written in standard notation as

$$k_1 = f(x_n, t_n)\Delta t \tag{3.59a}$$
$$k_2 = f(x_n + k_1/2, t_n + \Delta t/2)\Delta t \tag{3.59b}$$
$$k_3 = f(x_n + k_2/2, t_n + \Delta t/2)\Delta t \tag{3.59c}$$
$$k_4 = f\ (x_n + k_3, t_n + \Delta t)\ \Delta t, \tag{3.59d}$$

and

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{3.60}$$

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

$$k_{1v} = a(x_n, v_n, t_n)\Delta t \tag{3.61a}$$
$$k_{1x} = v_n \Delta t \tag{3.61b}$$
$$k_{2v} = a(x_n + k_{1x}/2, v_n + k_{1v}/2, t_n + \Delta t/2)\Delta t \tag{3.61c}$$
$$k_{2x} = (v_n + k_{1v}/2)\Delta t \tag{3.61d}$$
$$k_{3v} = a(x_n + k_{2x}/2, v_n + k_{2v}/2, t_n + \Delta t/2)\Delta t \tag{3.61e}$$
$$k_{3x} = (v_n + k_{2v}/2)\Delta t \tag{3.61f}$$
$$k_{4v} = a(x_n + k_{3x}, v_n + k_{3v}, t + \Delta t) \tag{3.61g}$$
$$k_{4x} = (v_n + k_{3x})\Delta t, \tag{3.61h}$$

and

$$v_{n+1} = v_n + \frac{1}{6}(k_{1v} + 2k_{2v} + 2k_{3v} + k_{4v}) \tag{3.62a}$$
$$x_{n+1} = x_n + \frac{1}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}) \quad \text{(fourth-order Runge–Kutta)}. \tag{3.62b}$$

Because Runge–Kutta algorithms are self-starting, they are frequently used to obtain the first few iterations for an algorithm that is not self-starting.

As we have discussed, one way to determine the accuracy of a solution is to calculate it twice with two different values of the time step. One way to make this comparison is to choose time steps Δt and $\Delta t/2$ and compare the solution at the desired time. If the difference is small, the error is assumed to be small. This estimate of the error can be used to adjust the value of the time step. If the error is too large, than the time step can be halved. And if the error is much less than the desired value, the time step can be increased so that the program runs faster.

A better way of controlling the step size was developed by Fehlberg who showed that it is possible to evaluate the rate in such a way as to simultaneously obtain two Runge–Kutta approximations with different orders. For example, it is possible to run a fourth-order and

fifth-order algorithm in tandem by evaluating five rates. We thus obtain different estimates of the true solution using different weighed averages of these rates:

$$y_{n+1} = y_n + c_1k_1 + c_2k_2 + c_3k_3 + c_4k_4 + c_5k_5 \tag{3.63a}$$
$$y^*_{n+1} = y_n + c^*_1k_1 + c^*_2k_2 + c^*_3k_3 + c^*_4k_4. \tag{3.63b}$$

Because we can assume that the fifth-order solution is closer to the true solution than the fourth-order algorithm, the difference $|y - y^*|$ provides a good estimate of the error of the fourth-order method. If this estimated error is larger than the desired tolerance, then the step size is decreased. If the error is smaller than the desired tolerance, the step size is increased. The `RK45` ODE solver in the numerics package implements this technique for choosing the optimal step size.

In applications where the accuracy of the numerical solution is important, adaptive time step algorithms should always be used. As stated in *Numerical Recipes*, "Many small steps should tiptoe through treacherous terrain, while a few great strides should speed through uninteresting countryside. The resulting gains in efficiency are not mere tens of percents or factors of two; they can sometimes be factors of ten, a hundred, or more."

Adaptive step size algorithms are not well suited for tabulating functions or for simulation because the intervals between data points are not constant. An easy way to circumvent this problem is to use a method that takes multiple adaptive steps while checking to insure that the last step does not overshoot the desired fixed step size. The `ODEMultistepSolver` implements this technique. The solver acts like a fixed step size solver, even though the solver monitors its internal step size so as to achieve the desired accuracy.

It is also possible to combine the results from a calculation using two different values of the time step to yield a more accurate expression. Consider the Taylor series expansion of $f(t + \Delta t)$ about t:

$$f(t + \Delta t) = f(t) + f'(t)\Delta t + \frac{1}{2!}f''(t)(\Delta t)^2 + \cdots. \tag{3.64}$$

Similarly, we have

$$f(t - \Delta t) = f(t) - f'(t)\Delta t + \frac{1}{2!}f''(t)(\Delta t)^2 + \cdots. \tag{3.65}$$

We subtract (3.65) from (3.64) to find the usual central difference approximation for the derivative

$$f'(t) \approx D_1(\Delta t) = \frac{f(t + \Delta t) - f(t - \Delta t)}{2\Delta t} - \frac{(\Delta t)^2}{6}f'''(t). \tag{3.66}$$

The truncation error is order $(\Delta t)^2$. Next consider the same relation but for a time step that is twice as large:

$$f'(t) \approx D_1(2\Delta t) = \frac{f(t + 2\Delta t) - f(t - 2\Delta t)}{4\Delta t} - \frac{4(\Delta t)^2}{6}f'''(t). \tag{3.67}$$