to approximate a true dynamical trajectory of the system under consideration.

This attractive feature of algorithms that can be derived from a discretized action has inspired Elber and co-workers to construct a novel class of MD algorithms that are designed to yield reasonable long-time dynamics with very large time steps [76,77]. In fact, Elber and co-workers do not base their approach on the discretization of the classical action but on the so-called Onsager-Machlup action [78]. The reason for selecting this more general action is that the Onsager-Machlup action is a *minimum* for the true trajectory, while the Lagrangian action is only an extremum. It would carry too far to discuss the practical implementation of the algorithm based on the Onsager-Machlup action. For details, we refer the reader to refs. [76,77].

4.4 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

$$k_B T = \frac{\langle 2\mathcal{K} \rangle}{f}. \tag{4.4.1}$$

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., [79])

$$P = \rho k_B T + \frac{1}{dV} \left\langle \sum_{i<j} \mathbf{f}(\mathbf{r}_{ij}) \cdot \mathbf{r}_{ij} \right\rangle, \tag{4.4.2}$$

where d is the dimensionality of the system, and $f(\mathbf{r}_{ij})$ is the force between particles i and j at a distance 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.* [80] have derived a general procedure to convert averages from one ensemble to another. A more recent (and more accessible) description of these interensemble transformations has been given by Allen and

Tildesley [41]. 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:

$$\langle \mathcal{K}^2 \rangle_{NVE} - \langle \mathcal{K} \rangle_{NVE}^2 = \frac{3k_B^2 T^2}{2N} \left(1 - \frac{3k_B}{2C_V} \right). \tag{4.4.3}$$

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. Methods to calculate these properties are discussed in Chapter 7.

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. In Algorithm 7 an implementation to compute the radial distribution function is described. 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

Algorithm 7 (The Radial Distribution Function)

```
subroutine gr(switch)                radial distribution function
                                     switch = 0 initialization,
                                     = 1 sample, and = 2 results
if (switch.eq.0) then                initialization
  ngr=0
  delg=box/(2*nhis)                  bin size
  do i=0,nhis                        nhis total number of bins
     g(i)=0
  enddo
else if (switch.eq.1) then           sample
  ngr=ngr+1
  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
      r=sqrt(xr**2)
      if (r.lt.box/2) then           only within half the box length
         ig=int(r/delg)
         g(ig)=g(ig)+2               contribution for particle i and j
      endif
    enddo
  enddo
else if (switch.eq.2) then           determine g(r)
  do i=1,nhis
    r=delg*(i+0.5)                   distance r
    vb=((i+1)**3-i**3)*delg**3       volume between bin i+1 and i
    nid=(4/3)*pi*vb*rho              number of ideal gas part. in vb
    g(i)=g(i)/(ngr*npart*nid)        normalize g(r)
  enddo
endif
return
end
```

Comments to this algorithm:

1. *For efficiency reasons the sampling part of this algorithm is usually combined with the force calculation (for example, Algorithm 5).*
2. *The factor* `pi` = *3.14159....*

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.

4.4.1 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 j of the diffusing species is proportional to the negative gradient in the concentration of that species:

$$\mathrm{j} = -D\nabla c, \tag{4.4.4}$$

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 where 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:

$$\frac{\partial c(r,t)}{\partial t} + \nabla \cdot \mathrm{j}(r,t) = 0. \tag{4.4.5}$$

Combining equation (4.4.5) with equation (4.4.4), we obtain

$$\frac{\partial c(r,t)}{\partial t} - D\nabla^2 c(r,t) = 0. \tag{4.4.6}$$

We can solve equation (4.4.6) 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:

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

where we have used the fact that we have imposed

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

We can directly obtain an equation for the time evolution of $\langle r^2(t)\rangle$ by multiplying equation (4.4.6) by r^2 and integrating over all space. This yields

$$\frac{\partial}{\partial t}\int \mathbf{dr}\; r^2 c(r,t) = D\int \mathbf{dr}\; r^2\nabla^2 c(r,t). \tag{4.4.7}$$

