Chapter 4

Molecular Dynamics Simulations

Adapted from:

"Computer Simulations of Liquids"
M. P. Allen and D. J. Tildesley
Oxford University Press (2002)

In the previous Chapter 3, we have seen how one can calculate thermodynamic ensemble properties by using a stochastic process to generate phase space configurations. In this Chapter, we will generate the configurations of a thermodynamic ensemble in a deterministic way, by following the *classical molecular dynamics* of a system in time.

4.1 Equations of motion for atomic systems

In this chapter, we deal with the techniques used to solve the classical equations of motion for a system of N molecules interacting via a potential $\mathcal{V}$. These equations may be written down in various ways. You are probably most familiar with the Newton equations of motion. A fully equivalent, and perhaps more fundamental form is the Lagrangian equation of motion

$$\frac{\mathrm{d}}{\mathrm{d}t}(\partial \mathcal{L}/\partial \dot{q}_k) - (\partial \mathcal{L}/\partial q_k) = 0 \tag{4.1}$$

where the Lagrangian function $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})$ is defined in terms of kinetic and potential energies, $\mathcal{K}$ and $\mathcal{V}$, respectively as

$$\mathcal{L} = \mathcal{K} - \mathcal{V} \tag{4.2}$$

and is considered to be a function of the generalized coordinates $\mathbf{q_k}$ and their time derivatives $\dot{\mathbf{q}}_\mathbf{k}$. If we consider a system of atoms, with Cartesian coordinates $\mathbf{r}_i$, and the usual definitions of $\mathcal{K}$ and $\mathcal{V}$, Eq. (4.1) turns indeed into the Newton equation

$$m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i \tag{4.3}$$

where $\mathrm{m_i}$, is the mass of atom i and

$$\mathbf{f}_i = \nabla_{\mathbf{r}_i} \mathcal{L} = -\nabla_{\mathbf{r}_i} \mathcal{V} \tag{4.4}$$

is the force on that atom. These equations also apply to the centre of mass motion of a molecule, with $\mathbf{f_i}$ representing the total force on molecule i. The generalized momentum $\mathbf{p_k}$ conjugate to $\mathbf{q_k}$ is defined as

$$p_k = \partial \mathcal{L}/\partial \dot{q}_k \tag{4.5}$$

Computing centre of mass trajectories, thus involves solving a system of N second-order differential equations of the form of Eq. (4.3). Before considering how to do this, we can make some very general remarks regarding the equations themselves.

A consequence of Eq.(4.1), is that in certain circumstances a particular generalized momentum $\mathbf{p_k}$ may be conserved, i.e, $\dot{\mathbf{p}}_\mathbf{k} = 0$. The requirement is that $\mathcal{L}$, shall be independent of the corresponding generalized coordinate $\mathbf{q_k}$. For any set of particles, it is possible to choose six generalized coordinates, changes in which correspond to translations of the centre of mass, and

rotations about the centre of mass, for the system as a whole (changes in the remaining $3N-6$ coordinates involving motion of the particles relative to one another). If the potential function $\mathcal{V}$ depends only on the magnitudes of particle separations (as is usual) and there is no external field applied then $\mathcal{V}$ and $\mathcal{L}$ are manifestly independent of these six generalized coordinates. The corresponding conjugate momenta, in Cartesian coordinates, are the total linear momentum

$$\mathbf{P} = \sum_i \mathbf{p}_i \tag{4.6}$$

and the total angular momentum

$$\mathbf{L} = \sum_i \mathbf{r}_i \times \mathbf{p}_i = \sum_i m_i \mathbf{r}_i \times \dot{\mathbf{r}}_i \tag{4.7}$$

where we take the origin at the centre of mass of the system. Thus, these are conserved quantities for a completely isolated set of interacting molecules. In practice, we rarely consider completely isolated systems. A more general criterion for the existence of these conservation laws is provided by symmetry considerations. If the system is invariant to translation in a particular direction, then the corresponding momentum component is conserved. If the system is invariant to rotation about an axis, then the corresponding angular momentum component is conserved. Thus, we occasionally encounter systems enclosed in a spherical box, and so all three components of total angular momentum about the centre of symmetry will be conserved, but total translational momentum will not be. If the surrounding walls formed a cubical box, none of these quantities would be conserved. This is an important case in practice, since for the simulations of e.g. liquids, we often use periodic boundary conditions, i.e. we enclose the system in a periodically repeated box (as illustrated in Figure 4.1).

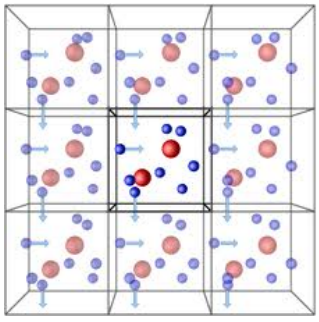

Fig. 4.1
Illustration of periodic boundary conditions
(from isaacs.sourceforge.net).

In the case of such periodic boundary conditions, it is easy to see that translational invariance is preserved, and hence total linear momentum is conserved. Several different box geometries can be considered (e.g. cubic, orthorhombic, or truncated octahedron), but none of them can be spherically symmetrical; in fact it is impossible (in Euclidean space) to construct a spherically symmetric periodic system. Hence, total angular momentum is not

conserved in most molecular dynamics simulations.
We have left until last the most important conservation law. It is easy to show, assuming $\mathcal{V}$ and $\mathcal{K}$ do not depend explicitly on time (so that the total energy $\mathcal{H}=\mathcal{V}+\mathcal{K}$ and $\delta\mathcal{H}/\delta t = 0$), that the form of the equations of motion guarantees that the total derivative $\dot{H} = d\mathcal{H}/dt$ is zero, i.e. the total energy is a ***constant of the motion***. This energy conservation law applies whether or not an external potential exists: the essential condition is that no explicitly time-dependent (or velocity-dependent) forces shall act on the system.
The second point concerning the equations of motion is that they are ***reversible in time***. By changing the signs of all the velocities or momenta, we will cause the molecules to retrace their trajectories. If the equations of motion are solved correctly, the computer-generated trajectories will also have this property.
Our final observation concerning Eqs. (4.3) and (4.4) is that the spatial derivative of the potential appears. This leads to a qualitative difference in the form of the motion, and the way in which the equations are solved, depending upon whether or not $\mathcal{V}$ *is a continuous function of particle positions*. To use the finite time-step method of solution to be described in the next section, it is essential that the particle positions vary smoothly with time: a Taylor expansion of $\mathbf{r}(t)$ about time t may be necessary, for example. Whenever the potential varies sharply (as in the hard sphere and square well cases or by using distance cutoffs for long-range interactions) impulsive 'collisions' between particles occur at which the velocities (typically) change discontinuously.

