Chapter 3

Monte Carlo Simulations

Adapted from:

"Understanding Molecular Simulation"
Daan Frenkel and Berend Smit
Academic Press (2001)
Chapter 3

In this chapter, the basic principles of the Monte Carlo (MC) method are described. In particular, we will focus on MC simulations of systems of a fixed number of particles (N) in a given volume (V) at a temperature (T), i.e. simulations in the canonical (N, V, T) ensemble.
In the previous chapter, we introduced some of the basic concepts of (classical) statistical mechanics. Our next aim is to indicate where the Monte Carlo method comes in. We start from the classical expression for the partition function Q, equation (2.24):

$$Q = c \int d\mathbf{p}^N d\mathbf{r}^N \exp[-\mathcal{H}(\mathbf{r}^N\, \mathbf{p}^N)/k_B T] \tag{3.1}$$

where $\mathbf{r}^N$ stands for the coordinates of all N particles, and $\mathbf{p}^N$ for the corresponding momenta. The function $\mathcal{H}(\mathbf{r}^N,\mathbf{p}^N)$ 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 in equation (2.24) approaches the classical partition function in the limit of $\hbar \to 0$. For instance, for a system of N identical atoms, $c = 1/(h^{3N}\, N!)$.
The classical equation corresponding to equation (2.25) is

$$\langle A \rangle = \frac{\int d\mathbf{p}^N d\mathbf{r}^N\, A(\mathbf{p}^N, \mathbf{r}^N) \exp[-\beta \mathcal{H}(\mathbf{p}^N, \mathbf{r}^N)]}{\int d\mathbf{p}^N d\mathbf{r}^N\, \exp[-\beta \mathcal{H}(\mathbf{p}^N, \mathbf{r}^N)]} \tag{3.2}$$

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 <A> in equation (3.2) 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 $o(100)$. 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 (3.2) 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 preceding example clearly demonstrates that better numerical techniques are needed to compute thermal averages. One such a technique is the Monte Carlo method or, more precisely, the Monte Carlo importance sampling algorithm introduced 1953 by Metropolis et al. (*N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.N. Teller, and E. Teller, J. Chem. Phys. 21, 1087 (1953)*). The application of this method to the numerical simulation of dense molecular systems is the subject of the present chapter.

3.1 Importance Sampling

Before discussing importance sampling, let us first look at the simplest Monte Carlo technique; that is, random sampling. Suppose we wish to evaluate a one-dimensional integral I

$$I = \int_a^b dx\, f(x) \tag{3.3}$$

Instead of using a conventional quadrature where the integrand is evaluated at predetermined values of the abscissa, we could do something else. Note that equation (3.3) can be rewritten as

$$I = (b-a)\,\langle f(x)\rangle \tag{3.4}$$

where $\langle f(x)\rangle$ denotes the unweighted average of $f(x)$ over the interval [a,b].

In brute force Monte Carlo, this average is determined by evaluating $f(x)$ at a large number (say, L) of x values randomly distributed over the interval [a,b] . It is clear that, as $L\rightarrow\infty$, this procedure should yield the correct value for I. However, as with the conventional quadrature procedure, this method is of little use to evaluate averages such as in equation (3.2) because most of the computing is spent on points where the Boltzmann factor is negligible. Clearly, it would be much preferable to sample many points in the region where the Boltzmann factor is large and few elsewhere. This is the basic idea behind importance sampling.

How should we distribute our sampling through configuration space? To see this, let us first consider a simple, one-dimensional example. Suppose we wish to compute the definite integral in equation (3.3) by Monte Carlo sampling but with the sampling points distributed nonuniformly over the

sampling interval [a,b] (for convenience we assume a = 0 and b = 1), according to some nonnegative probability density $w(x)$. Clearly, we can rewrite equation (3.3) as

$$I = \int_0^1 dx\, w(x) \frac{f(x)}{w(x)} \tag{3.5}$$

Let us assume that we know that $w(x)$ is the derivative of another (nonnegative, nondecreasing) function $u(x)$, with $u(0) = 0$ and $u(1) = 1$ (these boundary conditions imply that $w(x)$ is normalized). Then I can be written as

