r2i*r6i*(r6i-0.5)         Lennard-Jones potential
       f(i)=f(i)+ff*xr                 update force
       f(j)=f(j)-ff*xr
       en=en+4*r6i*(r6i-1)-ecut        update energy
     endif
  enddo
enddo
return
end
```

Comments to this algorithm:

1. *For efficiency reasons the factors 4 and 48 are usually taken out of the force loop and taken into account at the end of the calculation for the energy.*

2. *The term* ecut *is the value of the potential at* $r = r_c$*; for the Lennard-Jones potential, we have*

$$\texttt{ecut} = 4\left(\frac{1}{r_c^{12}} - \frac{1}{r_c^{6}}\right).$$

Although the examples in this appendix apply to Monte Carlo simulations, the same techniques can also be used in a Molecular Dynamics simulation. However, in the present, simple example (see Algorithm 3) we will not attempt to make the program particularly efficient and we shall, in fact, consider all possible pairs of particles explicitly.

We first compute the current distance in the x, y, and z directions between each pair of particles i and j. These distances are indicated by xr. As in the Monte Carlo case, we use periodic boundary conditions. In the present example, we use a cutoff at a distance r_c in the explicit calculation of intermolecular interactions, where r_c is chosen to be less than half the diameter of the periodic box. In that case we can always limit the evaluation of intermolecular interactions between i and j to the interaction between i and the nearest periodic image of j.

In the present case, the diameter of the periodic box is denoted by box. If we use simple cubic periodic boundary conditions, the distance in any direction between i and the nearest image of j should always be less (in absolute value) than box/2. A compact way to compute the distance between i and

the nearest periodic image of `j` uses the nearest integer function (`nint(x)` in `FORTRAN`). The `nint` function simply rounds a real number to the nearest integer.[2] Starting with the x-distance (say) between `i` and any periodic image of `j`, `xr`, we compute the x-distance between `i` and the nearest image of `j` as `xr=xr-box*nint(xr/box)`. Having thus computed all Cartesian components of $\mathbf{r}_{ij}$, the vector distance between `i` and the nearest image of `j`, we compute r_{ij}^2 (denoted by `r2` in the program). Next we test if r_{ij}^2 is less than r_c^2, the square of the cutoff radius. If not, we immediately skip to the next value of `j`. It perhaps is worth emphasizing that we do not compute $|r_{ij}|$ itself, because this would be both unnecessary and expensive (as it would involve the evaluation of a square root).

If a given pair of particles is close enough to interact, we must compute the force between these particles, and the contribution to the potential energy. Suppose that we wish to compute the x-component of the force

$$\begin{aligned} f_x(r) &= -\frac{\partial u(r)}{\partial x} \\ &= -\left(\frac{x}{r}\right)\left(\frac{\partial u(r)}{\partial r}\right). \end{aligned}$$

For a Lennard-Jones system (in reduced units),

$$f_x(r) = \frac{48x}{r^2}\left(\frac{1}{r^{12}} - 0.5\frac{1}{r^6}\right).$$

1.2.3 Integrating the Equations of Motion

Now that we have computed all forces between the particles, we can integrate Newton's equations of motion. Algorithms have been designed to do this. Some of these will be discussed in a bit more detail. In the program (Algorithm 4), we have used the so-called Verlet algorithm. This algorithm is not only one of the simplest, but also usually the best.

To derive it, we start with a Taylor expansion of the coordinate of a particle, around time t,

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{f(t)}{2m}\Delta t^2 + \frac{\Delta t^3}{3!}\dddot{r} + \mathcal{O}(\Delta t^4),$$

similarly,

$$r(t-\Delta t) = r(t) - v(t)\Delta t + \frac{f(t)}{2m}\Delta t^2 - \frac{\Delta t^3}{3!}\dddot{r} + \mathcal{O}(\Delta t^4).$$

Summing these two equations, we obtain:

$$r(t+\Delta t) + r(t-\Delta t) = 2r(t) + \frac{f(t)}{m}\Delta t^2 + \mathcal{O}(\Delta t^4)$$

or

$$r(t+\Delta t) \approx 2r(t) - r(t-\Delta t) + \frac{f(t)}{m}\Delta t^2. \tag{1.5}$$

The estimate of the new position contains an error that is of order Δt^4, where Δt is the time step in our Molecular Dynamics scheme. Note that the

[2] Unfortunately, many FORTRAN compilers yield very slow `nint` functions. It is often cheaper to write your own code to replace the `nint` library routine.

Algorithm 4 (Integrating the Equations of Motion)

```
subroutine integrate(f,en)          integrate equations of motion
sumv=0
sumv2=0
do i=1,npart                        MD loop
   xx=2*x(i)-xm(i)+delt**2*f(i)     Verlet algorithm (1.5)
   vi=(xx-xm(i))/(2*delt)           velocity (1.6)
   sumv=sumv+vi                     velocity center of mass
   sumv2=sumv2+vi**2                total kinetic energy
   xm(i)=x(i)                       update positions previous time
   x(i)=xx                          update positions current time
enddo
temp=sumv2/(3*npart)                instantaneous temperature
etot=(en+sumv2)/(2*npart)           total energy per particle
return
end
```

Comments to this algorithm:

1. *The total energy* `etot` *should remain approximately constant during the simulation. A drift of this quantity may signal programming errors. It therefore is important to monitor this quantity. Similarly, the velocity of the center of mass* `sumv` *should remain zero.*

2. *In this subroutine we use the Verlet algorithm (1.5) to integrate the equations of motion. The velocities are calculated using equation (1.6).*

Verlet algorithm does not use the velocity to compute the new position. One, however, can derive the velocity from knowledge of the trajectory, using

$$r(t + \Delta t) - r(t - \Delta t) = 2v(t)\Delta t + \mathcal{O}(\Delta t^3)$$

or

$$v(t) = \frac{r(t + \Delta t) - r(t - \Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^2). \tag{1.6}$$

This expression for the velocity is only accurate to order Δt^2. However, it is possible to obtain more accurate estimates of the velocity (and thereby of the kinetic energy) using a Verlet-like algorithm (i.e., an algorithm that yields trajectories identical to the one given by equation (1.5)). In our program, we use the velocities only to compute the kinetic energy and, thereby the instantaneous temperature.

Now that we have computed the new positions, we may discard the positions at time $t - \Delta t$. The current positions become the old positions and the new positions become the current positions.

After each time step, we compute the current temperature (`temp`), the current potential energy (`en`) calculated in the force loop, and the total energy (`etot`). Note that the total energy should be conserved.

This completes the introduction to the Molecular Dynamics method. The reader should now be able to write a basic Molecular Dynamics program for liquids or solids consisting of spherical particles. In what follows, we shall do

two things. First of all, we discuss, in a bit more detail, the methods available to integrate the equations of motion. Next, we discuss measurements in Molecular Dynamics simulations.

1.3 Equations of Motion

It is obvious that a good Molecular Dynamics program requires a good algorithm to integrate Newton's equations of motion. In this sense, the choice of algorithm is crucial. However, although it is easy to recognize a *bad* algorithm, it is not immediately obvious what criteria a *good* algorithm should satisfy. Let us look at the different points to consider.

Although, at first sight, speed seems important, it is usually not very relevant because the fraction of time spent on integrating the equations of motion (as opposed to computing the interactions) is small, at least for atomic and simple molecular systems.

Accuracy for large time steps is more important, because the longer the time step that we can use, the fewer evaluations of the forces are needed per unit of simulation time. Hence, this would suggest that it is advantageous to use a sophisticated algorithm that allows use of a long time step.

Algorithms that allow the use of a large time step, achieve this by storing information on increasingly higher-order derivatives of the particle coordinates. As a consequence, they tend to require more memory storage. For a typical simulation, this usually is not a serious drawback because, unless one considers very large systems, the amount of memory needed to store these derivatives is small compared to the total amount available even on a normal workstation.

Energy conservation is an important criterion, but actually we should distinguish two kinds of energy conservation; namely, short time and long time. The sophisticated higher-order algorithms tend to have very good energy conservation for short times (i.e., during a few time steps). However, they often have the undesirable feature that the overall energy drifts for long times. In contrast, Verlet-style algorithms tend to have only moderate short-term energy conservation but little long-term drift.

It would seem to be most important to have an algorithm that accurately predicts the trajectory of all particles for both short and long times. In fact, no such algorithm exists. For essentially all systems that we study by MD simulations, we are in the regime where the trajectory of the system through phase space (i.e., the 6N-dimensional space spanned by all particle coordinates and momenta) depends sensitively on the initial conditions. This means that two trajectories that are initially very close will diverge exponentially as time progresses. We can consider the integration error caused by the algorithm as the source for the initial small difference between the "true" trajectory of the system and the trajectory generated in our simulation. We should expect that any integration error, no matter how small, will always cause our simulated trajectory to diverge exponentially from the true trajectory compatible with the same initial conditions. This so-called Lyapunov instability (see section 1.3.1) would seem to be a devastating blow to the whole idea of Molecular Dynamics simulations but we have good reasons to assume that even this problem need not be serious.

Clearly, this statement requires some clarification. First of all, one should realize that the aim of an MD simulation is *not* to predict precisely what will happen to a system that has been prepared in a precisely known initial condition: we are always interested in statistical predictions. We wish to predict the average behavior of a system that was prepared in an initial state about which

we know something (e.g., the total energy) but by no means everything. In this respect, MD simulations differ fundamentally from numerical schemes to predict the trajectory of satellites through space: in the latter case, we really wish to predict the true trajectory. We cannot afford to launch an ensemble of satellites and make statistical predictions about their destination. However, in MD simulations, statistical predictions are good enough. Still, this would not justify the use of inaccurate trajectories unless the trajectories obtained numerically, in some sense, are close to true trajectories.