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

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

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

$$\begin{aligned}
\frac{\partial\langle r^2(t)\rangle}{\partial t} &= D\int \mathbf{dr}\; r^2\nabla^2 c(r,t)\\
&= D\int \mathbf{dr}\; \boldsymbol{\nabla}\cdot(r^2\boldsymbol{\nabla} c(r,t)) - D\int \mathbf{dr}\; \boldsymbol{\nabla} r^2\cdot\boldsymbol{\nabla} c(r,t)\\
&= D\int \mathbf{dS}\; (r^2\boldsymbol{\nabla} c(r,t)) - 2D\int \mathbf{dr}\; \mathbf{r}\cdot\boldsymbol{\nabla} c(r,t)\\
&= 0 - 2D\int \mathbf{dr}\; (\boldsymbol{\nabla}\cdot \mathbf{r}c(r,t)) + 2D\int \mathbf{dr}\; (\boldsymbol{\nabla}\cdot\mathbf{r})c(r,t)\\
&= 0 + 2dD\int \mathbf{dr}\; c(r,t)\\
&= 2dD.
\end{aligned} \tag{4.4.8}$$

Equation (4.4.8) 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-squared 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:

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

This plot would look like the one that will be shown later in Figure 4.6. 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

$$2D = \lim_{t\to\infty} \frac{\partial \langle x^2(t)\rangle}{\partial t}, \tag{4.4.9}$$

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

$$\begin{aligned}
\langle x^2(t)\rangle &= \left\langle \left(\int_0^t dt' \, v_x(t') \right)^2 \right\rangle \\
&= \int_0^t \int_0^t dt'dt'' \, \langle v_x(t')v_x(t'')\rangle \\
&= 2\int_0^t \int_0^{t'} dt'dt'' \, \langle v_x(t')v_x(t'')\rangle .
\end{aligned} \tag{4.4.10}$$

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 (4.4.10) in equation (4.4.9), we obtain

$$\begin{aligned}
2D &= \lim_{t\to\infty} 2\int_0^t dt'' \, \langle v_x(t-t'')v_x(0)\rangle \\
D &= \int_0^\infty d\tau \, \langle v_x(\tau)v_x(0)\rangle .
\end{aligned} \tag{4.4.11}$$

In the last line of equation (4.4.11) 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* (see Appendix C for some details). Green-Kubo relations have been derived for many other transport coefficients, such as the shear viscosity η,

$$\eta = \frac{1}{Vk_BT}\int_0^\infty dt\ \langle \sigma^{xy}(0)\sigma^{xy}(t)\rangle \tag{4.4.12}$$

with

$$\sigma^{xy} = \sum_{i=1}^{N}\left(m_i v_i^x v_i^y + \frac{1}{2}\sum_{j\neq i} x_{ij} f_y(r_{ij})\right); \tag{4.4.13}$$

the thermal conductivity λ_T,

$$\lambda_T = \frac{1}{Vk_BT^2}\int_0^\infty dt\ \langle j_z^e(0) j_z^e(t)\rangle \tag{4.4.14}$$

with

$$j_z^e = \frac{d}{dt}\sum_{i=1}^{N} z_i \frac{1}{2}\left(m_i v_i^2 + \sum_{j\neq i} v(r_{ij})\right); \tag{4.4.15}$$

and electrical conductivity σ_e

$$\sigma_e = \frac{1}{Vk_BT}\int_0^\infty dt\ \langle j_x^{el}(0) j_x^{el}(t)\rangle \tag{4.4.16}$$

with

$$j_x^{el} = \sum_{i=1}^{N} q_i v_i^x. \tag{4.4.17}$$

For details, see, for example, [79]. 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. In Algorithm 8 an implementation of the calculation of the mean-squared displacement and velocity autocorrelation function is described.