$$I = \int_0^1 du\, \frac{f[x(u)]}{w[x(u)]} \tag{3.6}$$

In equation (3.6) we have written $x(u)$ to indicate that, if we consider u as the integration variable, then x must be expressed as a function of u. The next step is to generate L random samples of u uniformly distributed in the interval [0,1]. We then obtain the following estimate for I:

$$I \approx \frac{1}{L} \sum_{i=1}^{L} \frac{f[x(u_i)]}{w[x(u_i)]} \tag{3.7}$$

What have we gained by rewriting I in this way? The answer depends crucially on our choice for $w(x)$. To see this, let us estimate σ^2_L, the variance in I_L, where I_L denotes the estimate for I obtained from equation (3.7) with L random sample points:

$$\sigma_I^2 = \frac{1}{L^2} \sum_{i=1}^{L} \sum_{j=1}^{L} \left\langle \left(\frac{f[x(u_i)]}{w[x(u_i)]} - \langle f/w \rangle \right) \left(\frac{f[x(u_j)]}{w[x(u_j)]} - \langle f/w \rangle \right) \right\rangle \tag{3.8}$$

where the angular brackets denote the true average; that is, the one that would be obtained in the limit $L \to \infty$. As different samples i and j are assumed to be totally independent, all cross terms in equation (3.8) vanish, and we are left with

$$\sigma_I^2 = \frac{1}{L^2} \sum_{i=1}^{L} \left\langle \left(\frac{f[x(u_i)]}{w[x(u_i)]} - \langle f/w \rangle \right)^2 \right\rangle$$

$$= \frac{1}{L} \left[\langle (f/w)^2 \rangle - \langle f/w \rangle^2 \right] \tag{3.9}$$

Equation (3.9) shows that the variance in I still goes as $1/L$, but the magnitude of this variance can be reduced greatly by choosing $w(x)$ such that $f(x)/w(x)$ is a smooth function of x. Ideally, we should have $f(x)/w(x)$ constant, in which

case the variance would vanish altogether. In contrast, if $w(x)$ is constant, as is the case for the brute force Monte Carlo sampling, then the relative error in I can become very large. For instance, if we are sampling in a (multidimensional) configuration space of volume Ω, of which only a small fraction f is accessible (for instance, $f = 10^{-260}$, see previous section), then the relative error that results in a brute force MC sampling will be of order $1/(Lf)$. As the integrand in equation (3.2) is nonzero only for those configurations where the Boltzmann factor is nonzero, it would clearly be advisable to carry out a nonuniform Monte Carlo sampling of configuration space, such that the weight function w is approximately proportional to the Boltzmann factor. Unfortunately, the simple importance sampling scheme described previously cannot be used to sample multidimensional integrals over configuration space, such as equation (3.2). The reason is simply that we do not know how to construct a transformation such as the one from equation (3.5) to equation (3.6) that would enable us to generate points in configuration space with a probability density proportional to the Boltzmann factor. In fact, a necessary (but not nearly sufficient) condition for the solution to the latter problem is that we must be able to compute analytically the partition function of the system under study. If we could do that for the systems of interest to us, there would be hardly any need for computer simulation.

3.2 The Metropolis Method

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 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

$$\langle A \rangle = \frac{\int d\mathbf{r}^N \exp[-\beta \mathcal{U}(\mathbf{r}^N)] A(\mathbf{r}^N)}{\int d\mathbf{r}^N \exp[-\beta \mathcal{U}(\mathbf{r}^N)]}. \tag{3.10}$$

Hence, we wish to know the ratio of two integrals. What Metropolis et al. showed is that it is possible to devise an efficient Monte Carlo scheme to sample such a ratio. To understand the Metropolis method, let us first look more closely at the structure of equation (3.10). In what follows we denote the configurational part of the partition function by $\mathcal{Z}$:

$$Z \equiv \int d\mathbf{r}^N \exp[-\beta \mathcal{U}(\mathbf{r}^N)]. \tag{3.11}$$

Note that the ratio $\exp(-\beta \mathcal{U})/\mathcal{Z}$ in equation (3.10) 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;

$$\langle A \rangle \approx \frac{1}{L} \sum_{i=1}^{L} n_i A(\mathbf{r}_i^N) \tag{3.12}$$

