Lecture 7: Partial Differential Equations

Euler, Runge – Kutta and Adams Methods

- Our ultimate goal is to solve very general nonlinear **partial differential equations (非线性偏微分方程)** of elliptic (椭圆的), hyperbolic (双曲线的), parabolic (抛物线的), or mixed type.

- However, a variety of basic techniques are required from the solutions of **ordinary differential equations (常微分方程)**. By understanding the basic ideas for computationally solving initial and boundary value problems (初值和边值问题) for differential equations, we can solve more complicated partial differential equations.

Differential Equations

- The development of numerical solution techniques for initial and boundary value problems originates (起源于) from the simple concept of the Taylor expansion.
- Implementation, however, often requires ingenuity (新颖), insight, and clever application of the basic principles. In some sense, our numerical solution techniques reverse (逆转) our understanding of calculus (微积分). Whereas calculus(微积分) teaches us to take a limit (用极限) in order to define a derivative or integral, in numerical computations we take the derivative or integral of the governing equation (控制方程) and go backwards to define it as the difference.

学过中学数学的人对于方程是比较熟悉的；在初等数学中就有各种各样的方程，比如线性方程、二次方程、高次方程、指数方程、对数方程、三角方程和方程组等等。这些方程都是要把研究的问题中的已知数和未知数之间的关系找出来，列出包含一个未知数或几个未知数的一个或者多个方程式，然后取求方程的解。

但是在实际工作中，常常出现一些特点和以上方程完全不同的问题。比如：物质在一定条件下的运动变化，要寻求它的运动、变化的规律；某个物体在重力作用下自由下落，要寻求下落距离随时间变化的规律；火箭在发动机推动下在空间飞行，要寻求它飞行的轨道，等等，要以现有数据求得出形式上的函数解析式，而不是以已知函数来计算特定的未知数。

物质运动和它的变化规律在数学上是用函数关系来描述的，因此，这类问题就是要去寻求满足某些条件的一个或者几个未知函数。也就是说，凡是这类问题都不是简单地去求一个或者几个固定不变的数值，而是要求一个或者几个未知的函数。

在数学上，解这类方程，要用到微分和导数的知识。因此，凡是表示未知函数的导数以及自变量之间的关系的方程，就叫做微分方程。

定义1：凡含有参数，未知函数和未知函数导数 (或微分) 的方程，称为微分方程，有时简称为方程，未知函数是一元函数的微分方程称作常微分方程，未知函数是多元函数的微分方程称作偏微分方程。微分方程中出现的未知函数最高阶导数的阶数，称为微分方程的阶。定义式如下： $F(x, y, y', y'', \cdots, y^{(n)}) = 0$

方程最初是“等式”的同义词，发展到现在，又特指“含未知量的等式”．解方程的过程，就是通过摆弄方程，求出未知量．

我们从小学就开始接触到的方程，通常又被叫做“代数方程”．代数方程的未知量就是数字，我们通过在方程两边加减乘除乘方开方，或者在多个方程之间进行加减乘除乘方开方等操作，计算出这个（些）数字到底是什么，这样就解出了一个代数方程．由于解方程时所用到的操作全都是代数运算，所以称之为代数方程．

微分方程的未知量则是函数，而方程中会涉及对这些函数求微分．解微分方程要做的是，得到具体的函数表达式．

举例而言，如果我们知道一个物体在空气中受到的阻力正比于其速度，那么这个物体在重力作用下在空气中下落时，该怎么求出其每时每刻的速度大小呢？这里，未知量就是速度函数，其自变量是时间．如果用 v 来表示物体竖直向下的速度大小，那么空气对它造成的加速度就可以写成 $-kv$，重力对它造成的加速度则是 g，其中 k, g 都是正实数．我们自然可以根据问题描述，写下如下关系：

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -kv + g \tag{1}$$

求解这个微分方程，就是得到类似 $v = \frac{g}{k} - C\mathrm{e}^{-kt}$ 的结果，其中 C 是一个待定常数，由该物体的初速度决定．

Euler, Runge-Kutta and Adams methods

The solutions of general partial differential equations (偏微分方程) rely heavily on the techniques developed for ordinary differential equations (常微分方程). Thus we begin by considering systems of **differential equations** of the form

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}, t)$$

where *y* represents a vector and the initial conditions are given by

$$\mathbf{y}(0) = \mathbf{y}_0$$

with $t \in [0, T]$. Although very simple in appearance, this equation cannot be solved analytically (解析求解) in general. Of course, there are certain cases for which the problem can be solved analytically, but it will generally be important to **rely on numerical solutions** for insight.

The simplest algorithm for solving this system of differential equations is known as the **Euler method (欧拉法)**. The Euler method is derived by **making use of the definition of the derivative**:

$$\frac{d\mathbf{y}}{dt} = \lim_{\Delta t \to 0} \frac{\Delta \mathbf{y}}{\Delta t}$$

Thus over a time span $\Delta t = t_{n+1} - t_n$ we can approximate the original differential equation by

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}, t) \quad \Rightarrow \quad \frac{\mathbf{y}_{n+1} - \mathbf{y}_n}{\Delta t} \approx f(\mathbf{y}_n, t_n)$$

The approximation can easily be rearranged to give

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot f(\mathbf{y}_n, t_n)$$

Thus the Euler method gives an iterative scheme by which the future values of the solution can be determined. Generally, the algorithm structure is of the form

$$\mathbf{y}(t_{n+1}) = F(\mathbf{y}(t_n))$$

where $F(\mathbf{y}(t_n)) = \mathbf{y}(t_n) + \Delta t \cdot f(t_n, \mathbf{y}(t_n))$

The graphical representation of this iterative process is illustrated in Fig. 1 where the slope (derivative) of the function is responsible for generating each subsequent approximation to the solution *y(t)*. Note that the Euler method is exact as the step size descreases to zero: $\Delta t \to 0$.