This latter statement that is generally believed to be true although, to our knowledge, it has not been proven for any class of systems that is of interest for MD simulations. However, considerable numerical evidence (see, e.g., [6]) suggest that there exist so-called shadow orbits. A shadow orbit is a true trajectory of a many-body system that closely follows the numerical trajectory for a time that is long compared to the time it takes the Lyapunov instability to develop. Hence, the results of our simulation are representative of a true trajectory in phase space, even though we cannot tell a priori which. Surprisingly (and fortunately), it appears that shadow orbits are better behaved (i.e., track the numerical trajectories better) for systems in which small differences in the initial conditions lead to an exponential divergence of trajectories than for the, seemingly, simpler systems that show no such divergence [6]. In spite of this reassuring evidence, it should be emphasized that it is just evidence and not proof. Hence, our trust in Molecular Dynamics simulation as a tool to study the time evolution of many-body systems is based largely on belief. To conclude this discussion, let us say that there is clearly still a corpse in the cupboard. We believe this corpse will not haunt us, and we quickly close the cupboard. For more details, the reader is referred to [7,8].

Newton's equation of motion are time reversible, and so should be our algorithms. In fact, many algorithms (for instance the predictor-corrector schemes, see Appendix E of ref. [5], and many of the schemes used to deal with constraints) are *not* time reversible. That is, future and past phase space coordinates do not play a symmetric role in such algorithms. As a consequence, if one were to reverse the momenta of all particles at a given instant, the system would not trace back its trajectory in phase space, even if the simulation would be carried out with infinite numerical precision. Only in the limit of an infinitely short time step will such algorithms become reversible. However, what is more important, many seemingly reasonable algorithms differ in another crucial respect from Hamilton's equation of motion: true Hamiltonian dynamics leaves the magnitude of any volume element in phase space unchanged, but many numerical schemes, in particular those that are not time reversible, do not reproduce this area-preserving property. This may sound like a very esoteric objection to an algorithm, but it is not. Again, without attempting to achieve a rigorous formulation of the problem, let us simply note that all trajectories that correspond to a particular energy E are contained in a (hyper) volume Ω in phase space. If we let Hamilton's equation of motion act on all points in this volume (i.e., we let the volume evolve in time), then we end up with exactly the same volume. However, a non-area-preserving algorithm will map the volume Ω on another (usually larger) volume Ω'. After sufficiently long times, we expect that the non-area-preserving algorithm will have greatly expanded the volume of our system in phase space. This is not compatible with energy conservation. Hence, it is plausible that nonreversible algorithms will have serious long-term energy drift problems. Reversible, area-preserving algorithms will not change the magnitude of the volume in phase space. This property is not sufficient to guarantee the absence of long-term en-

ergy drift, but it is at least compatible with it. It is possible to check whether an algorithm is area preserving by computing the Jacobian associated with the transformation of old to new phase space coordinates.

Finally, it should be noted that even when we integrate a time-reversible algorithm, we shall find that the numerical implementation is hardly ever truly time reversible. This is so, because we work on a computer with finite machine precision using floating-point arithmetic that results in rounding errors (on the order of the machine precision).

In the remainder of this section, we shall discuss some of these points in more detail. Before we do so, let us first consider how the Verlet algorithm scores on these points. First of all, the Verlet algorithm is fast. But we had argued that this is relatively unimportant. Second, it is not particularly accurate for long time steps. Hence, we should expect to compute the forces on all particles rather frequently. Third, it requires about as little memory as is at all possible. This is useful when we simulate very large systems, but in general it is not a crucial advantage. Fourth, its short-term energy conservation is fair (in particular in the versions that use a more accurate expression for the velocities) but, more important, it exhibits little long-term energy drift. This is related to the fact that the Verlet algorithm is time reversible and area preserving. In fact, although the Verlet algorithm does not conserve the total energy of this system exactly, strong evidence indicates that it does conserve a pseudo-Hamiltonian approaching the true Hamiltonian in the limit of infinitely short time steps. The accuracy of the trajectories generated with the Verlet algorithm is not impressive. But then, it would hardly help to use a better algorithm. Such an algorithm may postpone the unavoidable exponential growth of the error in the trajectory by a few hundred time steps (see section 1.3.1), but no algorithm is good enough that it will keep the trajectories close to the true trajectories for a time comparable to the duration of a typical Molecular Dynamics run.[3]

1.3.1 Lyapunov Instability

To end this discussion of algorithms, we wish to illustrate the extreme sensitivity of the trajectories to small differences in initial conditions. Let us consider the position ($\mathbf{r}^N$) of one of the N particles at time t. This position is a function of the initial positions and momenta at $t = 0$:

$$\mathbf{r}(t) = f\left[\mathbf{r}^N(0), \mathbf{p}^N(0); t\right].$$

Let us now consider the position at time t that would result if we perturbed the initial conditions (say, some of the momenta) by a small amount ϵ. In that case, we would obtain a different value for r at time t:

$$\mathbf{r}'(t) = f\left[\mathbf{r}^N(0), \mathbf{p}^N(0) + \boldsymbol{\epsilon}; t\right].$$

We denote the difference between $\mathbf{r}(t)$ and $\mathbf{r}'(t)$ by $\Delta\mathbf{r}(t)$. For sufficiently short times, $\Delta\mathbf{r}(t)$ is linear in ϵ. However, the coefficient of the linear dependence diverges exponentially; that is

$$|\Delta\mathbf{r}(t)| \sim \epsilon \exp(\lambda t). \tag{1.7}$$

This so-called Lyapunov instability of the trajectories is responsible for our inability to accurately predict a trajectory for all but the shortest simulations. The

[3] Error free integration of the equations of motion is possible for certain discrete models, such as lattice-gas cellular automata. But these models do not follow Newton's equation of motion.

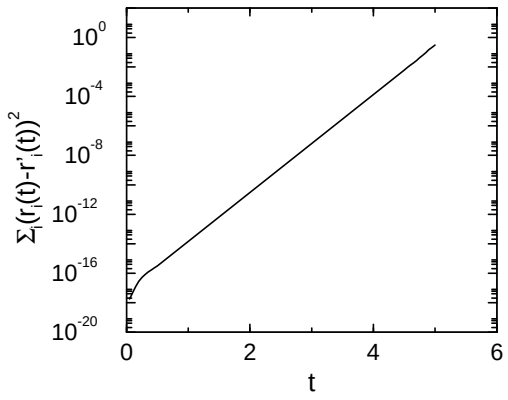

Figure 1: Illustration of the Lyapunov instability in a simulation of a Lennard-Jones system. The figure shows the time dependence of the sum of squared distances between two trajectories that were initially very close (see text). The total length of the run in reduced units was 5, which corresponds to 1000 time steps. Note that, within this relatively short time, the two trajectories become essentially uncorrelated.

exponent λ is called the Lyapunov exponent (more precisely, the largest Lyapunov exponent; there are more such exponents, 6N in fact, but the largest dominates the long-time exponential divergence of initially close trajectories). Suppose that we wish to maintain a certain bound Δ_{max} on $|\Delta\mathbf{r}(t)|$, in the interval $0 < t < t_{max}$. How large an initial error (ϵ) can we afford? From equation (1.7), we deduce

$$\epsilon \sim \Delta_{max} \exp(-\lambda t_{max}).$$

Hence, the acceptable error in our initial conditions decreases exponentially with t_{max}, the length of the run. To illustrate that this effect is real, we show the result of two almost identical simulations: the second differs from the first in that the x components of the velocities of 2 particles (out of 1000) have been changed by $+10^{-10}$ and -10^{-10} (in reduced units). We monitor the sum of the squares of the differences of the positions of all particles:

$$\sum_{i=1}^{N} |\mathbf{r}_i(t) - \mathbf{r}'_i(t)|^2 .$$

As can be seen in Figure 1, this measure of the distance does indeed grow exponentially with time.

After 1000 time steps, the two systems that were initially very close have become very nearly uncorrelated. It should be stressed this run was performed using perfectly normal parameters (density, temperature, time step). The only unrealistic thing about this simulation is that it is extremely short. Most Molecular Dynamics simulations require many tens, if not hundreds, or thousands of time steps.

1.4 Computer Experiments

Now that we have a working Molecular Dynamics program, we wish to use it to "measure" interesting properties of many-body systems. What properties

are interesting? First of all, of course, those quantities that can be compared with real experiments. Simplest among these are the thermodynamic properties of the system under consideration, such as, the temperature T, the pressure P, and the heat capacity C_V. As mentioned earlier, the temperature is measured by computing the average kinetic energy per degree of freedom. For a system with f degrees of freedom, the temperature T is given by

$$T = \frac{\langle 2\mathcal{K} \rangle}{f}. \tag{1.8}$$

There are several different (but equivalent) ways to measure the pressure of a classical N-body system. The most common among these is based on the virial equation for the pressure. For pairwise additive interactions, we can write (see, e.g., [9])

$$P = \rho k_B T + \frac{1}{dV} \left\langle \sum_{i<j} \mathbf{f}(\mathbf{r}_{ij}) \cdot \mathbf{r}_{ij} \right\rangle, \tag{1.9}$$

where d is the dimensionality of the system, and $\mathbf{f}(\mathbf{r}_{ij})$ is the force between particles i and j at a distance $\mathbf{r}_{ij}$. Note that this expression for the pressure has been derived for a system at constant N, V, and T, whereas our simulations are performed at constant N, V, and E. In fact, the expression for the pressure in the microcanonical ensemble (constant N, V, E) is not identical to the expression that applies to the canonical (constant N, V, T) ensemble. Lebowitz *et al.* [10] have derived a general procedure to convert averages from one ensemble to another. A more recent (and more accessible) description of these inter-ensemble transformations has been given by Allen in [4]. An example of a relation derived by such a transformation is the expression for the heat capacity at constant volume, as obtained from the fluctuations in the kinetic energy in the microcanonical ensemble:

$$\langle \mathcal{K}^2 \rangle_{NVE} - \langle \mathcal{K} \rangle^2_{NVE} = \frac{3k_B^2 T^2}{2N} \left(1 - \frac{3k_B}{2C_V} \right). \tag{1.10}$$

However, one class of thermodynamic functions cannot be measured directly in a simulation, in the sense that these properties cannot be expressed as a simple average of some function of the coordinates and momenta of all the particles in the system. Examples of such properties are the entropy S, the Helmholtz free energy F, and the Gibbs free energy G. Separate techniques are required to evaluate such thermal quantities in a computer simulation.

