4. Runge-Kutta Formula For Differential Equations

To solve the differential equations numerically, the most useful formula is called Runge-Kutta formula which has been widely used in numerical analysis.

For a dynamic system without input, it is generally expressed as the following first-order differential equation:

$$\dot{x}(t) = f(t, x(t)), \qquad x(t_0)=\alpha \tag{4-1}$$

where $t=t_0$ is the initial time and $x(t_0)=\alpha$ is the initial condition. The problem to solve $x(t)$ in (4-1) for $t>t_0$ is called the initial value problem, or IVP in brief. For example, the following equation

$$\dot{x}(t) = -x(t) + t\,, \qquad x(0)=1 \tag{4-2}$$

is an IVP and its solution can be obtained in closed form as below:

$$x(t) = 2e^{-t} + t - 1, \qquad t\geq 0 \tag{4-3}$$

However, if (4-1) is more complicated, such as

$$\dot{x}(t) = -x^2(t) + t \cdot sin(t), \qquad x(0)=1 \tag{4-4}$$

then it is impossible to find the solution $x(t)$ in closed form. Hence, it is required to solve (4-4) in a numerical method.

The simplest numerical method is called the Euler formula, which was propsed by Euler in 1768. With the use of fixed grid size $\Delta t=h$, the grid points along t are denoted as t_0, t_1, t_2,… in order, where

$$t_i = t_0 + i \cdot h\,, \qquad i=1,2,3,\ldots\ldots \tag{4-5}$$

and the value $x(t)$ at t_i is defined as

$$x_i = x(t_i), \qquad i=0,1,2,3,\ldots\ldots \tag{4-6}$$

where the initial value $x_0=x(t_0)=\alpha$ is known. According to the definition of derivative, we have

$$\dot{x}(t) = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t}, \tag{4-7}$$

which implies the derivative of $x(t)$ at $t=t_i$ can be approximately expressed as

$$\dot{x}(t_i) \approx \frac{x(t_i + \Delta t) - x(t_i)}{\Delta t} = \frac{x(t_{i+1}) - x(t_i)}{h} = \frac{x_{i+1} - x_i}{h} \tag{4-8}$$

Substituting (4-8) into (4-1) at $t=t_i$ leads to

$$\frac{x_{i+1}-x_i}{h} \approx f(t_i, x_i), \qquad x_0=\alpha \tag{4-9}$$

and then

$$x_{i+1} = x_i + h \cdot f(t_i, x_i), \qquad x_0=\alpha \tag{4-10}$$

which is the famous Euler formula. Clearly, if we try to achieve a solution more precisely, the grid size *h* should be chosen as small as possible. However, reducing the grid size *h* is inefficient since the cost of calculation time may increase tremendously.

Now, let's introduce Taylor's expansion to explain the error caused by Euler formula (4-10). According to Taylor's expansion, a function $x(t)$ continuous at $t=t_i$ can be expressed as

$$x(t) = a_0 + a_1(t-t_i) + a_2(t-t_i)^2 + \cdots + a_n(t-t_i)^n + \cdots \tag{4-11}$$

whose higher order derivatives are

$$\dot{x}(t) = a_1 + 2a_2(t-t_i) + 3a_3(t-t_i)^2 + \cdots + na_n(t-t_i)^{n-1} + \cdots \tag{4-12}$$

$$\ddot{x}(t) = 2!\,a_2 + 3\cdot 2a_3(t-t_i) + 4\cdot 3a_4(t-t_i)^2 + \cdots + \cdots + n(n-1)a_n(t-t_i)^{n-2} + \cdots \tag{4-13}$$

$$\dddot{x}(t) = 3!\,a_3 + 4\cdot 3\cdot 2a_4(t-t_i) + 5\cdot 4\cdot 3a_5(t-t_i)^2 + \cdots + \cdots + n(n-1)(n-2)a_n(t-t_i)^{n-3} + \cdots \tag{4-14}$$

