In Case Study 4, this method was used to compute the standard deviation of the energy in a Molecular Dynamics simulation. In Figure 4.4 a typical plot of this estimate of the variance as a function of block size is shown. For small values of M, the number of blocking operations, the data are correlated and as a consequence the variance will increase if we perform the blocking operation. For very high values of M we have only a few samples, and as a result, the statistical error in our estimate of $\sigma^2(A)$ will be large. The plateau in between gives us the value of $\sigma^2(A)$ we are interested in.

Appendix E

Integration Schemes

E.1 Higher-Order Schemes

The basic idea behind the predictor-corrector algorithms is to use information about the position and its first n derivatives at time t to arrive at a prediction for the position and its first n derivatives at time $t + \Delta t$. We then compute the forces (and thereby the accelerations) at the predicted positions. And then we find that these accelerations are *not* equal to the values that we had predicted. So we adjust our predictions for the accelerations to match the facts. But we do more than that. On the basis of the observed discrepancy between the predicted and observed accelerations, we also try to improve our estimate of the positions and the remaining $n-1$ derivatives. This is the "corrector" part of the predictor-corrector algorithm. The precise "recipe" used in applying this correction is a compromise between accuracy and stability. Here, we shall simply show a specific example of a predictor-corrector algorithm, without attempting to justify the form of the corrector part.

Consider the Taylor expansion of the coordinate of a given particle at time $t + \Delta t$:

$$r(t+\Delta t) = r(t) + \Delta t \frac{\partial r}{\partial t} + \frac{\Delta t^2}{2!}\frac{\partial^2 r}{\partial t^2} + \frac{\Delta t^3}{3!}\frac{\partial^3 r}{\partial t^3} + \cdots .$$

Using the notation

$$\begin{aligned} x_0(t) &\equiv r(t) \\ x_1(t) &\equiv \Delta t \frac{\partial r}{\partial t} \\ x_2(t) &\equiv \frac{\Delta t^2}{2!}\frac{\partial^2 r}{\partial t^2} \\ x_3(t) &\equiv \frac{\Delta t^3}{3!}\frac{\partial^3 r}{\partial t^3}, \end{aligned}$$

we can write the following *predictions* for $x_0(t+\Delta t)$ through $x_3(t+\Delta t)$:

$$\begin{aligned} x_0(t+\Delta t) &= x_0(t) + x_1(t) + x_2(t) + x_3(t) \\ x_1(t+\Delta t) &= x_1(t) + 2x_2(t) + 3x_3(t) \\ x_2(t+\Delta t) &= x_2(t) + 3x_3(t) \\ x_3(t+\Delta t) &= x_3(t). \end{aligned}$$

Now that we have $x_0(t+\Delta t)$, we can compute the forces at the predicted position, and thus compute the corrected value for $x_2(t+\Delta t)$. We denote the difference between $x_2^{\text{corrected}}$ and $x_2^{\text{predicted}}$ by Δx_2:

$$\Delta x_2 \equiv x_2^{\text{corrected}} - x_2^{\text{predicted}}.$$

We now estimate "corrected" values for x_0 through x_3, as follows:

$$x_n^{\text{corrected}} = x_n^{\text{predicted}} + C_n \Delta x_2, \qquad \text{(E.1.1)}$$

where the C_n are constants fixed for a given order algorithm. As indicated, the values for C_n are such that they yield an optimal compromise between the accuracy and the stability of the algorithm. For instance, for a fifth-order predictor-corrector algorithm (i.e., one that uses x_0 through x_4), the values for C_n are

$$\begin{aligned} C_0 &= \frac{19}{120} \\ C_1 &= \frac{3}{4} \\ C_2 &= 1 \qquad \text{(of course)} \\ C_3 &= \frac{1}{2} \\ C_4 &= \frac{1}{12}. \end{aligned}$$

One may iterate the predictor and corrector steps to self-consistency. However, there is little point in doing so because (1) every iteration requires a force calculation. One would be better off spending the same computer time to run with a *shorter* time step and only one iteration because (2) even if we iterate the predictor-corrector algorithm to convergence, we still do not get the *exact* trajectory: the error is still of order Δt^n for an nth-order algorithm. This is why we gain more accuracy by going to a shorter time step than by iterating to convergence at a fixed value of Δt.

E.2 Nosé-Hoover Algorithms

As discussed in section 6.1.2, it is advantageous to implement the Nosé thermostat using the formulation of Hoover, equations (6.1.24)–(6.1.27). Since the velocity also appears on the right-hand side of equation (6.1.25), this scheme cannot be implemented directly into the velocity Verlet algorithm (see also section 4.3). To see this, consider a standard constant-N,V,E simulation, for which the velocity Verlet algorithm is of the form

$$\begin{aligned} r(t+\Delta t) &= r(t) + v(t)\Delta t + f(t)\Delta t^2/(2m) \\ v(t+\Delta t) &= v(t) + \frac{f(t+\Delta t) + f(t)}{2m}\Delta t. \end{aligned}$$

When we use this scheme for the Nosé-Hoover equations of motion, we obtain for the positions and velocities

$$r_i(t+\Delta t) = r_i(t) + v(t)\Delta t + [f_i(t)/m_i - \xi(t)v_i(t)]\,\Delta t^2/2 \qquad \text{(E.2.1)}$$

$$\begin{aligned} v_i(t+\Delta t) = v_i(t) &+ [f_i(t+\Delta t)/m_i - \xi(t+\Delta t)v_i(t+\Delta t) \\ &+ f_i(t)/m_i - \xi(t)v_i(t)]\,\Delta t/2. \end{aligned} \qquad \text{(E.2.2)}$$

The first step of the velocity Verlet algorithm can be carried out without difficulty. In the second step, we first update the velocity, using the old "forces" to the intermediate value $v(t+\Delta t/2) \equiv v'$. And then we must use the new "forces" to update v':

$$v_i(t+\Delta t) = v'_i + [f_i(t+\Delta t)/m_i - \xi(t+\Delta t)v_i(t+\Delta t)]\,\Delta t/2. \qquad \text{(E.2.3)}$$

In these equations $v_i(t+\Delta t)$ appears on the right- and left-hand sides; therefore, these equations cannot be integrated exactly.[1] For this reason the Nosé-Hoover method is usually implemented using a predictor-corrector scheme or solved iteratively [138]. This has a disadvantage that the solution is no longer time reversible. Martyna *et al.* [85] have developed a set of explicit reversible integrators using the Liouville approach (see section 4.3.3) for this type of extended systems.

[1] For the harmonic oscillator it is possible to find an analytic solution (see Case Study 12).