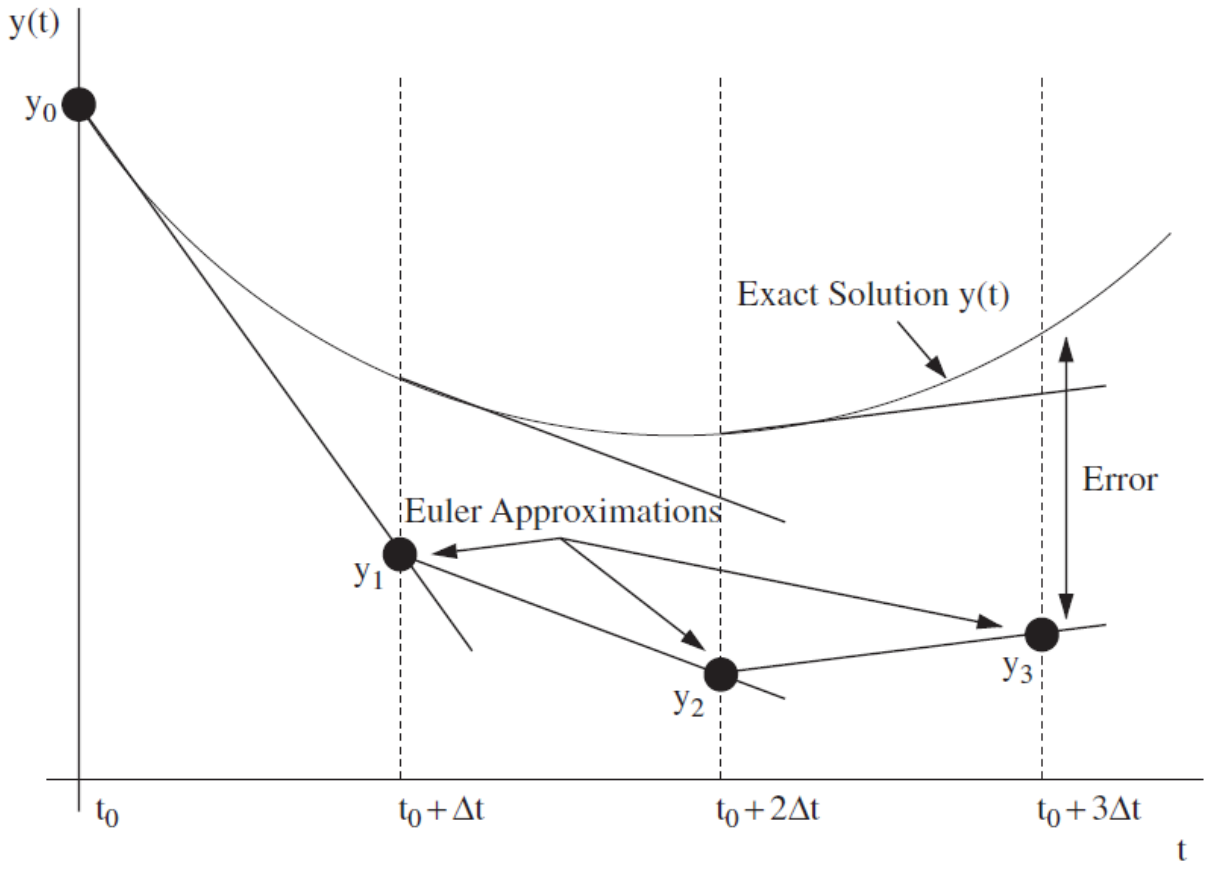


Figure 7.1: Graphical description of the iteration process used in the Euler method. Note that each subsequent approximation is generated from the slope of the previous point. This graphical illustration suggests that smaller steps Δt should be more accurate.

The Euler method can be generalized to the following iterative scheme:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot \phi \,.$$

where the function ϕ is chosen to reduce the error over a single time step Δt and $y_n = y(t_n)$.

The function ϕ is no longer constrained (固定), as in the Euler scheme, to make use of the derivative at the left end point of the computational step.

Rather, the derivative at the mid-point of the time-step and at the right end of the time-step may also be used to possibly improve accuracy. In particular, by generalizing to include the slope at the left and right ends of the time-step Δt, we can generate an iteration scheme of the following form:

$$\mathbf{y}(t+\Delta t) = \mathbf{y}(t) + \Delta t\left[Af(t,\mathbf{y}(t)) + Bf(t+P\cdot\Delta t, \mathbf{y}(t) + Q\Delta t\cdot f(t,\mathbf{y}(t)))\right]$$

where *A*,*B*, *P* and *Q* are arbitrary constants.

Upon Taylor expanding the last term, we find

$$f(t + P \cdot \Delta t, \mathbf{y}(t) + Q\Delta t \cdot f(t, \mathbf{y}(t))) =$$
$$f(t, \mathbf{y}(t)) + P\Delta t \cdot f_t(t, \mathbf{y}(t)) + Q\Delta t \cdot f_{\mathbf{y}}(t, \mathbf{y}(t)) \cdot f(t, \mathbf{y}(t)) + O(\Delta t^2)$$

where f_t and f_y denote differentiation with respect to t and y respectively, and $O(\Delta t^2)$ denotes all terms that are of size Δt^2 and smaller.

Plugging in this last result into the original iteration scheme (5.1.8) results in the following:

$$\mathbf{y}(t + \Delta t) = \mathbf{y}(t) + \Delta t(A + B) f(t, \mathbf{y}(t))$$
$$+PB\Delta t^2 \cdot f_t(t, \mathbf{y}(t)) + BQ\Delta t^2 \cdot f_{\mathbf{y}}(t, \mathbf{y}(t)) \cdot f(t, \mathbf{y}(t)) + O(\Delta t^3)$$

which is valid up to $O(\Delta t^2)$.

