北京化工大学

Beijing University of Chemical Technology

计算方法讲义 (六)

常微分方程

Cheng Yong

目录

创建日期：2019 年 7 月 5 日

更新日期：2020 年 2 月 11 日

第 1 章　常微分方程

在工程和科学技术的实际问题中，常需要求解微分方程，如在高等数学中我们见过以下常微分方程：

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases} \tag{1.1}$$

$$\begin{cases} y'' = f(x, y, y') \\ y(x_0) = y_0, y'(x_0) = y_0' \end{cases} \tag{1.2}$$

$$\begin{cases} y'' = f(x, y, y') \qquad a \le x \le b \\ y(a) = y_a, y(b) = y_b \end{cases} \tag{1.3}$$

前两个问题称为初值问题，最后一个问题称为边值问题。

另外，在实际应用中还经常需要求解常微分方程组:

$$\begin{cases} y_1' = f_1(x, y_1, y_2) \qquad y_1(x_0) = y_{10} \\ y_2' = f_2(x, y_1, y_2) \qquad y_2(x_0) = y_{20} \end{cases} \tag{1.4}$$

只有简单的和典型的微分方程可以求出解析解，而在实际问题中的微分方程往往无法求出解析解，而必须采用数值方法求解，而差分方法是一类解微分方程的有效方法。

1.1　差分法

差分方法是一类重要的数值方法，这类方法回避了求 $y(x)$ 的解析表达式，而是要寻求它在一系列等距离离散节点

$$a = x_0 < x_1 < x_2 < x_3 < \cdots < x_n < \cdots \tag{1.5}$$

上的近似数值解 $y_0, y_1, y_2, y_3, \cdots y_n \cdots$，$h = (x_n - x_0)/n$，$x_i = x_0 + i * h$

我们首先介绍初值问题

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases} \tag{1.6}$$

的数值解。

初值问题的各种差分方法都采用 “步进式”，即求解过程顺着节点排列的次序一步一步地向前推进。描述这类算法，只要给出从已知信息 $y_0, y_1, y_2, y_3, \cdots, y_n$ 计算 y_{n+1} 的递推公式即可。这类计算格式统称为差分格式。

1.2 Euler 方法

由

$$\begin{cases} y^{'} = f\left(x, y\right) \\ y\left(x_0\right) = y_0 \end{cases} \tag{1.7}$$

可知

$$y^{'}(x_n) = f\left(x_n, y(x_n)\right) \tag{1.8}$$

用向前差商代替导数：

$$y^{'}(x_n) \approx \frac{y(x_{n+1}) - y(x_n)}{h} \tag{1.9}$$

代入 (1.8) 得到：

$$y(x_{n+1}) \approx y(x_n) + hf(x_n, y(x_n)) \tag{1.10}$$

用 y_n 作为 $y(x_n)$ 的近似值，并将所得结果作为 y_{n+1}，得到

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{1.11}$$

将 y_{n+1} 作为 $y(x_{n+1})$ 的近似值，由此得到 (向前)Euler 格式：

$$\begin{cases} y_0 = y\left(x_0\right) \\ y_{n+1} = y_n + hf(x_n, y_n) \end{cases} \tag{1.12}$$

初值 y_0 是已知的,则依据上式即可逐步算出微分方程初值问题的数值解 $y_1, y_2, y_3, \cdots, y_n, \cdots$。

例 1. 用 *Euler* 格式求解初值问题：

$$\begin{cases} y^{'} = y - \frac{2x}{y} \quad 0 \le x \le 1 \qquad h = 0.1 \\ y(0) = 1 \end{cases} \tag{1.13}$$

解：显然 $f(x, y) = y - \frac{2x}{y}, x_0 = 0, y_0 = 1, h = 0.1$，由欧拉格式：

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{1.14}$$

$$= y_n + h\left(y_n - \frac{2x_n}{y_n}\right) \quad n = 1, 2, 3, \cdots, 10 \tag{1.15}$$