By now you are almost certainly confused about the difference, if any, between equation (3.12) and equation (3.7). The difference is that in the case of equation (3.7) we know *a priori* the probability of sampling a point in a (hyper)volume $d\mathbf{r}^N$ around $\mathbf{r}^N$. In other words we know both $\exp(-\beta\mathcal{U})/\mathcal{Z}$ and $\mathcal{Z}$. In contrast, in equation (3.12) we know only $\exp(-\beta\mathcal{U})/\mathcal{Z}$, that is, we know only the relative but not the absolute probability of visiting different points in configuration space. This may sound rather abstract: let us therefore try to clarify the difference with the help of a simple example (see Figure 3.1). In this figure, we compare two ways to measure the depth of the river Nile, by

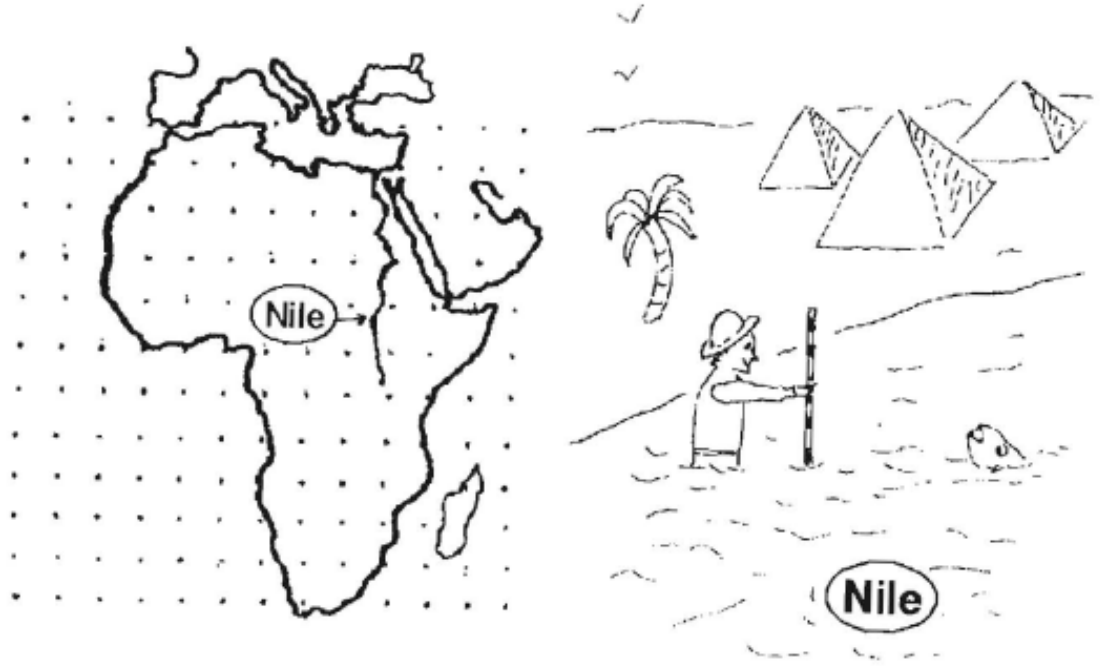

Figure 3.1: Measuring the depth of the Nile: a comparison of conventional quadrature (left), with the Metropolis scheme (right).

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 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 $\mathcal{Z}$.

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:

$$\mathcal{N}(o)\pi(o \to n) = \mathcal{N}(n)\pi(n \to o) \tag{3.13}$$

Many possible forms of the transition matrix $\pi(o \to n)$ satisfy equation (3.13). 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 i to j by $\alpha(o \to n)$; where α is usually referred to as the underlying matrix of the Markov chain. 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 $acc(o \to n)$. Clearly,

$$\pi(o \to n) = \alpha(o \to n) \times acc(o \to n) \tag{3.14}$$

In the original Metropolis scheme, α is chosen to be a symmetric matrix ($acc(o \rightarrow n) = acc(n \rightarrow o)$). However, in later sections we shall see several examples where α is not symmetric. If α is symmetric, we can rewrite equation (3.13) in terms of the $acc(o \rightarrow n)$:

$$\mathcal{N}(o) \times acc(o \rightarrow n) = \mathcal{N}(n) \times acc(n \rightarrow o) \tag{3.15}$$