To proceed further, we simply note that the Taylor expansion for $\mathbf{y}(t+\Delta t)$ gives:

$$\mathbf{y}(t+\Delta t) = \mathbf{y}(t) + \Delta t \cdot f(t,\mathbf{y}(t)) + \frac{1}{2}\Delta t^2 \cdot f_t(t,\mathbf{y}(t)) + \frac{1}{2}\Delta t^2 \cdot f_{\mathbf{y}}(t,\mathbf{y}(t))f(t,\mathbf{y}(t)) + O(\Delta t^3)$$

Comparing this Taylor expansion with (5.1.10) gives the following relations:

$$A + B = 1$$
$$PB = \frac{1}{2}$$
$$BQ = \frac{1}{2}$$

which yields three equations for the four unknowns *A*,*B*, *P* and *Q*. Thus one degree of freedom is granted, and a wide variety of schemes can be implemented.

Two of the more commonly used schemes are known as Heun's (休恩) method and Modified Euler-Cauchy (second order Runge-Kutta). These schemes assume *A = 1/2* and *A = 0* respectively, and are given by:

$$\mathbf{y}(t+\Delta t) = \mathbf{y}(t) + \frac{\Delta t}{2}\left[f(t,\mathbf{y}(t)) + f(t+\Delta t, \mathbf{y}(t)+\Delta t\cdot f(t,\mathbf{y}(t)))\right]$$

$$\mathbf{y}(t+\Delta t) = \mathbf{y}(t) + \Delta t\cdot f\left(t+\frac{\Delta t}{2}, \mathbf{y}(t)+\frac{\Delta t}{2}\cdot f(t,\mathbf{y}(t))\right).$$

Generally speaking, these methods for iterating forward in time given a single initial point are known as Runge-Kutta (龙格一库塔) methods. By generalizing the assumption (5.1.8), we can construct stepping schemes (构造多步方法) which have arbitrary accuracy. Of course, the level of algebraic (代数的) difficulty in deriving these higher accuracy schemes also increases significantly from Heun's method and Modified Euler-Cauchy.

<Lec 7c>

$\dot{y} = f(y)$, or more generally $\dot{y} = f(t, y)$

Forward Euler: $y_{k+1} = y_k + \Delta t\, f(t_k, y_k)$

Backward Euler: $y_{k+1} = y_k + \Delta t\, f(t_{k+1}, y_{k+1})$

* Both methods have poor global error $\sim O(\Delta t)$

Let's try a generic integrator (tune it for small error)

~~Generat~~

Generic integrator : $y_{k+1} = y_k + \Delta t\, \phi$

we choose to ϕ to reduce error.

Idea : Instead of ϕ being the slope at (t_k, y_k) (F.E.)

or (t_{k+1}, y_{k+1}) (B.E.),

lets use the slope at (t_k, y_k) AND at another point $(t_k + p\Delta t,\ y_k + q\Delta t f(t_k, y_k))$.

$$y_{k+1} = y_k + \Delta t \underbrace{\left[A f(t_k, y_k) + B f(t_k + p\Delta t,\ y_k + q\Delta t f(t_k, y_k)) \right]}_{\phi} \quad (*)$$

We get to choose A, B, P, α to match Taylor series. $y(t_k+\Delta t)$

If $P=\alpha$ then $\left(t_k+P\Delta t,\ y_k+P\Delta t f(t_k,y_k)\right)$ is a small $P\Delta t$ Forward Euler step.

Taylor expand the last term in (*):

$$f\left(t_k+P\Delta t,\ y_k+P\Delta t f(t_k,y_k)\right)$$

$$= f(t_k,y_k) + P\Delta t \frac{\partial f}{\partial t}(t_k,y_k) + P\Delta t \frac{\partial f}{\partial y}(t_k,y_k)\cdot f(t_k,y_k) + \mathcal{O}(\Delta t^2)$$

to plugin eq.(*) becomes:

$$(**)\quad y_{k+1} = y_k + \Delta t(A+B) f(t_k,y_k) + PB\Delta t^2 f_t(t_k,y_k) + PB\Delta t^2 f_y(t_k,y_k) f(t_k,y_k) + \mathcal{O}(\Delta t^3)$$

($\frac{\partial f}{\partial t}$ and $\frac{\partial f}{\partial y}$)

Compare with Taylor Series for exact $y(t_k+\Delta t)$:

$$y(t_k+\Delta t) = y(t_k) + \Delta t \frac{dy}{dt}(t_k) + \frac{\Delta t^2}{2}\frac{d^2y}{dt^2}(t_k) + \mathcal{O}(\Delta t^3)$$

$$= y_k + \Delta t f(t_k, y_k) + \frac{\Delta t^2}{2}\frac{d}{dt} f(t_k, y_k) + \mathcal{O}(\Delta t^3)$$

$$= y_k + \Delta t f(t_k, y_k) + \frac{\Delta t^2}{2}\left[f_t(t_k, y_k) + f_y(t_k, y_k) \cdot \underbrace{f(t_k, y_k)}_{\dot{y}(t_k)} \right] + \mathcal{O}(\Delta t^3)$$

$$(***) \quad = y_k + \Delta t f(t_k, y_k) + \frac{\Delta t^2}{2} f_t(t_k, y_k) + \frac{\Delta t^2}{2} f_y(t_k, y_k) f(t_k, y_k) + \mathcal{O}(\Delta t^3)$$

$$y_{k+1} = y_k + \Delta t \left[A f(t_k, y_k) + B f(t_k + P\Delta t,\ y_k + Q\Delta t f(t_k, y_k)) \right] \quad (*)$$

Comparing (**) with (***), we have:

$$\begin{cases} A + B = 1 \\ pB = 1/2 \\ p = \alpha \end{cases}$$

unknown A, B, p, α. (many sto solutions, all agree on the eq with $O(\Delta t^3)$)

Most popular choice:

$A = 0$, $B = 1$, $p = \alpha = 1/2$

$$y_{k+1} = y_k + \Delta t f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f(t_k, y_k)\right)$$

"Second-order Runge-Kutta" ("ode23" in matlab)

* Explicit v.s. implicit
* 3rd order local error $O(\Delta t^3)$, 2nd order global error.