得

$$y_1 = y_0 + h\left(y_0 - \frac{2x_0}{y_0}\right) \tag{1.16}$$

$$= 1 + 0.1\left(1 - \frac{2 \cdot 0}{1}\right) = 1.1 \tag{1.17}$$

$$y_2 = y_1 + h\left(y_1 - \frac{2x_1}{y_1}\right) \tag{1.18}$$

$$= 1.1 + 0.1\left(1.1 - \frac{2\cdot 0.1}{1.1}\right) = 1.1918 \tag{1.19}$$

依此类推，得结果见下表：

x_n	y_n	$f(x_n)$
0	1.0000	1.0000
0.1000	1.1000	1.0954
0.2000	1.1918	1.1832
0.3000	1.2774	1.2649
0.4000	1.3582	1.3416
0.5000	1.4351	1.4142
0.6000	1.5090	1.4832
0.7000	1.5803	1.5492
0.8000	1.6498	1.6125
0.9000	1.7178	1.6733
1.0000	1.7848	1.7321

图 1.1: 计算结果

从上表中可以看出解的准确值和近似值两者相差较大，可见，Euler 格式的精度较差，Euler 格式具有一阶代数精度。实际上此微分方程的解析解为：$y = \sqrt{(1+2x)}$。其函数曲线和欧拉解见下图。

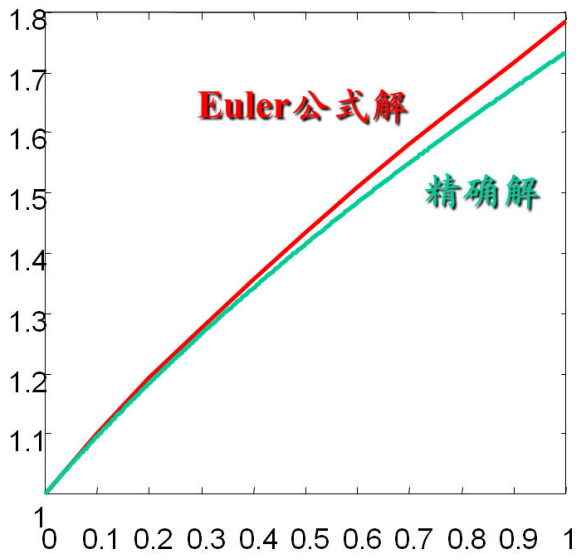

图 1.2: 函数曲线和欧拉解

```c
#include <stdio.h>
#define MAXSIZE 50

double f(double x,double y);

void main(void) {
    double a,b,h,x[MAXSIZE],y[MAXSIZE];
    long i,n;
    printf("\n请输入求解区间a,b: ");
    scanf("%lf,%lf",&a,&b);
    printf("\n请输入步长h: ");
    scanf("%lf",&h);
    n=(long)((b-a)/h);
    x[0]=a;

    printf("\n请输入起点x[0]=%lf处的纵坐标y[0]: ",x[0]);
    scanf("%lf",&y[0]);
    for(i=0; i<n; i++) {
        x[i+1]=x[i]+h;
        y[i+1]=y[i]+h*f(x[i],y[i]);
    }

    printf("\n计算结果为: ");

    for(i=0; i<=n; i++)
        printf("\nx[%ld]=%lf,y[%ld]=%lf",i,x[i],i,y[i]);
}

double f(double x,double y) {
    return y - 2*x/y; /*计算并返回函数值f(x,y)*/
}
```