From equation (3.15) follows

$$\frac{acc(o \rightarrow n)}{acc(n \rightarrow o)} = \frac{\mathcal{N}(n)}{\mathcal{N}(o)} = \exp\{-\beta[\mathcal{U}(n) - \mathcal{U}(o)]\} \tag{3.16}$$

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

$$\begin{aligned} acc(o \rightarrow n) &= \mathcal{N}(o)/\mathcal{N}(n) \quad \text{if } \mathcal{N}(n) < \mathcal{N}(o) \\ &= 1 \qquad\qquad\quad \text{if } \mathcal{N}(n) \geq \mathcal{N}(o) \end{aligned} \tag{3.17}$$

Other choices for $acc(o \rightarrow n)$ are possible 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.
In summary, then, in the Metropolis scheme, the transition probability for going from state o to state n is given by

$$\begin{aligned} \pi(o \rightarrow n) &= \alpha(o \rightarrow n) && \mathcal{N}(n) \geq \mathcal{N}(o) \\ &= \alpha(o \rightarrow n)[\mathcal{N}(n)/\mathcal{N}(o)] && \mathcal{N}(n) < \mathcal{N}(o) \\ \pi(o \rightarrow o) &= 1 - \textstyle\sum_{n \neq o} \pi(o \rightarrow n). \end{aligned} \tag{3.18}$$

Note that we still have not specified the matrix α, except for the fact that it must be symmetric. This reflects considerable freedom in the choice of our trial moves. We will come back to this point in subsequent sections.
One thing that we have not yet explained is how to decide whether a trial move is to be accepted or rejected. The usual procedure is as follows. Suppose that we have generated a trial move from state o to state n with $\mathcal{U}(n) > \mathcal{U}(o)$. According to equation (3.16) this trial move should be accepted with a probability

$$acc(o \rightarrow n) = \exp\{-(\beta[\mathcal{U}(n) - \mathcal{U}(o)]\} < 1$$

In order to decide whether to accept or reject the trial move, we generate a random number, denoted by $Ranf$, from a uniform distribution in the interval [0, 1]. Clearly, the probability that $Ranf$ is less than $acc(o \rightarrow n)$ is equal to $acc(o \rightarrow n)$. We now accept the trial move if $Ranf < acc(o \rightarrow n)$ and reject it otherwise. This rule guarantees that the probability to accept a move from o to n is indeed equal to $acc(o \rightarrow n)$. Obviously, it is very important that our random

number generator does indeed generate numbers uniformly in the interval [0, 1]. Otherwise the Monte Carlo sampling will be biased. The quality of random number generators should never be taken for granted.
Thus far, we have not mentioned another condition that $\pi(o \rightarrow n)$ should satisfy, namely that it is ergodic (i.e., every accessible point in configuration space can be reached in a finite number of Monte Carlo steps from any other point). Although some simple MC schemes are guaranteed to be ergodic, these are often not the most efficient schemes. Conversely, many efficient Monte Carlo schemes have either not been proven to be ergodic or, worse, been proven to be nonergodic. The solution is usually to mix the efficient nonergodic scheme with an occasional trial move of the ergodic scheme. The method as a whole will then be ergodic (at least, in principle).

3.3 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 works.

3.3.1 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, 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; $\mathbf{r}' = \mathbf{r} + \Delta$, and calculate its new energy $\mathcal{U}(\mathbf{r}'^N)$.
3. Accept the move from $\mathbf{r}^N$ to $\mathbf{r}'^N$ with probability

$$\mathrm{acc}(o \rightarrow n) = \min\left(1, \exp\{-\beta[\mathcal{U}(\mathbf{r}'^N) - \mathcal{U}(\mathbf{r}^N)]\}\right) \tag{3.19}$$

3.4 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 understudy. Rather, we shall simply assume that we have an atomic or molecular model system in a 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.