Fourth – Order Runge–Kutta "ode45"

$$y_{k+1} = y_k + \frac{\Delta t}{6}\left[f_1 + 2f_2 + 2f_3 + f_4\right]$$

where

$$f_1 = f(t_k, y_k)$$

$$f_2 = f\left(t_k + \frac{\Delta t}{2},\ y_k + \frac{\Delta t}{2} f_1\right)$$

$$f_3 = f\left(t_k + \frac{\Delta t}{2},\ y_k + \frac{\Delta t}{2} f_2\right)$$

$$f_4 = f\left(t_k + \Delta t,\ y_k + \Delta t f_3\right)$$

Global error $\sim O(\Delta t^4)$

Local error $\sim O(\Delta t^5)$

Other integrators:

- Adams – Bashforth
- Stiff integrators
- Energy conserving (symplectic)

Applications of "ode45" :

Van der Pol Oscillator

$$\begin{cases} \dot{x} = v \\ \dot{v} = \mu(1-x^2)v + x \end{cases}$$

Vector form :

$$y = \begin{pmatrix} x \\ v \end{pmatrix} \qquad \dot{y} = f(y)$$

Codes : vdp.m , simvdp.m .

$\dot{y} = f(t, y)$

Last time we saw the 4th order Runge-Kutta integrator ("ode45")

$$y_{k+1} = y_k + \frac{\Delta t}{6}\left[f_1 + 2f_2 + 2f_3 + f_4\right]$$

$f_1 = f(t_k, y_k)$

$f_2 = f(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f_1)$ /* $\frac{\Delta t}{2}$ "half F.E. based on f_1 */

$f_3 = f(t_k + \frac{\Delta t}{2}, y_k + \frac{\Delta t}{2} f_2)$ /* another half F.E. using f_2 */

$f_4 = f(t_k + \Delta t, y_4 + \Delta t f_3)$ /* full F.E using f_3 */

- very accurate $\mathcal{O}(\Delta t^5)$ local, $\mathcal{O}(\Delta t^4)$ global
- uses <u>four</u> evaluations of $f(t, y)$

Page 2

Example : Lorenz Model (1963, atomospheric convection model)

$$\begin{cases} \dot{x} = \sigma(y-x) \\ \dot{y} = x(\rho - z) - y \\ \dot{z} = xy - \beta z \end{cases}$$

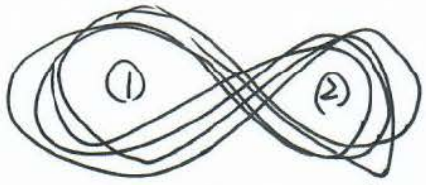
parameters may lead to "chaos"

$\sigma = 10$

$\beta = 8/3$

$\rho = 28$

$\Rightarrow$



(no way to predict which side it is at time t.)

1963年，Lorenz发现了第一个混沌吸引子——Lorenz系统，从此揭开了混沌研究的序幕。

概念

在数学中，一个动力系统被称为自治的，当且仅当这个系统由一组常微分方程组成，并且这些方程的表达式与动力系统的自变量无关。在有关物理的动力系统中，自变量通常是时间。这时自治系统通常表示其中的物理规律不再随时间变化的系统，也就是说空间中每一点的性质在过去、现在和将来都是一样的。自治系统是动力系统中很重要的一个组成部分。理论上来说，所有的动力系统都可以转化为自治系统。对于自治微分系统来说，要出现混沌现象，其维数必须要大于2.典型的一个例子就是Lorenz模型，它是由美国气象学家Lorenz在研究大气运动的时候，通过对对流模型简化，只保留三个变量提出的一个完全确定性的三阶自治常微分方程组，其方程形式如下：

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = \rho x - y - xz \\ \frac{dz}{dt} = xy - \beta z \end{cases}$$

其中，三个参数分别为：σ为普朗特数，ρ是瑞利数，β是方向比。

Lorenz模型已经成为混沌领域的经典模型，系统中三个参数的选择对系统会不会进入混沌状态起着重要的作用。

4th-order Runge-Kutta

Perhaps the most popular general stepping scheme used in practice is known as the *4th* order Runge-Kutta (龙格—库塔) method. The term “4th order” refers to the fact that the Taylor series local truncation error (泰勒级数局部截断误差) is pushed to $O(t^5)$. The total cumulative (global) error is then $O(t^4)$ and is responsible for the scheme name of “4th order”. The scheme is as follows:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{6}\left[f_1 + 2f_2 + 2f_3 + f_4\right]$$

where

$$f_1 = f(t_n, \mathbf{y}_n)$$

$$f_2 = f\left(t_n + \frac{\Delta t}{2}, \mathrm{y}_n + \frac{\Delta t}{2} f_1\right)$$

$$f_3 = f\left(t_n + \frac{\Delta t}{2}, \mathrm{y}_n + \frac{\Delta t}{2} f_2\right)$$

$$f_4 = f\left(t_n + \Delta t, \mathrm{y}_n + \Delta t \cdot f_3\right)$$

9.5.1 关于该方法的讨论

式(7)的完整推导超出了本书的范围，可在高级教程中找到，但从这里也能了解一些情况。考虑解曲线 $y = y(t)$ 在第一个子区间 $[t_0, t_1]$ 上的曲线，式(7)中的函数值是该曲线斜率的近似

值。其中，f_1 是左端点的斜率，f_2 和 f_3 为中间两点的斜率的估计，而 f_4 是右端点的斜率，见图 9.9(a)。然后通过对斜率函数

$$y(t_1) - y(t_0) = \int_{t_0}^{t_1} f(t, y(t))\, dt \tag{8}$$

积分得到下一个点 (t_1, y_1)。

如果应用辛普森公式和步长 $h/2$，式(8)的积分近似为

$$\int_{t_0}^{t_1} f(t, y(t))\, dt \approx \frac{h}{6}(f(t_0, y(t_0)) + 4f(t_{1/2}, y(t_{1/2})) + f(t_1, y(t_1))) \tag{9}$$