E.2.1 Canonical Ensemble

For M chains, the Nosé-Hoover equations of motion are given by (see also section 6.1.3)

$$\begin{aligned}
\dot{r}_i &= p_i/m_i \\
\dot{p}_i &= F_i - \frac{p_{\xi_1}}{Q_1} p_i \\
\dot{\xi}_k &= \frac{p_{\xi_k}}{Q_k} \qquad\qquad k = 1, \ldots, M \\
\dot{p}_{\xi_1} &= \left(\sum_i p_i^2/m_i - L k_B T \right) - \frac{p_{\xi_2}}{Q_2} p_{\xi_1} \\
\dot{p}_{\xi_k} &= \left[\frac{p_{\xi_{k-1}}^2}{Q_{k-1}} - k_B T \right] - \frac{p_{\xi_{k+1}}}{Q_{k+1}} p_{\xi_k} \\
\dot{p}_{\xi_M} &= \left[\frac{p_{\xi_{M-1}}^2}{Q_{M-1}} - k_B T \right].
\end{aligned}$$

The Liouville operator for the equation of motions is defined as (see section 4.3.3)

$$iL \equiv \dot{\eta} \frac{\partial}{\partial \eta}$$

with $\eta = (\mathbf{r}^N, \mathbf{p}^N, \xi^M, p_\xi^M)$. Using the equations of motion, $\mathbf{p}_i = m_i \mathbf{v}_i$, and $p_{\xi_k} = Q_k \nu_{\xi_k}$, we obtain as Liouville operator for the Nosé-Hoover chains

$$\begin{aligned}
iL_{NHC} = &\sum_{i=1}^{N} \mathbf{v}_i \cdot \nabla_{\mathbf{r}_i} + \sum_{i=1}^{N} \left[\frac{\mathbf{F}_i(\mathbf{r}_i)}{m_i} \right] \cdot \nabla_{\mathbf{v}_i} - \sum_{i=1}^{N} \nu_{\xi_1} \mathbf{v}_i \cdot \nabla_{\mathbf{v}_i} + \sum_{k=1}^{M} \nu_{\xi_k} \frac{\partial}{\partial \xi_k} \\
&+ \sum_{k=1}^{M-1} (G_k - \nu_{\xi_k} \nu_{\xi_{k+1}}) \frac{\partial}{\partial \nu_{\xi_k}} + G_M \frac{\partial}{\partial \nu_{\xi_M}}
\end{aligned}$$

with

$$\begin{aligned}
G_1 &= \frac{1}{Q_1} \left(\sum_{i=1}^{N} m_i v_i^2 - L k_B T \right) \\
G_k &= \frac{1}{Q_k} \left(Q_{k-1} \nu_{\xi_{k-1}}^2 - k_B T \right).
\end{aligned}$$

As explained in section 4.3.3, the Liouville equation combined with the Trotter formula is a powerful technique for deriving a time-reversible algorithm for solving the equations of motion numerically. Here we will use this technique to derive such a scheme for the Nosé-Hoover thermostats. We use a simplified version; a more complete description can be found in ref. [85].

We have to make an intelligent separation of the Liouville operator. The first step is to separate the part of the Liouville operator that only involves the positions (iL_r) and the velocities (iL_v) from the parts that involve the Nosé-Hoover thermostats (iL_C):

$$iL_{NHC} = iL_r + iL_v + iL_C$$

with

$$\begin{aligned}
iL_r &= \sum_{i=1}^{N} \mathbf{v}_i \cdot \nabla_{\mathbf{r}_i} \\
iL_v &= \sum_{i=1}^{N} \frac{\mathbf{F}_i(\mathbf{r}_i)}{m_i} \cdot \nabla_{\mathbf{v}_i} \\
iL_C &= \sum_{k=1}^{M} \nu_{\xi_k} \frac{\partial}{\partial \xi_k} - \sum_{i=1}^{N} \nu_{\xi_1} \mathbf{v}_i \cdot \nabla_{\mathbf{v}_i} \\
&\quad + \sum_{k=1}^{M-1} (G_k - \nu_{\xi_k} \nu_{\xi_{k+1}}) \frac{\partial}{\partial \nu_{\xi_k}} + G_M \frac{\partial}{\partial \nu_{\xi_M}}.
\end{aligned}$$

There are several ways to factorize iL_{NHC} using the Trotter formula; we follow the one used by Martyna *et al.* [85]:

$$e^{(iL\Delta t)} = e^{(iL_C \Delta t/2)} e^{(iL_v \Delta t/2)} e^{(iL_r \Delta t)} e^{(iL_v \Delta t/2)} e^{(iL_C \Delta t/2)} + \mathcal{O}\left(\Delta t^3\right). \quad \text{(E.2.4)}$$

The Nosé-Hoover chain part L_C has to be further factorized. Here, we will do this for a chain of length $M = 2$; the more general case is discussed in ref. [85]. The Nosé-Hoover part of the Liouville operator for this chain length can be separated into five terms:

$$iL_C = iL_\xi + iL_{C_v} + iL_{G1} + iL_{v\xi_1} + iL_{G2},$$

where the terms are defined as

$$\begin{aligned}
iL_\xi &\equiv \sum_{k=1}^{2} \nu_{\xi_k} \frac{\partial}{\partial \xi_k} \\
iL_{C_v} &\equiv -\sum_{i=1}^{N} \nu_{\xi_1} \mathbf{v}_i \cdot \nabla_{\mathbf{v}_i} \\
iL_{G1} &\equiv G_1 \frac{\partial}{\partial \nu_{\xi_1}} \\
iL_{v\xi_1} &\equiv -(\nu_{\xi_1} \nu_{\xi_2}) \frac{\partial}{\partial \nu_{\xi_1}} \\
iL_{G2} &\equiv G_2 \frac{\partial}{\partial \nu_{\xi_2}}.
\end{aligned}$$

The factorization for the Trotter equation that we use is[2]

$$\begin{aligned} e^{(iL_C\Delta t/2)} &= e^{(iL_{G2}\Delta t/4)}e^{(iL_{v\xi_1}\Delta t/4+iL_{G1}\Delta t/4)} \\ &\quad \times e^{(iL_\xi\Delta t/2)}e^{(iL_{Cv}\Delta t/2)}e^{(iL_{G1}\Delta t/4+iL_{v\xi_1}\Delta t/4)}e^{(iL_{G2}\Delta t/4)} \\ &= e^{(iL_{G_2}\Delta t/4)}\left[e^{(iL_{v\xi_1}\Delta t/8)}e^{(iL_{G1}\Delta t/4)}e^{(iL_{v\xi_1}\Delta t/8)}\right] \\ &\quad \times e^{(iL_\xi\Delta t/2)}e^{(iL_{Cv}\Delta t/2)} \\ &\quad \times \left[e^{(iL_{v\xi_1}\Delta t/8)}e^{(iL_{G1}\Delta t/4)}e^{(iL_{v\xi_1}\Delta t/8)}\right]e^{(iL_{G2}\Delta t/4)}. \end{aligned} \tag{E.2.5}$$

Our numerical algorithm is now fully defined by equations (E.2.4) and (E.2.5). This seemingly complicated set of equations is actually relatively easy to implement in a simulation.