4.2 Finite difference methods

A standard method for solution of ordinary differential equations such as Eq.(4.3) is the finite difference approach. The general idea is as follows. Given the molecular positions, velocities, and other dynamic information at time t, we attempt to obtain the positions, velocities etc. at a later time $t + \Delta t$, to a sufficient degree of accuracy. The equations are solved on a step-by-step basis; the choice of the time interval Δt will depend somewhat on the method of solution, but Δt will be significantly smaller than the typical time taken for a molecule to travel its own length. Many different algorithms fall into the general finite difference pattern. To illustrate the principles of the method, we will choose one (a ***predictor-corrector algorithm***) and then proceed to discuss the technical details, which affect the choice in practice.
If the classical trajectory is continuous, then an estimate of the positions, velocities etc. at time $t + \Delta t$ may be obtained by Taylor expansion about time t:

$$\mathbf{r}^p(t+\Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t) + 1/2\Delta t^2\mathbf{a}(t) + 1/6\Delta t^3\mathbf{b}(t) + \ldots$$
$$\mathbf{v}^p(t+\Delta t) = \mathbf{v}(t) + \Delta t\mathbf{a}(t) + 1/2\Delta t^2\mathbf{b}(t) + \ldots$$
$$\mathbf{a}^p(t+\Delta t) = \mathbf{a}(t) + \Delta t\mathbf{b}(t) + \ldots$$
$$\mathbf{b}^p(t+\Delta t) = \mathbf{b}(t) + \ldots \qquad (4.8)$$

The superscript marks these as 'predicted' values: we shall be 'correcting' them shortly. Just as $\mathbf{r}$ and $\mathbf{v}$ stand for the complete set of positions and

velocities, so **a** is short for all the accelerations, and **b** denotes all the third time derivatives of **r**. If we truncate the expansion, retaining (for example) just the terms given explicitly in Eq. (4.8), then we seem to have achieved our aim of (approximately) advancing the values of the stored coordinates and derivatives from one time step to the next. In this example, we would store four 'vectors' **r**, **v**, **a**, and **b**. Equivalent alternatives would be to base the prediction on **r, v** and 'old' values of the velocities $\mathbf{v}(t - \Delta t)$, $\mathbf{v}(t - 2\Delta t)$, or on **r, v,** **a** and 'old' accelerations $\mathbf{a}(t - \Delta t)$, using slightly different predictor equations. However, there is a snag. An equation like Eq. (4.8) will not generate correct trajectories as time advances, because we have not introduced the equations of motion. These enter through the correction step. We may calculate, from the new positions $\mathbf{r}^p$, the forces at time $t+\Delta t$, and hence the correct accelerations $\mathbf{a}^c(t+\Delta t)$. These can be compared with the predicted accelerations from Eq.(4.8), to estimate the size of the error in the prediction step:

$$\Delta\mathbf{a}(t+\Delta t) = \mathbf{a}^c(t+\Delta t) - \mathbf{a}^p(t+\Delta t) \qquad (4.9)$$

This error, and the results of the predictor step, are fed into the corrector step, which reads, typically,

$$\begin{aligned}
\mathbf{r}^c(t+\Delta t) &= \mathbf{r}^p(t+\Delta t) + c_o\,\Delta a(t+\Delta t)\\
\mathbf{v}^c(t+\Delta t) &= \mathbf{v}^p(t+\Delta t) + c_1\,\Delta a(t+\Delta t)\\
\mathbf{a}^c(t+\Delta t) &= \mathbf{a}^p(t+\Delta t) + c_2\,\Delta a(t+\Delta t)\\
\mathbf{b}^c(t+\Delta t) &= \mathbf{b}^p(t+\Delta t) + c_3\,\Delta a(t+\Delta t)
\end{aligned} \qquad (4.10)$$

The idea is that $\mathbf{r}^c(t+\Delta t)$ etc. are now better approximations to the true positions, velocities etc. Gear has discussed the 'best' choice for the coefficients c_o, c_1 , c_2, c_3 ••• (i.e. the choice leading to optimum stability and accuracy of the trajectories). Different values of the coefficients are required if we include more (or fewer) position derivatives in our scheme, and the coefficients also depend upon the order of the differential equation being solved (here it is second order, since the double time derivative of position is being compared with the accelerations computed from **r**).
The corrector step may be iterated: new 'correct' accelerations are calculated from the positions $\mathbf{r}^c$ and compared with the current values of $\mathbf{a}^c$, so as to further refine the positions, velocities etc. through an equation like Eq. (4.10).
In many applications this iteration is the key to obtaining an accurate solution. The predictor provides an initial guess at the solution, which in principle does not have to be a very good one, since the successive corrector iterations should then converge rapidly onto the correct answer. In molecular dynamics, however, the evaluation of accelerations (i.e. forces) from particle positions is the most time-consuming part of a simulation, and since this is implicit in each corrector step, a large number of corrector iterations would be very expensive. Normally, just one (occasionally two) corrector steps are carried out, and so an accurate predictor stage (such as the Taylor series of Eq. (4.8)) is essential. The general scheme of a stepwise MD simulation, based on a predictor-corrector algorithm, may be summarized as follows:

(a) predict the positions, velocities, accelerations etc., at a time $t + \Delta t$, using the current values of these quantities;
(b) evaluate the forces, and hence accelerations $\mathbf{a_i} = \mathbf{f_i}/m_i$, from the new positions;
(c) correct the predicted positions, velocities, accelerations etc., using the new accelerations;
(d) calculate any variables of interest, such as the energy, virial, order parameters, ready for the accumulation of time averages, before returning to (a) for the next step.

The simple predictor-corrector algorithm just described is only one of many possibilities. Which is the best algorithm to use in MD? The choice is not so wide as it may seem at first. Firstly, a large class of algorithms may be interconverted by simple matrix transformations and hence are essentially equivalent, although there may be small differences due to round-off errors.
A shopping list of desirable qualities for a successful simulation algorithm might be as follows:

(a) It should be fast, and require little memory.
(b) It should permit the use of a long time step Δt.
(c) It should duplicate the classical trajectory as closely as possible.
(d) It should satisfy the known conservation laws for energy and momentum, and be time-reversible.
(e) It should be simple in form and easy to program.

For molecular dynamics, not all of the above points are very important. Compared with the time-consuming force calculation, which is carried out at every time step, the raw speed of the integration algorithm is not crucial. It is far more important to be able to employ a long time step Δt: in this way, a given period of 'simulation' time can be covered in a modest number of integration steps, i.e, in an acceptable amount of computer time. Clearly, the larger Δt, the less accurately will our solution follow the correct classical trajectory. How important are points (c) and (d) above?
The accuracy and stability of a simulation algorithm are measured by its local and global truncation errors, and algorithms may be tested on a simple model, such as the harmonic oscillator. The algorithms used in molecular dynamics simulations have been chosen with these criteria in mind, but it is unreasonable to expect that any approximate method of solution will dutifully follow the exact classical trajectory indefinitely. Any two classical trajectories, which are initially very close will eventually diverge from one another exponentially with time. In the same way, any small perturbation, even the tiny error associated with finite precision arithmetic, will tend to cause a computer-generated trajectory to diverge from the true classical trajectory with which it is initially coincident. This effect is illustrated in Fig. 4.2: using one simulation as a reference, it is shown that a small perturbation applied at time t = 0 causes the trajectories in the perturbed simulation to diverge from the reference trajectories, and become statistically uncorrelated, within a few hundred time

steps. In this example, we show the growing average 'distance in configuration space', defined as $|\Delta \mathbf{r}|$ where $|\Delta \mathbf{r}|^2 = (1/N) \Sigma |\mathbf{r}_i(t) - \mathbf{r}_i^0(t)|^2$, $\mathbf{r}_i^0(t)$ being the position of molecule i at time t in a reference simulation, and $\mathbf{r}_i(t)$ being the position of the same molecule at the same time in the perturbed simulation. In the three cases illustrated here, all the molecules in the perturbed runs are initially displaced in random directions from their reference positions at t = 0, by $10^{-3}\sigma$, $10^{-6}\sigma$, and $10^{-9}\sigma$, respectively, where σ is the molecular diameter. In all other respects, the runs are identical; in particular, each corresponds to essentially the same total energy. As the runs proceed, however, other mechanical quantities eventually become statistically uncorrelated. In Fig. 4.2, we show the percentage difference in kinetic energies between perturbed and reference simulations. On the scale of the figure, the kinetic energies remain very close for a period whose length depends on the size of the initial perturbation; after this point the differences become noticeable very rapidly. Presumably, both the reference trajectory and the perturbed trajectory are diverging from the true solution of Newton's equations.