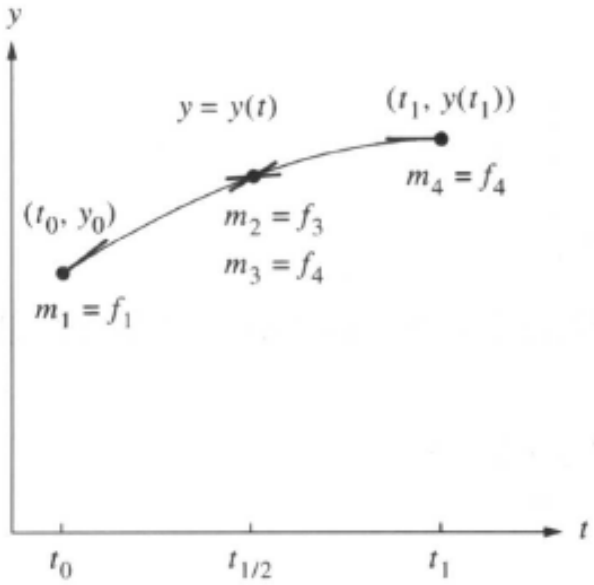
其中 $t_{1/2}$ 为区间中点。需要 3 次函数求值，因此显然可以选择 $f(t_0, y(t_0)) = f_1$ 和 $f(t_1, y(t_1)) \approx f_4$。中点的值则选择为 f_2 和 f_3 的平均值：

$$f(t_{1/2}, y(t_{1/2})) \approx \frac{f_2 + f_3}{2}$$

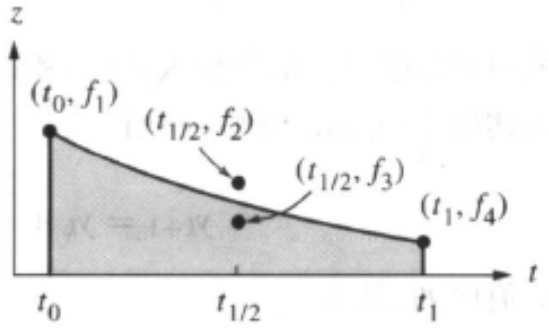
将这些值代入式(9)，并用它和式(8)求 y_1：

$$y_1 = y_0 + \frac{h}{6}\left(f_1 + \frac{4(f_2 + f_3)}{2} + f_4\right) \tag{10}$$

该式简化后得到式(6)和 $k=0$。式(9)的积分图见图 9.9(b)。



(a) 解曲线 $y = y(t)$ 的预报斜率 m_j



(b) 积分逼近：

$$y(t_1) - y_0 = \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$$

图 9.9　$N=4$ 阶龙格－库塔方法中的 $y = y(t)$ 和 $z = f(t, y(t))$ 曲线

The key to this method, as well as any of the other Runge-Kutta schemes, is the use of intermediate time-steps to improve accuracy.

For the *4th* order scheme presented here, a graphical representation of this derivative sampling at intermediate time-steps is shown in Fig. 2.

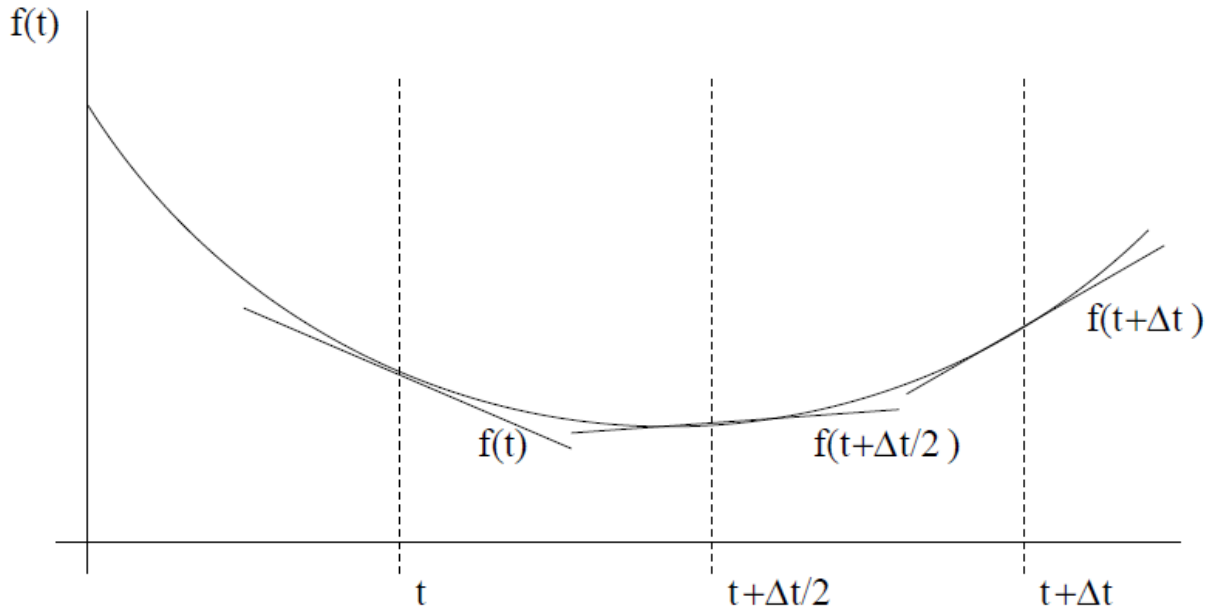


Figure 2: Graphical description of the initial, intermediate, and final slopes used in the 4th order Runke-Kutta iteration scheme over a time Δt.