From the above equations, we have $a_0 = x(t_i)$ and $a_k = \frac{1}{k!}x^{(k)}(t_i)$ for k=1,2,…,∞. Thus, (4-11) can be rewritten as

$$x(t) = x(t_i) + \dot{x}(t_i)(t-t_i) + \frac{\ddot{x}(t_i)}{2!}(t-t_i)^2 + \cdots + \frac{x^{(n)}(t_i)}{n!}(t-t_i)^n + \cdots \tag{4-15}$$

Let $t=t_{i+1}$, we have

$$x(t_{i+1}) = x(t_i) + \dot{x}(t_i)(t_{i+1}-t_i) + \frac{\ddot{x}(t_i)}{2!}(t_{i+1}-t_i)^2 + \cdots + \cdots + \frac{x^{(n)}(t_i)}{n!}(t_{i+1}-t_i)^n + \cdots \tag{4-16}$$

i.e.,

$$\begin{aligned} x_{i+1} &= x_i + f(t_i, x_i)h + \frac{f'(t_i, x_i)}{2!}h^2 + \cdots + \frac{f^{(n-1)}(t_i, x_i)}{n!}h^n + \cdots \\ &= x_i + f(t_i, x_i)h + O(h^2) \end{aligned} \tag{4-17}$$

where

$$O(h^2) = \frac{f'(t_i, x_i)}{2!}h^2 + \cdots + \frac{f^{(n-1)}(t_i, x_i)}{n!}h^n + \cdots \tag{4-18}$$

Comparing (4-17) with (4-10), we know that the term $O(h^2)$ is the error of Euler formula, which is proportional to h^2.

In order to reduce the error to h^3, we further modify Euler form (4-10) as below:

$$\frac{x_{i+1} - x_i}{h} = \beta_0 f(t_i, x_i) + \beta_1 f(t_i + \gamma h, x_i + \delta h), \tag{4-19}$$

where β_0, β_1, γ and δ are variables to be determined. Since $f(t, x(t))$ depends on $x(t)$ and t, its Taylor's expansion at $x(t)=x_i$ and $t=t_i$ can be expressed as

$$\begin{aligned} f(t, y(t)) = {} & a_0 + a_t(t - t_i) + a_x(x(t) - x_i) + a_{tt}(t - t_i)^2 + a_{tx}(x(t) - x_i)(t - t_i) \\ & + a_{xx}(x(t) - x_i)^2 + a_{ttt}(t - t_i)^3 + a_{ttx}(t - t_i)^2(x(t) - x_i) \\ & + a_{txx}(t - t_i)(x(t) - x_i)^2 + a_{xxx}(x(t) - x_i)^3 + \cdots \end{aligned} \tag{4-20}$$

where all the coefficients a_0, a_t, a_x, a_{tt}, a_{tx} … are constant. The partial derivatives of $f(t, x(t))$ are

$$\begin{aligned} f_t(t, x(t)) &= \frac{\partial f(t, x(t))}{\partial t} \\ &= a_t + 2a_{tt}(t - t_i) + a_{tx}(x(t) - x_i) + 3a_{ttt}(t - t_i)^2 \\ &\quad + 2a_{ttx}(t - t_i)(x(t) - x_i) + a_{txx}(x(t) - x_i)^2 + \cdots \end{aligned} \tag{4-21}$$

$$\begin{aligned} f_x(t, x(t)) &= \frac{\partial f(t, x(t))}{\partial x(t)} \\ &= a_x + a_{tx}(t - t_i) + 2a_{xx}(x(t) - x_i) + a_{ttx}(t - t_i)^2 \\ &\quad + 2a_{txx}(t - t_i)(x(t) - x_i) + 3a_{xxx}(x(t) - x_i)^2 + \cdots \end{aligned} \tag{4-22}$$