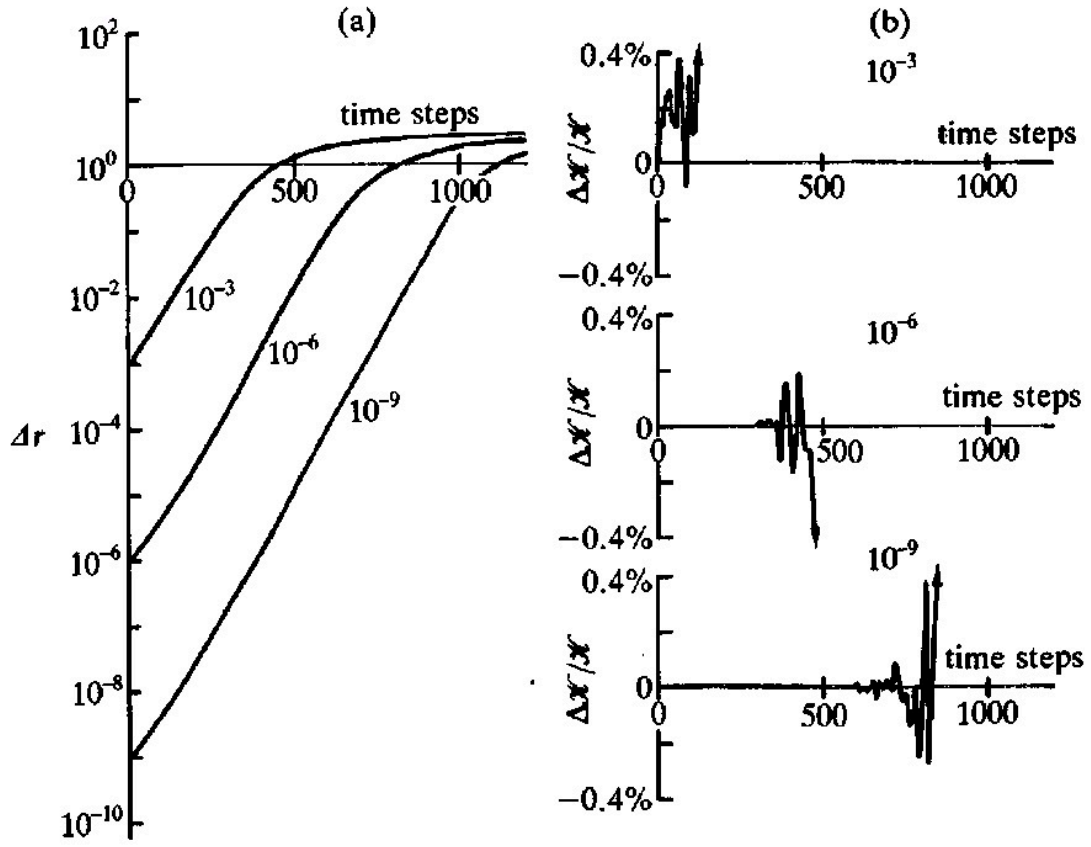

Fig. 4.2 The divergence of trajectories in molecular dynamics. The curves are labeled with the initial displacement in units of σ. (a) Δr is the phase space separation between perturbed and reference trajectories. (b) $\Delta \mathcal{K}/\mathcal{K}$ is the percentage difference in kinetic energies.

Clearly, no integration algorithm will provide an essentially exact solution for a

very long time. Fortunately, we do not need to do this. Remember that we use molecular dynamics to generate states sampled from the e.g. microcanonical ensemble, We do not need exact classical trajectories to do this, but must lay great emphasis on energy conservation as being of primary importance for this reason. Momentum conservation is also important, but this can usually be easily arranged. The point is that the particle trajectories must stay on the appropriate constant-energy hypersurface in phase space, otherwise correct ensemble averages will not be generated. Energy conservation is degraded as the time step is increased, and so all simulations involve a trade-off between economy and accuracy: a good algorithm permits a large time step to be used while preserving acceptable energy conservation. Other factors dictating the energy-conserving properties are the shape of the potential energy curves and the typical particle velocities. Thus, shorter time steps are used at high temperatures, for light molecules, and for rapidly varying potential functions.

The final quality an integration algorithm should possess is simplicity. A simple algorithm will involve the storage of only a few coordinates, velocities etc., and will be easy to program. Bearing in mind that solution of ordinary differential equations is a fairly routine task, there is little point in wasting valuable man-hours on programming a very complicated algorithm, when the time might be better spent checking and optimizing the calculation of forces. Little computer time is to be gained by increases in algorithm speed, and the consequences of making a mistake in coding a complicated scheme might be significant. We now turn to specific examples of algorithms in common use.

4.2.1 The Verlet algorithm

Perhaps the most widely used method of integrating the equations of motion is that initially adopted by Verlet.

This method is a direct solution of the second-order equations (4.3). The method is based on positions $\mathbf{r}(t)$, accelerations $\mathbf{a}(t)$, and the positions $\mathbf{r}(t - \Delta t)$ from the previous step. The equation for advancing the positions reads as follows:

$$\mathbf{r}(t +\Delta t) = 2\mathbf{r}(t) – \mathbf{r}(t-\Delta t) + \Delta t^2\mathbf{a}(t) \qquad (4.11)$$

There are several points to note about Eq.(4.11). It will be seen that the velocities do not appear at all. They have been eliminated by addition of the equations obtained by Taylor expansion about $\mathbf{r}(t)$:

$$\mathbf{r}(t +\Delta t) = \mathbf{r}(t) +\Delta t\mathbf{v}(t) + 1/2\Delta t^2\mathbf{a}(t) + \ldots$$
$$\mathbf{r}(t -\Delta t) = \mathbf{r}(t) - \Delta t\mathbf{v}(t) + 1/2\Delta t^2\mathbf{a}(t) - \ldots \qquad (4.12)$$

The velocities are not needed to compute the trajectories, but they are useful for estimating the kinetic energy (and hence the total energy). They may be obtained from the formula

$$\boldsymbol{v}(t) = \frac{\boldsymbol{r}(t+\Delta t)-\boldsymbol{r}(t-\Delta t)}{2\Delta t} \qquad (4.13)$$

Whereas Eq. (4.11) is correct except for errors of order Δt^4 (the local error) the velocities from Eq. (4.13) are subject to errors of order Δt^2. More accurate estimates of $\mathbf{v}(t)$ can be made, if more variables are stored, but this adds to the inconvenience already implicit in Eq. (4.13), namely that $\mathbf{v}(t)$ can only be computed once $r(t+\Delta t)$ is known. A second observation regarding the Verlet algorithm is that it is properly centred (i.e. $r(t-\Delta t)$ and $r(t+\Delta t)$ play symmetrical roles in Eq. (4.11)), making it time-reversible. Thirdly, the advancement of positions takes place all in one go, rather than in two stages as in the predictor-corrector example. The overall scheme is illustrated in Fig. 4.3.