To see how the implementation works, we need to know how each operator works on our coordinates $\eta = (\mathbf{r}^N, \mathbf{v}^N, \xi_1, v_{\xi_1}, \xi_2, v_{\xi_2})$. If we start at $t = 0$ with initial condition η, the position at time $t = \Delta t$ follows from

$$e^{iL_{NHC}\Delta t}f[\mathbf{r}^N, \mathbf{v}^N, \xi_1, v_{\xi_1}, \xi_2, v_{\xi_2}].$$

Because of the Trotter expansion, we can apply each term in iL_{NHC}, sequentially. For example, if we let the first term of the Liouville operator, iL_{G2}, act on the initial state η,

$$\begin{aligned} &\exp\left(\frac{\Delta t}{4}G_2\frac{\partial}{\partial v_{\xi_2}}\right)f\left[\mathbf{r}^N, \mathbf{p}^N, \xi_1, v_{\xi_1}, \xi_2, v_{\xi_2}\right] \\ &= \sum_{n=0}^{\infty}\frac{(G_2\Delta t/4)^n}{n!}\frac{\partial^n}{\partial v_{\xi_2}^n}f\left[\mathbf{r}^N, \mathbf{p}^N, \xi_1, v_{\xi_1}, \xi_2, v_{\xi_2}\right] \\ &= f\left[\mathbf{r}^N, \mathbf{p}^N, \xi_1, v_{\xi_1}, \xi_2, v_{\xi_2} + G_2\Delta t/4\right]. \end{aligned}$$

This shows that the effect of iL_{G2} is to shift v_{ξ_2} without affecting the other coordinates. This gives as transformation rule for this operator:

$$e^{(iL_{G2}\Delta t/4)}: \qquad v_{\xi_2} \to v_{\xi_2} + G_2\Delta t/4. \tag{E.2.6}$$

The operators ($iL_{v\xi_1}$ and iL_{Cv}) are of the form $\exp(ax\partial/\partial x)$; such operators give a scaling of the x coordinate:

$$\begin{aligned} \exp\left(ax\frac{\partial}{\partial x}\right)f(x) &= \exp\left(a\frac{\partial}{\partial \ln(x)}\right)f\{\exp[\ln(x)]\} \\ &= f\{\exp[\ln(x) + a]\} = f[x\exp(a)] \end{aligned}$$

[2]The second factorization, indicated by $[\cdots]$, is used to avoid a hyperbolic sine function, which has a possible singularity. See ref. [85] for details.

If we apply this result[3] to $iL_{v\xi_1}$, we obtain for this operator

$$\begin{aligned} &\exp\left(-\frac{\Delta t}{8}v_{\xi_2}v_{\xi_1}\frac{\partial}{\partial v_{\xi_1}}\right)f\left[\mathbf{r}^N, \mathbf{p}^N, \xi_1, v_{\xi_1}, \xi_2, v_{\xi_2}\right] \\ &= f\left[\mathbf{r}^N, \mathbf{p}^N, \xi_1, \exp\left(-\frac{\Delta t}{8}v_{\xi_2}\right)v_{\xi_1}, \xi_2, v_{\xi_2}\right], \end{aligned}$$

giving the transformation rule

$$e^{(iL_{v\xi_1}\Delta t/8)}: \qquad v_{\xi_1} \to \exp\left[-v_{\xi_2}\Delta t/8\right]v_{\xi_1}. \tag{E.2.7}$$

In a similar way we can derive for the other terms

$$e^{(iL_{G1}\Delta t/4)}: \qquad v_{\xi_1} \to v_{\xi_1} + G_1\Delta t/4 \tag{E.2.8}$$

$$e^{(iL_\xi\Delta t/2)}: \qquad \xi_1 \to \xi_1 - v_{\xi_1}\Delta t/2 \tag{E.2.9}$$

$$\xi_2 \to \xi_2 - v_{\xi_2}\Delta t/2 \tag{E.2.10}$$

$$e^{(iL_{Cv}\Delta t/2)}: \qquad v_i \to \exp\left[-v_{\xi_1}\Delta t/2\right]v_i. \tag{E.2.11}$$

Finally, the transformation rules that are associated to iL_v and iL_r are similar to the velocity Verlet algorithm, i.e.,

$$e^{(iL_v\Delta t/2)}: \qquad \mathbf{v}_i \to \mathbf{v}_i + \mathbf{F}_i\Delta t/(2m) \tag{E.2.12}$$

$$e^{(iL_r\Delta t)}: \qquad \mathbf{r}_i \to \mathbf{r}_i + \mathbf{v}_i\Delta t. \tag{E.2.13}$$

With these transformation rules (E.2.6)–(E.2.13) we can write down our numerical algorithm by subsequently applying the transformation rules according to the order defined by equations (E.2.4) and (E.2.5). If we start with initial coordinate $\eta(0) = (\mathbf{r}^N, \mathbf{v}^N, \xi_1, v_{\xi_1}, \xi_2, v_{\xi_2})$, we have to apply first $e^{iL_C}\eta$. Since this operator is further factorized according to equation (E.2.5), the first step in our algorithm is to apply $e^{(iL_{G2}\Delta t/4)}$. According to transformation rule (E.2.6) applying this operator on η gives as new state

$$v_{\xi_2}(\Delta t/4) = v_{\xi_2} + G_2\Delta t/4.$$

The output of this rule is the new state on which we apply the next operator in equation (E.2.5), $iL_{v\xi_1}$, with transformation rule (E.2.9):

$$v_{\xi_1}(\Delta t/8) = \exp\left[-v_{\xi_2}(\Delta t/4)\Delta t/8\right]v_{\xi_1}.$$

[3]This can be generalized, giving the identity

$$\begin{aligned} \exp\left(a\frac{\partial}{\partial g(x)}\right)f(x) &= \exp\left(a\frac{\partial}{\partial g(x)}\right)f\left\{g^{-1}[g(x)]\right\} \\ &= \exp\left(a\frac{\partial}{\partial y}\right)f\left[g^{-1}(y)\right] \\ &= f\left\{g^{-1}[y + a]\right\} = f\left\{g^{-1}[g(x) + a]\right\}. \end{aligned}$$

Algorithm 30 (Equations of Motion: Nosé-Hoover)

```
subroutine integrate     integrate equations of motion
                         Nosé-Hoover thermostat
call chain(uk)
call pos_vel(uk)
call chain(uk)
return
end
```

- *Comments to this algorithm:*

1. *This subroutine solves the equations of motion for a single time step* Δt *using the Trotter equations (E.2.4) and (E.2.5).*
2. *In the subroutine* `chain` *we apply* $e^{iL_C \Delta t/4}$ *to the current state (see Algorithm 31).*
3. *In the subroutine* `pos_vel` *we apply* $e^{(iL_r+iL_p)\Delta t}$ *to the current state (see Algorithm 32).*
4. `uk` *is the total kinetic energy.*

The next step is to apply iL_{G1}, followed by again $iL_{v_{\xi 1}}$, etc. In this way we continue to apply all operators on the output of the previous step.

Applying the Nosé-Hoover part of the Liouville operator changes ξ_k, v_{ξ_k}, and $\mathbf{v}_i$. The other two Liouville operators change $\mathbf{v}_i$ and $\mathbf{r}_i$. This makes it convenient to separate the algorithm into two parts in which the positions and velocities of the particles and the Nosé-Hoover chains are considered separately. An example of a possible implementation is shown in Algorithm 30.