A second class of observable properties are the functions that characterize the local structure of a fluid. Most notable among these is the so-called radial distribution function $g(r)$. The radial distribution function is of interest for two reasons: first of all, neutron and X-ray scattering experiments on simple fluids, and light-scattering experiments on colloidal suspensions, yield information about $g(r)$. Second, $g(r)$ plays a central role in theories of the liquid state. Numerical results for $g(r)$ can be compared with theoretical predictions and thus serve as a criterion to test a particular theory. In a simulation, it is straightforward to measure $g(r)$: it is simply the ratio between the average number density $\rho(r)$ at a distance r from any given atom (for simplicity we assume that all atoms are identical) and the density at a distance r from an atom in an ideal gas at the same overall density. By construction, $g(r) = 1$ in an ideal gas. Any deviation of $g(r)$ from unity reflects correlations between the particles due to the intermolecular interactions.

Both the thermodynamic properties and the structural properties mentioned previously do not depend on the time evolution of the system: they are static equilibrium averages. Such averages can be obtained by Molecular Dynamics simulations or equally well by Monte Carlo simulations. However, in addition to the static equilibrium properties, we can also measure dynamic equilibrium properties in a Molecular Dynamics simulation (but not with a Monte Carlo simulation). At first sight, a dynamic equilibrium property appears to be a contradiction: in equilibrium all properties are independent of time, hence any time dependence in the macroscopic properties of a system seems to be related to nonequilibrium behavior. This is true, but it turns out that the time-dependent behavior of a system that is only weakly perturbed is completely described by the dynamic equilibrium properties of the system. Later, we shall provide a simple introduction to the quantities that play a central role in the theory of time-dependent processes near equilibrium, in particular the so-called time-correlation functions. However, we shall not start with a general description of nonequilibrium processes. Rather we start with a discussion of a simple specific example that allows us to introduce most of the necessary concepts.

1.4.1 Diffusion

Diffusion is the process whereby an initially nonuniform concentration profile (e.g., an ink drop in water) is smoothed in the absence of flow (no stirring). Diffusion is caused by the molecular motion of the particles in the fluid. The macroscopic law that describes diffusion is known as Fick's law, which states that the flux $\mathbf{j}$ of the diffusing species is proportional to the negative gradient in the concentration of that species:

$$\mathbf{j} = -D\boldsymbol{\nabla} c, \tag{1.11}$$

where D, the constant of proportionality, is referred to as the *diffusion coefficient*. In what follows, we shall be discussing a particularly simple form of diffusion, namely, the case that the molecules of the diffusing species are identical to the other molecules but for a label that does not affect the interaction of the labeled molecules with the others. For instance, this label could be a particular polarization of the nuclear spin of the diffusing species or a modified isotopic composition. Diffusion of a labeled species among otherwise identical solvent molecules is called *self-diffusion*.

Let us now compute the concentration profile of the tagged species, under the assumption that, at time $t = 0$, the tagged species was concentrated at the origin of our coordinate frame. To compute the time evolution of the concentration profile, we must combine Fick's law with an equation that expresses conservation of the total amount of labeled material:

$$\frac{\partial c(r,t)}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{j}(r,t) = 0. \tag{1.12}$$

Combining equation (1.12) with equation (1.11), we obtain

$$\frac{\partial^2 c(r,t)}{\partial t^2} - D\nabla^2 c(r,t) = 0. \tag{1.13}$$

We can solve equation (1.13) with the boundary condition

$$c(r,0) = \delta(r)$$

($\delta(r)$ is the Dirac delta function) to yield

$$c(r,t) = \frac{1}{(4\pi Dt)^{d/2}} \exp\left(-\frac{r^2}{4Dt}\right).$$

As before, d denotes the dimensionality of the system. In fact, for what follows we do not need $c(r,t)$ itself, but just the time dependence of its second moment:

$$\left\langle r^2(t)\right\rangle \equiv \int d\mathbf{r}\; c(r,t) r^2,$$

where we have used the fact that we have imposed

$$\int d\mathbf{r}\; c(r,t) = 1.$$

We can directly obtain an equation for the time evolution of $\langle r^2(t)\rangle$ by multiplying equation (1.13) by r^2 and integrating over all space. This yields

$$\frac{\partial}{\partial t}\int d\mathbf{r}\; r^2 c(r,t) = D\int d\mathbf{r}\; r^2 \nabla^2 c(r,t). \tag{1.14}$$

The left-hand side of this equation is simply equal to

$$\frac{\partial \left\langle r^2(t)\right\rangle}{\partial t}.$$

Applying partial integration to the right-hand side, we obtain

$$\begin{aligned}
\frac{\partial \left\langle r^2(t)\right\rangle}{\partial t} &= D\int d\mathbf{r}\; r^2\nabla^2 c(r,t)\\
&= D\int d\mathbf{r}\; \nabla(r^2\nabla c(r,t)) - D\int d\mathbf{r}\; \boldsymbol{\nabla} r^2\cdot\boldsymbol{\nabla} c(r,t)\\
&= D\int d\mathbf{S}\; (r^2\boldsymbol{\nabla} c(r,t)) - 2D\int d\mathbf{r}\; \mathbf{r}\cdot\boldsymbol{\nabla} c(r,t)\\
&= 0 - 2D\int d\mathbf{r}\; (\boldsymbol{\nabla}\cdot\mathbf{r}c(r,t)) + 2D\int d\mathbf{r}\; (\boldsymbol{\nabla}\cdot\mathbf{r})c(r,t)\\
&= 0 + 2dD\int d\mathbf{r}\; c(r,t)\\
&= 2dD.
\end{aligned} \tag{1.15}$$

Equation (1.15) relates the diffusion coefficient D to the width of the concentration profile. This relation was first derived by Einstein. It should be realized that, whereas D is a macroscopic transport coefficient, $\langle r^2(t)\rangle$ has a microscopic interpretation: it is the mean-square distance over which the labeled molecules have moved in a time interval t. This immediately suggests how to measure D in a computer simulation. For every particle i, we measure the distance traveled in time t, $\Delta\mathbf{r}_i(t)$ and we plot the mean square of these distances as a function of the time t:

$$\left\langle \Delta r(t)^2\right\rangle = \frac{1}{N}\sum_{i=1}^{N}\Delta\mathbf{r}_i(t)^2.$$

This plot would look like the one that will be shown later in Figure 5. We should be more specific about what we mean by the displacement of a particle

in a system with periodic boundary conditions. The displacement that we are interested in is simply the time integral of the velocity of the tagged particle:

$$\Delta\mathbf{r}(t) = \int_0^t dt'\ \mathbf{v}(t').$$

In fact, there is a relation that expresses the diffusion coefficient directly in terms of the particle velocities. We start with the relation

$$2D = \lim_{t\to\infty} \frac{\partial \left\langle x^2(t)\right\rangle}{\partial t} \tag{1.16}$$

where, for convenience, we consider only one Cartesian component of the mean-square displacement. If we write $x(t)$ as the time integral of the x component of the tagged-particle velocity, we get

$$\begin{aligned}
\left\langle x^2(t)\right\rangle &= \left\langle \left(\int_0^t dt'\ v_x(t') \right)^2 \right\rangle \\
&= \int_0^t \int_0^t dt' dt''\ \left\langle v_x(t') v_x(t'')\right\rangle \\
&= 2\int_0^t \int_0^{t'} dt' dt''\ \left\langle v_x(t') v_x(t'')\right\rangle . \qquad (1.17)
\end{aligned}$$

The quantity $\langle v_x(t')v_x(t'')\rangle$ is called the velocity autocorrelation function. It measures the correlation between the velocity of a particle at times t' and t''. The velocity autocorrelation function (VACF) is an equilibrium property of the system, because it describes correlations between velocities at different times along an equilibrium trajectory. As equilibrium properties are invariant under a change of the time origin, the VACF depends only on the difference of t' and t''. Hence,

$$\langle v_x(t')v_x(t'')\rangle = \langle v_x(t'-t'')v_x(0)\rangle .$$

Inserting equation (1.17) in equation (1.16), we obtain

$$\begin{aligned}
2D &= \lim_{t\to\infty} 2\int_0^t dt''\ \langle v_x(t-t'')v_x(0)\rangle \\
D &= \int_0^\infty d\tau\ \langle v_x(\tau)v_x(0)\rangle . \qquad (1.18)
\end{aligned}$$

In the last line of equation (1.18) we introduced the coordinate $\tau \equiv t - t''$. Hence, we see that we can relate the diffusion coefficient D to the integral of the velocity autocorrelation function. Such a relation between a transport coefficient and an integral over a time-correlation function is called a *Green-Kubo relation*. Green-Kubo relations have been derived for many other transport coefficients, such as the shear viscosity η,

$$\eta = \frac{1}{Vk_BT}\int_0^\infty dt\ \langle \sigma^{xy}(0)\sigma^{xy}(t)\rangle \tag{1.19}$$

with

$$\sigma^{xy} = \sum_{i=1}^{N}\left(m_i v_i^x v_i^y + \frac{1}{2}\sum_{j\neq i} x_{ij} f_y(r_{ij})\right); \tag{1.20}$$

the thermal conductivity λ_T,

$$\lambda_T = \frac{1}{Vk_B T^2} \int_0^\infty dt \ \langle j_z^e(0) j_z^e(t) \rangle \tag{1.21}$$

with

$$j_z^e = \frac{d}{dt} \sum_{i=1}^{N} z_i \frac{1}{2} \left(m_i v_i^2 + \sum_{j \neq i} v(r_{ij}) \right); \tag{1.22}$$

and electrical conductivity σ_e

$$\sigma_e = \frac{1}{Vk_B T} \int_0^\infty dt \ \left\langle j_x^{el}(0) j_x^{el}(t) \right\rangle \tag{1.23}$$

with

$$j_x^{el} = \sum_{i=1}^{N} q_i v_i^x. \tag{1.24}$$

For details, see, for example, [9]. Time correlation functions can easily be measured in a Molecular Dynamics simulation. It should be emphasized that for classical systems, the Green-Kubo relation for D and the Einstein relation are strictly equivalent. There may be practical reasons to prefer one approach over the other, but the distinction is never fundamental.