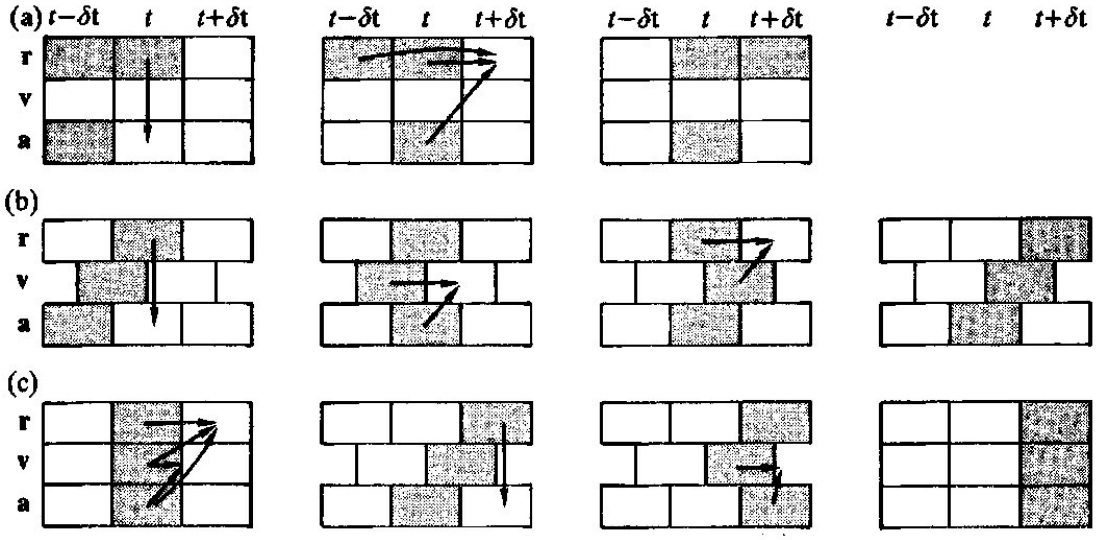

Fig. 4.3 Various forms of the Verlet algorithm. (a) Verlet's original method. (b) The leap-frog form. (c) The velocity form. Successive steps in the implementation of each algorithm are shown. In each case, the stored variables are in grey boxes.

As we can see, the Verlet method requires essentially 9N words of storage, making it very compact, and it is simple to program. The algorithm is exactly reversible in time and, given conservative forces, is guaranteed to conserve linear momentum. The method has been shown to have excellent energy-conserving properties even with long time steps. Typical energy conservation are typically 0.001-0.01%. Against the Verlet algorithm, one must say that the handling of velocities is rather awkward, and that the form of the algorithm may needlessly introduce some numerical imprecision. This arises because, in Eq. 4.11, a small term ($O(\delta t^2)$) is added to a difference of large terms ($O(\delta t^0)$), in order to generate the trajectory.

Modifications to the basic Verlet scheme have been proposed to tackle these deficiencies. One of these is a so-called half-step ***'leap-frog'*** scheme. The origin of the name becomes apparent when we write the algorithm down:

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) - \Delta t\mathbf{v}(t+1/2\Delta t) \qquad (4.14a)$$

$$\mathbf{v}(t+1/2\Delta t) = \mathbf{v}(t-1/2\Delta t) + \Delta t\mathbf{a}(t) \quad (4.14b)$$

The stored quantities are the current positions $\mathbf{r}(t)$ and accelerations $\mathbf{a}(t)$ together with the mid-step velocities $\mathbf{v}(t-1/2\Delta t)$. The velocity equation (4.14b) is implemented first, and the velocities leap over the coordinates to give the next mid-step values $\mathbf{v}(t+1/2\Delta t)$. During this step, the current velocities may be calculated

$$\mathbf{v}(t) = \tfrac{1}{2}(\mathbf{v}(t+1/2\Delta t) + \mathbf{v}(t-1/2\Delta t)) \quad (4.15)$$

This is necessary so that the energy $\mathcal{H}=\mathcal{K}+\mathcal{V}$ at time t can be calculated, as well as any other quantities that require positions and velocities at the same instant. Following this, Eq. (4.14a) is used to propel the positions once more ahead of the velocities. After this, the new accelerations may be evaluated ready for the next step. This is illustrated in Fig. 4.3. Elimination of the velocities from these equations shows that the method is algebraically equivalent to Verlet's algorithm. There are some advantages in programming Eqs. (4.14-4.15), however, since the velocities (admittedly not at time t) appear explicitly; for example, adjusting the simulation energy is usually achieved by appropriately scaling the velocities. Numerical benefits derive from the fact that at no stage do we take the difference of two large quantities to obtain a small one; this minimizes loss of precision on a computer. If there is a desperate need to conserve storage space, the accelerations may be directly accumulated onto the velocities, thus making the overall requirements of order 6N words. The cost is that Eq.(4.15) may no longer be used, and it becomes necessary to estimate the kinetic energy at time t from the known mid-step values. Finally, note that the leapfrog approach may be applied to other algorithms as well as Verlet's.

As Eq.(4.15), leap-frog methods still do not handle the velocities in a completely satisfactory manner. A Verlet-equivalent algorithm, which does store positions, velocities, and accelerations all at the same time t, and which minimizes round-off error, has also been proposed. This ***'velocity Verlet'*** algorithm takes the form

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) - \Delta t\mathbf{v}(t)+1/2\Delta t^2\mathbf{a}(t) \quad (4.16a)$$
$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) +1/2\Delta t(\mathbf{a}(t) + \mathbf{a}(t+\Delta t)) \quad (4.16b)$$

Again, the Verlet algorithm may be recovered by eliminating the velocities. In this form, the method resembles a three-value predictor-corrector algorithm, where the position corrector coefficient is zero. The algorithm only requires storage of **r, v**, and **a**. Although it is not implemented in exactly the form of a Gear predictor-corrector, it does involve two stages, with a force evaluation in between. Firstly, the new positions at time $t+\Delta t$ are calculated using Eq. (4.16a), and the velocities at mid-step are computed using

$$\mathbf{v}(t+1/2\Delta t) = \mathbf{v}(t) +1/2\Delta t\mathbf{a}(t) \quad (4.17)$$

The forces and accelerations at time $t+\Delta t$ are then computed, and the velocity

move completed

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t+1/2\Delta t)+Dt\mathbf{a}(t+\Delta t) \qquad (4.18)$$

At this point, the kinetic energy at time $t+\Delta t$ is available. The potential energy at this time will have been evaluated in the force loop. The whole process is shown in Fig. 4.3. The method once more uses 9N words of storage, and its numerical stability, convenience, and simplicity make it perhaps the most attractive proposed to date. The code for the velocity version of Verlet's method is a straightforward transcription of Eqs. (4.16)-(4.18).
All the methods described in this section are essentially equivalent in that they have identical global errors and in fact generate identical position trajectories.