E.2.2 The Isothermal-Isobaric Ensemble

Similar to the canonical ensemble we can derive a time-reversible integration scheme for simulation in the NPT ensemble. The equations of motions are given by expressions (6.2.1)–(6.2.8):

$$\begin{aligned}
\dot{\mathbf{r}}_i &= \frac{\mathbf{p}_i}{m_i} + \frac{p_\epsilon}{W}\mathbf{r}_i \\
\dot{\mathbf{p}}_i &= \mathbf{F}_i - \left(1 + \frac{d}{dN}\right)\frac{p_\epsilon}{W}\mathbf{p}_i - \frac{p_{\xi_1}}{Q_1}\mathbf{p}_i \\
\dot{V} &= dV p_\epsilon / W
\end{aligned}$$

Algorithm 31 (Propagating the chain)

```
subroutine chain(uk)              apply equation (E.2.5)
                                  to the current position

G2=(Q1*vxi1*vxi1-T)
vxi2=vxi2+G2*delt4                Update v_ξ2 using equation (E.2.6)
vxi1=vxi1*exp(-vxi2*delt8)        Update v_ξ1 using equation (E.2.7)
G1=(2*uk-L*T)/Q1
vxi1=vxi1+G1*delt4                Update v_ξ1 using equation (E.2.8)
vxi1=vxi1*exp(-vxi2*delt8)        Update v_ξ1 using equation (E.2.7)
xi1=xi1+vxi1*delt2                Update ξ1 using equation (E.2.9)
xi2=xi2+vxi2*delt2                Update ξ2 using equation (E.2.10)
s=exp(-vxi1*delt2)                Scale factor in equation (E.2.11)
do i=1,npart
  v(i)=s*v(i)                     update v_i using equation (E.2.11)
enddo
uk=uk*s*s                         update kinetic energy
vxi1=vxi1*exp(-vxi2*delt8)        Update v_ξ1 using equation (E.2.7)
G1=(2*uk-L*T)/Q1
vxi1=vxi1+G1*delt4                Update v_ξ1 using equation (E.2.8)
vxi1=vxi1*exp(-vxi2*delt8)        Update v_ξ1 using equation (E.2.7)
G2=(Q1*vxi1*vxi1-T)/Q2
vxi2=vxi2+G2*delt4                Update v_ξ2 using equation (E.2.6)
return
end
```

Comments to this algorithm:

1. *In this subroutine* `T` *is the imposed temperature,* `delt`$= \Delta t$, `delt2` $= \Delta t/2$, `delt4` $= \Delta t/4$, *and* `delt8` $= \Delta t/8$.
2. `uk` *is the total kinetic energy.*

Algorithm 32 (Propagating the Positions and Velocities)

```
subroutine pos_vel(uk)        apply equation (E.2.4)
                              to the current position
uk=0
do i=1,npart
  x(i)=x(i)+v(i)*delt2        update x_i using equation (E.2.13)
enddo
call force                    calculate the force
do i=1,npart
  v(i)=v(i)+f(i)*delt/m       update v_i using equation (E.2.12)
  x(i)=x(i)+v(i)*delt2        update x_i using equation (E.2.13)
  uk=uk+m*v(i)*v(i)/2         update kinetic energy
enddo
return
end
```

Comments to this algorithm:

1. *In this subroutine* `delt` $= \Delta t$ *and* `delt2` $= \Delta t/2$.
2. *The subroutine* `force` *calculates the force on the particles.*

$$
\begin{aligned}
\dot{p}_\epsilon &= dV(P_{int} - P_{ext}) + \frac{1}{N}\sum_{i=1}^{N}\frac{p_i^2}{m_i} - \frac{p_{\xi_1}}{Q_1}p_\epsilon \\
\dot{\xi}_k &= \frac{p_{\xi_k}}{Q_k} \qquad \text{for } k = 1,\ldots,M \\
\dot{p}_{\xi_1} &= \sum_{i=1}^{N}\frac{p_i^2}{m_i} + \frac{p_\epsilon^2}{W} - (dN+1)k_BT - \frac{p_{\xi_2}}{Q_2}p_{\xi_1} \\
\dot{p}_{\xi_k} &= \frac{p_{\xi_{k-1}}^2}{Q_{k-1}} - k_BT - \frac{p_{\xi_{k+1}}}{Q_{k+1}}p_{\xi_k} \qquad \text{for } k = 2,\ldots,M-1 \\
\dot{p}_{\xi_M} &= \frac{p_{\xi_{M-1}}^2}{Q_{M-1}} - k_BT.
\end{aligned}
$$

To derive a time-reversible numerical integration scheme to solve the equations of motion we use again the Liouville approach.

A state is characterized by the variables $\eta = (r^N, p^N, \epsilon, p_\epsilon, \xi^M, p_\xi^M)$. The Liouville operator is defined by

$$iL_{NPT} \equiv \dot{\eta}\frac{\partial}{\partial\eta}.$$

For a chain of length $M = 2$, using $p_i = m_i v_i (\neq m_i \dot{r}_i)$, $p_{\xi_k} = Q_k v_{\xi_k}$, $\epsilon = (\ln V)/d$, and $p_\eta = W v_\eta$, the Liouville operator for these equations of motion can be written as

$$iL_{NPT} = iL_r + iL_v + iL_{CP},$$

in which we define the operators

$$
\begin{aligned}
iL_r &= \sum_{i=1}^{N}(v_i + v_\epsilon r_i)\cdot\nabla_{r_i} + v_\epsilon\frac{\partial}{\partial\epsilon} \\
iL_v &= \sum_{i=1}^{N}\left[\frac{F_i(r)}{m_i}\right]\cdot\nabla_{v_i} \\
iL_{CP} &= -\sum_{i=1}^{N}v_{\xi_1}v_i\cdot\nabla_{v_i} + \sum_{k=1}^{M}v_{\xi_k}\frac{\partial}{\partial\xi_k} + \sum_{k=1}^{M-1}(G_k - v_{\xi_k}v_{\xi_{k+1}})\frac{\partial}{\partial v_{\xi_k}} \\
&\quad + G_M\frac{\partial}{\partial v_{\xi_M}} - \left(1+\frac{1}{N}\right)\sum_{i=1}^{N}v_\epsilon v_i\cdot\nabla_{v_i} + (G_\epsilon - v_\epsilon v_{\xi_1})\frac{\partial}{\partial v_\epsilon}
\end{aligned}
$$

with

$$
\begin{aligned}
G_1 &= \frac{1}{Q}\left[\sum_{i=1}^{N}m_i v_i^2 + W v_\epsilon^2 - (N_f+1)k_BT\right] \\
G_k &= \frac{1}{Q_k}\left(Q_{k-1}v_{\xi_{k-1}}^2 - k_BT\right) \\
G_\epsilon &= \frac{1}{W}\left[\left(1+\frac{1}{N}\right)\sum_{i=1}^{N}m_i v_i^2 + \sum_{i=1}^{N}r_i\cdot F_i(r)m_i - dV\frac{\partial U(r,V)}{\partial V} - dP_{ext}V\right].
\end{aligned}
$$

An appropriate Trotter equation for the equations of motion is [85]

$$e^{(iL_{NPT}\Delta t)} = e^{(iL_{CP}\Delta t/2)}e^{(iL_v\Delta t/2)}e^{(iL_r\Delta t)}e^{(iL_v\Delta t/2)}e^{(iL_{CP}\Delta t/2)} + \mathcal{O}\left(\Delta t^3\right). \tag{E.2.14}$$