$$\begin{aligned} f_{tt}(t, y(t)) &= \frac{\partial^2 f(t, x(t))}{\partial t^2} = \frac{\partial f_t(t, x(t))}{\partial t} \\ &= 2a_{tt} + 3 \cdot 2a_{ttt}(t - t_i) + 2a_{ttx}(x(t) - x_i) + \cdots \end{aligned} \tag{4-23}$$

$$\begin{aligned} f_{tx}(t, y(t)) &= \frac{\partial^2 f(t, x(t))}{\partial x(t) \partial t} = \frac{\partial f_x(t, x(t))}{\partial t} \\ &= a_{tx} + 2a_{ttx}(t - t_i) + 2a_{txx}(x(t) - x_i) + \cdots \end{aligned} \tag{4-24}$$

$$\begin{aligned} f_{xx}(t, x(t)) &= \frac{\partial^2 f(t, x(t))}{(\partial x(t))^2} = \frac{\partial f_x(t, x(t))}{\partial x(t)} \\ &= 2a_{xx} + 2a_{txx}(t - t_i) + 3 \cdot 2a_{xxx}(x(t) - x_i) + \cdots \end{aligned} \tag{4-25}$$

Clearly, all the coefficients can be derived from the above partial derivatives and expressed as

$$a_0 = f(t_i, x_i),\ \ a_t = f_t(t_i, x_i),\ \ a_x = f_x(t_i, x_i),$$

$$a_{tt} = \frac{1}{2!} f_{tt}(t_i, x_i),\ \ a_{tx} = f_{tx}(t_i, x_i),\ \ a_{xx} = \frac{1}{2!} f_{xx}(t_i, x_i), \cdots.$$

Substituting them into (4-20) yields

$$\begin{aligned} f(t, x(t)) = {} & f(t_i, x_i) + f_t(t_i, x_i)(t - t_i) + f_x(t_i, x_i)(x(t) - x_i) + \frac{1}{2!} f_{tt}(t_i, x_i)(t - t_i)^2 \\ & + f_{tx}(t_i, x_i)(t - t_i)(x(t) - x_i) + \frac{1}{2!} f_{xx}(t_i, x_i)(x(t) - x_i)^2 \\ & + \frac{1}{3!} f_{ttt}(t_i, x_i)(t - t_i)^3 + \frac{1}{3!} f_{xxx}(t_i, x_i)(x(t) - x_i)^3 + \cdots \end{aligned} \tag{4-26}$$

Let $t=t_i+\gamma h$ and $x(t)=x_i+\delta h$, then

$$\begin{aligned} & f(t_i + \gamma h, x_i + \delta h) \\ = {} & f(t_i, x_i) + h\gamma f_t(t_i, x_i) + h\delta f_x(t_i, x_i) + \frac{1}{2} h^2 \gamma^2 f_{tt}(t_i, x_i) \\ & + h^2 \gamma\delta f_{tx}(t_i, x_i) + \frac{1}{2} h^2 \delta^2 f_{xx}(t_i, x_i) + O(h^3) \end{aligned} \tag{4-27}$$

Hence, (4-19) can be rearranged and rewritten as

$$\begin{aligned} x_{i+1} = {} & x_i + h[(\beta_0 + \beta_1) f(t_i, x_i)] + \frac{h^2}{2!}[2\beta_1\gamma f_t(t_i, x_i) + 2\beta_1\delta f_x(t_i, x_i)] \\ & + \frac{h^3}{3!}[3\beta_1\gamma^2 f_{tt}(t_i, x_i) + 6\beta_1\gamma\delta f_{tx}(t_i, x_i) + 3\beta_1\delta^2 f_{xx}(t_i, x_i)] \\ & + O(h^4) \end{aligned} \tag{4-28}$$