Adams method: multi-stepping techniques

The development of the Runge-Kutta schemes rely on the definition of the derivative and Taylor expansions. Another approach to solving (5.1.1) is **to start with the fundamental theorem of calculus (微积分)**. Thus the differential equation can be integrated over a time-step *t* to give

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}, t) \quad \Rightarrow \quad \mathbf{y}(t + \Delta t) - \mathbf{y}(t) = \int_t^{t+\Delta t} f(t, y)dt$$

And once again using our iteration notation we find

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} f(t, y)dt$$

This iteration relation is simply a restatement (重述) of (5.1.7) with $\Delta t \cdot \phi = \int_{t_n}^{t_{n+1}} f(t, \mathbf{y}) dt$. However, at this point, no approximations have been made and (5.1.17) is exact. The numerical solution will **be found by approximating** $f(t, \mathbf{y}) \approx p(t, \mathbf{y})$ **where *p(t, y)* is a polynomial**. Thus the iteration scheme in this instance will be given by

$$\mathbf{y}_{n+1} \approx \mathbf{y}_n + \int_{t_n}^{t_{n+1}} p(t, y) dt$$

It only remains to determine the form of the polynomial to be used in the approximation. The Adams-Bashforth suite (系列) of computational methods uses the current point and a determined number of (一定的数量) past points to evaluate the future solution. As with the Runge-Kutta schemes, the order of accuracy is determined by the choice of ϕ . In the Adams-Bashforth case, this relates directly to the choice of the polynomial approximation *p(t, y)*. A first-order scheme can easily be constructed by allowing

$$p_1(t) = \text{constant} = f(t_n, \mathbf{y}_n)$$

where the present point and no past points are used to determine the value of the polynomial. Inserting this first-order approximation into (5.1.18) results in the previously found Euler scheme

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot f(t_n, \mathbf{y}_n)$$

Alternatively, we could assume that the polynomial used both the current point and the previous point so that a second-order scheme resulted. The linear polynomial which passes through these two points is given by

$$p_2(t) = f_{n-1} + \frac{f_n - f_{n-1}}{\Delta t}(t - t_{n-1}).$$

When inserted into (5.1.18), this linear polynomial yields

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \int_{t_n}^{t_{n+1}} \left(f_{n-1} + \frac{f_n - f_{n-1}}{\Delta t}(t - t_n) \right) dt$$

一、微分方程

微分方程 的定义：凡是含有 参数、未知函数、未知函数的导数（或者微分）的方程

定义式： $F(x, y, y', y'', \ldots, y^{(n)})$

- 常微分方程：未知函数是一元函数
- 偏微分方程：未知函数是多元函数

微分方程的 阶：未知函数最高阶导数的阶数（注意和后面龙格-库塔法的阶做区分）

例01：

$$\begin{cases} \frac{dy}{dx} = 2x \\ y|_{x=1} = 2 \end{cases}$$ 就是一个一阶微分方程

【例2.1】一曲线通过点(1,2)，且在该曲线上任一点M(x,y)处的切线斜率为$2x$，求该曲线的方程。

解：设所求曲线为$y=y(x)$，由已知条件可得

$$\begin{cases} \frac{dy}{dx} = 2x \\ y|_{x=1} = 2 \end{cases}$$ ——此即为**微分方程**

$$\Rightarrow dy = 2xdx \Rightarrow \int dy = \int 2xdx \Rightarrow y = x^2 + c$$ c为积分常数

由另一已知条件： $y|_{x=1} = 2 \Rightarrow 2 = 1^2 + c \Rightarrow c = 1$

故函数解析表达式为： $y = x^2 + 1$

例02：

$$\begin{cases} \frac{d^2S}{d^2t} = -0.4 \\ S|_{t=0} = 0 \\ (\frac{dS}{dt})|_{t=0} = 20 \end{cases}$$

就是一个二阶微分方程

【例2.2】列车在平直线路上以20m/s的速度行驶，当制动时，列车获得加速度-0.4m/s^2，求从制动开始到停止所需的时间t以及列车滑行距离s的关系。

解：由已知条件得如下

$$\begin{cases} \dfrac{d^2S}{dt^2} = -0.4 \\ S\Big|_{t=0} = 0,\ (\dfrac{dS}{dt})_{t=0} = 20 \end{cases}$$

$$\frac{d^2S}{dt^2} = -0.4 \Rightarrow \int \frac{d^2S}{dt^2}dt = \int -0.4dt \Rightarrow \frac{dS}{dt} = -0.4t + c_1$$

$$\Rightarrow \int \frac{dS}{dt}dt = \int (-0.4t + c_1)dt \Rightarrow S = -0.2t^2 + c_1t + c_2$$

二、常微分方程的数值求解

求解微分方程的困难：

- 求解微分方程往往计算量很大，甚至无法求解
- 在实际应用中，可以解出 **近似值** 即可——**数值求解**

2.1 数值求解：

概念：利用给定常微分方程及边界条件，求解函数 $y(x)$ 在若干 **离散点** 的 **近似值**（再拟合成折现或曲线）的方法

即：在区间 $[a,b]$ 上有若干离散点： $a = x_0 < x_1 < \ldots < x_n = b$ ，求出函数 $y(x)$ 在**离散点** x_k 处的**近似值** y_k ，作为**精确值** $y(x_k)$ 的近似

（这里强调一下： x_k 是第k个离散点； y_k 是**近似值**； $y(x_k)$ 是**精确值**）

2.2 数值求解法分类

分类方式一：

- **单步法：**计算 y_{i+1} 时，只用到了前面的一个近似值 y_i
- **多步法：**计算 y_{i+1} 时，用到了前面的多个近似值 y_i、y_{i-1}、$y_{y-2}\cdots$
  一般单步法的运算量少，但是精度不高

3.1 欧拉法（欧拉折线法）

令常微分方程：

$$\begin{cases} y'(x) = f(x,y) \\ y|_{x=0} = y_0 \end{cases}, (a = x_0 \leqslant x_1 \leqslant \ldots \leqslant x_n = b)$$

起始点 y_0 已知 ， h 等距步长 = $x_{k+1} - x_k$ ，步长取的越小，运算精度越高，但运算量增大

欧拉公式： $y_{k+1} = y_k + h * f(x_k, y_k)$

例03（欧拉法 数值求解微分方程）

【例2.3-2】求解常微分方程初值问题，取步长$h=0.1$，计算到$t=0.5$。

$$y' = -y + t + 1 \qquad t \geq 0$$
$$y(0) = 1$$

解： $y_{i+1} = y_i + h f(t_i, y_i) = y_i + 0.1 \times (-y_i + t_i + 1) = 0.9y_i + 0.1t_i + 0.1$ ← 欧拉公式得出