4.4.2 Order-n Algorithm to Measure Correlations

The calculation of transport coefficients from the integral of a time-correlation function, or from a (generalized) Einstein relation, may require a lot of memory and CPU time, in particular if fluctuations decay slowly. As an example, we consider again the calculation of the velocity autocorrelation function and the measurement of the diffusion coefficient. In a dense

```
      subroutine dif(switch,nsamp)         diffusion; switch = 0 init.
                                           = 1 sample, and = 2 results
      if (switch.eq.0) then                Initialization
         ntel=0                            time counter
         dtime=dt*nsamp                    time between two samples
         do i=1,tmax                       tmax total number of time step
            ntime(i)=0                     number of samples for time i
            vacf(i)=0
            r2t(i)=0
         enddo
      else if (switch.eq.1) then           sample
         ntel=ntel+1
         if (mod(ntel,it0).eq.0) then      decide to take a new t = 0
            t0 = t0 + 1                    update number of t = 0
            tt0=mod(t0-1,t0max)+1          see note 1
            time0(tt0)=ntel                store the time of t = 0
            do i=,npart
               x0(i,tt0)=x(i)              store position for given t = 0
               vx0(i,tt0)=vx(i)            store velocity for given t = 0
            enddo
         endif
         do t=1,min(t0,t0max)              update vacf and r2, for t = 0
           delt=ntel-time0(t)+1            actual time minus t = 0
           if (delt.lt.tmax) then
            ntime(delt)=ntime(delt)+1
            do i=1,npart
               vacf(delt)=vacf(delt)+      update velocity autocorr.
     +            vx(i)*vx0(i,t)
               r2t(delt)=r2t(delt)+        update mean-squared displ.
     +               (x(i)-x0(i,t))**2
            enddo
           endif
         enddo
       else if (switch.eq.2) then          determine results
         do i=1,tmax
           time=dtime*(i+0.5)              time
           vacf(i)=vacf(i)                 volume velocity autocorr.
     +               /(npart*ntime(i))
           r2t(i)=r2t(i)                   mean-squared displacement
     +              /(npart*ntime(i))
         enddo
      endif
      return
      end
```

Comments to this algorithm:

1. *We define a new* `t` $=$ `0` *after each* `it0` *times this subroutine has been called. For each* `t` $=$ `0`*, we store the current positions and velocities. The term* `t0max` *is the maximum number of* `t` $=$ `0` *we can store. If we sample more, the first* `t` $=$ `0` *will be removed and replaced by a new one. This limits the maximum time we collect data to* `t0max*it0`*; this number should not be smaller than* `tmax`*, the total number of time steps we want to sample.*
2. *Because* `nsamp` *gives the frequency at which this subroutine is called, the time between two calls is* `nsamp*delt`*, where* `delt` *is the time step.*

medium, the velocity autocorrelation function changes rapidly on typically microscopic time scales. It therefore is important to have an even shorter time interval between successive samples of the velocity. Yet, when probing the long-time decay of the velocity autocorrelation function, it is not necessary to sample with the same frequency. The conventional schemes for measuring correlation functions do not allow for such adjustable sampling frequencies. Here, we describe an algorithm that allows us to measure fast and slow decay simultaneously at minimal numerical cost. This scheme can be used to measure the correlation function itself, but in the example that we discuss, we show how it can be used to compute the transport coefficient.

Let us denote by Δt the time interval between successive measurements of the velocity of the particles in the system. We can define block sums of the velocity of a given particle as follows:

$$\mathbf{v}^{(i)}(j) \equiv \sum_{l=(j-1)n+1}^{jn} \mathbf{v}^{(i-1)}(l) \tag{4.4.18}$$

with

$$\mathbf{v}^{(0)}(l) \equiv \mathbf{v}(l), \tag{4.4.19}$$

where $\mathbf{v}(l)$ is the velocity of a particle at time l. Equation (4.4.18) is a recursive relation between block sums of level i and $i-1$. The variable n determines the number of terms in the summation. For example, $\mathbf{v}^{(3)}(j)$ can

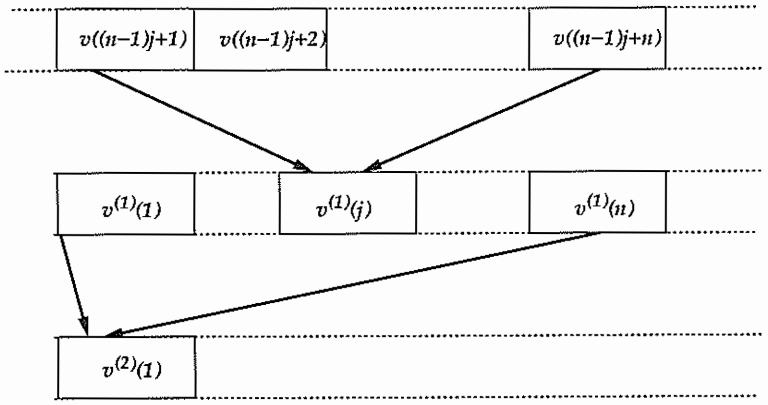

Figure 4.2: Coarse graining the velocities.

