Note that the truncation error is again order $(\Delta t)^2$, but is four times bigger. We can eliminate this error to leading order by dividing (3.67) by 4 and subtracting it from (3.66):

$$f'(t) - \frac{1}{4}f'(t) = \frac{3}{4}f'(t) \approx D_1(\Delta t) - \frac{1}{4}D_1(2\Delta t),$$

or

$$f'(t) \approx \frac{4D_1(\Delta t) - D_1(2\Delta t)}{3}. \tag{3.68}$$

It is easy to show that the error for $f'(t)$ is order $(\Delta t)^4$. Recursive difference formulas for derivatives can be obtained by canceling the truncation error at each order. This method is called *Richardson extrapolation.*

Another class of algorithms are *predictor-corrector* algorithms. The idea is to first *predict* the value of the new position:

$$x_p = x_{n-1} + 2v_n\Delta t \quad \text{(predictor)}. \tag{3.69}$$

The predicted value of the position allows us to predict the acceleration a_p. Then using a_p, we obtain the *corrected* values of v_{n+1} and x_{n+1}:

$$v_{n+1} = v_n + \frac{1}{2}(a_p + a_n)\Delta t \tag{3.70a}$$

$$x_{n+1} = x_n + \frac{1}{2}(v_{n+1} + v_n)\Delta t \quad \text{(corrected)}. \tag{3.70b}$$

The corrected values of x_{n+1} and v_{n+1} are used to obtain a new predicted value of a_{n+1} and, hence, a new predicted value of v_{n+1} and x_{n+1}. This process is repeated until the predicted and corrected values of x_{n+1} differ by less than the desired value.

Note that the predictor-corrector algorithm is not self-starting. The predictor-corrector algorithm gives more accurate positions and velocities than the leapfrog algorithm and is suitable for very accurate calculations. However, it is computationally expensive, needs significant storage (the forces at the last two stages and the coordinates and velocities at the last step), and becomes unstable for large time steps.

As we have emphasized, there is no single algorithm for solving Newton's equations of motion that is superior under all conditions. It is usually a good idea to start with a simple algorithm and then to try a higher order algorithm to see if any real improvement is obtained.

We now discuss an important class of algorithms, known as *symplectic* algorithms, which are particularly suitable for doing long time simulations of Newton's equations of motion when the force is only a function of position. The basic idea of these algorithms derives from the Hamiltonian theory of classical mechanics. We first give some basic results needed from this theory to understand the importance of symplectic algorithms.

In Hamiltonian theory the generalized coordinates q_i and momenta p_i take the place of the usual positions and velocities familiar from Newtonian theory. The index i labels both a particle and a component of the motion. For example, in a two-particle system in two dimensions, i would run from 1 to 4. The Hamiltonian (which for our purposes can be thought of as the total energy) is written as

$$H(q_i, p_i) = T + V, \tag{3.71}$$

where T is the kinetic energy and V is the potential energy. Hamilton's theory is most relevant for nondissipative systems, which we consider here. For example, for a two-particle system in two dimensions connected by a spring, H would take the form

$$H = \frac{p_1^2}{2m} + \frac{p_2^2}{2m} + \frac{p_3^2}{2m} + \frac{p_4^2}{2m} + \frac{1}{2}k(q_1 - q_3)^2 + \frac{1}{2}k(q_2 - q_4)^2, \tag{3.72}$$

where if the particles are labeled as A and B, we have $p_1 = p_{x,A}$, $p_2 = p_{y,A}$, $p_3 = p_{x,B}$, $p_4 = p_{y,B}$, and similarly for the q_i. The equations of motion are written as first-order differential equations known as Hamilton's equations:

$$\dot{p}_i = -\frac{\partial H}{\partial q_i} \tag{3.73a}$$

$$\dot{q}_i = \frac{\partial H}{\partial p_i}, \tag{3.73b}$$

which are equivalent to Newton's second law and an equation relating the velocity to the momentum. The beauty of Hamiltonian theory is that these equations are correct for other coordinate systems such as polar coordinates, and they also describe rotating systems where the momenta become angular momenta, and the position coordinates become angles.

Because the coordinates and momenta are treated on an equal footing, we can consider the properties of flow in phase space where the dimension of phase space includes both the coordinates and momenta. Thus, one particle moving in one dimension corresponds to a two-dimensional phase space. If we imagine a collection of initial conditions in phase space forming a volume in phase space, then one of the results of Hamiltonian theory is that this volume does not change as the system evolves. A slightly different result, called the *symplectic* property, is that the sum of the areas formed by the projection of the phase space volume onto the planes q_i, p_i for each pair of coordinates and momenta also does not change with time. Numerical algorithms that have this property are called symplectic algorithms. These algorithms are built from the following two statements which are repeated M times for each time step:

$$p_i^{(k+1)} = p_i^{(k)} + a_k F_i^{(k)}\delta t \tag{3.74a}$$

$$q_i^{(k+1)} = q_i^{(k)} + b_k p_i^{(k+1)}\delta t, \tag{3.74b}$$

where $F_i^{(k)} \equiv -\partial V(q_i^{(k)})/\partial q_i^{(k)}$. The label k runs from 0 to $M-1$ and one time step is given by $\Delta t = M\delta t$. (We will see that δt is the time step of an intermediate calculation that is made during the time step Δt.) Note that in the update for q_i, the already updated p_i is used. For simplicity, we assume that the mass is unity.

Different algorithms correspond to different values of M, a_k, and b_k. For example, $a_0 = b_0 = M = 1$ corresponds to the Euler–Cromer algorithm, and $M = 2$, $a_0 = a_1 = 1$, $b_0 = 2$, and $b_1 = 0$ is equivalent to the Verlet algorithm as we will now show. If we substitute in the appropriate values for a_k and b_k into (3.74), we have

$$p_i^{(1)} = p_i^{(0)} + F_i^{(0)}\delta t \tag{3.75a}$$

$$q_i^{(1)} = q_i^{(0)} + 2p_i^{(1)}\delta t \tag{3.75b}$$

$$p_i^{(2)} = p_i^{(1)} + F_i^{(1)}\delta t \tag{3.75c}$$

$$q_i^{(2)} = q_i^{(1)}. \tag{3.75d}$$