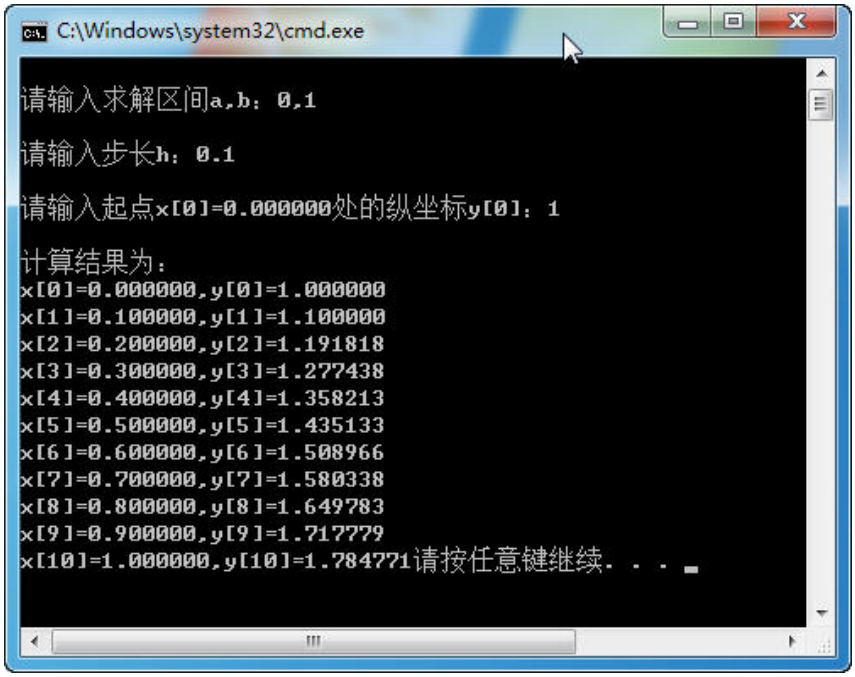

图 1.3: Euler 格式计算结果

隐式 Euler 格式

由

$$\begin{cases} y^{'} = f\left(x, y\right) \\ y\left(x_0\right) = y_0 \end{cases} \tag{1.20}$$

可知

$$y^{'}(x_{n+1}) = f\left(x_{n+1}, y(x_{n+1})\right) \tag{1.21}$$

用向后差商代替导数：

$$y^{'}(x_{n+1}) \approx \frac{y(x_{n+1}) - y(x_n)}{h} \tag{1.22}$$

代入 (1.21) 得到：

$$y(x_{n+1}) \approx y(x_n) + hf(x_{n+1}, y(x_{n+1})) \tag{1.23}$$

用 y_n 作为 $y(x_n)$ 的近似值，y_{n+1} 作为 $y(x_{n+1})$ 的近似值，得

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) \tag{1.24}$$

由此得到 (向后)Euler 格式：

$$\begin{cases} y_0 = y\left(x_0\right) \\ y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) \end{cases} \tag{1.25}$$

它具有和向前 Euler 格式相当的精度。

需要注意的是：向后 Euler 格式右端含有 y_{n+1}，因此此类格式称为隐式的。而向前 Euler 格式右端不含 y_{n+1}，此类格式称为显式的。

无论显式 Euler 公式还是隐式 Euler 公式，在计算 y_{n+1} 时只要用到前一个值 y_n，这种类型的方法称为单步格式或单步法。

Euler 两步格式

为改善精度，可采用中心差商。由

$$\begin{cases} y^{'} = f\left(x, y\right) \\ y\left(x_0\right) = y_0 \end{cases} \tag{1.26}$$

可知

$$y^{'}(x_n) = f\left(x_n, y(x_n)\right) \tag{1.27}$$

用中心差商代替导数：

$$y^{'}(x_n) \approx \frac{y(x_{n+1}) - y(x_{n-1})}{2h} \tag{1.28}$$

代入 (1.27) 得到如下计算公式：

$$y(x_{n+1}) \approx y_{n-1} + 2hf(x_n, y(x_n)) \tag{1.29}$$

这一公式称为 Euler 两步格式。在计算 y_{n+1} 时不仅要用到前一个值 y_n，还要用到更前一个值 y_{n-1}。

Euler 两步格式较向前 Euler 公式和向后 Euler 公式具有更高的精度，但它是一种两步法，它不能由初值直接求解。实际使用时除初值 y_0 外必须借助某种方法得到另一个值 y_1，这样就增加的计算的复杂性。