The operator iL_{CP} has to be further factorized:

$$iL_{CP} = iL_\xi + iL_{C_v} + iL_{G\epsilon} + iL_{v\epsilon} + iL_{G1} + iL_{v\xi_1} + iL_{G2},$$

where the terms are defined as

$$
\begin{aligned}
iL_\xi &\equiv \sum_{k=1}^{2} v_{\xi_k} \frac{\partial}{\partial \xi_k} \\
iL_{C_v} &\equiv -\sum_{i=1}^{N} \left[v_{\xi_1} + \left(1 + \frac{d}{dN}\right) \right] \mathbf{v}_i \cdot \nabla_{\mathbf{v}_i} \\
iL_{G\epsilon} &\equiv G_\epsilon \frac{\partial}{\partial v_\epsilon} \\
iL_{v\epsilon} &\equiv -(v_{\xi_1} v_\epsilon) \frac{\partial}{\partial v_\epsilon} \\
iL_{G1} &\equiv G_1 \frac{\partial}{\partial v_{\xi_1}} \\
iL_{v\xi_1} &\equiv -(v_{\xi_1} v_{\xi_2}) \frac{\partial}{\partial v_{\xi_1}} \\
iL_{G2} &\equiv G_2 \frac{\partial}{\partial v_{\xi_2}}.
\end{aligned}
$$

The Trotter expansion of the term iL_C is

$$
\begin{aligned}
e^{(iL_C \Delta t/2)} &= e^{(iL_{G2}\Delta t/4 + iL_{v\xi_1}\Delta t/4)} e^{(iL_{G1}\Delta t/4)} e^{(iL_{G\epsilon}\Delta t/4 + iL_{v\epsilon}\Delta t/4)} \\
&\quad \times e^{(iL_\xi \Delta t/2)} e^{(iL_{C_v}\Delta t/2)} \\
&\quad \times e^{(iL_{v\epsilon}\Delta t/4 + iL_{G\epsilon}\Delta t/4)} e^{(iL_{G1}\Delta t/4 + iL_{v\xi_1}\Delta t/4)} e^{(iL_{G2}\Delta t/4)} \\
&= e^{(iL_{G_2}\Delta t/4)} \left[e^{(iL_{v\xi_1}\Delta t/8)} e^{(iL_{G1}\Delta t/4)} e^{(iL_{v\xi_1}\Delta t/8)} \right] \\
&\quad \times \left[e^{(iL_{v\epsilon}\Delta t/8)} e^{(iL_{G\epsilon}\Delta t/4)} e^{(iL_{v\epsilon}\Delta t/8)} \right] \\
&\quad \times e^{(iL_\xi \Delta t/2)} e^{(iL_{C_v}\Delta t/2)} \left[e^{(iL_{v\epsilon}\Delta t/8)} e^{(iL_{G\epsilon}\Delta t/4)} e^{(iL_{v\epsilon}\Delta t/8)} \right] \\
&\quad \times \left[e^{(iL_{v\xi_1}\Delta t/8)} e^{(iL_{G1}\Delta t/4)} e^{(iL_{v\xi_1}\Delta t/8)} \right] e^{(iL_{G2}\Delta t/4)}. \quad \text{(E.2.15)}
\end{aligned}
$$

Similar to the NVT version the transformation rules of the various operators can be derived and translated into an algorithm. Such an algorithm is presented in ref. [85].

Appendix F

Saving CPU Time

The energy or force calculation is the most time-consuming part of almost all Molecular Dynamics and Monte Carlo simulations. If we consider a model system with pairwise additive interactions (as is done in many molecular simulations), we have to consider the contribution to the force on particle i, by all its neighbors. If we do not truncate the interactions, this implies that, for a system of N particles, we must evaluate $N(N-1)/2$ pair interactions. And even if we do truncate the potential, we still would have to compute all $N(N-1)/2$ pair distances to describe which pairs can interact. This implies that, if we use no tricks, the time needed for the evaluation of the energy scales as N^2. There exist efficient techniques for speeding up the evaluation of both short-range and long-range interactions in such a way that the computing time scales as $N^{3/2}$, rather than N^2. The techniques for the long-range interactions were discussed in Chapter 12.1; here, we discuss some of the techniques used for the short-range interactions. These techniques are:

1. Verlet list
2. Cell (or linked) list
3. Combination of Verlet and cell lists

F.1 Verlet List

If we simulate a large system and use a cutoff that is smaller than the simulation box, many particles do not contribute to the energy of a particle i. It is advantageous therefore to exclude the particles that do not interact from the expensive energy calculation. Verlet [13] developed a bookkeeping technique, commonly referred to as the Verlet list or neighbor list, which is illustrated in Figure F.1. In this method a second cutoff radius $r_v > r_c$ is in-

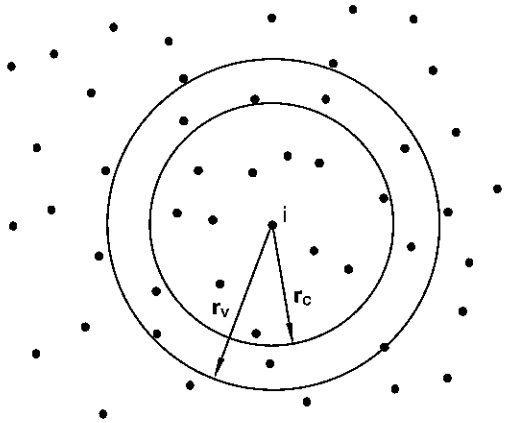

Figure F.1: The Verlet list: a particle i interacts with those particles within the cutoff radius r_c; the Verlet list contains all the particles within a sphere with radius $r_v > r_c$.

troduced, and before we calculate the interactions, a list is made (the Verlet list) of all particles within a radius r_v of particle i. In the subsequent calculation of the interactions, only those particles in this list have to be considered. Until now we have not saved any CPU time. We gain such time when we next calculate the interactions; if the maximum displacement of the particles is less than $r_v - r_c$, then we have to consider only the particles in the Verlet list of particle i. This is a calculation of order N. As soon as one of the particles is displaced more than $r_v - r_c$, we have to update the Verlet list. The latter operation is of order N^2, and although this step is not performed each time an interaction is calculated, it will dominate for a very large number of particles.

The Verlet list can be used for both Molecular Dynamics and Monte Carlo simulations. However, there are some small differences in the implementation. For example, in a Molecular Dynamics simulation, the force on all particles is calculated at the same time. It is sufficient therefore to have a Verlet list with half the number of particles for each particle as long as the interaction i-j is accounted for in either the list of particle i or that of j. In a Monte Carlo simulation each particle is considered separately, therefore it is convenient to have for each particle the complete list. Algorithm 33 shows the use of the Verlet list in a Monte Carlo simulation.

