Numerical Integration Methods for Differential Equation

Hai-Qing Lin

CSRC &

Department of Physics
Chinese University of Hong Kong

Numerical Integration of First-Order Differential Equation

$$\frac{dx}{dt} = f\left(x,t\right)$$

- The Euler method (1st order)
- The Runge-Kutta method (2nd order)
- The Runge-Kutta method (4th order)

The Euler Method (1st Order)

$$t_{n+1} = t_n + \Delta t$$

$$x_{n+1} = x_n + f(x_n, t_n)\,\Delta t$$

The Second-order Runge-Kutta Method

$$k_1 = f(x_n, t_n)\,\Delta t$$

$$k_2 = f(x_n + k_1/2,\ t_n + \Delta t/2)\,\Delta t$$

$$x_{n+1} = x_n + k_2 + O((\Delta t)^3)$$

The Fourth-order Runge-Kutta Method

$$k_1 = f(x_n, t_n)\, \Delta t$$

$$k_2 = f(x_n + k_1/2,\, t_n + \Delta t/2)\, \Delta t$$

$$k_3 = f(x_n + k_2/2,\, t_n + \Delta t/2)\, \Delta t$$

$$k_4 = f(x_n + k_3,\, t_n + \Delta t)\, \Delta t$$

$$x_{n+1} = x_n + (k_1 + 2k_2 + 2k_3 + k_4)/6 + O((\Delta t)^5)$$

Two different ways at the middle of the interval.

The Runge-Kutta Method

- The most common finite difference method for solving ordinary differential equations.
- Basically, it evaluates $f(x,t)$ multiple times in the interval $[t, t+\Delta t]$, then update x using a weighted average of the intermediate rates k_i: $x_{n+1} = x_n + \Sigma\, c_i\, k_i$. e.g., 2nd order, $c_1=0$, $c_2=1$; 4th order $x_{n+1} = x_n + (k_1 + 2k_2 + 2k_3 + k_4)/6$
- The R-K coefficients were chosen to cancel as many terms in the Taylor series expansion of $f(x,t)$ as possible.

Numerical Integration of Second-Order Differential Equation

$$\frac{dx}{dt} = v(t); \quad \frac{dv}{dt} = a(t)$$

- The Euler method (1st order)
- The Mid-point method (2nd order for x)
- The Half-step method (leap-frog, 2nd order)
- The Euler-Richardson method (2nd order)
- The Runge-Kutta method (2nd order)
- The Verlet (velocity) method (3rd order)
- The Runge-Kutta method (4th order)

The Euler Method (1st Order)

$$t_{n+1} = t_n + \Delta t$$

Euler

$$v_{n+1} = v_n + a_n \Delta t$$

$$x_{n+1} = x_n + v_n \Delta t$$

Euler-Cromer (could be 2nd order)

$$x_{n+1} = x_n + v_{n+1} \Delta t \qquad v_{n+1} = v_n + a_n \Delta t$$

$$x_{n+1} = x_n + v_n \Delta t \qquad v_{n+1} = v_n + a_{n+1} \Delta t$$

$$x_{n+1} = x_n + v_{n+1} \Delta t \qquad v_{n+1} = v_n + a_{n+1} \Delta t$$

The Mid-Point Method (2nd Order for x)

$$v_{n+1} = v_n + a_n \, \Delta t$$

$$\begin{aligned} x_{n+1} &= x_n + (v_n + v_{n+1}) \, \Delta t / 2 \\ &= x_n + v_n \Delta t + a_n \, \Delta t^2 / 2 \end{aligned}$$

Exact for constant acceleration?

It could be unstable as the Euler algorithm.

One may modify it to $v_{n+1} = v_n + a_{n+1} \, \Delta t$, etc.

The Half-Step Method (Leap-frog)

$$v_{1/2} = v_0 + a_0\,\Delta t / 2$$

$$v_{n+1/2} = v_{n-1/2} + a_n\,\Delta t$$

$$x_{n+1} = x_n + v_{n+1/2}\,\Delta t$$

This algorithm is stable, so it is a common textbook algorithm.

The Euler-Richardson Method (2nd Order)

$$a_n = F(x_n, v_n, t_n) / m$$

$$t_m = t_n + \Delta t / 2$$

$$v_m = v_n + a_n \Delta t / 2$$

$$x_m = x_n + v_n \Delta t / 2$$

$$a_m = F(x_m, v_m, t_m) / m$$

$$v_{n+1} = v_n + a_m \Delta t$$

$$x_{n+1} = x_n + v_m \Delta t$$

The Runge-Kutta Method (2nd Order)

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

$$k_{1x} = v_n\,\Delta t$$

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

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

$$v_{n+1} = v_n + k_{2v}$$

$$x_{n+1} = x_n + k_{2x}$$

The Runge-Kutta Method (4th Order)

$k_{1v} = a\,(x_n, v_n, t_n)\,\Delta t\,;$ $\quad k_{1x} = v_n\,\Delta t$

$k_{2v} = a\,(x_n+k_{1x}/2, v_n+k_{1v}/2, t_n+\Delta t/2)\,\Delta t\,;$ $\quad k_{2x} = (v_n+k_{1v}/2)\,\Delta t$

$k_{3v} = a\,(x_n+k_{2x}/2, v_n+k_{2v}/2, t_n+\Delta t/2)\,\Delta t\,;$ $\quad k_{3x} = (v_n+k_{2v}/2)\,\Delta t$

$k_{4v} = a\,(x_n+k_{3x}, v_n+k_{3v}, t_n+\Delta t)\,\Delta t\,;$ $\quad k_{4x} = (v_n+k_{3v})\,\Delta t$

$$v_{n+1} = v_n + (k_{1v} + 2k_{2v} + 2k_{3v} + k_{4v})/6$$

$$x_{n+1} = x_n + (k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x})/6$$

The Verlet (Velocity) Method (3rd Order)

$$x_{n+1} = x_n + v_n \Delta t + a_n(\Delta t)^2 / 2$$

$$v_{n+1} = v_n + (a_{n+1} + a_n)\, \Delta t / 2$$

It is self-starting & minimizes round-off errors

More on derivations and application later.

Symplectic Algorithms

Gray et al., J. Chem. Phys. 101, 4062-4072(1994).

- The basic idea of them derives from the Hamiltonian theory of classical mechanics, where coordinates and momenta are treated on an equal footing. The algorithm, $\Delta t = M\delta t$

  $p^{k+1}_i = p^k_i + a_k F^k_i \delta t,\ q^{k+1}_i = q^k_i + b_k p^k_i \delta t$

  where $F^k_i = \Delta V(q^k_i)/\Delta q^k_i,\ k=0,\ \ldots,\ M\text{-}1.$
- Different algorithms correspond to different values of M, a_k, and b_k. Examples:
  - $M=1,\ a_0=b_0=1,$ *the Euler-Cromer*
  - $M=2,\ a_0=a_1=1,\ b_0=2,\ b_1=0,$ *the Verlet*
- These algorithms are frequently more accurate and stable than nonsymplectic algorithms.

Accuracy of Algorithms

- A common way to determine the accuracy of a solution is to calculate it twice with Δt and $\Delta t\,/2$, and adjust Δt from the error.
- It is possible to combine the results from a calculation using two different values of Δt to yield a more accurate expression.
- There is no single algorithm for a given problem is superior under all conditions. It is usually a good idea to start with a simple one and then try a higher order algorithm.