4.2.2 The Gear predictor-corrector

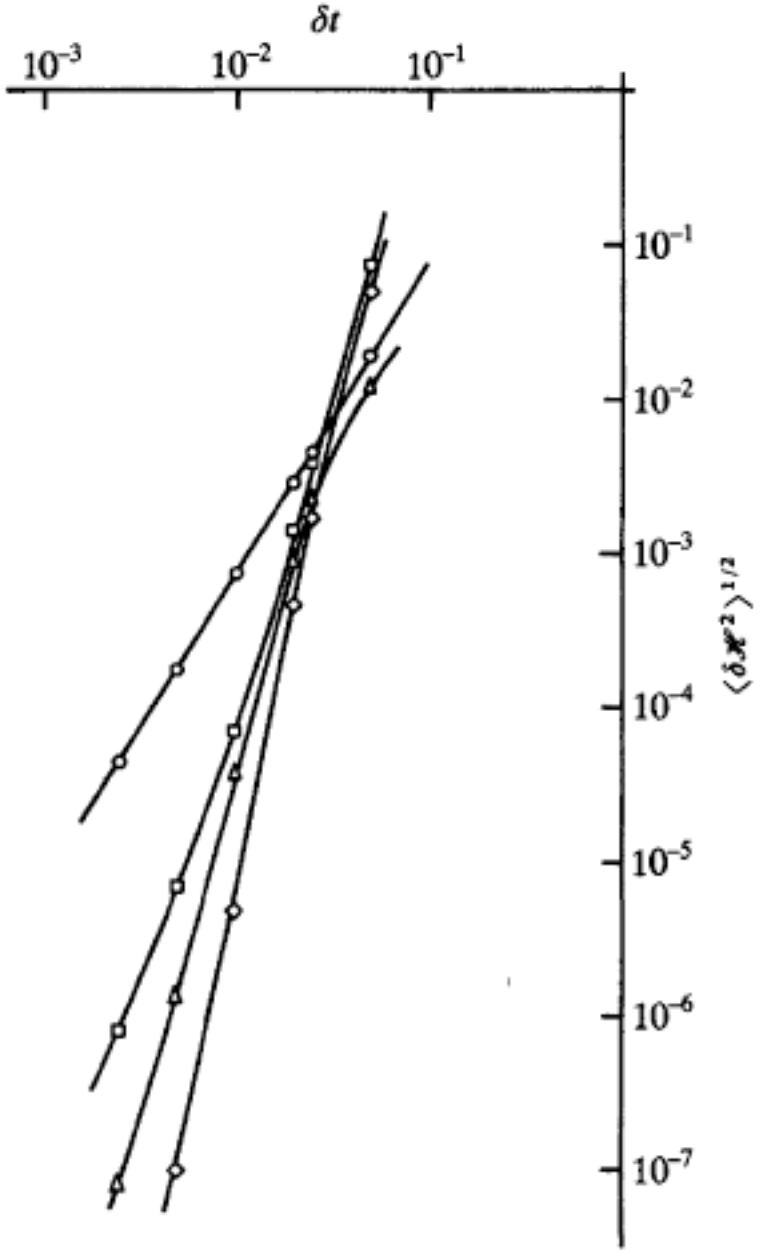

Fig. 4.4 Energy conservation of various algorithms. RMS energy fluctuations

$<\delta\mathcal{H}^2>^{1/2}$ for various runs starting from the same initial conditions, and proceeding for the same total simulation time t_{run} but using different time steps Δt and corresponding numbers of steps $t_{run}= t_{run}/\Delta t$. The plot uses log-log scales. The curves correspond to velocity Verlet (circles), Gear fourth-order (squares), Gear fifth-order (triangles], and Gear sixth-order (diamonds) algorithms.

We have already discussed the basic predictor-corrector algorithm with Gear's set of corrector coefficients. How does it compare with the various forms of the Verlet algorithm? In the form described here, a four-value Gear algorithm (as described earlier) requires 15N words of storage (**r, v, a, b**, and the new accelerations/forces); a more accurate five-value method would need 18N words. This is a large requirement compared with Verlet, and this may be an important factor. Unfortunately, increasing the order of a Gear method does not result in a great improvement in accuracy for molecular dynamics. It is instructive to consider the reasons for this. In a liquid, the forces on a molecule, and hence its motion during a time step, are dictated by the motion of its neighbours, particularly the close neighbours, which move into and out of a small region of strong interaction with the molecule. This makes any Taylor series predictor, which takes no account of the motion of the neighbours, unreliable, and a high-order predictor is no significant improvement over a low-order one. Since the success of the predictor-corrector method relies on the accuracy of the predictor, especially if we perform only one corrector iteration, the higher-order Gear algorithms have little to offer over the simpler low-order Verlet type methods. If we measure performance by calculating the root-mean-square energy fluctuations $<\delta\mathcal{H}^2>^{1/2}$ for a short run using a particular time step Δt, then the results typically take the form of Fig. 4.4. At short Δt, the higher-order Gear methods are more accurate, but in most simulations we are interested in making Δt as high as possible. With a longer time step, the Verlet algorithm is more attractive. In fact, $<\delta\mathcal{H}^2>^{1/2}$ is closely proportional to Δt^2 for Verlet-equivalent algorithms, while energy conservation for the higher-order methods worsens much more rapidly with increasing Δt.

4.3 Constraint dynamics

In polyatomic systems, it becomes necessary to consider not only the stretching of interatomic bonds, but also bending motions, which change the angle between bonds, and twisting motions, which alter torsional angles. These torsional motions are, typically, of much lower frequency than bond vibrations, and are very important in long-chain organic molecules: they lead to conformational interconversion and have a direct influence on polymer dynamics. Clearly, these effects must be treated properly in molecular dynamics, within the classical approximation. It would be quite unrealistic to assume total rigidity of such a molecule, although bond lengths can be thought of as fixed and a case might be made out for a similar approximation in the case of bond bending angles.

Of course, for any system with such *holonomic constraints* applied (i.e. a set of algebraic equations connecting the coordinates) it is possible to construct a set of generalized coordinates obeying constraint-free equations of motion

(i.e. ones in which the constraints appear implicitly). For any molecule of moderate complexity, such an approach would be very complicated, although it was used in the first simulations of butane. The equations of motion in such a case are derived from first principles, starting with the Lagrangian (Eqs. (4.1) and (4.2)).

A special technique has been developed to handle the dynamics of a molecular system in which certain arbitrarily selected degrees of freedom (such as bond lengths) are constrained, while others remain free to evolve under the influence of intermolecular and intramolecular forces. This constraint dynamics approach in effect uses a set of undetermined multipliers to represent the magnitudes of forces directed along the bonds, which are required to keep the bond lengths constant. The technique is to solve the equations of motion for one time step in the absence of the constraint forces, and subsequently determine their magnitudes and correct the atomic positions. The method can be applied equally well to totally rigid and non-rigid molecules. Its great appeal is that it reduces even a complex polyatomic liquid simulation to the level of difficulty of an atomic calculation plus a constraint package based on molecular geometry. We will illustrate an application of this method with a simple example. In this section, as in the last one, $\mathbf{r}_a$ will represent the position of atom a in a specific molecule.

Consider a bent triatomic molecule such as water, in which we wish to constrain two of the bonds to be of fixed length, but allow the remaining bond, and hence the inter-bond angle, to vary under the influence of the intramolecular potential. Numbering the central (oxygen) atom 2, and the two outer (hydrogen) atoms 1 and 3, we write the equations of motion in the form

$$m_1\ddot{\mathbf{r}}_1 = \boldsymbol{f}_1 + \boldsymbol{g}_1 \qquad (4.19a)$$
$$m_2\ddot{\mathbf{r}}_2 = \boldsymbol{f}_2 + \boldsymbol{g}_2 \qquad (4.19b)$$
$$m_3\ddot{\mathbf{r}}_3 = \boldsymbol{f}_3 + \boldsymbol{g}_3 \qquad (4.19c)$$

Here $\mathbf{f}_1$, $\mathbf{f}_2$ and $\mathbf{f}_3$ are the forces due to intermolecular interactions and to those intramolecular effects that are explicitly included in the potential. The remaining terms $\mathbf{g}_1$ etc. are the constraint forces: their role is solely to keep the desired bond lengths constant, that is to ensure that the equations

$$\chi_{12} = r_{12}^2(t) - d_{12}^2 = 0 \qquad (4.20a)$$
$$\chi_{23} = r_{23}^2(t) - d_{23}^2 = 0 \qquad (4.20a)$$