3.4.1 Translational Moves

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 $-\Delta$ and $+\Delta$ to the x, y, and z coordinates of the molecular center of mass:

$$\begin{aligned} x_i' &\rightarrow x_i + \Delta\,(\mathrm{Ranf} - 0.5) \\ y_i' &\rightarrow y_i + \Delta\,(\mathrm{Ranf} - 0.5) \\ z_i' &\rightarrow z_i + \Delta\,(\mathrm{Ranf} - 0.5) \end{aligned} \tag{3.20}$$

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 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. 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.

Using this criterion it is easy to show that for simulations of condensed phases it is usually advisable to perform random displacements of one particle at a time (as we shall see later, the situation is different for correlated displacements). To see why random single-particle moves are preferred, consider a system of N spherical particles, interacting through a potential energy function $\mathcal{U}(\mathbf{r}^N)$. Typically, we expect that a trial move will be rejected if the potential energy of the system changes by much more than k_BT. At the same time, we try to make the Monte Carlo trial steps as large as is possible without having a very low acceptance. A displacement that would, on average, give rise to an increase of the potential energy by k_BT would still have a reasonable acceptance. In the case of a single-particle trial move, we then have

$$\begin{aligned} \langle \Delta\mathcal{U} \rangle &= \left\langle \frac{\partial \mathcal{U}}{\partial r_i^\alpha} \right\rangle \overline{\Delta r_i^\alpha} + \frac{1}{2} \left\langle \frac{\partial^2 \mathcal{U}}{\partial r_i^\alpha \partial r_i^\beta} \right\rangle \overline{\Delta r_i^\alpha \Delta r_i^\beta} + \cdots \\ &= 0 + f(\mathcal{U})\, \overline{\Delta r_i^2} + \mathcal{O}(\Delta^4) \end{aligned} \qquad (3.21)$$

where the angle brackets denote averaging over the ensemble and the horizontal bar denotes averaging over random trial moves. The second derivative of $\mathcal{U}$ has been absorbed into the function $f(\mathcal{U})$, the precise form of which does not concern us here. If we now equate $(\Delta\mathcal{U})$ on the right-hand side of equation (3.21) to k_BT, we find the following expression for $\overline{\Delta r_i^2}$:

$$\overline{\Delta r_i^2} \approx k_B T / f(\mathcal{U}) \qquad (3.22)$$

If we attempt to move N particles, one at a time, most of the computation is spent on the evaluation of the change in potential energy. Assuming that we use a neighbor list or a similar time-saving device, the total time spent on evaluation the potential energy change is proportional to nN, where n is the average number of interaction partners per molecule. The sum of the mean-square displacements will be proportional to $N\overline{\Delta r^2} \sim N k_B T / f(\mathcal{U})$. Hence, the mean-square displacement per unit of CPU time will still be proportional to: $k_BT/(nf(\mathcal{U}))$. Now suppose that we try to move all particles at once. The cost in CPU time will still be proportional to nN. But, using the same reasoning as in equations (3.21) and (3.22), we estimate that the sum of the mean-square displacements is smaller by a factor $1/N$. Hence the total efficiency will be down by this same factor. This simple argument explains why most simulators use single-particle, rather than collective trial moves. It is important to note that we have assumed that a collective MC trial

move consists of N independent trial displacements of the particles. It is also possible to construct efficient collective MC moves, in which the trial displacements of the individual particles are not chosen independently.
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 time required to test whether a trial move is accepted depends on the magnitude of the move (see Figure 3.2). 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.