1.5 Some Applications

Let us illustrate the results of the previous sections with an example. Like in the section on Monte Carlo simulations we choose the Lennard-Jones fluid as our model system. We use a truncated and shifted potential

$$u^{tr-sh}(r) = \begin{cases} u^{lj}(r) - u^{lj}(r_c) & r \leq r_c \\ 0 & r > r_c \end{cases},$$

where $u^{lj}(r)$ is the Lennard-Jones potential and for these simulations $r_c = 2.5\sigma$ is used.

Case Study 1 (Static Properties of the Lennard-Jones Fluid)
Let us start a simulation with 108 particles on a simple cubic lattice. We give the system an initial temperature $T = 0.728$ and density $\rho = 0.8442$, which is close to the triple (gas-liquid-solid) point of the Lennard-Jones fluid [11–13].

In Figure 2 the evolution of the total energy, kinetic energy, and potential energy is shown. It is important to note that the total energy remains constant and does not show a (slow) drift during the entire simulation. The kinetic and potential energy do change initially (the equilibration period) but during the end of the simulation they oscillate around their equilibrium value. This figure shows that, for the calculation of the average potential energy or kinetic energy, we need ca. 1000 time steps to equilibrate the simulation. The figure also shows significant fluctuations in the potential energy, some of which may take several (100) time steps before they disappear.

Appendix D of ref. [5] shows in detail how to calculate statistical error in the data of a simulation. In this example, we use the method of Flyvbjerg and Petersen [14]. The following operations on the set of data points are performed: we start by calculating of the standard deviation of all the data points, then we group two consecutive data points and determine again the standard deviation

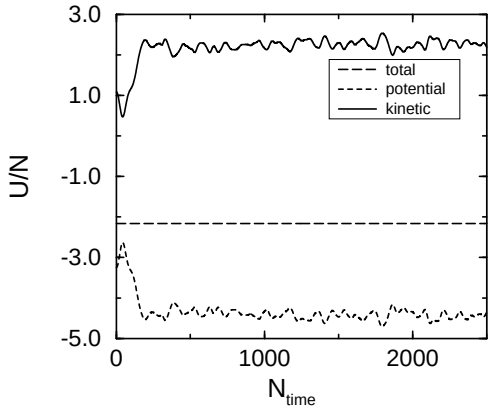

Figure 2: Total, potential, and kinetic energy per particle U/N as a function of the number of time steps N_{time}

of the new, blocked, data set This new data set contains half the number of data points of the original set. The procedure is repeated until there are not enough data points to compute a standard deviation, the number of times we perform this operation is called M. What do we learn from this?

First of all, let us assume that the time between two samples is so large that the data points are uncorrelated. If the data are uncorrelated the standard deviation (as calculated according to the formula in Appendix D of ref. [5], i.e., correcting for the fact we have fewer data points) should be invariant to this blocking operation and we should get a standard deviation that is independent of M. In a simulation, however, the time between two data points is usually too short to obtain a statistically independent sample, as a consequence consecutive data points would be (highly) correlated. If we would calculate a standard deviation using these data, this standard deviation will be too optimistic. The effect of the block operation will be that after grouping two consecutive data points, the correlation between the two new data points will be less. This, however, will increase the standard deviation; the data will have more noise since consecutive data points no longer resemble each other that closely. This decrease of accuracy as a function of the number of blocking operations will continue until we have grouped so many data points that two consecutive point are really uncorrelated. This is exactly the standard deviation we want to determine. It is important to note that we have to ensure that the standard deviations we are looking at are significant; therefore, we have to determine the standard deviation of the error at the same time.

The results of this error calculation for the potential energy is shown in Figure 3, as expected, for a low value of M, the error increases until a plateau is reached. For high values of M, we have only a few data points, which results in a large standard deviation in the error. The advantage of this method is that we have a means of finding out whether or not we have simulated enough; if we do not find such a plateau, the simulation must have been too short. In addition we find a reliable estimate of the standard deviation. The figure also shows the effect of increasing the total length of the simulation by a factor of 4; the statistical error in the potential energy has indeed decreased by a factor of 2.

In this way we obtained the following results. For the potential energy

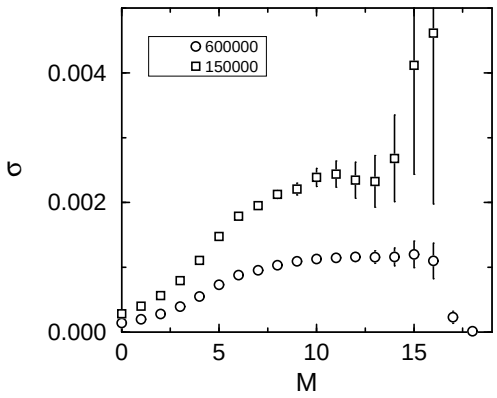

Figure 3: The standard deviation σ in the potential energy as a function of the number of block operations M for a simulation of 150,000 and 600,000 time steps.

$U = -4.4190 \pm 0.0012$ and for the kinetic energy $K = 2.2564 \pm 0.0012$, the latter corresponds to an average temperature of $T = 1.5043 \pm 0.0008$. For the pressure, we have obtained: 5.16 ± 0.02.

In Figure 4, the radial distribution function is shown. This distribution function shows the characteristics of a dense liquid. We can use the radial distribution function to calculate the energy and pressure. The potential energy per particle can be calculated from

$$\begin{aligned} U/N &= \frac{1}{2}\rho \int_0^\infty d\mathbf{r}\, u(r)g(r) \\ &= 2\pi\rho \int_0^\infty dr\, r^2 u(r)g(r) \end{aligned} \tag{1.25}$$

and for the pressure from

$$\begin{aligned} P &= \rho T - \frac{1}{3}\frac{1}{2}\rho^2 \int_0^\infty d\mathbf{r}\, \frac{du(r)}{dr} r g(r) \\ &= \rho T - \frac{2}{3}\pi\rho^2 \int_0^\infty dr\, \frac{du(r)}{dr} r^3 g(r), \end{aligned} \tag{1.26}$$

where $u(r)$ is the pair potential.

Equations (1.25) and (1.26) can be used to check the consistency of the energy and pressure calculations and the determination of the radial distribution function. In our example, we obtained from the radial distribution function for the potential energy $U/N = -4.419$ and for the pressure $P = 5.181$, which is in good agreement with the direct calculation.

Case Study 2 (Dynamic Properties of the Lennard-Jones Fluid)
As an example of a dynamic property we have determined the diffusion coefficient. As shown in the previous section, the diffusion coefficient can be determined from the mean square displacement or from the velocity autocorrelation function.

In Figure 5 the mean-square displacement is shown as a function of the simulation time. From the mean-square displacement we can determine the

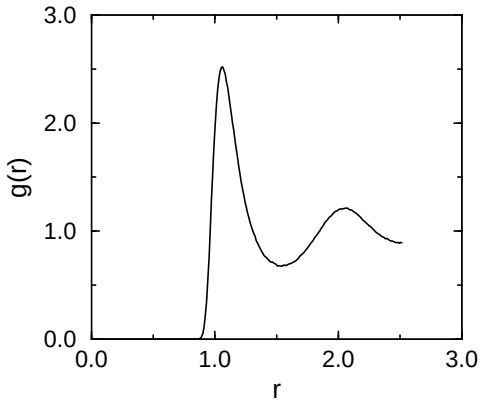

Figure 4: Radial distribution function of a Lennard-Jones fluid close to the triple point: $T = 1.5043 \pm 0.0008$ and $\rho = 0.8442$

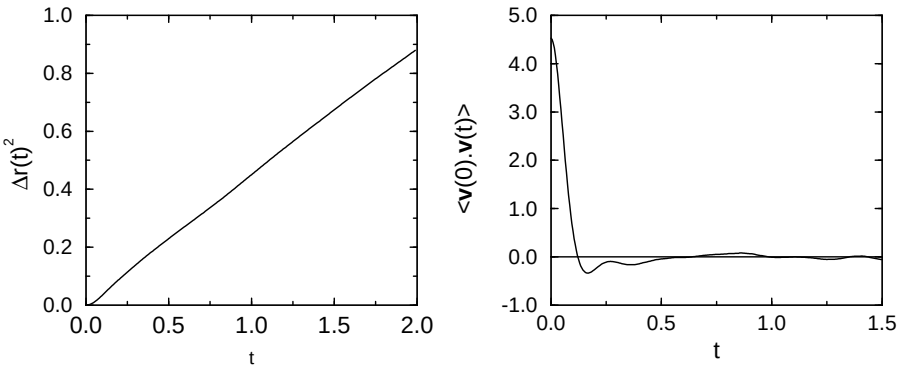

Figure 5: (left) Mean square displacement $\Delta r(t)^2$ as a function of the simulation time t. Note that for long times, $\Delta r(t)^2$ varies linearly with t. The slope is then given by $2dD$, where d is the dimensionality of the system and D the self-diffusion coefficient. (right) Velocity autocorrelation function $\langle \mathbf{v}(0) \cdot \mathbf{v}(t) \rangle$ as a function of the simulation time t.

diffusion using equation (1.16). This equation, however, is valid only in the limit $t \to \infty$. In practice this means that have to verify that we have simulated enough that the mean square displacement is really proportional to t and not another power of t.

The velocity autocorrelation function can be used as an independent route to test the calculation of the diffusion coefficient. The diffusion coefficient follows from equation (1.18). In this equation we have to integrate to $t \to \infty$. Knowing whether or not we have simulated sufficiently to perform this integration reliably is equivalent to the determination of the slope in the mean square displacement. A simple trick is to determine the diffusion coefficient as a function of the truncation of the integration; if a plateau has been reached over a sufficient number of integration limits, the calculation is probably reliable.

2 Monte Carlo simulations

Conceptually, the Molecular Dynamics method is straightforward: we simply compute the time-averaged properties of a many-body system by solving Newton's equation of motion. To understand the Monte Carlo method, we need to discuss some statistical mechanics. I try to limit this discussion to its bare essentials.

