Finally, we use (3.48a) for x_{n+1} to eliminate $x_{n+1} - x_n$ from (3.50); after some algebra we obtain the desired result (3.48b).

Another useful algorithm that avoids the roundoff errors of the leapfrog algorithm is due to Beeman and Schofield. We write the *Beeman* algorithm in the form

$$x_{n+1} = x_n + v_n\Delta t + \frac{1}{6}(4a_n - a_{n-1})(\Delta t)^2 \tag{3.51a}$$

$$v_{n+1} = v_n + \frac{1}{6}(2a_{n+1} + 5a_n - a_{n-1})\Delta t \quad \text{(Beeman algorithm)}. \tag{3.51b}$$

Note that $v_{n+1} = v_n + \frac{1}{12}(5a_{n+1} + 8a_n - a_{n-1})\Delta t$ agrees to v_{n+1} to $O(\Delta t^3)$
Why not?
Then also if
$x_{n+1} = x_n + v_n\Delta t + \frac{1}{24}(3a_{n+1} + 5a_n - a_{n-1})\Delta t^2$
agrees to $O(\Delta t^4)$

Note that (3.51) does not calculate particle trajectories more accurately than the Verlet algorithm. Its advantage is that it generally does a better job of maintaining energy conservation. However, the Beeman algorithm is not self-starting.

The most common finite difference method for solving ordinary differential equations is the *Runge–Kutta* method. To explain the many algorithms based on this method, we consider the solution of the first-order differential equation

$$\frac{dx}{dt} = f(x, t). \tag{3.52}$$

Runge–Kutta algorithms evaluate the rate $f(x, t)$ multiple times in the interval $[t, t + \Delta t]$. For example, the classic fourth-order Runge–Kutta algorithm, which we will discuss in the following, evaluates $f(x, t)$ at four times t_n, $t_n + a_1\Delta t$, $t_n + a_2\Delta t$, and $t_n + a_3\Delta t$. Each evaluation of $f(x, t)$ produces a slightly different rate r_1, r_2, r_3, and r_4. The idea is to advance the solution using a weighted average of the intermediate rates:

$$\cancel{x}_{n+1} = \cancel{x}_n + (c_1r_1 + c_2r_2 + c_3r_3 + c_4r_4)\Delta t. \tag{3.53}$$

The various Runge–Kutta algorithms correspond to different choices for the constants a_i and c_i. These algorithms are classified by the number of intermediate rates $\{r_i, i = 1, \ldots, N\}$. The determination of the Runge–Kutta coefficients is difficult for all but the lowest order methods, because the coefficients must be chosen to cancel as many terms in the Taylor series expansion of $f(x, t)$ as possible. The first nonzero expansion coefficient determines the order of the Runge–Kutta algorithm. Fortunately, these coefficients are tabulated in most numerical analysis books.

To illustrate how the various sets of Runge–Kutta constants arise, consider the case $N = 2$. The second-order Runge–Kutta solution of (3.52) can be written using standard notation as

$$x_{n+1} = x_n + k_2 + O\big((\Delta t)^3\big), \tag{3.54a}$$

where

$$k_2 = f(x_n + k_1/2, t_n + \Delta t/2)\Delta t \tag{3.54b}$$

$$k_1 = f(x_n, t_n)\Delta t. \tag{3.54c}$$

Note that the weighted average uses $c_1 = 0$ and $c_2 = 1$. The interpretation of (3.54) is as follows. The Euler algorithm assumes that the slope $f(x_n, t_n)$ at (x_n, t_n) can be used to extrapolate to the next step, that is, $x_{n+1} = x_n + f(x_n, t_n)\Delta t$. A plausible way of making a more accurate determination of the slope is to use the Euler algorithm to extrapolate to the midpoint of the interval and then to use the midpoint slope across the full width of the interval. Hence, the Runge–Kutta estimate for the rate is $f(x^*, t_n + \frac{1}{2}\Delta t)$, where $x^* = x_n + \frac{1}{2}f(x_n, t_n)\Delta t$ (see (3.54c)).

The application of the second-order Runge–Kutta algorithm to Newton's equation of motion (3.31) yields

$$k_{1v} = a_n(x_n, v_n, t_n)\Delta t \tag{3.55a}$$

$$k_{1x} = v_n\Delta t \tag{3.55b}$$

$$k_{2v} = a(x_n + k_{1x}/2, v_n + k_{1v}/2, t + \Delta t/2)\Delta t \tag{3.55c}$$

$$k_{2x} = (v_n + k_{1v}/2)\Delta t, \tag{3.55d}$$

and

$$v_{n+1} = v_n + k_{2v} \tag{3.56a}$$

$$x_{n+1} = x_n + k_{2x} \quad \text{(second-order Runge–Kutta)}. \tag{3.56b}$$

Note that the second-order Runge–Kutta algorithm in (3.55) and (3.56) is identical to the Euler–Richardson algorithm.

Other second-order Runge–Kutta type algorithms exist. For example, if we set $c_1 = c_2 = \frac{1}{2}$ we obtain the endpoint method:

$$y_{n+1} = y_n + \frac{1}{2}k_1 + \frac{1}{2}k_2, \tag{3.57a}$$

where

$$k_1 = f(x_n, t_n)\Delta t \tag{3.57b}$$

$$k_2 = f(x_n + k_1, t_n + \Delta t)\Delta t. \tag{3.57c}$$

And if we set $c_1 = \frac{1}{3}$ and $c_2 = \frac{2}{3}$, we obtain Ralston's method:

$$y_{n+1} = y_n + \frac{1}{3}k_1 + \frac{2}{3}k_2, \tag{3.58a}$$

where

$$k_1 = f(x_n, t_n)\Delta t \tag{3.58b}$$

$$k_2 = f(x_n + 3k_1/4, t_n + 3\Delta t/4)\Delta t. \tag{3.58c}$$

Note that Ralston's method does not calculate the rate at uniformly spaced subintervals of Δt. In general, a Runge–Kutta method adjusts the partition of Δt as well as the constants a_i and c_i so as to minimize the error.

In the *fourth-order* Runge–Kutta algorithm, the derivative is computed at the beginning of the time interval, in two different ways at the middle of the interval, and again at the end of the interval. The two estimates of the derivative at the middle of the interval are given twice the weight of the other two estimates. The algorithm for the solution of (3.52) can be