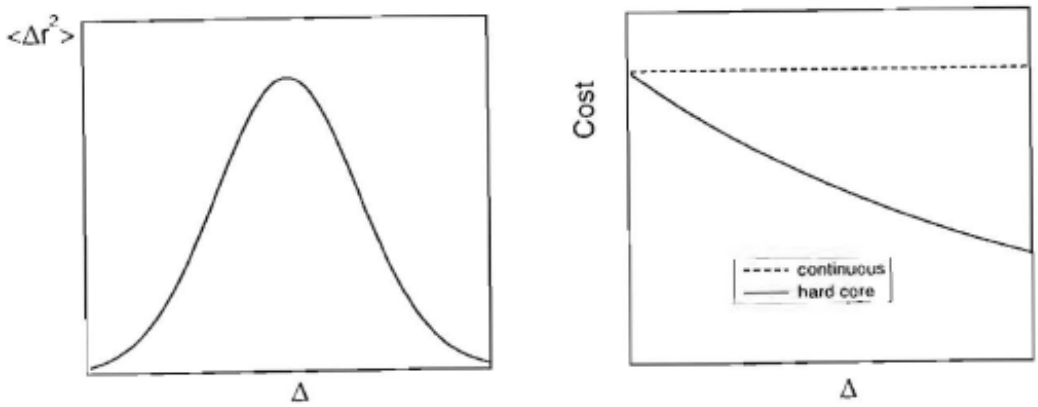

Figure 3.2: (left) Typical dependence of the mean-square displacement of a particle on the average size Δ of the trial move. (right) Typical dependence of the computational cost of a trial move on the step-size Δ. For continuous potentials, the cost is constant, while for hard-core potentials it decreases rapidly with the size of the trial move.

Exactly how much depends on the nature of the program, 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.

A distinct disadvantage of the efficiency criterion discussed previously is that it does not allow us to detect if the sampling of configuration space is ergodic. To take a specific example, suppose that our system consists of a number of particles that are trapped in different potential energy minima. Clearly, we can sample the vicinity of these minima quite well and still have totally inadequate sampling of the whole of configuration space. A criterion that would detect such nonergodicity has been proposed by Mountain and Thirumalai. These authors consider the difference between the variance of the time average of the (potential) energy of all particles. Let us denote the time average of the energy of particle j in time interval t by $e_j(t)$:

$$e_j(t) = \frac{1}{t}\int_0^t dt'\, e_j(t')$$

And the average single particle energy for this interval is

$$\bar{e}(t) \equiv \frac{1}{N}\sum_{j=1}^{N} e_j(t)$$

The variance of interest is

$$\sigma_E^2(t) \equiv \frac{1}{N}\sum_{j=1}^{N} [e_j(t) - \bar{e}(t)]^2$$

If all particles sample the whole of configuration space, $\sigma_E^2(t)$ will approach zero as $t \to \infty$:

$$\sigma_E^2(t)/\sigma_E^2(0) \to \tau_E/t$$

where τ_E is a measure for the characteristic time to obtain uncorrelated samples. However, if the system is nonergodic, as in a (spin) glass, Ω_E will not decay to zero. The work of Mountain and Thirumalai suggests that a good method to optimize the efficiency of a Monte Carlo scheme is to minimize the product of τ_E and the computer time per trial move. Using this scheme, Mountain and Thirumalai concluded that, even for the Lennard-Jones system, a trial move acceptance of 50% is far from optimal. They found that an acceptance probability of 20% was twice as efficient.

Of course, in some situations an efficiency criterion based on ergodicity is not useful. By construction, it cannot be used to optimize simulations of glasses. But also when studying interfaces (e.g., solid-liquid or liquid-vapor) the ergodicity criterion would suggest that every particle should have ample time to explore both coexisting phases. This is clearly unnecessary: ice can be in equilibrium with water, even though the time of equilibration is far too short to allow complete exchange of the molecules in the two phases.

3.4.2 Orientational Moves

If we are simulating molecules rather than atoms we must also generate trial moves that change the molecular orientation. As we discussed already, it almost requires an effort to generate translational trial moves with a distribution that does not satisfy the symmetry requirement of the underlying Markov chain. For rotational moves, the situation is very different. It is only too easy to introduce a systematic bias in the orientational distribution function of the molecules by using a nonsymmetrical orientational sampling scheme. Several different strategies to generate rotational displacements exist, here we only mention one possible approach.