be written as

$$\begin{aligned}
v^{(3)}(j) &= \sum_{l_1=(j-1)n+1}^{jn} v^{(2)}(l_1) \\
&= \sum_{[l_1=(j-1)n+1]}^{jn} \sum_{[l_2=(l_1-1)n+1]}^{l_1 n} \sum_{[l_3=(l_2-1)n+1]}^{l_2 n} v(l_3) \\
&= \sum_{l=(j-1)n^3+1}^{n^3 j} v(l) \\
&\approx \frac{1}{\Delta t} \int_{l=(j-1)n^3+1}^{n^3 j} dt\, v(t) = \frac{\mathbf{r}(n^3 j) - \mathbf{r}(n^3(j-1)+1)}{\Delta t}.
\end{aligned}$$

Clearly, the block sum of the velocity is related to the displacement of the particle in a time interval $n^i \Delta t$. In Figure 4.2 the blocking operation is illustrated. From the preceding block sums, it is straightforward to compute the velocity autocorrelation function with a resolution that decreases with increasing time. At each level of blocking, we need to store $n \times N$ block sums, where N is the number of particles (in practice, it will be more convenient to store the *block-averaged* velocities).

The total storage per particle for a simulation of length $t = n^i \Delta t$ is $i \times n$. This should be compared to the conventional approach where, to study correlations over the same time interval, the storage per particle would be n^i. In the conventional calculation of correlation functions, the number of floating-point operations scales at t^2 (or $t \ln t$, if the fast Fourier technique is used). In contrast, in the present scheme the number of operations scales as t.

At each time step we have to update $v^{(0)}(t)$ and correlate it with all n entries in the $\mathbf{v}^{(0)}$-array. The next block sum has to be updated and correlated once every n time steps, the third every every n^2 steps, etc. This yields, for the total number of operations,

$$\frac{t}{\Delta t} \times n \left(1 + \frac{1}{n} + \frac{1}{n^2} + \cdots + \frac{1}{n^i}\right) < \frac{t}{\Delta t} n \frac{n}{n-1}.$$

Using this approach, we can quickly and efficiently compute a wide variety of correlation functions, both temporal and spatial. However, it should be stressed that each blocking operation leads to more coarse graining. Hence, any high-frequency modulation of long-time behavior of such correlation functions will be washed out.

Interestingly enough, even though the velocity autocorrelation function itself is approximate at long times, we can still compute the integral of the velocity autocorrelation function (i.e., the diffusion coefficient), with no loss in numerical accuracy. Next, we discuss in some detail this technique for computing the diffusion coefficient.

Let us define

$$\Delta\bar{\mathbf{r}}^{(i)}(j) \equiv \sum_{l=0}^{j} \mathbf{v}^{(i)}(l)\Delta t = \mathbf{r}(n^i) - \mathbf{r}(0). \tag{4.4.20}$$

The square of the displacement of the particle in a time interval $n^i\Delta t$ can be written as

$$(\Delta\bar{\mathbf{r}}^2)^{(i)}(j) = \left[\mathbf{r}(n^i) - \mathbf{r}(0)\right]^2 = \Delta\bar{\mathbf{r}}^{(i)}(j) \cdot \Delta\bar{\mathbf{r}}^{(i)}(j). \tag{4.4.21}$$

To compute the diffusion coefficient, we should follow the time dependence of the mean-squared displacement. As a first step, we must determine $\Delta\bar{\mathbf{r}}^{(i)}(j)$ for all i and all j. In fact, to improve the statistics, we wish to use every sample point as a new time origin. To achieve this, we again create arrays of length n. However, these arrays do not contain the same block sums as before, but partial block sums (see Algorithm 9).

1. At every time interval Δt, the lowest-order blocking operation is performed through the following steps:

   (a) We first consider the situation that all lowest-order accumulators have already been filled at least once (this is true if $t > n\Delta t$). The value of the current velocity $v(t)$ is added to

   $$\mathbf{v}_{\text{sum}}(1, j) = \mathbf{v}_{\text{sum}}(1, j+1) + \mathbf{v}(t)$$

   for `j = 1,n-1`, and

   $$\mathbf{v}_{\text{sum}}(1, j) = \mathbf{v}(t)$$

   for `j = n`.