Bekker *et al.* have developed an elegant extension of the Verlet list for systems with periodic boundary conditions [530]. To calculate the force or potential energy of particle i one has to locate the nearest image of the particles in the Verlet list of particle j (see, Algorithm 34). Bekker *et al.* have

Algorithm 33 (Use of Verlet List in a Monte Carlo Move)

```
  SUBROUTINE mcmove_verlet              attempts to displace a particle
                                        using a Verlet list
  o=int(ranf()*npart)+1                 select a particle at random
  if (abs(x(o)-xv(o)).gt.(rv-rc)        check to make a new list
+  /2) call new_vlist
  call en_vlist(o,x(o),eno)             energy old configuration
  xn=x(o)+(ranf()-0.5)*delx             random displacement
  if (abs(xn-xv(o)).gt.(rv-rc)/2)       check to make a new list
+  call new_vlist
  call en_vlist(o,xn,enn)               energy new configuration
  arg=exp(-beta*(enn-eno))
  if (ranf().lt.arg)
+  x(o)=xn                              accepted: replace x(o) by xn
  return
  end
```

Comments to this algorithm:

1. *The algorithm is based on Algorithm 2.*
2. *Subroutine* `newvlist` *makes the Verlet list (see Algorithm 34) and subroutine* `en_vlist` *calculates the energy of a particle at the given position using the Verlet list (see Algorithm 35).*

shown that this nearest image calculation in the inner loop of a MD or MC simulation can be avoided.

In a periodic system, the total force on particle i can be written as

$$\mathbf{F}_i = \sum_{j=1}^{N} \sum_{k=-13}^{13}{}' \mathbf{F}_{i(j.k)},$$

where the prime denotes that the summation is performed over the nearest image of particle j in the central box ($k = 0$) or in one of its 26 periodic images. Here, (j.k) denotes the periodic image of particle j in box k. Box k is defined by the integer numbers n_x, n_y, n_z:

$$k = 9n_x + 3n_y + n_z$$

and

$$t_k = n_x L_x + n_y L_y + n_z L_z,$$

Algorithm 34 (Making a Verlet List)

```
SUBROUTINE new_vlist                    makes a new Verlet list
do i=1,npart                            initialize list
  nlist(i)=0
  xv(i)=x(i)                            store position of particles
enddo
do i=1,npart-1
  do j=i+1,npart
    xr=x(i)-x(j)
    if (xr.gt.hbox) then                nearest image
      xr=xr-box
    else if (xr.lt.-hbox) then
      xr=xr+box
    endif
    if (abs(xr).lt.rv) then             add to the lists
      nlist(i)=nlist(i)+1
      nlist(j)=nlist(j)+1
      list(i,nlist(i))=j
      list(j,nlist(j))=i
    endif
  enddo
enddo
return
end
```

Comments to this algorithm:

1. *Array* `list(i,itel)` *is the Verlet list of particle* `i`*, the total number of particles in the Verlet list of particle* `i` *is given by* `nlist(i)`*, and the array* `xv(i)` *contains the position of the particles at the moment the list is made (is used to see when a new list has to be made).*
2. *Note that in this algorithm we assume* all *particles are in the simulation box; hence* `x(i)` $\in$ *[0,box].*

where t_k is the translation vector of the central box to its periodic image k. A particle in the central box is denoted by $(i.0) = i$. Using this notation, we can write, for the interaction between particles i and j,

$$\mathbf{F}_{i(j.k)} = \mathbf{F}_{(i.-k)j} = -\mathbf{F}_{(j.k)i} = -\mathbf{F}_{j(i.-k)}.$$

Algorithm 35 (Calculating the Energy Using a Verlet List)

```
SUBROUTINE en_vlist(i,xi,en)            calculates energy using
                                        the Verlet list
en=0
do jj=1,nlist(i)                        loop over the particles in the list
  j=list(i,jj)                          next particle in the list
  en=en+enij(i,xi,j,x(j))
enddo
return
end
```

Comment to this algorithm:

1. *Array* `list(i,itel)` *and* `nlist` *are made in Algorithm 34 and* `enij` *gives the energy between particles* `i` *and* `j` *at the given positions.*

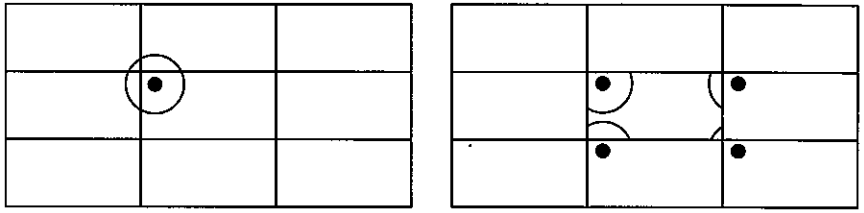

Figure F.2: Verlet lists: (left) conventional approach in which each particle has a Verlet list; (right) the approach of Bekker *et al.* in which each periodic image of a particle has its own Verlet list that contains only those particles in the central box.

We can write

$$\mathbf{F}_i = \sum_{j=1}^{N} \sum_{k=-13}^{13} {}'\mathbf{F}_{(i.k)j}.$$

The importance of this seemingly trivial result is that the summation is over all particles j in the *central* box with the nearest image of particle i. The difference between the two approaches is shown in Figure F.2.

This method is implemented using different Verlet lists for each periodic image of particle i. These lists contain only those particles that interact with particle i *and* are in the central box. If these lists are used, it is not necessary to

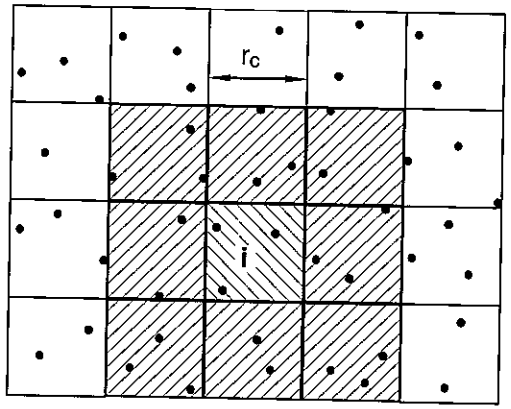

Figure F.3: The cell list: the simulation cell is divided into cells of size $r_c \times r_c$; a particle i interacts with those particles in the same cell or neighboring cells (in 2D there are 9 cells; and in 3D, 27 cells).

use the nearest image operation during the calculation of the force or energy. In [530] the use of these lists is shown to speed up an MD simulation by a factor 1.5. In addition, Bekker *et al.* have shown that a similar trick can be used to take the calculation of the virial (pressure) out of the inner loop.

F.2 Cell Lists

An algorithm that scales with N is the cell list or linked-list method [24]. The idea of the cell list is illustrated in Figure F.3. The simulation box is divided into cells with a size equal to or slightly larger than the cutoff radius r_c; each particle in a given cell interacts with only those particles in the same or neighboring cells. Since the allocation of a particle to a cell is an operation that scales with N and the total number of cells that needs to be considered for the calculation of the interaction is independent of the system size, the cell list method scales as N. Algorithm 36 shows how a cell list can be used in a Monte Carlo simulation.