Rigid, Linear Molecules

Consider a system consisting of N linear molecules. We specify the orientation of the ith molecule by a unit vector $\mathbf{u}_i$. One possible procedure to change $\mathbf{u}_i$ by a small, random amount is the following. First, we generate a unit $\mathbf{v}$ with a random orientation. Next we multiply this random unit $\mathbf{v}$ by a scale factor y. The magnitude of y determines the magnitude of the trial rotation. We now add $\gamma\mathbf{v}$ to $\mathbf{u}_i$. Let us denote the resulting sum vector by $\mathbf{t}$: $\mathbf{t} = \gamma\mathbf{v} + \mathbf{u}_i$. Note that $\mathbf{t}$ is not a unit vector. Finally, we normalize $\mathbf{t}$, and the result is our trial orientation vector $\mathbf{u}_i'$. We still have to fix γ, which determines the acceptance probability for the orientational trial move. The value of γ is determined by essentially the same criteria as for translational moves. We have not yet indicated whether or not the translational and orientational trial moves should be performed simultaneously. Both procedures are acceptable. However, if rotation and translation separate moves, then the selection of the type of move should be probabilistic rather than deterministic.

Rigid, Nonlinear Molecules

Only slightly more complex is the case of a nonlinear rigid molecule. It is conventional to describe the orientation of nonlinear molecules in terms of the Eulerian angles (ϕ,θ,ψ). However, for most simulations, use of these angles is less convenient because all rotation operations should then be expressed in terms of trigonometric functions, and these are computationally expensive. It is usually better to express the orientation of such a molecule in terms of quaternion parameters. The rotation of a rigid body can be specified by a quaternion of unit norm $\mathbf{Q}$. Such a quaternion may be thought of as a unit vector in four-dimensional space:

$$\mathbf{Q} \equiv (q_0, q_1, q_2, q_3) \quad \text{with } q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \tag{3.23}$$

There is a one-to-one correspondence between the quaternion components q_α and the Eulerian angles:

$$\begin{aligned} q_0 &= \cos\frac{\theta}{2}\cos\left(\frac{\phi+\psi}{2}\right) \\ q_1 &= \sin\frac{\theta}{2}\cos\left(\frac{\phi-\psi}{2}\right) \\ q_2 &= \sin\frac{\theta}{2}\sin\left(\frac{\phi-\psi}{2}\right) \\ q_3 &= \cos\frac{\theta}{2}\sin\left(\frac{\phi+\psi}{2}\right) \end{aligned} \tag{3.24}$$

and the rotation matrix R, which describes the

rotation of the molecule-fixed vector in the laboratory frame, is given by:

$$\mathsf{R} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \tag{3.25}$$

To generate trial rotations of nonlinear rigid bodies, we must rotate the vector (q_0, q_1, q_2, q_3) on the four-dimensional (4D) unit sphere. The procedure just described for the rotation of a 3D unit vector is easily generalized to 4D.

Nonrigid Molecules

If the molecules under consideration are not rigid then we must also consider Monte Carlo trial moves that change the internal degrees of freedom of a molecule. In practice, it makes an important difference whether or not we have frozen out some of the internal degrees of freedom of a molecule by imposing rigid constraints on, say, bond lengths and possibly even some bond angles. If not, the situation is relatively simple: we can carry out normal trial moves on the Cartesian coordinates of the individual atoms in the molecule (in addition to center of mass moves). If some of the atoms are strongly bound, it is advisable to carry out small trial moves on those particles (no rule forbids the use of trial moves of different size for different atoms, as long as the moves for one particular atom are always sampled from the same distribution).

However, when the bonds between different atoms become very stiff, this procedure does not sample conformational changes of the molecule efficiently. In Molecular Dynamics simulations it is common practice to replace very stiff intramolecular interactions by rigid constraints.

For Monte Carlo simulations this is also possible. In fact, elegant techniques have been developed for this purpose. However, the corresponding MD techniques are so much easier to use, in particular for large molecules, that one cannot recommend the use of the Monte Carlo technique for any but the smallest flexible molecules with internal constraints.