If we want to take the precision to h^2, from (4-17) and (4-28) we have

$$\begin{aligned} x_{i+1} & \approx x_i + f(t_i, x_i)h + \frac{f'(t_i, x_i)}{2!} h^2 + O(h^3) \\ & \approx x_i + h[(\beta_0 + \beta_1) f(t_i, x_i)] \\ & \quad + \frac{h^2}{2!}[2\beta_1\gamma f_t(t_i, x_i) + 2\beta_1\delta f_x(t_i, x_i)] + O(h^3) \end{aligned} \tag{4-29}$$

Since $\dot{x}(t_i) = f(t_i, x_i)$, the term $f'(t_i, x_i)$ can be changed into the following form:

$$\begin{aligned} f'(t_i, x_i) & = f_t(t_i, x_i) + f_x(t_i, x_i)\dot{x}(t_i) \\ & = f_t(t_i, x_i) + f_x(t_i, x_i) f(t_i, x_i) \end{aligned} \tag{4-30}$$

As a result, from (4-29) and (4-30) we obtain

$$\begin{aligned} & f(t_i, x_i)h + \frac{h^2}{2!}(f_t(t_i, x_i) + f_x(t_i, x_i) f(t_i, x_i)) \\ = {} & h[(\beta_0 + \beta_1) f(t_i, x_i)] + \frac{h^2}{2!}[2\beta_1\gamma f_t(t_i, x_i) + 2\beta_1\delta f_x(t_i, x_i)] \end{aligned} \tag{4-31}$$

which leads to

$$\beta_0 + \beta_1 = 1 \tag{4-32}$$

$$2\beta_1\gamma f_t(t_i, x_i) + 2\beta_1\delta f_x(t_i, x_i) = f_t(t_i, x_i) + f_x(t_i, x_i) f(t_i, x_i) \tag{4-33}$$

From (4-33), wehave

$$2\beta_1\gamma = 1 \tag{4-34}$$

$$2\beta_1\delta = f(t_i, x_i) \tag{4-35}$$

Since there are four variables β_0, β_1, γ and δ in three equations (4-32), (4-34) and (4-35), the choice of these variables is not unique and commonly they are selected as

$$\beta_0 = \beta_1 = \frac{1}{2}, \quad \gamma = 1, \quad \delta = f(t_i, x_i) \tag{4-36}$$

That means the numerical solution (4-28) is chosen as

$$x_{i+1} = x_i + \frac{h}{2} f(t_i, x_i) + \frac{h}{2} f(t_i + h, x_i + hf(t_i, x_i)), \tag{4-37}$$

which is called the second-order Runge-Kutta formula. For the convenience of programming, the formula is often rearranged as below:

$$x_{i+1} = x_i + \frac{1}{2}(k_0 + k_1) \tag{4-38}$$

where

$$\begin{cases} k_0 = h \cdot f(t_i, x_i) \\ k_1 = h \cdot f(t_i + h, x_i + k_0) \end{cases} \tag{4-39}$$

In case that a higher precision is required, we oftem employ the higher order Runge-Kutta formula. For example, if a precision of h^4 is needed, then the fourth-order Runge-Kutta formula must be used, which is often given as

$$x_{i+1} = x_i + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3) \tag{4-40}$$

where

$$\begin{cases} k_0 = h \cdot f(t_i, x_i), & k_1 = h \cdot f\left(t_i + \frac{h}{2}, x_i + \frac{k_0}{2}\right) \\ k_2 = h \cdot f\left(t_i + \frac{h}{2}, x_i + \frac{k_1}{2}\right), & k_3 = h \cdot f(t_i, x_i + k_2) \end{cases} \tag{4-41}$$

This formula has been widely applied to a lot of applications in engineering due to its simplicity and acceptable accuracy.

