8.4 ■ THE NUMERICAL ALGORITHM

Now that we have specified the interaction between the particles, we need to introduce a numerical method for computing the trajectory of each particle. As we have learned, the criteria for a good numerical integration method include that it conserves the phase-space volume and is consistent with the known conservation laws, is time reversible, and is accurate for relatively large time steps to reduce the CPU time needed for the total time of the simulation. These requirements mean that we should use a symplectic algorithm for the relatively long times of interest in molecular dynamics simulations. We adopt the commonly used second-order algorithm:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 \tag{8.4a}$$

$$v_{n+1} = v_n + \frac{1}{2}(a_{n+1} + a_n)\Delta t. \tag{8.4b}$$

To simplify the notation, we have written the algorithm for only one component of the particle's motion. The new position is used to find the new acceleration a_{n+1}, which is used together with a_n to obtain the new velocity v_{n+1}. The algorithm represented by (8.4) is known as the Verlet (or sometimes the velocity Verlet) algorithm (see Appendix 3A). We will use the `Verlet` implementation of the `ODESolver` interface to implement the algorithm. Thus, the x, v_x, y, and v_y values for the ith particle will be stored in the `state` array at `state[4*i]`, `state[4*i+1]`, `state[4*i+2]`, and `state[4*i+3]`, respectively.

8.5 ■ PERIODIC BOUNDARY CONDITIONS

A useful simulation must incorporate as many of the relevant features of the physical system of interest as possible. Usually we want to simulate a gas, liquid, or a solid in the bulk, that is, systems of at least $N \sim 10^{23}$ particles. In such systems the fraction of particles near the walls of the container is negligibly small. The number of particles that can be studied in a molecular dynamics simulation is typically 10^3–10^5, although we can simulate the order of 10^9 particles using clusters of computers. For these relatively small systems, the fraction of particles near the walls of the container is significant, and hence the behavior of such a system would be dominated by surface effects.

The most common way of minimizing surface effects and to simulate more closely the properties of a bulk system is to use what are known as *periodic boundary conditions*, although the *minimum image approximation* would be a more accurate name. This boundary condition is familiar to anyone who has played the Pacman computer game. Consider N particles that are constrained to move on a line of length L. The application of periodic boundary conditions is equivalent to considering the line to be a circle, and hence the maximum separation between any two particles is $L/2$ (see Figure 8.2). The generalization of periodic boundary conditions to two dimensions is equivalent to imagining a box with opposite edges joined so that the box becomes the surface of a torus (the shape of a doughnut or a bagel). The three-dimensional version of periodic boundary conditions cannot be visualized easily, but the same methods can be used.

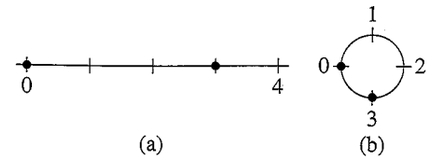

Figure 8.2 (a) Two particles at $x = 0$ and $x = 3$ on a line of length $L = 4$; the distance between the particles is 3. (b) The application of periodic boundary conditions for short range interactions is equivalent to thinking of the line as forming a circle of circumference L. In this case the minimum distance between the two particles is 1.

The implementation of periodic boundary conditions is straightforward. If a particle leaves the box by crossing a boundary in a particular direction, we add or subtract the length L of the box in that direction to the position. One simple way is to use an `if else` statement as shown:

Listing 8.1 Calculation of the position of particle in the central cell.

```
private double pbcPosition(double s, double L) {
   if(s > L) {
      s -= L;
   } else if(s < 0) {
      s += L;
   }
   return s;
}
```

To compute the minimum distance `ds` in a particular direction between two particles, we can use the method `pbcSeparation` (see Figure 8.2):

Listing 8.2 Calculation of the minimum separation.

```
private double pbcSeparation(double ds, double L) {
   if(ds > 0.5*L) {
      ds -= L;
   } else if(ds < -0.5*L) {
      ds += L;
   }
   return ds;
}
```

The equivalent static methods, `PBC.position` and `PBC.separation` in the Open Source Physics `numerics` package can also be used.

Exercise 8.2 Use of the % operator

(a) Another way to compute the position of a particle in the central cell is to use the `%` (modulus) operator. For example, `17 % 5` equals 2 because 17 divided by 5 leaves a remainder of 2. The `%` operator can also be used with floating point numbers. For example, `10.2 % 5 = 0.2`. Write a little test program to see how the `%` function works and determine the result of `10.2 % 3.3`, `-10.2 % 3.3`, `10.2 % -3.3`, and `-10.2 % -3.3`. In what way does `%` act like a remainder operator?