(where d_{12} and d_{23} are the bond lengths, and $r_{12} = |\mathbf{r}_1 - \mathbf{r}_2|$ etc.) are satisfied at all times. The Lagrangian equations of motion are derived from these constraints; they are Eqs (4.19) with

$$\boldsymbol{g}_a = \tfrac{1}{2}\lambda_{12}\boldsymbol{\nabla}_{\boldsymbol{r}_a}\chi_{12} + \tfrac{1}{2}\lambda_{23}\boldsymbol{\nabla}_{\boldsymbol{r}_a}\chi_{23} \qquad (4.21)$$

and λ_{12} and λ_{23} are undetermined (Lagrangian) multipliers. The factors of (1/2) are introduced so that this definition of the multipliers agrees with later equations. So far, we have made no approximations, and, in principle, could solve for the constraint forces. However, because we are bound to solve the

equations of motion approximately, using finite difference methods, in practice this will lead to bond lengths that steadily diverge from the desired values. Instead, one can use an approach in which the constraint forces are calculated so as to guarantee that the constraints are satisfied at each time step; by implication, the constraint forces themselves are only correct to the same order of accuracy as the integration algorithm. Thus, we write

$$m_a \ddot{\mathbf{r}}_\mathbf{a} = \boldsymbol{f}_a + \boldsymbol{g}_a \approx \boldsymbol{f}_a + \boldsymbol{g}_a^{(r)} \tag{4.22}$$

where $\mathbf{g_a^{(r)}}$ is an approximation (the form of which will be given below) to the true forces of constraint, $\mathbf{g_a}$ acting on each atom a. By considering the way in which these forces enter into the Verlet algorithm, Eq. (4.11), we can write

$$\boldsymbol{r}_\mathbf{a}(t+\Delta t) = \boldsymbol{r'}_a(t+\Delta t) + (\frac{\Delta t^2}{m_a})\boldsymbol{g}_a^{(r)}(t) \tag{4.23}$$

where $\mathbf{r'_a}(t+\Delta t)$ is the position which would have been reached in the absence of any constraints. Returning to our example of water, and recognizing that the constraint forces must be directed along the bonds and must conform to Newton's third law, we see that

$$\boldsymbol{g}_\mathbf{1}^{(r)} = \lambda_{12}\boldsymbol{r}_{12} \tag{4.24a}$$
$$\boldsymbol{g}_\mathbf{2}^{(r)} = \lambda_{23}\boldsymbol{r}_{23} - \lambda_{12}\boldsymbol{r}_{12} \tag{4.24b}$$
$$\boldsymbol{g}_\mathbf{3}^{(r)} = -\lambda_{23}\,\boldsymbol{r_{23}} \tag{4.24c}$$

where λ_{12} and λ_{23} are the undetermined multipliers. These may be calculated if we write out Eq. (4.23) explicitly:

$$\boldsymbol{r_1}(t+\Delta t) = \boldsymbol{r'}_\mathbf{1}(t+\Delta t) + (\frac{\Delta t^2}{m_1})\lambda_{12}\boldsymbol{r_{12}}(t) \tag{4.25a}$$
$$\boldsymbol{r_2}(t+\Delta t) = \boldsymbol{r'}_\mathbf{2}(t+\Delta t) + (\frac{\Delta t^2}{m_2})\lambda_{23}\boldsymbol{r_{23}}(t) - (\frac{\Delta t^2}{m_2})\lambda_{12}\boldsymbol{r_{12}}(t) \tag{4.25b}$$
$$\boldsymbol{r_3}(t+\Delta t) = \boldsymbol{r'}_\mathbf{3}(t+\Delta t) + (\frac{\Delta t^2}{m_3})\lambda_{23}\boldsymbol{r_{23}}(t) \tag{4.25c}$$

Thus

$$\boldsymbol{r_{12}}(t+\Delta t) = \boldsymbol{r'}_{\mathbf{12}}(t+\Delta t) + \Delta t^2\left(1/m_1 + 1/m_2\right)\lambda_{12}\boldsymbol{r_{12}}(t) - \Delta t^2 m_2^{-1}\lambda_{23}\boldsymbol{r_{23}}(t) \tag{4.26a}$$
$$\boldsymbol{r_{23}}(t+\Delta t) = \boldsymbol{r'}_{\mathbf{23}}(t+\Delta t) - \frac{\Delta t^2}{m_2}\lambda_{12}\boldsymbol{r_{12}}(t) + \Delta t^2\left(1/m_2 + 1/m_3\right)m_2^{-1}\lambda_{23}\boldsymbol{r_{23}}(t) \tag{4.26b}$$

Now we can take the square modulus of both sides, and apply our desired constraints: $|\mathbf{r_{12}}(t+\Delta t)|^2 = |\mathbf{r_{12}}(t)|^2 = d_{12}$ and similarly for $\mathbf{r_{23}}$. The result is a pair of quadratic equations in λ_{12} and λ_{23}, the coefficients in which are all known (given that we already have the 'unconstrained' positions and which can be solved for the undetermined multipliers. In practice, since terms linear in λ_{12}

and λ_{23} are proportional to Δt^2, while the second order terms are proportional to Δt^4, these equations are solved in an iterative fashion. The quadratic terms are dropped and the remaining linear equations solved for λ_{12} and λ_{23}; these values are substituted into the quadratic terms to give new linear equations, which yield improved estimates of λ_{12} and λ_{23}, and so on. Finally, these values are used in Eq. (4.25).

We have examined this case in some detail so as to bring out the important features in a more general scheme. Bond angle (as opposed to bond length) constraints present no fundamental difficulty, and may be handled by introducing additional length constraints. For example, the H-O-H bond angle in water may be fixed by constraining the H-H distance, in addition to the O-H bond lengths. Instead of Eq. (4.25) we would then have

$$\boldsymbol{r_1}(t+\Delta t) = \boldsymbol{r'_1}(t+\Delta t) + (\frac{\Delta t^2}{m_1})\lambda_{12}\boldsymbol{r_{12}}(t) - (\frac{\Delta t^2}{m_1})\lambda_{31}\boldsymbol{r_{31}}(t) \quad (4.27a)$$

$$\boldsymbol{r_2}(t+\Delta t) = \boldsymbol{r'_2}(t+\Delta t) + (\frac{\Delta t^2}{m_2})\lambda_{23}\boldsymbol{r_{23}}(t) - (\frac{\Delta t^2}{m_2})\lambda_{12}\boldsymbol{r_{12}}(t) \quad (4.27b)$$

$$\boldsymbol{r_3}(t+\Delta t) = \boldsymbol{r'_3}(t+\Delta t) + (\frac{\Delta t^2}{m_3})\lambda_{31}\boldsymbol{r_{31}}(t) - (\frac{\Delta t^2}{m_3})\lambda_{23}\boldsymbol{r_{23}}(t) \quad (4.27c)$$

and Eq. (4.26) would be replaced by

$$\boldsymbol{r_{12}}(t+\Delta t) = \boldsymbol{r'_{12}}(t+\Delta t) + \Delta t^2(1/m_1 + 1/m_2)\lambda_{12}\boldsymbol{r_{12}}(t) - \Delta t^2 m_2^{-1}\lambda_{23}\boldsymbol{r_{23}}(t) - \Delta t^2 m_1^{-1}\lambda_{31}\boldsymbol{r_{31}}(t) \quad (4.28a)$$

$$\boldsymbol{r_{23}}(t+\Delta t) = \boldsymbol{r'_{23}}(t+\Delta t) - \Delta t^2 m_3^{-1}\lambda_{31}\boldsymbol{r_{31}}(t) + \Delta t^2(1/m_2 + 1/m_3)\lambda_{23}\boldsymbol{r_{23}}(t) - \Delta t^2 m_2^{-1}\lambda_{12}\boldsymbol{r_{12}}(t) \quad (4.28b)$$

$$\boldsymbol{r_{31}}(t+\Delta t) = \boldsymbol{r'_{31}}(t+\Delta t) + \Delta t^2 m_1^{-1}\lambda_{12}\boldsymbol{r_{12}}(t) - \Delta t^2 m_3^{-1}\lambda_{23}\boldsymbol{r_{23}}(t) + \Delta t^2(1/m_3 + 1/m_1)\lambda_{31}\boldsymbol{r_{31}}(t) \quad (4.28c)$$

This process of 'triangulating' the molecule by introducing fictitious bonds is straightforwardly applied to more complex systems.