F.3 Combining the Verlet and Cell Lists

It is instructive to compare the efficiency of the Verlet list and cell list in more detail. In the Verlet list the number of particles for which the distance needs to be calculated is in three dimensions, given by

$$n_v = \frac{4}{3}\pi\rho r_v^3;$$

Algorithm 36 (Use of Cell List in a Monte Carlo Move)

```
  SUBROUTINE mcmove_neigh        attempts to displace a particle
                                 using a cell list
  call newnlist(rc)              make the cell list
  o=int(ranf()*npart)+1          select a particle at random
  call en_nlist(o,x(o),eno)      calculate energy old configuration
  xn=x(o)+(ranf()-0.5)*delx      give particle random displacement
  call en_nlist(o,xn,enn)        calculate energy new configuration
  arg=exp(-beta*(enn-eno))
  if (ranf().lt.arg)
+   x(o)=xn                      accepted: replace x(o) by xn
  return
  end
```

Comments to this algorithm:

1. *This algorithm is based on Algorithm 2.*
2. *Subroutine* `new_nlist` *makes the cell list (see Algorithm 37) and subroutine* `en_nlist` *calculates the energy of a particle at the given position using the cell list (see Algorithm 38). Note that it is possible, at the expense of some extra bookkeeping, to update the list once a move is accepted instead of making a new list every move.*

for the cell list the corresponding number is

$$n_l = 27\rho r_c^3.$$

If we use typical values for the parameters in these equations (Lennard-Jones potential with $r_c = 2.5\sigma$ and $r_v = 2.7\sigma$), we find that n_l is five times larger than n_v. As a consequence, in the Verlet scheme, the number of pair distances that needs to be calculated is 16 times less than in the cell list.

The observation that the Verlet scheme is more efficient in evaluating the interactions motivated Auerbach *et al.* [531] to use a combination of the two lists: use a cell list to construct a Verlet list. The use of the cell list removes the main disadvantage of the Verlet list for a large number of particles—scales as N^2—but keeps the advantage of an efficient energy calculation. An implementation of this method in a Monte Carlo simulation is shown in Algorithm 39.

Algorithm 37 (Making a Cell List)

```
SUBROUTINE new_nlist(rc)        makes a new cell list with cell size r_c
                                using a linked-list algorithm
rn=box/int(box/rc)              determine size of cells r_n ≥ r_c
do icel=0,ncel-1
  hoc(icel)=0                   set head of chain to 0 for each cell
enddo
do i=1,npart                    loop over the particles
  icel=int(x(i)/rn)             determine cell number
  ll(i)=hoc(icel)               link list the head of chain of cell icel
  hoc(icel)=i                   make particle i the head of chain
enddo
return
end
```

Comment to this algorithm:

1. *This algorithm uses the linked-list method. To each cell a particle* `i` *is named head of chain and stored in the array* `hoc(icel)`*. To this particle the next particle in the cell (chain) is linked via the linked-list array* `ll(i)`*. If the value of the* `ll(i)` *is 0 no more particles are in the cell (chain). The desired (optimum) cell size is* `rc`*, and* `rn` *is the closest size that fits in the box.*

F.4 Efficiency

The first question that arises is when to use which method. This depends very strongly on the details of the systems. In any event, we always start with a scheme as simple as possible, hence no tricks at all. Although the algorithm scales as N^2, it is straightforward to implement and therefore the probability of programming errors is relatively small. In addition we should take into account how often the program will be used.

The use of the Verlet list becomes advantageous if the number of particles in the list is significantly less than the total number of particles; in three dimensions this means

$$n_v = \frac{4}{3}\pi r_v^3 \rho \ll N.$$

If we substitute some typical values for a Lennard-Jones potential ($r_v = 2.7\sigma$ and $\rho = 0.8\sigma^{-3}$), we find $n_v \approx 66$, which means that only if the number of particles in the box is more than 100 does it make sense to use a Verlet list.

To see when to use one of the other techniques, we have to analyze the algorithms in somewhat more detail. If we use no tricks, the amount of CPU

Algorithm 38 (Calculating the Energy Using a Cell List)

```
SUBROUTINE ennlist(i,xi,en)     calculates energy using
                                the cell list
en=0
icel=int(xi/rn)                 determine the cell number
do ncel=1,neigh                 loop over the neighbor cells
  jcel=neigh(icel,ncel)         number of the neighbor
  j=hoc(jcel)                   head of chain of cell jcel
  do while (j.ne.0)
    if (i.ne.j)
+     en=en+enij(i,xi,j,x(j))
    j=ll(j)                     next particle in the list
  enddo
enddo
return
end
```

Comment to this algorithm:

1. *Array* `ll(i)` *and* `hoc(icel)` *are constructed in Algorithm 37;* `enij` *is a function that gives the energy between particles* `i` *and* `j` *at the given positions.* `neigh(icel,ncel)` *gives the location of the* `ncel`*th neighbor of cell* `icel`*.*

time to calculate the total energy is given by

$$\tau = cN(N-1)/2.$$

The constant gives the required CPU time for an energy calculation between a pair of particles. If we use the Verlet list, the CPU time is

$$\tau_v = cn_vN + \frac{c_v}{n_u}N^2,$$

where the first term arises from the calculation of the interactions and the second term from the update of the Verlet list, which is done every n_u^{th} cycle.

The cell list scales with N and the CPU time can be split into two contributions: one that accounts for the calculation of the energy and the other for the making of the list,

$$\tau_l = cn_lN + c_lN.$$

If we use a combination of the two lists, the total CPU time becomes

$$\tau_c = cn_vN + \frac{c_l}{n_u}N.$$

Algorithm 39 (Combination of Verlet and Cell Lists)

```
 SUBROUTINE mcmove_clist                displace a particle
                                        using a combined list
 o=int(ranf()*npart)+1                  select a particle at random
 if (abs(x(o)-xv(o)).gt.rv-rc)          check to make a new list
+  call new_clist
 call en_vlist(o,x(o),eno)              energy old configuration
 xn=x(o)+(ranf()-0.5)*delx              random displacement
 if (abs(xn-xv(o)).gt.rv-rc)            check to make a new list
+  call new_clist
 call en_vlist(o,xn,enn)                energy new configuration
 arg=exp(-beta*(enn-eno))
 if (ranf().lt.arg)
+  x(o)=xn                              accepted: replace x(o) by xn
 return
 end
```

Comments to this algorithm:

1. *The algorithm is based on Algorithm 33.*
2. *Subroutine* `newclist` *makes the Verlet list using a cell list (see Algorithm 40) and subroutine* `en_vlist` *calculates the energy of a particle at the given position using the Verlet list (see Algorithm 35).*

The way to proceed is to perform some test simulations to estimate the various constants, and from the equations, it will become clear which technique is preferred. In Case Study 26, we have made such an estimate for a simulation of the Lennard-Jones fluid.