数值解：$y_{i+1} = 0.9\, y_i + 0.1t_i + 0.1$ ← 几个离散点的数值解，需要解出前一个，才能接触后一个

解析解：$y = t + e^{-t}$

i:	0	1	2	3	4	5
t_i:	0	0.1	0.2	0.3	0.4	0.5
y_i:	1.0	1.0	1.01	1.029	1.0561	1.0905
$y(t)$:	1.0	1.0048	1.0187	1.0408	1.0703	1.1065

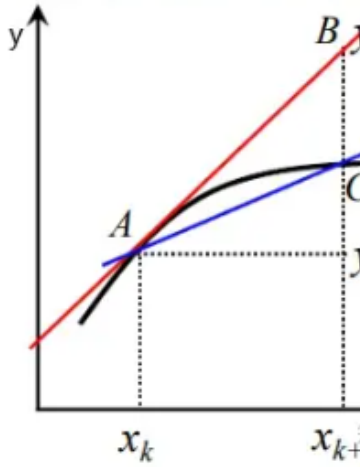
← 欧拉法的数值解(近似解)

← 解析解(精确解)

- **几何意义一：折线**，用 **步长** 和 **前面一个点的斜率** 计算后面一个点

（注意下图的纵坐标是 y ）

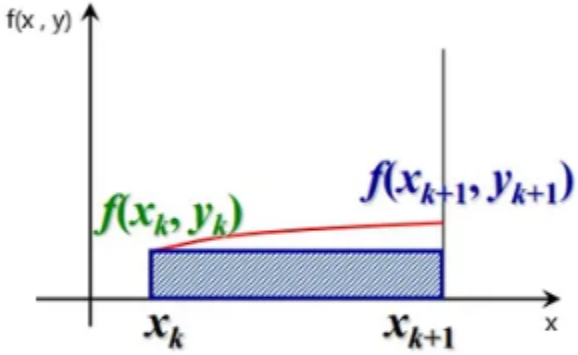
用线段AB近似代替曲线AC



- **几何意义二：用 矩形面积 代替 曲边梯形 面积**

注意：图中的纵坐标是 $f(x,y)$ 即上一张图的斜率， y 是面积

(1) 用矩形公式作近似计算：



几何意义：
用矩形面积近似代替曲边梯形面积

$$y_{k+1} - y_k = \int_{x_k}^{x_{k+1}} f(x,y)dx$$

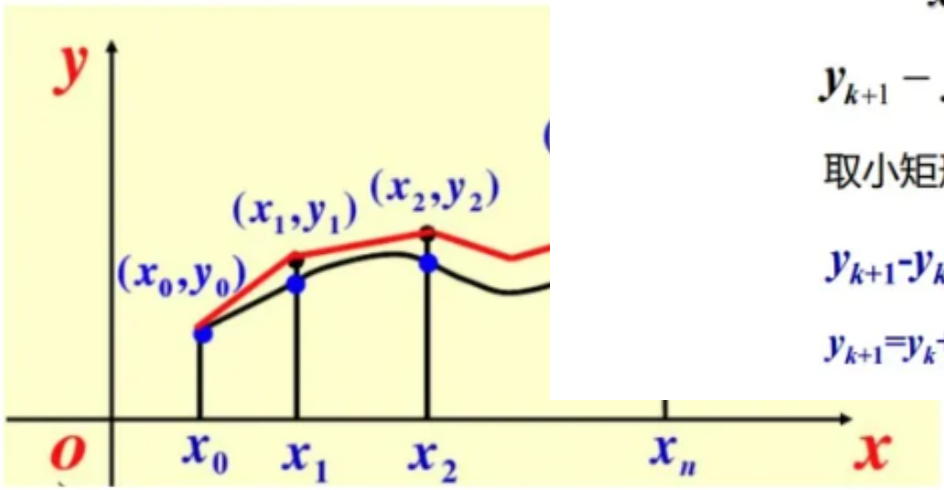
取小矩形面积：

$$y_{k+1}-y_k \approx (x_{k+1}-x_k)f(x_k,y_k)=hf(x_k,y_k)$$

$y_{k+1}=y_k+hf(x_k,y_k)$— 即得**欧拉公式**

注:
f(x , y)是图中的纵坐标
y是面积

欧拉公式的几何意义



欧拉法的几何意义：用一条折线近似代替积分曲线。

3.3 龙格——库塔法（Runger—Kutta法，简称R-K法

将函数 $y(x)$ 在 x_n 处泰勒展开：

$$y(x)=y(x_n)+y'(x_n)(x-x_n)+\frac{y''(x_n)}{2!}(x-x_n)^2+\frac{y^{(3)}(}{3}$$

n阶龙格-库塔法：取泰勒展开的前n阶导数项

将 $x=x_{n+1}$ 带入 $y(x)$ 计算出近似值 y_{n+1}

3.3.4 龙格-库塔法的一般形式

p阶龙格-库塔法： y_{n+1} 的值用 $f(x,y)$ 在某些点上函数值的**线性组合**成来计算，计算$f(x,y)$的次数是龙格-库塔法的阶，也是局部截断误差的阶。

从几何意义上看：龙格-库塔法可以看成用 p 个斜率 $K_r=f(x_r,y_r)$ 的一种 **加权平均逼近** 。
如：欧拉法只用了一个斜率 $f(x_n,y_n)$ ，即 K_1 ；而改进欧拉法是
$\frac{h}{2}f(x_k,y_k)+\frac{h}{2}f(x_{k+1},\tilde{y}_{k+1})$ ，即 $\frac{h}{2}K_1+\frac{h}{2}K_2$

