A higher order algorithm whose error is bounded is the *half-step* algorithm. In this algorithm, the average velocity during an interval is taken to be the velocity in the middle of the interval. The half-step algorithm can be written as

$$v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + a_n \Delta t \tag{3.37a}$$

$$x_{n+1} = x_n + v_{n+\frac{1}{2}} \Delta t \quad \text{(half-step algorithm)}. \tag{3.37b}$$

Note that the half-step algorithm is not self-starting, that is, (3.37a) does not allow us to calculate $v_{\frac{1}{2}}$. This problem can be overcome by adopting the Euler algorithm for the first half step:

$$v_{\frac{1}{2}} = v_0 + \frac{1}{2} a_0 \,\Delta t. \tag{3.37c}$$

Because the half-step algorithm is stable, it is a common textbook algorithm. The Euler–Richardson algorithm can be motivated as follows. We first write $x(t + \Delta t)$ as

$$x_1 \approx x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2} a(t)(\Delta t)^2. \tag{3.38}$$

The notation x_1 implies that $x(t + \Delta t)$ is related to $x(t)$ by one time step. We also may divide the step Δt into half steps and write the first half step, $x(t + \frac{1}{2}\Delta t)$, as

$$x(t + \Delta t/2) \approx x(t) + v(t)\frac{\Delta t}{2} + \frac{1}{2} a(t)(\Delta t/2)^2. \tag{3.39}$$

The second half step, $x_2(t + \Delta t)$, may be written as

$$x_2(t + \Delta t) \approx x(t + \Delta t/2) + v(t + \Delta t/2)\frac{\Delta t}{2} + \frac{1}{2} a(t + \Delta t/2)(\Delta t/2)^2. \tag{3.40}$$

We substitute (3.39) into (3.40) and obtain

$$x_2(t + \Delta t) \approx x(t) + \frac{1}{2}[v(t) + v(t + \Delta t/2)]\Delta t + \frac{1}{2}[a(t) + a(t + \Delta t/2)](\Delta t/2)^2. \tag{3.41}$$

Now $a(t + \frac{1}{2}\Delta t) \approx a(t) + \frac{1}{2}a'(t)\Delta t$. Hence to order $(\Delta t)^2$, (3.41) becomes

$$x_2(t + \Delta t) = x(t) + \frac{1}{2}[v(t) + v(t + \Delta t/2)]\Delta t + \frac{1}{2}[2a(t)](\Delta t/2)^2. \tag{3.42}$$

We can find an approximation that is accurate to order $(\Delta t)^3$ by combining (3.38) and (3.42) so that the terms to order $(\Delta t)^2$ cancel. The combination that works is $2x_2 - x_1$, which gives the Euler–Richardson result:

$$x_{\text{er}}(t + \Delta t) \equiv 2x_2(t + \Delta t) - x_1(t + \Delta t) = x(t) + v(t + \Delta t/2)\Delta t + O(\Delta t)^3. \tag{3.43}$$

The same reasoning leads to an approximation for the velocity accurate to $(\Delta t)^3$ giving

$$v_{\text{er}} \equiv 2v_2(t + \Delta t) - v_1(t + \Delta t) = v(t) + a(t + \Delta t/2)\Delta t + O(\Delta t)^3. \tag{3.44}$$

A bonus of the Euler–Richardson algorithm is that the quantities $|x_2 - x_1|$ and $|v_2 - v_1|$ give an estimate for the error. We can use these estimates to change the time step so that the error is always within some desired level of precision. We will see that the Euler–Richardson algorithm is equivalent to the second-order Runge–Kutta algorithm (see (3.54)).

One of the most common drift-free higher order algorithms is commonly attributed to Verlet. We write the Taylor series expansion for x_{n-1} in a form similar to (3.32b):

$$x_{n-1} = x_n - v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2. \tag{3.45}$$

If we add the forward and reverse forms, (3.32b) and (3.45) respectively, we obtain

$$x_{n+1} + x_{n-1} = 2x_n + a_n (\Delta t)^2 + O\big((\Delta t)^4\big), \tag{3.46}$$

or

$$x_{n+1} = 2x_n - x_{n-1} + a_n (\Delta t)^2 \quad \text{(leapfrog algorithm)}. \tag{3.47a}$$

Similarly, the subtraction of the Taylor series for x_{n+1} and x_{n-1} yields

$$v_n = \frac{x_{n+1} - x_{n-1}}{2\Delta t} \quad \text{(leapfrog algorithm)}. \tag{3.47b}$$

Note that the global error associated with the leapfrog algorithm (3.47) is third-order for the position and second-order for the velocity. However, the velocity plays no part in the integration of the equations of motion. The leapfrog algorithm is also known as the explicit central difference algorithm. Because this form of the leapfrog algorithm is not self-starting, another algorithm must be used to obtain the first few terms. An additional problem is that the new velocity (3.47b) is found by computing the difference between two quantities of the same order of magnitude. Such an operation results in a loss of numerical precision and may give rise to roundoff errors.

A mathematically equivalent version of the leapfrog algorithm is given by

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 \tag{3.48a}$$

$$v_{n+1} = v_n + \frac{1}{2}(a_{n+1} + a_n)\Delta t \quad \text{(velocity Verlet algorithm)}. \tag{3.48b}$$

We see that (3.48), known as the *velocity* form of the Verlet algorithm, is self-starting and minimizes roundoff errors. Because we will not use (3.47) in the text, we will refer to (3.48) as the Verlet algorithm.

We can derive (3.48) from (3.47) by the following considerations. We first solve (3.47b) for x_{n-1} and write $x_{n-1} = x_{n+1} - 2v_n \Delta t$. If we substitute this expression for x_{n-1} into (3.47a) and solve for x_{n+1}, we find the form (3.48a). Then we use (3.47b) to write v_{n+1} as

$$v_{n+1} = \frac{x_{n+2} - x_n}{2\Delta t}, \tag{3.49}$$

and use (3.47a) to obtain $x_{n+2} = 2x_{n+1} - x_n + a_{n+1}(\Delta t)^2$. If we substitute this form for x_{n+2} into (3.49), we obtain

$$v_{n+1} = \frac{x_{n+1} - x_n}{\Delta t} + \frac{1}{2} a_{n+1} \Delta t. \tag{3.50}$$