Case Study 26 (Comparison of Schemes for the Lennard-Jones Fluid)
It is instructive to make a detailed comparison of the various schemes to save CPU time for the Lennard-Jones fluid. We compare the following schemes:

1. Verlet list
2. Cell list
3. Combination of Verlet and cell lists
4. Simple N^2 algorithm

We have used the program of Case Study 1 as a starting point. At this point it is important to note that we have not tried to optimize the parameters (such

Algorithm 40 (Making a Verlet List Using a Cell List)

```
SUBROUTINE new_clist                    makes a new Verlet list
                                        using a cell list
call new_nlist(rv)                      make the cell lists
do i=1,npart                            initialize list
  nlist(i)=0
  xv(i)=x(i)                            store position of particles
enddo
do i=1,npart
 icel=int(x(i)/rn)                      determine cell number
 do ncel=1,neigh                        loop over the neighbor cells
   jcel=neigh(icel,ncel)                number of the neighbor
   j=hoc(jcel)                          head of chain of cell jcel
   do while (j.ne.0)
     if (i.ne.j) then
      xr=x(i)-x(j)
      if (xr.gt.hbox) then              nearest image
       xr=xr-box
      else if(xr.lt.-hbox)then
       xr=xr+box
      endif
      if (abs(xr).lt.rv) then           add to the Verlet lists
       nlist(i)=nlist(i)+1
       nlist(j)=nlist(j)+1
       list(i,nlist(i))=j
       list(j,nlist(j))=i
      endif
     endif
    j=ll(j)                             next particle in the cell list
   enddo
 enddo
enddo
return
end
```

Comments to this algorithm:

1. *Array* `list(i,itel)` *is the Verlet list of particle* `i`*, the number of particles in the Verlet list of particle* `i` *is given by* `nlist(i)`*, and the array* `xv(i)` *contains the position of the particles at the moment the list is made (is used to see when a new list has to be made). We assume that* all *particles are in the simulation box; hence* `x(i)` $\in$ *[0,*box*].*
2. *Subroutine* `new_nlist(rv,rn)` *makes a cell list (Algorithm 37). The desired cell size is* `rv` *and the actual cell size is* `rn`.

as the Verlet radius) for the various methods; we have simply taken some reasonable values.

For the Verlet list (and for the combination of Verlet and cell lists) it is important that the maximum displacement be smaller than twice the difference between the Verlet radius and cutoff radius. For the cutoff radius we have used $r_c = 2.5\sigma$, and for the Verlet radius $r_v = 3.0\sigma$. This limits the maximum displacement to $\Delta_x = 0.25\sigma$ and implies for the Lennard-Jones fluid that, if we want to use a optimum acceptance of 50%, we can use the Verlet method only for densities larger than $\rho > 0.6\sigma^{-3}$. For smaller densities, the optimum displacement is larger than 0.25. Note that this density dependence does not exist in a Molecular Dynamics simulation. In a Molecular Dynamics simulation, the maximum displacement is determined by the integration scheme and therefore is independent of density. This makes the Verlet method much more appropriate for a Molecular Dynamics simulation than for a Monte Carlo simulation. Only at high densities does it make sense to use the Verlet list.

The cell list method is advantageous only if the number of cells is larger than 3 in at least one direction. For the Lennard-Jones fluid this means that, if the number of particles is 400, the density should be lower than $\rho < 0.5\sigma^{-3}$. An important advantage of the cell list over the Verlet list is that this list can also be used for moves in which a particle is given a random position.

From these arguments it is clear that, if the number of particles is smaller than 200–500, the simple N^2 algorithm is the best choice. If the number of particles is significantly larger and the density is low, the cell list method is probably more efficient. At high density, all methods can be efficient and we have to make a detailed comparison.

To test these conclusions about the N dependence of the CPU time of the various methods, we have performed several simulations with a fixed number of Monte Carlo cycles. For the simple N^2 algorithm the CPU time per attempt is

$$\tau_{N^2} = cN,$$

where c is the CPU time required to calculate one interaction. This implies that the total amount of CPU time is independent of the density. For a calculation of the total energy, we have to do this calculation N times, which gives the scaling of N^2. Figure F.4 shows that indeed for the Lennard-Jones fluid, the τ_{N^2} increases linearly with the number of particles.

If we use the cell list, the CPU time will be

$$\tau_n = cV_l\rho + c_l p_l N,$$

where V_l is the total volume of the cells that contribute to the interaction (in three dimensions, $V_l = 27r_c^3$), c_n is the amount of CPU time required to

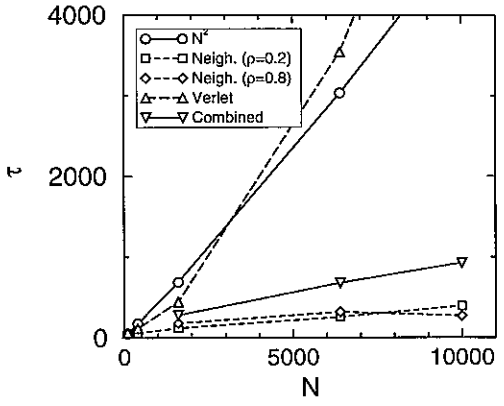

Figure F.4: Comparison of various schemes to calculate the energy: τ is in arbitrary units and N is the number of particles. As a test case the Lennard-Jones fluid is used. The temperature was $T^* = 2$ and per cycle the number of attempts to displace a particle was set to 100 for all systems. The lines serve to guide the eye.

make a cell list, and p_n is the probability that a new list has to be made. Figure F.4 shows that the use of a cell list reduces the CPU time for 10,000 particles with a factor 18. Interestingly, the CPU time does not increase with increasing density. We would expect an increase since the number of particles that contribute to the interaction of a particle i increases with density. However, the second contribution to τ_{Neigh} (p_n) is the probability that a new list has to be made, depends on the maximum displacement, which decreases when the density increases. Therefore, this last term will contribute less at higher densities.

For the Verlet scheme the CPU time is

$$\tau_v = cV_v\rho + c_v p_v N^2,$$

where V_v is the volume of the Verlet sphere (in three dimensions, $V_v = 4\pi r_v^3/3$), c_v is the amount of CPU time required to make the Verlet-list, and p_v is the probability that a new list has to be made. Figure F.4 shows that this scheme is not very efficient. The N^2 operation dominates the calculation. Note that we use a program in which a new list for all particles has to be made as soon as one of the particles has moved more than $(r_v - r_c)/2$; with some more bookkeeping it is possible to make a much more efficient program, in which a new list is made for only the particle that has moved out of the list.

The combination of the cell and Verlet lists removes the N^2 dependence of the simple Verlet algorithm. The CPU time is given by

$$\tau_c = cV_v\rho + c_v p_v c_n N.$$

Figure F.4 shows that indeed the N^2 dependence is removed, but the resulting scheme is not more efficient than the cell list alone.

This case study demonstrates that it is not simple to give a general recipe for which method to use. Depending on the conditions and number of particles, different algorithms are optimal. It is important to note that for a Molecular Dynamics simulation the conclusions may be different.

Appendix G

Reference States

G.1 Grand-Canonical Ensemble Simulation

In a grand-canonical ensemble simulation, we impose the temperature and chemical potential. Experimentally, however, usually the pressure rather than the chemical potential of the reservoir is fixed. To compare the experimental data with the simulation results it is necessary therefore to determine the pressure that corresponds to a given value of the chemical potential and temperature of our reservoir.

Preliminaries

The partition function of a system with N atoms in the N, V, T-ensemble is given by

$$Q(N,V,T) = \frac{V^N}{\Lambda^{3N}N!}\int ds^N \exp[-\beta\mathcal{U}(s^N)], \qquad \text{(G.1.1)}$$

where s^N are the scaled coordinates of the N particles. The free energy is related to the partition function via

$$F = -\frac{1}{\beta}\ln Q(N,V,T),$$

which gives us for the chemical potential

$$\mu \equiv \frac{\partial F}{\partial N} = -\frac{1}{\beta}\ln[Q(N+1,V,T)/Q(N,V,T)]. \qquad \text{(G.1.2)}$$