For very small molecules, as in the example above, the (linearized) constraint equations may be solved by straightforward algebra. For a larger polyatomic molecule, with n_c constraints, the solution of these equations essentially requires inversion of an n_c x n_c; matrix at each time step. This could become time-consuming for very large molecules, such as proteins. Assuming, however, that only near-neighbour atoms and bonds are related by constraint equations, the constraint matrix will be sparse, and special inversion techniques might be applicable. An alternative procedure is to go through the constraints one by one, cyclically, adjusting the coordinates so as to satisfy each in turn. The procedure may be iterated until all the constraints are satisfied to within a given tolerance. This approach has been called ***SHAKE*** and is most useful when large molecules are involved.

Problems-may arise in the construction of a constraint scheme for certain molecules. Consider the linear molecule CS_2: it has three atoms and five degrees of freedom (two rotational and three translational) so we require n_c = 3 x 3- 5 = 4 constraints. This is impossible with only three bond lengths available to be specified. Amore subtle example is that of benzene, modeled

as six united CH atoms in a hexagon. For six degrees of freedom (three rotational and three translational) we require n_c = 3 x 6- 6 = 12 constraints, and this number may indeed be accommodated. However, the constraint matrix is then found to be singular, i.e. its determinant vanishes. Physically, the problem is that all the constraints act in the plane of the molecule, and none of them act to preserve planarity. The solution to both these problems is to choose a subset of atoms sufficient to define the molecular geometry, apply constraints to those atoms, and express the coordinates of the remaining atoms as linear combinations of those of the primary 'core'. In computing the dynamics of the core, there is a simple prescription for transferring the forces acting on the 'secondary' atoms to the core atoms, so as to 'generate the correct linear and angular accelerations. Recently, the ***SHAKE*** method has been extended to handle more general geometrical constraints needed to specify (for example) the arrangement of side chains or substituent atoms in flexible hydrocarbons.

SHAKE is most easily applied to the Verlet algorithm, in which only positions and accelerations appear. The aim of a constraint package in an algorithm which includes velocities (and higher derivatives of position) is to ensure that not only do we satisfy the constraint equations but also we satisfy the derivatives of those equations: if $\boldsymbol{r_{ab}^2} = \boldsymbol{r_{ab}} \cdot \boldsymbol{r_{ab}}$ = constant, then we should ensure that $\frac{d(\boldsymbol{r_{ab}^2})}{dt} = 2\boldsymbol{r_{ab}} \cdot \dot{\boldsymbol{r}}_{\boldsymbol{ab}}$. Recently, a modification of the method of constraints, built around the velocity version of ***Verlet***'s algorithm has been proposed. Again, approximations to the true constraint forces are needed in the equations of motion to guarantee that the constraints are satisfied at all time steps. The ***velocity Verlet*** algorithm is a two-stage process, with each stage involving the forces, including the forces of constraint. Accordingly, at each stage an approximation to g is made, so as to ensure that the constraints are satisfied. Referring to Eqs. (4.16)-(4.18), we see that the constraint forces enter into the algorithm as follows. In the first stage we have

$$\boldsymbol{r}_{\mathbf{a}}(t + \Delta t) = \boldsymbol{r'}_{\boldsymbol{a}}(t + \Delta t) + 1/2(\frac{\Delta t^2}{m}_a)\boldsymbol{g}_{\boldsymbol{a}}^{(r)}(t) \tag{4.29}$$

and

$$\boldsymbol{v}_{\mathbf{a}}(t + 1/2\Delta t) = \boldsymbol{v'}_{\boldsymbol{a}}(t + 1/2\Delta t) + 1/2(\frac{\Delta t}{m}_a)\boldsymbol{g}_{\boldsymbol{a}}^{(r)}(t) \tag{4.30}$$

The constraint forces $\mathbf{g_a^{(r)}}$, for this stage are directed along the bond vectors $\mathbf{r_{ab}}(t)$. They are determined by solving Eq. (4.29) by matrix inversion or iteratively as in ***SHAKE***. At the same time, the velocities $\mathbf{v'_a}$ at time $t + 1/2\Delta t$ obtained using Eq. (4.17), are adjusted according to Eq. (4.30). The second part of the algorithm follows evaluation of the non-constraint forces $\mathbf{f_a}(t+\Delta t)$, which are used in Eq. (4.18) to give $\mathbf{v'_a}(d+\Delta t)$. The second stage is:

$$\boldsymbol{v}_{\mathbf{a}}(t + \Delta t) = \boldsymbol{v'}_{\boldsymbol{a}}(t + \Delta t) + 1/2(\frac{\Delta t}{m}_a)\boldsymbol{g}_{\boldsymbol{a}}^{(r)}(t + \Delta t) \tag{4.31}$$

These constraint forces $\mathbf{g}_a^{(r)}(t+\Delta t)$, are directed along the bonds $\mathbf{r}_{ab}(t+\Delta t)$, and are chosen so that the velocities satisfy the constraints exactly at time $t+\Delta t$. Note that, in the next integration step, a different approximation to these same constraint forces, namely $\mathbf{g}_a^{(r)}(t+\Delta t)$, will be used. This step follows immediately. By analogy with ***SHAKE***, the iterative solution of these equations is termed '***RATTLE***'.

For flexible molecules, we have ample choice as to where to apply constraints, and it is generally believed that, while constraining bond lengths is worthwhile, it is best to leave bond angles (and certainly torsion angles) free to evolve under the influence of appropriate terms in the potential energy. This is partly on the grounds of program efficiency: the ***SHAKE*** algorithm iterations converge very slowly when rigid 'triangulated' molecular units are involved, often necessitating a reduced time step, which might as well be used in a proper integration of the bond 'wagging' motions instead. The other reason is that the relatively low frequencies of these motions makes the constraint approximation less valid. A model with a strong harmonic potential is different from one in which the potential is replaced by a rigid constraint. This point has been recognized for some time in the field of polymer dynamics, and has been tested by computer simulation. In practical terms, for a model of a protein molecule, the introduction of bond length constraints into a model based otherwise on realistic intramolecular potential functions has little effect on the structure and dynamics, but the further introduction of constraints on bond angles seriously affects the torsion angle distributions and the all-important conformational interconversion rates. This effect can be countered by adding an additional constraint potential $\mathcal{V}_c$, which involves the calculation of the metric determinant IHI.

$$\mathcal{V}_c = \tfrac{1}{2} k_B T \ln |\mathbf{H}| \tag{4.32}$$

where H is given by

$$|\mathbf{H}| = |\mathbf{G}|/|\mathbf{G}^s| \tag{4.33}$$

and G is defined as

$$G_{kl} = \sum_{i=1}^{N} \sum_{\alpha} m_i \frac{\partial r_{i\alpha}}{\partial q_k} \frac{\partial r_{i\alpha}}{\partial q_l} \tag{4.34}$$

and $\mathrm{G_s}$ is a submatrix of G and runs over all constraint coordinates s

$$G_{kl}^s = \sum_{i=1}^{N} \sum_{\alpha} m_i \frac{\partial r_{i\alpha}}{\partial q_k^s} \frac{\partial r_{i\alpha}}{\partial q_l^s} \tag{4.35}$$