2.1 Statistical-mechanical background

We start from the classical expression for the partition function Q:

$$Q = c \int d\mathbf{p}^N d\mathbf{r}^N \exp[-\mathcal{H}(\mathbf{r}^N\ \mathbf{p}^N)/k_B T], \tag{2.1}$$

where $\mathbf{r}^N$ stands for the coordinates of all N particles, and $\mathbf{p}^N$ for the corresponding momenta. The function $\mathcal{H}(\mathbf{q}^N, \mathbf{p}^N)$ is the Hamiltonian of the system. It expresses the total energy of an isolated system as a function of the coordinates and momenta of the constituent particles: $\mathcal{H} = \mathcal{K} + \mathcal{U}$, where $\mathcal{K}$ is the kinetic energy of the system and $\mathcal{U}$ is the potential energy. Finally, c is a constant of proportionality, chosen such that the sum over quantum states approaches the classical partition function in the limit $\hbar \to 0$. For instance, for a system of N identical atoms, $c = 1/(h^{3N} N!)$. The classical equation is

$$\langle A \rangle = \frac{\int d\mathbf{p}^N d\mathbf{r}^N\ A(\mathbf{p}^N, \mathbf{r}^N) \exp[-\beta \mathcal{H}(\mathbf{p}^N, \mathbf{r}^N)]}{\int d\mathbf{p}^N d\mathbf{r}^N\ \exp[-\beta \mathcal{H}(\mathbf{p}^N, \mathbf{r}^N)]}, \tag{2.2}$$

where $\beta = 1/k_B T$. In this equation, the observable A has been expressed as a function of coordinates and momenta. As $\mathcal{K}$ is a quadratic function of the momenta the integration over momenta can be carried out analytically. Hence, averages of functions that depend on momenta only are usually easy to evaluate. The difficult problem is the computation of averages of functions $A(\mathbf{r}^N)$. Only in a few exceptional cases can the multidimensional integral over particle coordinates be computed analytically, in all other cases numerical techniques must be used.

Having thus defined the nature of the numerical problem that we must solve, let us next look at possible solutions. It might appear that the most straightforward approach would be to evaluate $\langle A \rangle$ in equation (2.2) by numerical quadrature, for instance using Simpson's rule. It is easy to see, however, that such a method is completely useless even if the number of independent coordinates DN (D is the dimensionality of the system) is still very small $\mathcal{O}(100)$. Suppose that we plan to carry out the quadrature by evaluating the integrand on a mesh of points in the DN-dimensional configuration space. Let us assume that we take m equidistant points along each coordinate axis. The total number of points at which the integrand must be evaluated is then equal to m^{DN}. For all but the smallest systems this number becomes astronomically large, even for small values of m. For instance, if we take 100 particles in three dimensions, and $m = 5$, then we would have to evaluate the integrand at 10^{210} points! Computations of such magnitude cannot be performed in the known universe. And this is fortunate, because the answer that would be obtained would have been subject to a large statistical error. After all, numerical quadratures work best on functions that are smooth over distances corresponding to the mesh size. But for most intermolecular potentials, the Boltzmann factor in equation (2.2) is a rapidly varying function of the particle coordinates. Hence

an accurate quadrature requires a small mesh spacing (i.e., a large value of m). Moreover, when evaluating the integrand for a dense liquid (say), we would find that for the overwhelming majority of points this Boltzmann factor is vanishingly small. For instance, for a fluid of 100 hard spheres at the freezing point, the Boltzmann factor would be nonzero for 1 out of every 10^{260} configurations!

The closing lines of the previous section suggest that it is in general not possible to evaluate an integral, such as $\int d\mathbf{r}^N \exp[-\beta\mathcal{U}(\mathbf{r}^N)]$, by direct Monte Carlo sampling. However, in many cases, we are not interested in the configurational part of the partition function itself but in averages of the type

$$\langle A \rangle = \frac{\int d\mathbf{r}^N \exp[-\beta\mathcal{U}(\mathbf{r}^N)] A(\mathbf{r}^N)}{\int d\mathbf{r}^N \exp[-\beta\mathcal{U}(\mathbf{r}^N)]}. \tag{2.3}$$

Hence, we wish to know the *ratio* of two integrals. What Metropolis *et al.* [15] showed is that it is possible to devise an efficient Monte Carlo scheme to sample such a ratio.[4] To understand the Metropolis method, let us first look more closely at the structure of equation (2.3). In what follows we denote the configurational part of the partition function by Z:

$$Z \equiv \int d\mathbf{r}^N \exp[-\beta\mathcal{U}(\mathbf{r}^N)]. \tag{2.4}$$

Note that the ratio $\exp(-\beta\mathcal{U})/Z$ in equation (2.3) is the probability density to find the system in a configuration around $\mathbf{r}^N$. Let us denote this probability density by

$$\mathcal{N}(\mathbf{r}^N) \equiv \frac{\exp[-\beta\mathcal{U}(\mathbf{r}^N)]}{Z}.$$

Clearly, $\mathcal{N}(\mathbf{r}^N)$ is nonnegative.

Suppose now that we are somehow able to randomly generate points in configuration space according to this probability distribution $\mathcal{N}(\mathbf{r}^N)$. This means that, on average, the number of points n_i generated per unit volume around a point $\mathbf{r}^N$ is equal to $L\mathcal{N}(\mathbf{r}^N)$, where L is the total number of points that we have generated. In other words;

$$\langle A \rangle \approx \frac{1}{L} \sum_{i=1}^{L} n_i \, A(\mathbf{r}_i^N). \tag{2.5}$$

By now the reader is almost certainly confused, let us therefore try to clarify this method with the help of a simple example (see Figure 6). In this figure, we compare two ways to measure the depth of the river Nile, by conventional quadrature (left) and by Metropolis sampling; that is, the construction of an importance-weighted random walk (right). In the conventional quadrature scheme, the value of the integrand is measured at a predetermined set of points. As the choice of these points does not depend on the value of the integrand, many points may be located in regions where the integrand vanishes. In contrast, in the Metropolis scheme, a random walk is constructed through that region of space where the integrand is nonnegligible (i.e., through the Nile itself). In this random walk, a trial move is rejected if it takes you out of the water and is accepted otherwise. After *every* trial move (accepted or not), the depth of the water is measured. The (unweighted) average of all these measurements yields an estimate of the average depth of the Nile. This, then, is the

[4] An interesting account of the early history of the Metropolis method may be found in H.L. Anderson, *J. Stat. Phys.* 43:731, 1986; and Wood [16] p. 3.

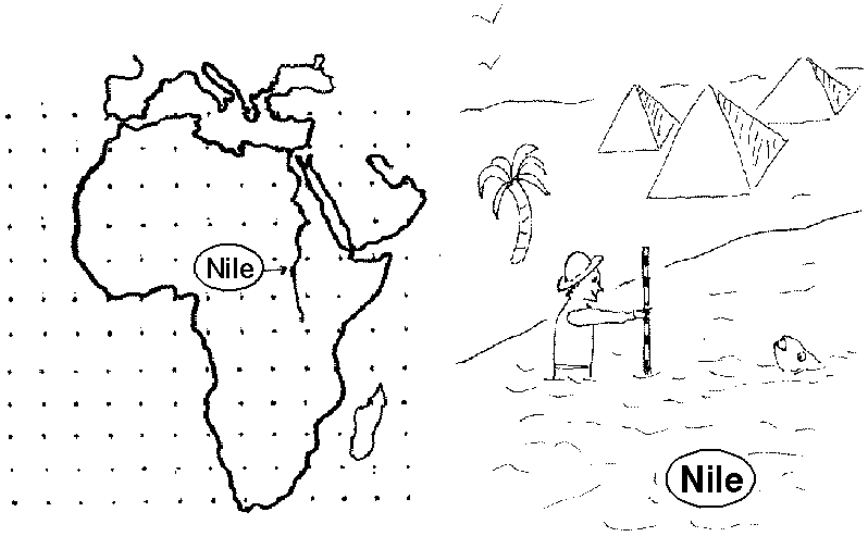

Figure 6: Measuring the depth of the Nile: a comparison of conventional quadrature (left), with the Metropolis scheme (right).

essence of the Metropolis method. In principle, the conventional quadrature scheme would also give results for the *total* area of the Nile. In the importance sampling scheme, however, information on the total area cannot be obtained directly, since this quantity is similar to Z.

Let us next consider how to generate points in configuration space with a relative probability proportional to the Boltzmann factor. The general approach is first to prepare the system in a configuration $\mathbf{r}^N$, which we denote by o (old), that has a nonvanishing Boltzmann factor $\exp[-\beta\mathcal{U}(o)]$. This configuration, for example, may correspond to a regular crystalline lattice with no hard-core overlaps. Next, we generate a new trial configuration $\mathbf{r}'^N$, which we denote by n (new), by adding a small random displacement Δ to o. The Boltzmann factor of this trial configuration is $\exp[-\beta\mathcal{U}(n)]$. We must now decide whether we will accept or reject the trial configuration. Many rules for making this decision satisfy the constraint that on average the probability of finding the system in a configuration n is proportional to $\mathcal{N}(n)$. Here we discuss only the Metropolis scheme, because it is simple and generally applicable.