梯形格式

对于初值问题：

$$\begin{cases} y' = f\left(x, y\right) \\ y\left(x_0\right) = y_0 \end{cases} \tag{1.30}$$

对上式在区间 $[x_n, x_{n+1}]$ 上积分

$$\int_{x_n}^{x_{n+1}} y' dx = \int_{x_n}^{x_{n+1}} f(x, y) dx \tag{1.31}$$

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(x, y) dx \tag{1.32}$$

设 $y(x_n)$ 已知，则计算 $y(x_{n+1})$ 就要计算积分：

$$\int_{x_n}^{x_{n+1}} f(x, y) dx \tag{1.33}$$

采用不同的方法来计算积分，就会得到不同的差分格式。

采用矩形方法

$$\int_{x_n}^{x_{n+1}} f(x,y)dx \approx hf(x_n, y(x_n)) \tag{1.34}$$

得到 $y(x_{n+1}) \approx y(x_n) + hf(x_n, y(x_n))$，即为 Euler 格式。

采用梯形方法

$$\int_{x_n}^{x_{n+1}} f(x,y)dx \approx \frac{h}{2}\bigg(f(x_n, y(x_n)) + f(x_{n+1}, y(x_{n+1}))\bigg) \tag{1.35}$$

得到 $y(x_{n+1}) \approx y(x_n) + \frac{h}{2}\bigg(f(x_n, y(x_n)) + f(x_{n+1}, y(x_{n+1}))\bigg)$。

此即为梯形格式，它是向前 Euler 格式与向后 Euler 格式的算术平均值，这个格式也是隐式格式，具有二阶代数精度。

改进的 Euler 格式

此方法是先用 Euler 格式：

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{1.36}$$

求得一个近似值，记为 $\bar{y}_{n+1}$，称之为预报值，然后用它替代梯形法右端的再直接计算：

$$y(x_{n+1}) \approx y(x_n) + \frac{h}{2}\bigg(f(x_n, y(x_n)) + f(x_{n+1}, \bar{y}_{n+1})\bigg) \tag{1.37}$$

得到校正值 y_{n+1}。这样建立的预报-校正系统称为改进的 Euler(欧拉) 格式。

改进 Euler 格式为：

初值：

$$y(x_0) = y_0 \tag{1.38}$$

预报：

$$\bar{y}_{n+1} = y_n + hf(x_n, y_n) \tag{1.39}$$

校正：

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, \bar{y}_{n+1})) \tag{1.40}$$

这是一步显式格式，它可表为如下平均化形式：

$$\begin{cases} y(x_0) = y_0 \\ y_p = y_n + hf(x_n, y_n) \\ y_c = y_n + hf(x_{n+1}, y_p) \\ y_{n+1} = \frac{1}{2}(y_p + y_c) \end{cases} \tag{1.41}$$

例 2. 用改进的 *Euler* 格式求解初值问题。

$$\begin{cases} y' = y - \frac{2x}{y} \quad 0 \le x \le 1, h = 0.1 \\ y(0) = 1 \end{cases} \tag{1.42}$$

解：显然 $f(x,y) = y - \frac{2x}{y}, x_0 = 0, y_0 = 1, h = 0.1$
由改进 Euler 格式，有：

$$\bar{y}_{n+1} = y_n + hf(x_n, y_n) \tag{1.43}$$
$$= y_n + h(y_n - \frac{2x_n}{y_n}) \tag{1.44}$$
$$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, \bar{y}_{n+1})) \tag{1.45}$$
$$= y_n + \frac{h}{2}\left((y_n - \frac{2x_n}{y_n}) + (\bar{y}_{n+1} - \frac{2x_{n+1}}{\bar{y}_{n+1}})\right) \tag{1.46}$$