To understand why Monte Carlo simulations of flexible molecules with a number of stiff (or even rigid) bonds (or bond angles) can become complicated, let us return to the original expression (3.2) for a thermal average of a function $\mathcal{A}(\mathbf{r}^N)$:

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

If we are dealing with flexible molecules, it is convenient to perform Monte sampling not on the Cartesian coordinates $\mathbf{r}^N$ but on the generalized coordinates $\mathbf{q}^N$, where q may be, for instance, a bond length or an internal angle. We must now express the Hamiltonian in equation (3.2) in terms of these generalized coordinates and their conjugate momenta. This is done most conveniently by first considering the Lagrangian $\mathcal{L} = \mathcal{K} - \mathcal{U}$, where $\mathcal{K}$ is the kinetic energy of the system $(\mathcal{K} = \sum (1/2) m \dot{r}^2)$ and $\mathcal{U}$ the potential energy. When we transform from Cartesian coordinates $\mathbf{r}$ to generalized coordinates

$\mathbf{q}$, $\mathcal{K}$ changes to

$$\begin{aligned} \mathcal{L} &= \sum_{i=1}^{N} \frac{1}{2} m_i \frac{\partial \mathbf{r}_i}{\partial q_\alpha} \frac{\partial \mathbf{r}_i}{\partial q_\beta} \dot{q}_\alpha \dot{q}_\beta - \mathcal{U}(\mathbf{q}^N) \\ &\equiv \frac{1}{2} \dot{\mathbf{q}} \cdot \mathsf{G} \cdot \dot{\mathbf{q}} - \mathcal{U}(\mathbf{q}^N). \end{aligned} \tag{3.26}$$

In the second line of equation (3.26) we have defined the matrix G. The momenta conjugate to $\mathbf{q}^N$ are easily derived using

$$p^\alpha \equiv \frac{\partial \mathcal{L}}{\partial \dot{q}_\alpha}$$

This yields $p^\alpha = \mathsf{G}_{\alpha\beta}\dot{q}_\beta$. We can now write down the Hamiltonian $\mathcal{H}$ in terms of the generalized coordinates and conjugate momenta:

$$\mathcal{H}(\mathbf{p}, \mathbf{q}) = \frac{1}{2} \dot{\mathbf{p}} \cdot \mathsf{G}^{-1} \cdot \dot{\mathbf{p}} - \mathcal{U}(\mathbf{q}^N) \tag{3.27}$$

If we now insert this form of the Hamiltonian into equation (3.2), and carry out the (Gaussian) integration over the momenta, we find

$$\begin{aligned} \langle A \rangle &= \frac{\int d\mathbf{q}^N \exp[-\beta \mathcal{U}(\mathbf{q}^N)] A(\mathbf{q}^N) \int d\mathbf{p}^N \exp(-\beta \mathbf{p} \cdot \mathsf{G}^{-1} \cdot \mathbf{p}/2)}{\int d\mathbf{q}^N d\mathbf{p}^N \exp(-\beta \mathcal{H})} \\ &= \frac{\int d\mathbf{q}^N \exp[-\beta \mathcal{U}(\mathbf{q}^N)] A(\mathbf{q}^N) |\mathsf{G}|^{\frac{1}{2}}}{\int d\mathbf{q}^N d\mathbf{p}^N \exp(-\beta \mathcal{H})} \end{aligned} \tag{3.28}$$

The problem with equation (3.28) is the term $|\mathsf{G}|^{1/2}$. Although the determinant $|\mathsf{G}|$ can be computed fairly easily for small flexible molecules, its evaluation can become quite an unpleasant task in the case of larger molecules.

Thus far we have considered the effect of introducing generalized coordinates only on the form of the expression for thermal averages. If we are considering a situation where some of the generalized coordinates are actually constrained to have a fixed value, then the picture changes again, because such hard constraints are imposed at the level of the Lagrangian equations of motion. Hard constraints therefore lead to a different form for the Hamiltonian in equation (3.27) and to another determinant in equation (3.28). Again, all this can be taken into account in the Monte Carlo sampling. An example of such a Monte Carlo scheme is the concerted rotation algorithm that has been developed by Theodorou and coworkers to simulate polymer melts and glasses. The idea of this algorithm is to select a set of adjacent skeletal bonds in a chain (up to seven bonds). These bonds are given a collective rotation while the rest of the chain is unaffected. By comparison, Molecular Dynamics simulations of flexible molecules with hard constraints have the advantage that these constraints enter directly into the equations of motion. The distinction between Molecular Dynamics and Monte Carlo, however, is more apparent than real, since it is possible to use MD techniques to generate collective Monte Carlo moves.