Algorithm 9 (Diffusion: Order-n Algorithm)

```
subroutine dif(switch,nsamp)     diffusion
                                 switch = 0 initialization,
                                 = 1 sample, and = 2 results
if (switch.eq.0) then            initialization
  ntel=0                         time counter for this subroutine
  dtime=dt*nsamp                 time between two samples
  do ib=1,ibmax                  ibmax max. number of blocks
   ibl(ib)=0                     length of current block
   do j=1,n                      n number of steps in a block
    tel(ib,j)=0                  counter number of averages
    delr2(ib,j)=0                running average mean-sq. displ.
    do i=1,npart
      vxsum(ib,j,i)=0            coarse-grained velocity particle i
    enddo
   enddo
  enddo
else if (switch.eq.2) then       print results
  do ib=1,max(ibmax,iblm)
    do j=2,min(ibl(ib),n)
     time=dtime*j*n**(ib-1)      time
     r2=delr2(ib,j)*dtime**2     mean-squared displacement
            /tel(ib,j)
    enddo
  enddo
   ...(continue)....
```

(b) These operations yield

$$\mathbf{v}_{\text{sum}}(1,l) = \sum_{j=t-n+l}^{j=t} \mathbf{v}(j).$$

The equation allow us to update the accumulators for the mean-squared displacement (4.4.21) for $l = 1, 2, \ldots, n$:

$$(\Delta \bar{\mathbf{r}}^2)^{(0)}(l) = (\Delta \bar{\mathbf{r}}^2)^{(0)}(l) + \mathbf{v}_{\text{sum}}^2(1,l)\Delta t^2.$$

2. If the current time step is a multiple of n, we perform the first blocking operation, if it is a multiple of n^2 the second, etc. Performing blocking operation i involves the following steps:

```
     ...(continue)....
   else if (switch.eq.1) then              sample
    ntel=ntel+1
    iblm=MaxBlock(ntel,n)                  maximum number of possible
                                           blocking operations
    do ib=1,iblm
     if (mod(ntel,n**(ib-1))               test if ntel is a multiple of n^ib
 +            .eq.0) then
      ibl(ib)=ibl(ib)+1                    increase current block length
      inm=max(ibl(ib),n)                   set maximum block length to n
      do i=1,npart
        if(ib.eq.1) then
          delx=vx(i)                       0th block: ordinary velocity
        else
          delx=vxsum(ib-1,1,i)             previous block velocity
        endif
        do in=1,inm
          if (inm.ne.n) then               test block length equal to n
            inp=in
          else
            inp=in+1
          endif
          if (in.lt.inm) then
           vxsum(ib,in,i)=
 +          vxsum(ib,inp,i)+delx           eqns. (4.4.22) or (4.4.25)
          else
           vxsum(ib,in,i)=delx             eqns. (4.4.23) or (4.4.26)
          endif
        enddo
        do in=1,inm
         tel(ib,in)=tel(ib,in)+1           counter number of updates
         delr2(ib,in)=delr2(ib,in)         update equation (4.4.24)
 +         +vxsum(ib,inm-in+1,i)**2
        enddo
      enddo
     endif
    enddo
   endif
   return
   end
```

Comment to this algorithm:

1. `MaxBlock(ntel,n)` *gives the maximum number of blocking operations that can be performed on the current time step* `ntel`.

(a) As before, we first consider the situation that all ith-order accumulators have already been filled at least once (i.e., $t > n^i\Delta t$). Using the $i-1$th block sum ($v_{sum}(i-1,1)$), we update

$$v_{sum}(i,j) = v_{sum}(i,j+1) + v_{sum}(i-1,1) \tag{4.4.22}$$

for `j = 1,n-1`, and

$$v_{sum}(i,j) = v_{sum}(i-1,1) \tag{4.4.23}$$

for `j = n`.

(b) These operations yield

$$v_{sum}(i,l) = \sum_{j=n-l+1}^{j=n} v_{sum}(i-1,j).$$

The equations allows us to update the accumulators for the mean-squared displacement, equation (4.4.21), for $l = 1,2,\ldots,n$:

$$(\Delta r^2)^{(i)}(l) = (\Delta r^2)^{(i)}(l) + v_{sum}^2(i,l)\Delta t^2. \tag{4.4.24}$$

3. Finally, we must consider how to handle arrays that have not yet been completely filled. Consider the situation that only nmax of the n locations of the array that contains the ith-level sums have been initialized. In that case, we should proceed as follows:

(a) Update the current block length: `nmax = nmax+1` (nmax $< n$).