Let us now "derive" the Metropolis scheme to determine the transition probability $\pi(o \to n)$ to go from configuration o to n. It is convenient to start with a thought experiment (actually a thought simulation). We carry out a very large number (say M) Monte Carlo simulations in parallel, where M is much larger than the total number of accessible configurations. We denote the number of points in any configuration o by $m(o)$. We wish that, on average, $m(o)$ is proportional to $\mathcal{N}(o)$. The matrix elements $\pi(o \to n)$ must satisfy one obvious condition: they do not destroy such an equilibrium distribution once it is reached. This means that, in equilibrium, the average number of accepted trial moves that result in the system leaving state o must be exactly equal to the number of accepted trial moves from all other states n to state o. It is convenient to impose a much stronger condition; namely, that in equilibrium the average number of accepted moves from o to any other state n is exactly canceled by the number of reverse moves. This detailed balance condition implies

the following:

$$\mathcal{N}(o)\pi(o \to n) = \mathcal{N}(n)\pi(n \to o). \tag{2.6}$$

Many possible forms of the transition matrix $\pi(o \to n)$ satisfy equation (2.6). Let us look how $\pi(o \to n)$ is constructed in practice. We recall that a Monte Carlo move consists of two stages. First, we perform a trial move from state o to state n. We denote the transition matrix that determines the probability to perform a trial move from o to n by $\alpha(o \to n)$; where α is usually referred to as the underlying matrix of Markov chain [17]. The next stage is the decision to either accept or reject this trial move. Let us denote the probability of accepting a trial move from o to n by $\mathrm{acc}(o \to n)$. Clearly,

$$\pi(o \to n) = \alpha(o \to n) \times \mathrm{acc}(o \to n). \tag{2.7}$$

In the original Metropolis scheme, α is chosen to be a symmetric matrix ($\mathrm{acc}(o \to n) = \mathrm{acc}(n \to o)$). However, in later sections we shall see several examples where α is *not* symmetric. If α is symmetric, we can rewrite equation (2.6) in terms of the $\mathrm{acc}(o \to n)$:

$$\mathcal{N}(o) \times \mathrm{acc}(o \to n) = \mathcal{N}(n) \times \mathrm{acc}(n \to o). \tag{2.8}$$

From equation (2.8) follows

$$\frac{\mathrm{acc}(o \to n)}{\mathrm{acc}(n \to o)} = \frac{\mathcal{N}(n)}{\mathcal{N}(o)} = \exp\{-\beta[\mathcal{U}(n) - \mathcal{U}(o)]\}. \tag{2.9}$$

Again, many choices for $\mathrm{acc}(o \to n)$ satisfy this condition (and the obvious condition that the probability $\mathrm{acc}(o \to n)$ cannot exceed 1). The choice of Metropolis *et al.* is

$$\begin{array}{rcll} \mathrm{acc}(o \to n) & = & \mathcal{N}(n)/\mathcal{N}(o) & \text{if } \mathcal{N}(n) < \mathcal{N}(o) \\ & = & 1 & \text{if } \mathcal{N}(n) \geq \mathcal{N}(o). \end{array} \tag{2.10}$$

Other choices for $\mathrm{acc}(o \to n)$ are possible (for a discussion, see for instance [1]), but the original choice of Metropolis *et al.* appears to result in a more efficient sampling of configuration space than most other strategies that have been proposed.

2.2 A Basic Monte Carlo Algorithm

It is difficult to talk about Monte Carlo or Molecular Dynamics programs in abstract terms. The best way to explain how such programs work is to write them down. This will be done in the present section.

Most Monte Carlo or Molecular Dynamics programs are only a few hundred to several thousand lines long. This is very short compared to, for instance, a typical quantum-chemistry code. For this reason, it is not uncommon that a simulator will write many different programs that are tailor-made for specific applications. The result is that there is no such thing as a standard Monte Carlo or Molecular Dynamics program. However, the cores of most MD/MC programs are, if not identical, at least very similar. Next, we shall construct such a core. It will be very rudimentary, and efficiency has been traded for clarity. But it should demonstrate how the Monte Carlo method work.

2.3 The Algorithm

The prime purpose of the kind of Monte Carlo or Molecular Dynamics program that we shall be discussing is to compute equilibrium properties of classical many-body systems. From now on, we shall refer to such programs simply as MC or MD programs, although it should be remembered that there exist many other applications of the Monte Carlo method (and, to a lesser extent, of the Molecular Dynamics method). Let us now look at a simple Monte Carlo program.

In the previous section, the Metropolis method was introduced as a Markov process in which a random walk is constructed in such a way that the probability of visiting a particular point $\mathbf{r}^N$ is proportional to the Boltzmann factor $\exp[-\beta\mathcal{U}(\mathbf{r}^N)]$. There are many ways to construct such a random walk. In the approach introduced by Metropolis *et al.* [15], the following scheme is proposed:

1. Select a particle at random, and calculate its energy $\mathcal{U}(\mathbf{r}^N)$.
2. Give the particle a random displacement; $r' = r + \Delta$, and calculate its new energy $\mathcal{U}(\mathbf{r'}^N)$.
3. Accept the move from $\mathbf{r}^N$ to $\mathbf{r'}^N$ with probability

$$\mathrm{acc}(o \to n) = \min\left(1, \exp\{-\beta[\mathcal{U}(\mathbf{r'}^N) - \mathcal{U}(\mathbf{r}^N)]\}\right). \tag{2.11}$$

An implementation of this basic Metropolis scheme is shown in Algorithms 5 and 6.

2.4 Trial Moves

Now that we have specified the general structure of the Metropolis algorithm, we should consider its implementation. We shall not go into the problem of selecting intermolecular potentials for the model system under study. Rather, we shall simply assume that we have an atomic or molecular model system in a suitable starting configuration and that we have specified all intermolecular interactions. We must now set up the underlying Markov chain; that is, the matrix α. In more down to earth terms: we must decide how we are going to generate trial moves. We should distinguish between trial moves that involve only the molecular centers of mass and those that change the orientation or possibly even the conformation of a molecule.

We start our discussion with trial moves of the molecular centers of mass. A perfectly acceptable method to create a trial displacement is to add random numbers between $-\Delta/2$ and $+\Delta/2$ to the x, y, and z coordinates of the molecular center of mass:

$$\begin{aligned} x_i' &\to x_i + \Delta\,(\mathrm{Ranf} - 0.5) \\ y_i' &\to y_i + \Delta\,(\mathrm{Ranf} - 0.5) \\ z_i' &\to z_i + \Delta\,(\mathrm{Ranf} - 0.5), \end{aligned} \tag{2.12}$$

where Ranf are random numbers uniformly distributed between 0 and 1. Clearly, the reverse trial move is equally probable (hence, α is symmetric).

We are now faced with two questions: how large should we choose Δ? and should we attempt to move all particles simultaneously or one at a time? In

Algorithm 5 (Basic Metropolis Algorithm)

```
  PROGRAM mc                          basic Metropolis algorithm

  do icycl=1,ncycl                    perform ncycl MC cycles
    call mcmove                       displace a particle
    if (mod(icycl,nsamp).eq.0)
+     call sample                     sample averages
  enddo
  end
```

Comments to this algorithm:

1. *Subroutine* `mcmove` *attempts to displace a randomly selected particle (see Algorithm 6).*
2. *Subroutine* `sample` *samples quantities every* `nsamp`*th cycle.*

Algorithm 6 (Attempt to Displace a Particle)

```
  SUBROUTINE mcmove                   attempts to displace a particle

  o=int(ranf()*npart)+1               select a particle at random
  call ener(x(o),eno)                 energy old configuration
  xn=x(o)+(ranf()-0.5)*delx           give particle random displacement
  call ener(xn,enn)                   energy new configuration
  if (ranf().lt.exp(-beta             acceptance rule (2.11)
+   *(enn-eno)) x(o)=xn               accepted: replace x(o) by xn
  return
  end
```

Comments to this algorithm:

1. *Subroutine* `ener` *calculates the energy of a particle at the given position.*
2. *Note that, if a configuration is rejected, the old configuration is retained.*
3. *The* `ranf()` *is a random number uniform in* $[0, 1]$.

the latter case we should pick the molecule that is to be moved at random to ensure that the underlying Markov chain remains symmetric. All other things being equal, we should choose the most efficient sampling procedure. But, to this end, we must first define what we mean by *efficient sampling*. In very vague terms, sampling is efficient if it gives you good value for money. Good value in a simulation corresponds to high statistical accuracy, and "money" is simply *money*: the money that buys your computer time and even your own time. For the sake of the argument, we assume the average scientific programmer is poorly paid. In that case we have to worry only about your computer budget.[5] Then we could use the following definition of an optimal sampling scheme: a Monte Carlo sampling scheme can be considered optimal if it yields the lowest statistical error in the quantity to be computed for a given expenditure of computing budget. Usually, computing budget is equivalent to CPU time.

From this definition it is clear that, in principle, a sampling scheme may be optimal for one quantity but not for another. Actually, the preceding definition is all but useless in practice (as are most definitions). For instance, it is just not worth the effort to measure the error estimate in the pressure for a number of different Monte Carlo sampling schemes in a series of runs of fixed length. However, it is reasonable to assume that the mean-square error in the observables is inversely proportional to the number of uncorrelated configurations visited in a given amount of CPU time. And the number of independent configurations visited is a measure for the distance covered in configuration space. This suggests a more manageable, albeit rather ad hoc, criterion to estimate the efficiency of a Monte Carlo sampling scheme: the sum of the squares of all accepted trial displacements divided by computing time. This quantity should be distinguished from the mean-square displacement per unit of computing time, because the latter quantity goes to zero in the absence of diffusion (e.g., in a solid or a glass), whereas the former does not.

Next, consider the choice of the parameter Δ which determines the size of the trial move. How large should Δ be? If it is very large, it is likely that the resulting configuration will have a high energy and the trial move will probably be rejected. If it is very small, the change in potential energy is probably small and most moves will be accepted. In the literature, one often finds the mysterious statement that an acceptance of approximately 50% should be optimal. This statement is not necessarily true. The optimum acceptance ratio is the one that leads to the most efficient sampling of configuration space. If we express efficiency as mean-square displacement per CPU time, it is easy to see that different Monte Carlo codes will have different optimal acceptance ratios. The reason is that it makes a crucial difference if the amount of computing required to test whether a trial move is accepted depends on the magnitude of the move (see Figure 7). In the conventional Metropolis scheme, all continuous interactions have to be computed before a move can be accepted or rejected. Hence, for continuous potentials, the amount of computation does not depend on the size of a trial move. In contrast, for simulations of molecules with hard repulsive cores, a move can be rejected as soon as overlap with any neighbor is detected. In that case, a rejected move is cheaper than an accepted one, and hence the average computing time per trial move goes down as the step size is increased. As a result, the optimal acceptance ratio for hard-core systems is appreciably lower than for systems with continuous interactions. Exactly how much depends on the nature of the program, in particular on whether it is a

[5] Still, we should stress that it is not worthwhile to spend a lot of time developing a fancy computational scheme that will be only marginally better than existing, simpler schemes, unless your program will run very often and speed is crucial.