四阶龙格-库塔法（常用）：

$$\begin{cases}y_{n+1}=y_n+\frac{h}{6}(K_1+2K_2+2K_3+K_4)\\K_1=f(x_n,y_n)\\K_2=f(x_n+\frac{h}{2},y_n+\frac{h}{2}K_1)\\K_3=f(x_n+\frac{h}{2},y_n+\frac{h}{2}K_2)\\K_4=f(x_n+h,y_n+hK_3)\end{cases}$$

（**再次强掉：** K_1,K_2,K_3,K_4 前面的系数是 **加权平均** 得到，计算顺序是从 K_1 到 K_4 ，根据这个规律特征，可以推出后续更高阶的龙格-库塔法）

> 问：为什么需要推出，不能直接写出其p阶的一般表达式吗。
> 回答：和”斐波那契数列“是一个道理，因为一般表达式很复杂，还不如推的简单。其一般形式如下

用了p个K_r的龙格-库塔法成为p阶龙格-库塔法，其一般形式：

$$\begin{cases}y_{n+1}=y_n+h\sum_{r=1}^{p}c_rK_r\\K_1=f(x_n,y_n)\\K_r=f(x_n+a_rh,y_n+h\sum_{s=1}^{r-1}b_{rs}K_s),r=2,\cdots,p\end{cases}$$

式中 λ_r,a_r,b_{rs} 等均为常数； $p\geq 1$为整数。

问：为什么需要推出，不能直接写出其p阶的一般表达式吗。
回答：和”斐波那契数列“是一个道理，因为一般表达式很复杂，还不如推的简单。其一般形式如下

用了p个K_r的龙格-库塔法成为p阶龙格-库塔法，其一般形式：

$$\begin{cases} y_{n+1} = y_n + h\sum_{r=1}^{p} c_r K_r \\ K_1 = f(x_n, y_n) \\ K_r = f(x_n + a_r h, y_n + h\sum_{s=1}^{r-1} b_{rs} K_s), r = 2, \cdots, p \end{cases}$$

式中 λ_r, a_r, b_{rs} 等均为常数； $p \geq 1$为整数。

例04（4阶龙格-库塔法 数值求解微分方程）

【例2.7-1】用四阶龙格-库塔法求解初值问题：

$$\begin{cases} \frac{dy}{dx} = x - y + 1 \\ y(0) = 1.0 \end{cases}$$，求解区间$0.0 \leq x \leq 0.5$

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_1) \\ K_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_2) \\ K_4 = f(x_n + h, y_n + hK_3) \end{cases}$$

解：由微分方程可得$f(x, y)=x-y+1$

取$h=0.1$，$x_0=0.0, x_1=0.1, x_2=0.2, x_3=0.3, x_4=0.4, x_5=0.5$.

求y_1：$K_1=f(x_0, y_0)=0.0-1.0+1.0=0.0$;

$K_2=f(x_0+0.5h, y_0+0.5hK_1)=0.05-1.0+1.0=0.05$;

$K_3=f(x_0+0.5h, y_0+0.5hK_2)=0.05-1.0-0.05*0.05+1.0=0.0475$

$K_4=f(x_0+h, y_0+hK_3)=0.1-1.0-0.1*0.0475+1.0=0.09525$

$y_1=y_0+h(K_1+2K_2+2K_3+K_4)/6$

$=1.0+0.1(0.0+0.1+0.095+0..09525)/6=1.0048375$

精确值$y(0.1)=1.004837$

而欧拉法：$y_1=1.0$　改进欧拉法：$y_1=1.005$

补充：二阶微分方程的 4阶R-K法

对于2阶微分方程，则需要拆分成2个一阶微分方程，此时的 f(x,y) 变成了 f(x,y,y') 需要分别用R-K法计算 y 和 y'

已知初值 $y(0)$ 和 $y'(0)$,以及 y'' 关于 x, y, y' 的函数，即 $y'' = f(x, y, y')$

例题：用4阶R-K法求下面的二阶微分方程的数值解，步长 h 取0.1

$$\begin{cases} 2\frac{d^2y}{dt} + (y^2 - 1)\frac{dy}{dt} + y = 0 \\ y(0) = 0.25 \\ y'(0) = 0 \end{cases}$$

即

$$\begin{cases} y'' = -\frac{y + y'(y^2-1)}{2} \\ y(0) = 0.25 \\ y'(0) = 0 \end{cases}$$

解：

将二阶微分方程拆成2个一阶微分方程，R-K法计算 y 和 y'

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(K_{11} + 2K_{12} + 2K13 + K_{14}) \\ y'_{n+1} = y'_n + \frac{h}{6}(K_{11} + 2K_{12} + 2K13 + K_{14}) \end{cases}$$

解：

将二阶微分方程拆成2个一阶微分方程，R-K法计算 y 和 y'

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(K_{11} + 2K_{12} + 2K13 + K_{14}) \\ y'_{n+1} = y'_n + \frac{h}{6}(K_{11} + 2K_{12} + 2K13 + K_{14}) \end{cases}$$

$$\begin{cases} K_{11} = f_1(x_n, y_n, y'_n) \\ K_{21} = f_2(x_n, y_n, y'_n) \\ \\ K_{12} = f_1(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_{11}, y'_n + \frac{h}{2}K_{21}) \\ K_{22} = f_2(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_{11}, y'_n + \frac{h}{2}K_{21}) \\ \\ K_{13} = f_1(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_{12}, y'_n + \frac{h}{2}K_{22}) \\ K_{23} = f_2(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_{12}, y'_n + \frac{h}{2}K_{22}) \\ \\ K_{14} = f_1(x_n + h, y_n + hK_{13}, y'_n + hK_{23}) \\ K_{24} = f_2(x_n + h, y_n + hK_{13}, y'_n + hK_{23}) \end{cases}$$

其中：K_{mn} 表示第 n 次迭代，m 次导数作为 $f(x, y)$ ；

其中：$\begin{cases} f_1 = y' \\ f_2 = y'' = -\frac{y + y'(y^2-1)}{2} \end{cases}$

然后根据上边的步骤，一步步计算出 y_{n+1} 和 y'_{n+1}