(b) For `j = 1,nmax-1`

$$v_{sum}(i,j) = v_{sum}(i,j) + v_{sum}(i-1,1). \tag{4.4.25}$$

(c) For `j = nmax`

$$v_{sum}(i,j) = v_{sum}(i-1,1). \tag{4.4.26}$$

The update of equation (4.4.21) remains the same.

In Case Study 6, a detailed comparison is made between the present algorithm and the conventional algorithm for the diffusion of the Lennard-Jones fluid.

4.5 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 (see also section 3.2.2):

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

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

Case Study 4 (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 [81–83].

In Figure 4.3 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 energies 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 approx. 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 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 [84]. The following operations on the set of data points are performed: we start by calculating the standard deviation of all the data points, then we group two consecutive data points and determine again the standard deviation 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, 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

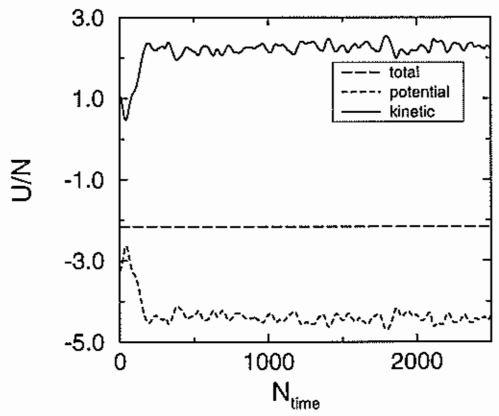

Figure 4.3: Total, potential, and kinetic energy per particle U/N as a function of the number of time steps N_{time}.

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 points 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 are shown in Figure 4.4, 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 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 $U = -4.4190 \pm 0.0012$ 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 4.5, the radial distribution function is shown. To determine this function we used Algorithm 7. 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

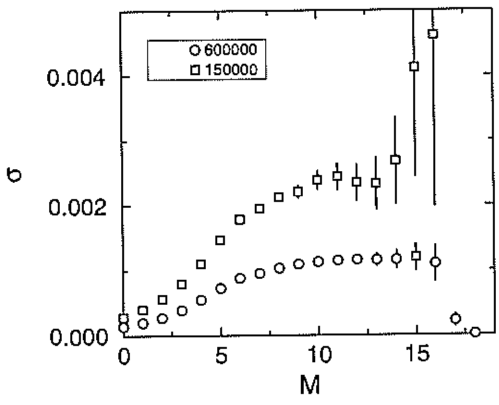

Figure 4.4: The standard deviation σ in the potential energy as a function of the number of block operations M for a simulation of 150,000 and 600,000 time steps. This variance is calculated using equation (D.3.4).

calculated from

$$\begin{aligned} U/N &= \frac{1}{2}\rho \int_0^\infty dr\, u(r)g(r) \\ &= 2\pi\rho \int_0^\infty dr\, r^2 u(r)g(r) \end{aligned} \tag{4.5.1}$$

and for the pressure from

$$\begin{aligned} P &= \rho k_B T - \frac{1}{3}\frac{1}{2}\rho^2 \int_0^\infty dr\, \frac{du(r)}{dr} r g(r) \\ &= \rho k_B T - \frac{2}{3}\pi\rho^2 \int_0^\infty dr\, \frac{du(r)}{dr} r^3 g(r), \end{aligned} \tag{4.5.2}$$

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

Equations (4.5.1) and (4.5.2) 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 5 (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-squared displacement or from the velocity autocorrelation function. We have determined these properties using Algorithm 8.

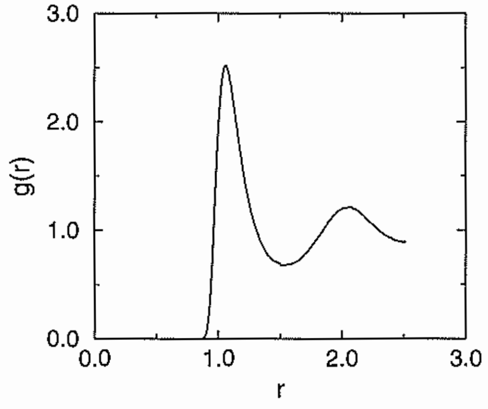

Figure 4.5: Radial distribution function of a Lennard-Jones fluid close to the triple point: $T = 1.5043 \pm 0.0008$ and $\rho = 0.8442$.