$$\bar{y}_1 = y_0 + h(y_0 - \frac{2x_0}{y_0}) = 1 + 0.1(1 - \frac{2 \cdot 0}{1}) = 1.1 \tag{1.47}$$
$$y_1 = y_0 + \frac{h}{2}\left((y_0 - \frac{2x_0}{y_0}) + (\bar{y}_1 - \frac{2x_1}{\bar{y}_1})\right) \tag{1.48}$$
$$= 1 + \frac{0.1}{2}\left((1 - \frac{2 \cdot 0}{1}) + (1.1 - \frac{2 \cdot 0.1}{1.1})\right) = 1.0959 \tag{1.49}$$
$$\bar{y}_2 = y_1 + h(y_1 - \frac{2x_1}{y_1}) = 1.0959 + 0.1(1.0959 - \frac{2 \cdot 0.1}{1.0959}) = 1.1872 \tag{1.50}$$
$$y_2 = y_1 + \frac{h}{2}\left((y_1 - \frac{2x_1}{y_1}) + (\bar{y}_2 - \frac{2x_2}{\bar{y}_2})\right) \tag{1.51}$$
$$= 1.0959 + \frac{0.1}{2}\left((1.0959 - \frac{2 \cdot 0.1}{1.0959}) + (1.1872 - \frac{2 \cdot 0.2}{1.1872})\right) = 1.1841 \tag{1.52}$$

依此类推，结果如下表：

x_n	y_n	$f(x_n, y_n)$
0.1	1.0959	1.0954
0.2	1.1841	1.1832
0.3	1.2662	1.2649
0.4	1.3434	1.3416
0.5	1.4164	1.4142
0.6	1.4860	1.4832
0.7	1.5525	1.5492
0.8	1.6165	1.6125
0.9	1.6782	1.6733
1.0	1.7379	1.7321