This is time-consuming and algebraically complicated for all but the simplest flexible molecules, and the lesson seems to be that, for realistic molecular dynamics simulations, bond length constraints are permissible, but bond

angle constraints should not be introduced without examining their effects.
Two final points should be made, in relation to the calculation of thermodynamic properties of model systems incorporating constraints. The calculation of the total kinetic energy of such a system is a simple matter of summing the individual atomic contributions in the usual way. When using this quantity to estimate the temperature $\mathcal{T}$

$$\mathcal{T} = \frac{2\mathcal{K}}{(3N - N_c)k_B} = \frac{1}{(3N - N_c)k_B} \sum_{i=1}^{N} |\mathbf{p}_i|^2 / m_i \tag{4.36}$$

we must divide by the number of degrees of freedom. It should be clear from the specification of the molecular model how many independent constraints have been applied, and hence what the number of degrees of freedom is. Secondly, in molecular systems quantities such as the pressure may be calculated in several ways, the two most important of which focus on the component atoms, and on the molecular centres of mass, respectively. Consider the evaluation of the virial function $\mathcal{W}$

$$\begin{aligned} \mathcal{W} &= \tfrac{1}{3} \sum_i \sum_{j>i} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} = -\tfrac{1}{3} \sum_i \sum_{j>i} \mathbf{r}_{ij} \cdot \nabla_{\mathbf{r}_{ij}} v(\mathbf{r}_{ij}) \\ &= -\tfrac{1}{3} \sum_i \sum_{j>i} w(r_{ij}) \end{aligned} \tag{4.37}$$

interpreting the sum as being taken over all atom-atom separations $\mathbf{r}_{ab}$ and forces $\mathbf{f}_{ab}$. In this case, all intramolecular contributions to $\mathcal{W}$ including the constraint forces should be taken into account. Now consider the alternative interpretation of the virial

$$\begin{aligned} \mathcal{W} &= \tfrac{1}{3} \sum_i \sum_{j>i} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} = -\tfrac{1}{3} \sum_i \sum_{j>i} \mathbf{r}_{ij} \cdot (\nabla_{\mathbf{r}_{ij}} \mathcal{V})_{\Omega_i \Omega_j} \\ &= -\tfrac{1}{3} \sum_i \sum_{j>i} w(r_{ij}) \end{aligned} \tag{4.38}$$

in which we take the $\mathbf{f}_{ij}$ to represent the sum of all the forces acting on a molecule i, due to its interactions with molecule j, and take each such force to act at the centre of mass. In this case, all the intramolecular forces, including the constraint forces, cancel out and can be omitted from the sum. It is easy to show that, at equilibrium, the average pressure computed by either route is the same.

4.4 Checks on accuracy

Is our MD program working properly? This is the first question that must be asked when a simulation is run for the first time, and the answer is frequently in the negative.

Here, we discuss the tell-tale signs of a non-functioning MD program.
The first check must be that the conservation laws are properly obeyed, and in particular that the energy should be 'constant'. In fact small changes in the energy will occur (see Fig. 4.4). For a simple Lennard-Jones system, fluctuations of order 1 part in 10^5 – 10^4 are generally considered to be acceptable, although some workers are less demanding, and some more so. No systematic investigation of this point has been carried out. Energy fluctuations may be reduced by decreasing the time step. If one of the Verlet algorithms is being used, then the following suggestion may be useful. Several short runs should be undertaken, each starting from the same initial configuration and covering the same total time t_{run}: each run should employ a different time step Δt, and hence consist of a different number of steps $\tau_{run} = t_{run} / \Delta t$. The RMS energy fluctuations for each run should be calculated. If the program is functioning correctly, and other sources of energy fluctuations (such as potential truncation) have been eliminated, then the Verlet algorithm should give RMS energy fluctuations which are accurately proportional to Δt^2 (see Fig. 4.4). A good initial estimate of Δt is that it should be roughly an order of magnitude less than the highest vibrational period $t_C = 2\pi/\omega_C$.
A slow upward drift of energy may also be due to a time step that is too long, to potential truncation effects, or might indicate a program error. Effects with a 'physical' origin and those due to time step problems can be distinguished by the procedure outlined above, i.e. duplicating a short run but using a larger number of smaller time steps. If the drift as a function of simulation time is unchanged, then it is presumably connected with the system under study, whereas if it is substantially reduced, the method used to solve the equations of motion (possibly the size of the time step) is responsible. In the category of program error, we should mention the possibility that the wrong quantity is being calculated. If the total energy varies significantly but the simulation is 'stable' in the sense that no inexorable climb in energy occurs, then the way in which the energy is calculated should be examined. Are potential and kinetic contributions added together correctly? Is the pairwise force (appearing in the double loop) in fact correctly derived from the potential? This last possibility may be tested by including statements that calculate the force on a given particle numerically, from the potential energy, by displacing it slightly in each of the three coordinate directions. The result may be compared with the analytical formula encoded in the program. As emphasized earlier, although small fluctuations are permissible, it is essential to eliminate any traces of a drift in the total energy over periods of thousands of time steps, if the simulation is to probe the microcanonical ensemble correctly.
Rather than a slow drift, a very rapid, even catastrophic, increase in energy may occur within the first few time steps. There are two possibilities here: either a starting configuration with particle overlaps has been chosen (so that the intermolecular forces are unusually large) or there is a serious program error. The starting configuration may be tested simply by printing out the initial coordinates and inspecting the numbers. Alternatively, particularly when the number of particles is large, statements may temporarily be incorporated into the force loop so as to test each of the pair separations and print out particle coordinates and identifiers whenever a very close pair is detected.

Tracking down a serious program error may be a difficult task. It is a favourite mistake, particularly when reading in the potential parameters in real (e.g. SI) units to make a small, but disastrous, error in unit conversion. There is much to be said for testing out a program on a small number of particles before tackling the full-size system, but beware! Is the potential cutoff distance still smaller than half of the box length? Frequent program errors involve mismatching of number, length or type of variables passed between routines in COMMON blocks or in argument lists. Simple typographical errors, while hard to spot, may have far-reaching effects. It is hard to overemphasize how useful modern software development tools can be in locating and eliminating mistakes of this kind. A good editor may be used to check the source code much more efficiently than simple visual inspection. Many compilers produce compilation listings, which include a summary of the types and lengths of all variables used in each routine. Examining these listings is a good way to spot misspelt variables. On modern computers, excellent interactive debugging facilities exist, which allow the program to be run under user control, with constant monitoring of the program flow and the values of variables of interest. Needless to say, a program written in a simple, logical, and modular fashion will be easier to debug (and will contain fewer errors!) than one which has not been planned in this way.

For molecular simulations, errors may creep into the program more easily than in the simple atomic case. Energy should be conserved just as for atomic simulations, although, for small molecules, a rather short time step may be needed to achieve this, since rotational motion occurs so rapidly. If nonconservation is a problem, several points may need checking. Incorrectly differentiating the potential on the way to the torques may be a source of error: this is e.g. more complicated for potentials incorporating multipolar terms. Again, this may be tested numerically, by subjecting selected molecules to small rotations, and observing the change in potential energy. If the angular part of the motion is suspect, the rest of the program may be tested by 'freezing out' the rotation. This is accomplished by disengaging the rotational algorithm; physically this corresponds to giving the molecules an infinite moment of inertia and zero angular velocity. Energy should still be conserved under these conditions. Conversely, the angular motion may be tested out by omitting, temporarily, the translational algorithm, thus fixing the molecular centres at their initial positions.

Two final points should be made. When the program appears to be running correctly, the user should check that the monitored quantities are in fact evolving in time. Even conserved variables will fluctuate a little if only due to round-off errors, and any quantity that appears to be constant to 10significant figures should be regarded with suspicion: it is probably not being updated at all. Excellent conservation, but no science, will result from a program that does not, in fact, move the particles (due to some error associated with the predictor and corrector routines, for example). A time step that is too small (or that has been accidentally set to zero) will be very wasteful of computer time, and the extent to which Δt can be increased without prejudicing the stability of the simulation should be investigated. Finally, the problems discussed above are all 'mechanical' rather than 'thermodynamic', i.e. they are associated with

the correct solution of the equations of motion. The quite separate question of attaining thermodynamic equilibrium is far from trivial. If a well-known system is being simulated (e.g, Lennard-Jones, soft-sphere potentials etc.) then it is obviously sensible to compare the simulation output, when equilibrium has been attained, with the known thermodynamic properties.