In Figure 4.6 the mean-squared displacement is shown as a function of the simulation time. From the mean-squared displacement we can determine the diffusion using equation (4.4.9). This equation, however, is valid only in the limit $t \to \infty$. In practice this means that we have to verify that we have simulated enough that the mean-squared displacement is really proportional to t and not to 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 (4.4.11). In this equation we have to integrate to $t \to \infty$. Knowing whether we have simulated sufficiently to perform this integration reliably is equivalent to determining the slope in the mean-squared 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.

Case Study 6 (Algorithms to Calculate the Mean-Squared Displacement)
In this case study, a comparison is made between the conventional (Algorithm 8) and the order-n methods (Algorithm 9) to determine the mean-squared displacement. For this comparison we determine the mean-squared displacement of the Lennard-Jones fluid.

In Figure 4.7 the mean-squared displacement as a function of time as computed with the conventional method is compared with that obtained from the order-n scheme. The calculation using the conventional scheme could not be extended to times longer than $t > 10$ without increasing the number of

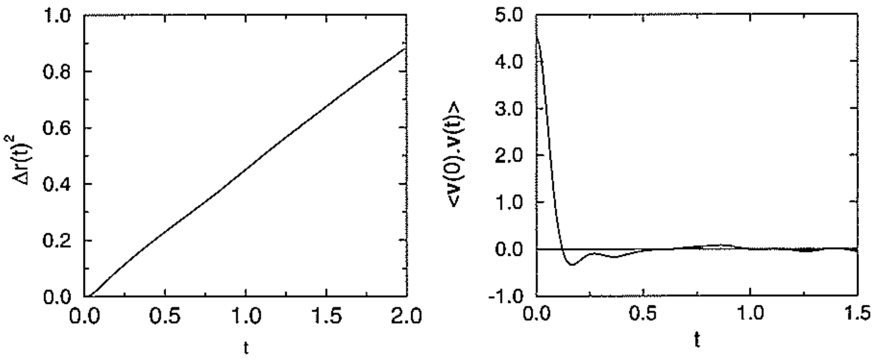

Figure 4.6: (left) Mean-squared displacement $\Delta r(t)^2$ as a function of the simulation time t. Note that for long times, $\Delta r(t)^2$ varies linearly with t. The slope is then given by 2dD, where d is the dimensionality of the system and D the self-diffusion coefficient. (right) Velocity autocorrelation function $\langle v(0) \cdot v(t) \rangle$ as a function of the simulation time t.

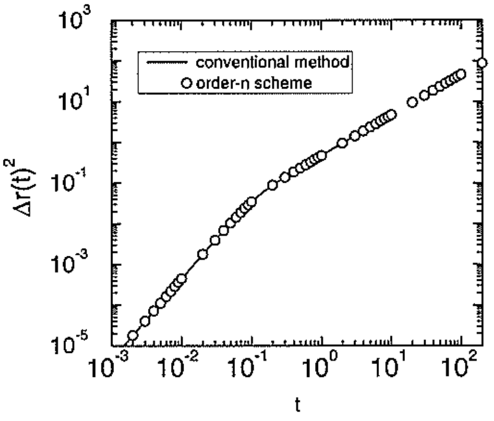

Figure 4.7: Mean-squared displacement as a function of time for the Lennard-Jones fluid ($\rho = 0.844$, $N = 108$, and $T = 1.50$); comparison of the conventional method with the order-n scheme .

time steps between two samples because of lack of memory. With the order-n scheme the calculation could be extended to much longer times with no difficulty. It is interesting to compare the accuracy of the two schemes. In the conventional scheme, the velocities of the particles at the current time step are used to update the mean-squared displacement of all time intervals. In

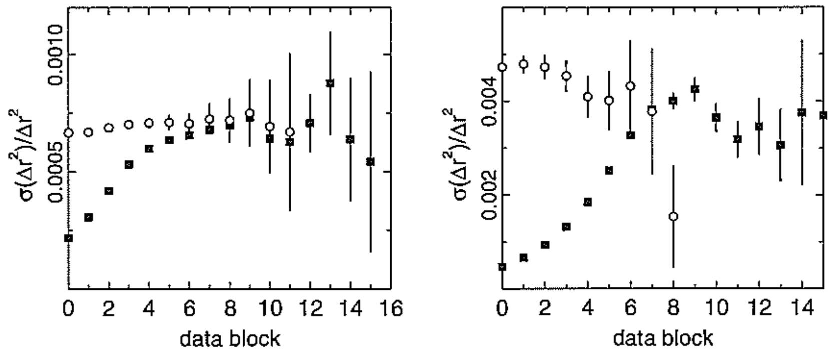

Figure 4.8: Relative error in the mean-squared displacement as a function of the number of data blocks as defined by Flyvbjerg and Petersen. The figures compare the conventional scheme (solid squares) with the order-n method (open circles) to determine the mean-squared displacement. The right figure is for $t = 0.1$ and the left figure for $t = 1.0$.