表 1.1: 计算结果

上述结果精度较 Euler 格式有明显提高。

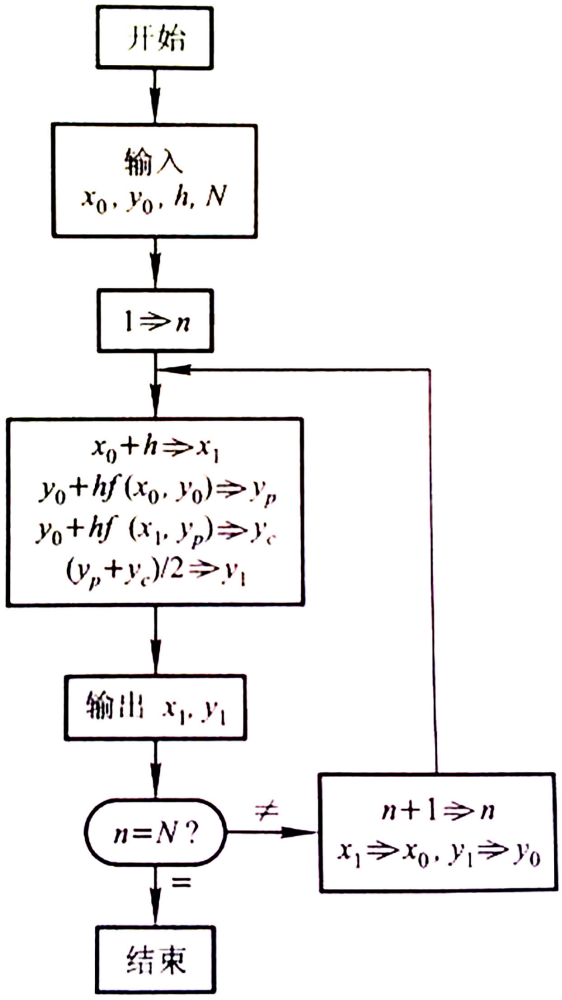

图 1.4: 改进的欧拉公式

```
#include<iostream>
#define N 10

using namespace std;

void modEuler(float (*f1)(float,float),float x0,float y0,float xn,int n) {
    int i;
    float yp,yc,x=x0,y=y0,h=(xn-x0)/n;
    // cout<<"x[0]="<<x<<'\t'<<"y[0]"<<y<<endl;
    for(i=1; i<=n; i++) {
        yp=y+h*f1(x,y);
        x=x0+i*h;
        yc=y+h*f1(x,yp);
        y=(yp+yc)/2.0;
        cout<<"x["<<i<<"]="<<x<<" y["<<i<<"]="<<y<<endl;
    }
}
void main() {
```

```
19
20      float xn=0.0,x0=-1.0,y0=3.0;
21      float f1(float ,float);
22      modEuler(f1,x0,y0,xn,N);
23  }
24  float f1(float x,float y) {
25      return 3*x-2*y*y-12;
26  }
```

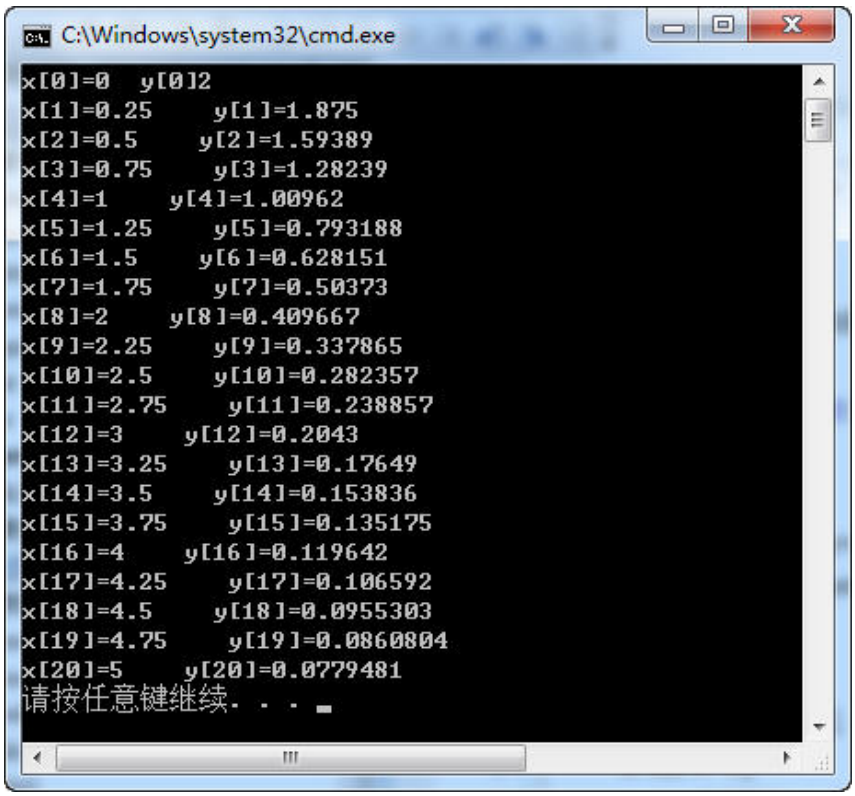

图 1.5: 改进的欧拉公式

Euler 格式的精度分析

这里依然运用代数精度来判定差分格式的精度。

定义 1. 定义 称某个差分格式具有 m 阶精度，如果它的近似关系式对于次数 $\leq m$ 的多项式均能准确成立，而对 $m+1$ 次多项式不能准确成立。

例 3. 考察 *Euler* 格式的精度。

解：由 Euler 格式

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{1.53}$$

可知，它对应的近似关系式：

$$y(x_{n+1}) \approx y(x_n) + hf(x_n, y(x_n)) \tag{1.54}$$

$$= y(x_n) + hy'(x_n) \tag{1.55}$$

当 $y(x) = 1$ ，左端 = 右端 =1

当 $y(x) = x$ ，左端 = 右端 $= x_{n+1} = x_n + h$

当 $y(x) = x^2$ ，左端为 $x_{n+1}^2 = (x_n + h)^2$，而右端为 $x_n^2 + 2hx_n$，因此左端 $\neq$ 右端。