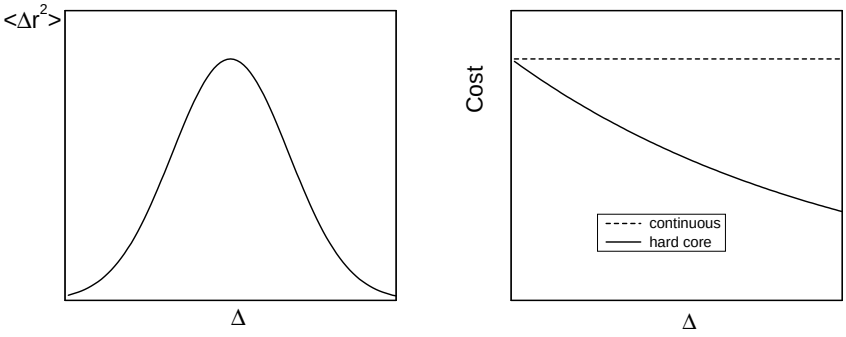

Figure 7: (left) Typical dependence of the mean-square displacement of a particle on the average size Δ of the trial move. (right) Typical dependence of the computational cost of a trial move on the step-size Δ. For continuous potentials, the cost is constant, while for hard-core potentials it decreases rapidly with the size of the trial move.

scalar or a vector code (in the latter case, hard-core systems are treated much like continuous systems), how the information about neighbor lists is stored, and even on the computational "cost" of random numbers and exponentiation. The consensus seems to be that for hard-core systems the optimum acceptance ratio is closer to 20 than to 50%, but this is just another rule of thumb that should be checked.

2.5 Canonical Ensemble

In a conventional Molecular Dynamics simulation, the total energy E and the total linear momentum $\mathbf{P}$ are constants of motion. Hence, Molecular Dynamics simulations measure (time) averages in an ensemble that is very similar to the microcanonical (see [18]); namely, the constant-NVE-$\mathbf{P}$ ensemble. In contrast, a conventional Monte Carlo simulation probes the canonical (i.e., constant-NVT) ensemble. The fact that these ensembles are different leads to observable differences in the statistical averages computed in Molecular Dynamics and Monte Carlo simulations. Most of these differences disappear in the thermodynamic limit and are already relatively small for systems of a few hundred particles. However, the choice of ensemble does make a difference when computing the mean-square value of fluctuations in thermodynamic quantities. Fortunately, techniques exist to relate fluctuations in different ensembles [10]. Moreover, nowadays it is common practice to carry out Molecular Dynamics simulations in ensembles other than the microcanonical. In particular, it is possible to do Molecular Dynamics at constant pressure, at constant stress, and at constant temperature. The choice of ensembles for Monte Carlo simulations is even wider: isobaric-isothermal, constant-stress-isothermal, grand-canonical (i.e., constant-μVT), and even microcanonical [19–24]. A more recent addition to this list is a Monte Carlo method that employs the Gibbs-ensemble technique [25], which was developed to study phase coexistence in moderately dense (multi component) fluids.

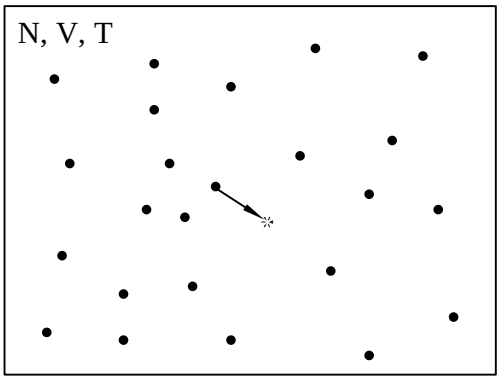

Figure 8: Canonical ensemble. The number of particles, volume, and temperature are constant. Shown is a Monte Carlo move in which a particle is displaced.

2.6 General Approach

In the following sections, we will use the following procedure to demonstrate the validity of our Monte Carlo algorithms:

1. Decide which distribution we want to sample. This distribution, denoted $\mathcal{N}$, will depend on the details of the ensemble.

2. Impose the condition of detailed balance,

$$K(o \to n) = K(n \to o), \tag{2.13}$$

   where $K(o \to n)$ is the flow of configuration o to n. This flow is given by the product of the probability of being in configuration o, the probability of generating configuration n, and the probability of accepting this move,

$$K(o \to n) = \mathcal{N}(o) \times \alpha(o \to n) \times \mathrm{acc}(o \to n). \tag{2.14}$$

3. Determine the probabilities of generating a particular configuration.

4. Derive the condition which needs to be fulfilled by the acceptance rules.

It is instructive to apply the preceding recipe to the ordinary Metropolis scheme. In the canonical ensemble, the number of particles, temperature, and volume are constant (see Figure 8). The partition function is

$$Q(N, V, T) \equiv \frac{1}{\Lambda^{3N} N!} \int d\mathbf{r}^N \, \exp[-\beta \mathcal{U}(\mathbf{r}^N)], \tag{2.15}$$

where $\Lambda = \sqrt{h^2/(2\pi m k_B T)}$ is the thermal de Broglie wavelength. From the partition function it follows that the probability of finding configuration $\mathbf{r}^N$ is given by distribution is

$$\mathcal{N}(\mathbf{r}^N) \propto \exp[-\beta \mathcal{U}(\mathbf{r}^N)]. \tag{2.16}$$

Equations (2.15) and (2.16) are the basic equations for a simulation in the canonical ensemble.

2.7 Monte Carlo Simulations

In the canonical ensemble, we have to sample distribution (2.16). This can be done using the following scheme:

1. Select a particle at random and calculate the energy of this configuration $\mathcal{U}(o)$.

2. Give this particle a random displacement (see Figure 8),

$$\mathbf{r}(o) \rightarrow \mathbf{r}(o) + \Delta(\text{Ranf} - 0.5),$$

   where $\Delta/2$ is the maximum displacement. The value of Δ should be chosen such that the sampling scheme is optimal (see section 2.4). The new configuration is denoted n and its energy $\mathcal{U}(n)$.

3. The move is accepted with a probability (see equation (2.10))

$$\text{acc}(o \rightarrow n) = \min\left(1, \exp\{-\beta[\mathcal{U}(n) - \mathcal{U}(o)]\}\right). \tag{2.17}$$

   If rejected, the old configuration is kept.

An implementation of this basic Metropolis scheme is shown in Section 2.2 (Algorithms 5 and 6).

2.8 Justification of the Algorithm

The probability of generating a particular configuration is constant and independent of the conformation of the system

$$\alpha(o \rightarrow n) = \alpha(n \rightarrow o) = \alpha.$$

Substitution of this equation in the condition of detailed balance (2.13) and substitution of the desired distribution (2.16) gives as condition for the acceptance rules

$$\frac{\text{acc}(o \rightarrow n)}{\text{acc}(n \rightarrow o)} = \exp\{-\beta[\mathcal{U}(n) - \mathcal{U}(o)]\}. \tag{2.18}$$

It is straightforward to demonstrate that acceptance rule (2.17) obeys this condition.

2.9 Sampling rejected states: a heresy?

The Markov Chain Monte Carlo method (MCMC) of Metropolis et al. was designed to estimate the average properties of systems (many-body systems are but one example) with a very large number of accessible states ("configurations"). To achieve this, the algorithm generates a random walk through configuration space such that any individual state is visited with a frequency proportional to its (Boltzmann) weight. The desired estimate is then obtained as an unweighted average over visited states.

In order to visit states with the correct frequency, the MCMC algorithm generates random trial moves from the current ("old") state (o) to a new state (n). In the case of molecular systems, such a move might for instance be the displacement of a single particle. These trial moves may be either accepted or rejected. The acceptance criterion is chosen such that the probability to find the system in state (i) is proportional to the Boltzmann weight ($P_B(i)$). The standard "dogma" of MCMC simulations is that rejected states should not be

included in the sampling. As a consequence, conventional MC algorithms are rather wasteful, because they tend to generate many trial states (often a majority) that are not used in the computation of averages. In this section, I argue that we can do better: the properties of rejected states *can* be included in the sampling. In the case where many trial states are generated simultaneously, the new approach may lead to a dramatic improvement in the statistical accuracy of Monte Carlo simulations [26].

MCMC algorithms traditionally construct a Markov-Chain using two steps: a. Starting from the current (old) state (o) a trial move is attempted to a new state (n) according to a trial probability α. b. The trial state is then accepted or rejected according to an acceptance rule that ensures detailed balance (or, at least, the less strict balance [27]) between sampled microstates.

$$\pi_{ij} = \alpha_{ij} P_{acc}(ij) \; \forall \, i \neq j, \quad \pi_{ii} = 1 - \sum_{j} \pi_{ij} \tag{2.19}$$

$$\sum_{j} \pi_{ij} = 1, \; \forall i \tag{2.20}$$

Here, and in what follows, we use the term "balance" to describe algorithms that leave the equilibrium distribution invariant. That is: when we apply one Monte Carlo step to an equilibrium distribution of initial states, the average flux into any given state i is exactly balanced by the total flux out of that state. As a consequence, the probability density is not changed by such the Monte Carlo algorithm. The stronger "detailed balance" condition states that, for any pair of states i and j in an equilibrium ensemble, the average flux from i to j is equal and opposite to the flux from j to i. Clearly, the latter condition can only be satisfied if the algorithm satisfies microscopic reversibility: i.e. if j can be reached from i then there is a finite probability to carry out the reverse move from j to i.

In Eqn.(2.19) π_{ij} is the ij^{th} element of the Markovian transition matrix. For $i \neq j$, π_{ij} is given by the product of two terms: the probability of attempting a trial move from state *i* to state *j*, and the probability of accepting *j* as the new state. The transition matrix is a stochastic matrix therefore its elements must obey Eqn.(2.20).

In a Monte Carlo simulation, thermodynamic properties are evaluated as the expectation values of the corresponding instantaneous properties. When a trial move is not accepted, the instantaneous property of the old state has to be recounted in the calculated average and no information about the rejected state is included in the computation of averages. Let us now consider MC algorithms that satisfy detailed balance. The condition of detailed balance is satisfied whenever equation (2.21) holds for any pair of old states (o) and new states (n).