the order-n scheme the current time step is only used to update the lowest-order array of v_{sum} (see Algorithm 9). The block sums of level i are updated only once every n^i time step. Therefore, for a total simulation of M time steps, the number of samples is much less for the order-n scheme; for the conventional scheme, we have M samples for all time steps, whereas the order-n scheme has M/n^i samples for the ith block velocity. Naively, one would think that the conventional scheme therefore is more accurate. In the conventional scheme, however, the successive samples will have much more correlation and therefore are not independent. To investigate the effect of these correlations on the accuracy of the results, we have used the method of Flyvbjerg and Petersen [84] (see Appendix D.3 and Case Study 4). In this method, the standard deviation is calculated as a function of the number of data blocks. If the data are correlated, the standard deviation will increase as a function of the number of blocks until the number of blocks is sufficient that the data in a data block are uncorrelated. If the data are uncorrelated, the standard deviation will be independent of the number of blocks. This limiting value is the standard deviation of interest.

In these simulations the time step was $\Delta t = 0.001$ and the block length was set to $n = 10$. For both methods the total number of time steps was equal. To calculate the mean-squared displacement, we have used 100,000 samples for all times in the conventional scheme. For the order-n scheme, we have used 100,000 samples for $t \in [0, 0.01]$, 10,000 for $t \in [0.01, 0.1]$, 1,000 for for $t \in [0.1, 1]$, etc. This illustrates that the number of samples in the order-n scheme is considerably less than in the conventional scheme. The

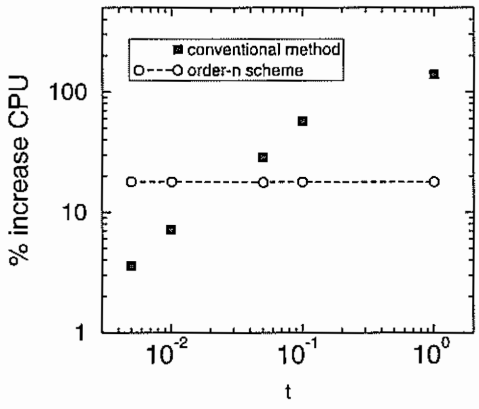

Figure 4.9: Percentage increase of the total CPU time as a function of the total time for which we determine the mean-squared displacement; comparison of the conventional scheme with the order-n scheme for the same system as is considered in Figure 4.7

accuracy of the results, however, turned out to be the same. This is shown in Figure 4.8 for $t = 0.1$ and $t = 1.0$. Since the total number of data blocking operations that can be performed on the data depends on the total number of samples, the number of blocking operations is less for the order-n method. Figure 4.8 shows that for $t = 0.1$ the order-n scheme yields a standard deviation that is effectively constant after three data blocking operations, indicating the samples are independent, whereas the standard deviation using the conventional method shows an increase for the first six to eight data blocking operations. For $t = 1.0$ the order-n method is independent of the number of data blocks, the conventional method only after 10 data blocks. This implies that one has to average over $2^{10} \approx 1000$ successive samples to have two independent data points. In addition, the figure shows that the plateau value of the standard deviation is essentially the same for the two methods, which implies that for this case the two methods are equally accurate.

In Figure 4.9 we compare the CPU requirements of the two algorithms for simulations with a fixed total number of time steps. This figure shows the increase of the total CPU time of the simulation as a function of the total time for which the mean-squared displacement has been calculated. With the order-n scheme the CPU time should be (nearly) independent of the total time for which we determine the mean-squared displacement, which is indeed what we observe. For the conventional scheme, however, the required CPU time increases significantly for longer times. At $t = 1.0$ the order-n scheme gives an increase of the total CPU time of 17%, whereas the conventional scheme shows an increase of 130%.