故 Euler 格式具有一阶代数精度。

同理可以证明，梯形格式和改进的 Euler 格式均具有二阶代数精度。

1.3 亚当斯方法

1.4 Runge-Kutta 方法

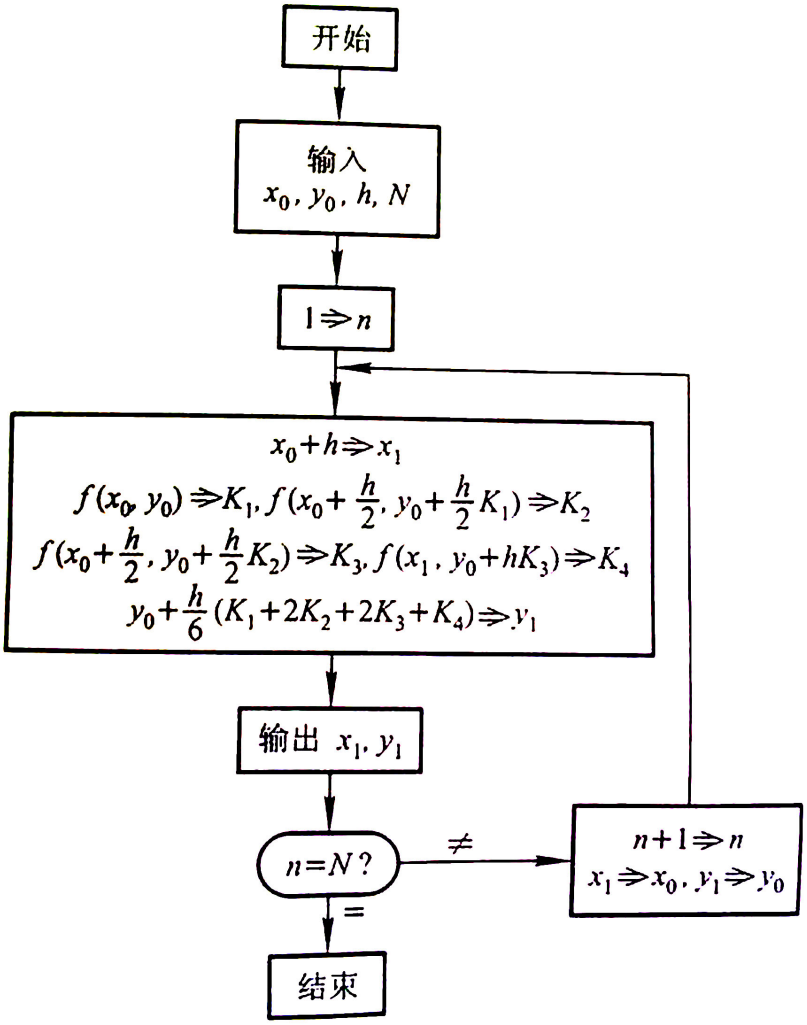

图 1.6: 四阶 Runge-Kutta 方法

1.5 本章小结

本章介绍了常微分方程的差分法，主要包括 Euler 方法、改进的 Euler 方法、Runge-Kutta 方法、Adams 方法及其他重要的线性多步格式，最后对算法的收敛性和稳定性进行了分析。

- Euler 方法；
- 改进的 Euler 方法；

本章习题

P118-120 页，第 3，12，24，25 题。

参考文献

[1] Author. *Title*. http://www.baidu.com, 2019.

[2] Author. *Title*. http://www.baidu.com, 2019.

[3] Author. *Title*. http://www.baidu.com, 2019.

[4] Author. *Title*. http://www.baidu.com, 2019.