Next, let's use an example to show the programming of the fourth-order

Runge-Kutta formula in MATLAB. Consider the following equation:

$$\dot{x}(t) = f(x(t),t) = -x(t)+1, \qquad x(0)=0.5 \tag{4-42}$$

and find the solution $x(t)$ for $t \geq 0$, which can be solved as below:

$$x(t) = -0.5e^{-t} + 1 \tag{4-43}$$

Now let's apply the fourth-order Runge-Kutta formula to verify (4-43) with the step size h=0.01. The precision is then in the order of $h^4=10^{-8}$. From (4-40), we have

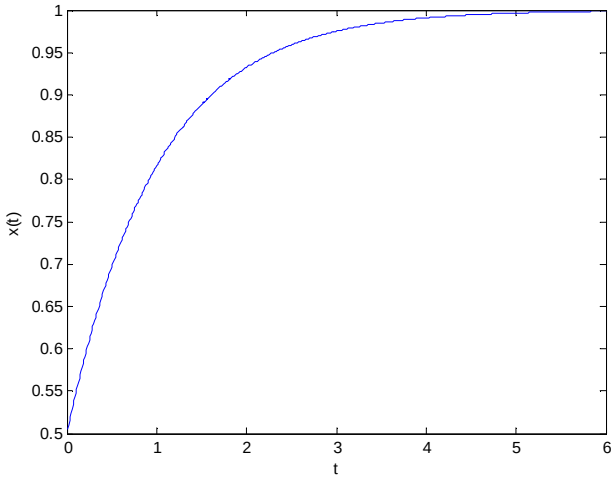
$$x_{i+1} = x_i + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3) \tag{4-44}$$

where

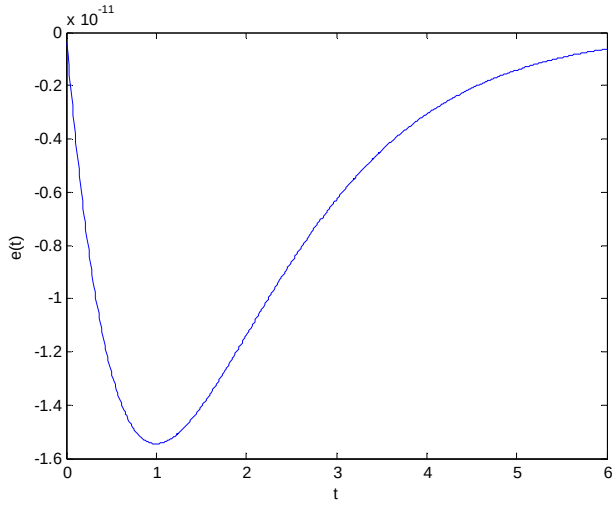
$$k_0 = h \cdot f(t_i, x_i) = h(-x_i + 1) = -h(x_i - 1)$$
$$k_1 = h \cdot f\left(t_i + \frac{h}{2}, x_i + \frac{k_0}{2}\right) = h\left(-\left(x_i + \frac{k_0}{2}\right) + 1\right) = -h\left(x_i + \frac{k_0}{2} - 1\right)$$
$$k_2 = h \cdot f\left(t_i + \frac{h}{2}, x_i + \frac{k_1}{2}\right) = h\left(-\left(x_i + \frac{k_1}{2}\right) + 1\right) = -h\left(x_i + \frac{k_1}{2} - 1\right)$$
$$k_3 = h \cdot f(t_i, x_i + k_2,) = h(-(x_i + k_2) + 1) = -h(x_i + k_2 - 1)$$

The programming in MATLAB is given as below:

```
================================================
>> % Fourth order Runge-Kutta method
>> x0=0.5; h=0.01;       % initial condition x(0)=0.5 and step size 0.01sec
>> k0=-h*(x0-1); k1=-h*(x0+k0/2-1); k2=-h*(x0+k1/2-1); k3=-h*(x0+k2-1);
>> x(1)=x0+(k0+2*k1+2*k2+k3)/6;
>> e(1)=x(1)-(-0.5*exp(-h)+1);       % numerical error at t=h
>> t(1)=h;
>> for n=1:599      % total simulation time 6 sec
>>       k0=-h*(x(n)-1);              k1=-h*(x(n)+k0/2-1);
>>       k2=-h*(x(n)+k1/2-1);         k3=-h*(x(n)+k2-1);
>>       x(n+1)=x(n)+(k0+2*k1+2*k2+k3)/6;
>>       t(n+1)=(n+1)*h;          % t-axis
>>       e(n+1)=x(n+1)-(-0.5*exp(-t(n+1))+1);      % numerical error
>> end
>> plot(t,x); xlabel('t'); ylabel('x(t)')
```



```
>> plot(t,e); xlabel('t'); ylabel('e(t)')
```