$$\rho_o \pi_{on} = \rho_o \alpha_{on} P_{acc}(on) = \rho_n \alpha_{no} P_{acc}(no) = \rho_n \pi_{no} \tag{2.21}$$

Equations (2.19-2.21) do not uniquely define $P_{acc}(on)$: there is therefore a relative freedom in the choice of the functional form. The most popular choice is the Metropolis rule [15], but (infinitely many) other valid rules exist. Below we present a more general derivation that allows the evaluation of equilibrium properties by combining importance sampling and local sampling of microstates using any combination of valid MCMC schemes. To make this explicit, we consider two sets of transition probabilities (that may or may not be

the same) : the first describes a conventional sampling of microstates while the second describes the local sampling of a group of microstates that are connected to the individual microstates of the first Markov chain. We use the term *"Markovian web"* to denote the set of microstates thus connected.

We assume that the Markov chain that connects the states of the Markovian web satisfies detailed balance. Then equation (2.21) applies for any given pair of connected microstates $\{o,n\}$. From equations (2.20) and (2.21) the balance condition may be derived:

$$\sum_m \rho_m \pi_{mn}^{web} = \sum_m \rho_n \pi_{nm}^{web} = \rho_n \sum_m \pi_{nm}^{web} = \rho_n \tag{2.22}$$

Let us now consider the sampling of an equilibrium property A of the system:

$$\langle A \rangle_\rho = \sum_n A_n \rho_n \Big/ \sum_n \rho_n \tag{2.23}$$

We now substitute ρ_n from Eqn.(2.22). By changing the order of summation (integration in the continuum case) over all microstates we get:

$$\frac{\sum_n A_n \rho_n}{\sum_n \rho_n} = \frac{\sum_n \sum_m A_n \rho_m \pi_{mn}^{web}}{\sum_n \sum_m \rho_m \pi_{mn}^{web}} = \frac{\sum_m \rho_m \sum_n A_n \pi_{mn}^{web}}{\sum_m \rho_m \sum_n \pi_{mn}^{web}}$$

$$= \frac{\sum_m \rho_m \sum_n A_n \pi_{mn}^{web}}{\sum_m \rho_m} \Leftrightarrow \langle A \rangle_\rho = \left\langle \sum_n \pi_{mn}^{web} A_n \right\rangle_{\rho_m} \tag{2.24}$$

Where the last step required the use of equation (2.20). Equation (2.24) is exact and is in principle sufficient to describe how the expectation value of a property can be evaluated by combining importance sampling and integration over the local states of a Markovian Web; i.e for every state $\{m\}$ sampled via importance sampling an integration is performed over all $\{n\}$ states for which $\pi_{mn}^{web} \neq 0$. Equation (2.24) is quite general and it may be implemented in many ways. In the present paper, we focus on an application to Monte Carlo sampling of the equilibrium properties of a simple off-lattice system. The transition matrix of the Markov chain that is used to generate the Markovian Web can be separated in two terms as in the case of MCMC discussed above (Eqn. (2.19)). The sum in equation (2.24) may be broken into two terms using Eqn. (2.19):

$$\sum_n A_n \pi_{mn}^{web} = \sum_{n \neq m} A_n \pi_{mn}^{web} + A_m \left(1 - \sum_{n \neq m} \pi_{mn}^{web} \right) \tag{2.25}$$

Substituting (2.25) into (2.24) and using the fact that $\sum_{n \neq m} \alpha_{mn} = 1$, a new expression for the average properties is derived:

$$\langle A \rangle_\rho = \left\langle \sum_{n \neq m} \alpha_{mn} \left[A_n P_{acc}(mn) + A_m \left(1 - P_{acc}(mn) \right) \right] \right\rangle_{\rho_m} \tag{2.26}$$

The summation in equation (2.26) may be performed in many ways. One possible approach is to express the sum as a weighted average over the trial probability α_{ij}.

$$\langle A \rangle_\rho = \left\langle \langle A_n P_{acc}(mn) + A_m (1 - P_{acc}(mn)) \rangle_{\alpha_{mn}} \right\rangle_{\rho_m} \quad (2.27)$$

Equation (2.27) implies that the sum required for equation (2.24) can be computed as a weighted average, where the weight is the trial probability α_{ij}. In practice, this means that we perform not only an importance sampling of the states n with corresponding weights ρ_m, but we also generate a finite subset of the trial states n, with weights α_{mn}.

We stress that equation (2.24) is general, whereas equation (2.27) represents one of the many possible implementations of the general case.

An important aspect of the proposed method is that the transition matrix of the Markovian Web is completely independent from the Markovian matrix that is used to generate the importance sampling (the importance sampling could even have been generated by completely different method, e.g. constant-temperature Molecular Dynamics). In other words: the two transition matrices may differ in their trial probability or in their acceptance probability (e.g. use of Metropolis or the symmetric rule) or in both.

It is advantageous to consider situations where the number of trial states is very large. Such a situation arises for instance when the system under consideration can be decomposed into non-interacting subsystems. An example is a system consisting of particles with intermolecular interactions that can be truncated beyond a finite cut-off distance r_c. If we divide the system into cells with diameter $d \geq r_c$, then non-adjacent cells will not interact. This means that the probability of acceptance of a trial move in one such cell does not depend on the possible outcome of a trial move inside all other non-adjacent cells. This makes it possible to construct "parallel" trial moves that attempt to move particles in a set of N non-interacting cells. In normal MCMC, the result of such a parallel move is that the system ends up in one specific final state (out of the 2^N possible states). With the present algorithm we can do much better. By generating M trial states in each of the subsystem we can evaluate the term $\sum A_n \pi_{mn}^{web}$ by summing all $(M+1)^N$ possible combinations of trial states for the subsystems. The number of trial microstates can be very large. Hence, special care should be taken to compute averages over trial states efficiently. Since the subsystems are independent one can calculate the probability of finding the total system in a specific macrostate (e.g. the collection of microstates with the same potential energy) as the convolution of the probabilities of the subsystems. It should be noted that the use of the convolution or even the independence between the subsystems is not a necessary requirement but it is expected to be crucial in the case of large number of possible trial states.

References

[1] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, Oxford, 1987.

[2] G. Ciccotti and W.G. Hoover (editors). *Proceedings of the 97th international school of Physics "Enrico Fermi" on Molecular-Dynamics Simulations of Statistical-Mechanical Systems*. North-Holland, Amsterdam, 1986.

[3] M. Meyer and V. Pontikis. *Proceedings of the NATO ASI on Computer Simulation in Materials Science*. Kluwer, Dordrecht, 1991.

[4] M.P. Allen and D.J. Tildesley. *Proceedings of the NATO ASI on Computer Simulation in Chemical Physics*. Kluwer, Dordrecht, 1993.

[5] D. Frenkel and B. Smit. *Understanding Molecular Simulations: from Algorithms to Applications*. Academic Press, San Diego, 1996.

[6] G.D. Quinlan and S. Tremaine. On the reliability of gravitational n-body integrations. *Mon. Not. R. Astron. Soc.*, 259:505–518, 1992.

[7] S. Toxvaerd. Hamiltonians for discrete dynamics. *Phys. Rev. E*, 50:2271–2274, 1994.

[8] F.J. Vesely. *Computational Physics. An Introduction*. Plenum, New York, 1994.

[9] J.P. Hansen and I.R. McDonald. *Theory of Simple Liquids*. Academic Press, London, 2nd edition, 1986.

[10] J.L. Lebowitz, J.K. Percus, and L. Verlet. Ensemble dependence of fluctuations with application to machine computations. *Phys. Rev.*, 153:250–254, 1967.

[11] J.P. Hansen and L. Verlet. Phase transitions of the Lennard-Jones system. *Phys. Rev.*, 184:151–161, 1969.

[12] A.J.C. Ladd and L.V. Woodcock. Triple-point coexistence properties of the Lennard-Jones system. *Chem. Phys. Lett.*, 51:155–159, 1977.

[13] A.J.C. Ladd and L.V. Woodcock. Interfacial and co-existence properties of the Lennard-Jones system at the triple point. *Mol. Phys.*, 36:611–619, 1978.

[14] H. Flyvbjerg and H.G. Petersen. Error estimates on averages of correlated data. *J. Chem. Phys.*, 91:461–466, 1989.

[15] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.N. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[16] W.W. Wood. Early history of computer simulation in statistical mechanics. In G. Ciccotti and W.G. Hoover, editors, *Molecular Dynamics Simulations of Statistical Mechanics Systems*, pages 2–14. Proceedings of the 97th Int. "Enrico Fermi" School of Physics, North Holland, Amsterdam, 1986.

[17] N.G. van Kampen. *Stochastic Processes in Physics and Chemistry*. North-Holland, Amsterdam, 1981.

[18] J.J. Erpenbeck and W.W. Wood. Molecular dynamics techniques for hard-core systems. In B.J. Berne, editor, *Statistical Mechanics, Part B*, pages 1–40. Plenum, New York, 1976.

[19] W.W. Wood. Monte Carlo calculations for hard disks in the isothermal-isobaric ensemble. *J. Chem. Phys.*, 48:415–434, 1968.

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

[21] R. Najafabadi and S. Yip. Observation of finite-temperature strain transformation (f.c.c. $\leftrightarrow$ b.c.c.) in Monte Carlo simulation of iron. *Scripta Metall.*, 17:1199–1204, 1983.

[22] G.E. Norman and V.S Filinov. Investigation of phase transitions by a Monte-Carlo method. *High Temp. (USSR)*, 7:216–222, 1969.

[23] D.J. Adams. Chemical potential of hard-sphere fluids by Monte Carlo methods. *Mol. Phys.*, 28:1241–1252, 1974.

[24] M. Creutz. Microcanonical Monte Carlo simulation. *Phys. Rev. Lett.*, 50:1411–1414, 1983.

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

[26] D. Frenkel. Speed-up of Monte Carlo simulations by sampling of rejected states. *Proc. Nat. Acad. Sci. USA*, 101:17571–17575, 2004.

[27] V.I. Manousiouthakis and M.W. Deem. Strict detailed balance is unnecessary in Monte Carlo simulation. *J. Chem. Phys.*, 110:2753–2756, 1999.