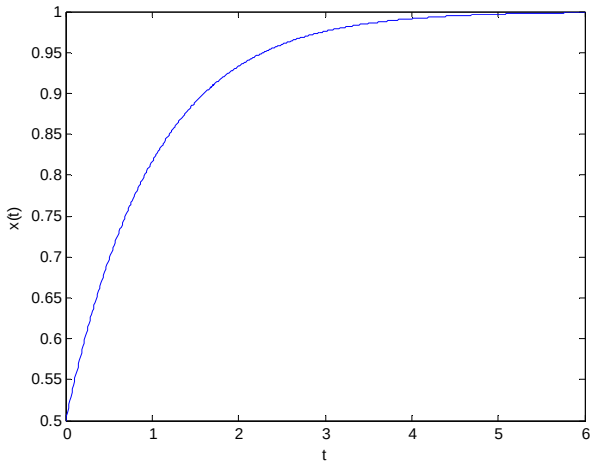
==========

From the above numerical results of error function $e(t)$, it is true that the precision is around 10^{-10} which is indeed less than $h^4(10^{-8})$. In MATLAB, some instructions are provided to solve the differential equations, such as ode23 and ode45. In fact, these instructions are also implemented by the Runge-Kutta method. Now, let's use the instruction ode23 to solve the system (4-42) and use the instruction ode45 to solve the system described as below:

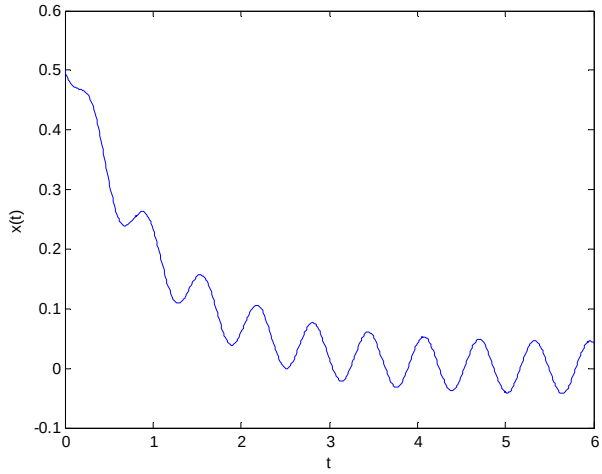
$$\dot{x}(t) = f(t, x(t)) = -x(t) + 0.5sin(sin10t), \qquad x(0)=0.5 \qquad (4\text{-}45)$$

whose solution can not be expressed in closed form. The programming in MATLAB is given in the following, which includes two m-files. The file first1.m is written for (4-43) and solved by ode23. The file first2.m is written for (4-45) and solved by ode45. Clearly, the result of (4-45) can not be expressed in closed form since it is more complicated than the result of (4-43).

```
==========================================
   Create m-file: first1.m
   function dx=first1(t,x)
   dx=-x+1;
==========================================
   Create m-file: first2.m
   function dx=first2(t,x)
   dx=-x+0.5*sin(sin(10*t));
==========================================
   >> % key in the following instructions
   >> [t,x]=ode23(@first1,[0:0.01:6],0.5)
   >> plot(t,x); xlabel('t'); ylabel('x(t)')
```




```
>> % key in the following instructions
>> [t,x]=ode45(@first2,[0:0.01:6],0.5)
>> plot(t,x); xlabel('t'); ylabel('x(t)')
```



==========