第九章 常微分方程初值问题的数值解法

内容提要

9.1 引言

9.2 简单的数值方法

9.3 龙格-库塔方法

9.4 单步法的收敛性与稳定性

9.5 线性多步法

9.7 一阶ODE方程组及高阶方程

9.1 引言

虽然求解微分方程有许多解析方法，但解析方法只能够求解一些特殊类型的方程。从实际意义上来讲，我们更关心的是某些 特定的自变量在某一个定义范围内的一系列离散点上的近似值。这组近似解称为微分方程在该范围内的数值解，寻找数值解的过程称为数值求解微分方程。

本章着重讨论一阶常微分方程初值问题

$$\begin{cases} y' = \dfrac{\mathrm{d}y}{\mathrm{d}x} = f(x,y), \quad x \in [x_0, b] \\ y(x_0) = y_0 \end{cases} \tag{9.1}$$

的数值解法。

domain定义域
range值域

定理9-1 设 f 在区域 $D=\{(x,y)\,|\,a\le x\le b, y\in R\}$ 上连续，关于 y 满足利普希茨条件，即 $|f(x,y)-f(x,\bar{y})|\le L|y-\bar{y}|$ 其中$L>0$，则对任意 $x_0\in[a,b]$，$y_0\in R$，常微分方程初值问题(9.1) 当 $x\in[a,b]$ 时存在唯一的连续可微解 $y(x)$。

Lipschitz条件 (Lipschitz Condition)

Definition 9.2. Given the rectangle $R = \{(t, y) : a \le t \le b, c \le y \le d\}$, assume that $f(t, y)$ is continuous on R. The function f is said to satisfy a ***Lipschitz condition*** in the variable y on R provided that a constant $L > 0$ exists with the property that

$$|f(t, y_1) - f(t, y_2)| \le L\,|y_1 - y_2| \tag{8}$$

whenever $(t, y_1), (t, y_2) \in R$. The constant L is called a ***Lipschitz constant*** for f. ▲

Theorem 9.1. Suppose that $f(t, y)$ is defined on the region R. If there exists a constant $L > 0$ so that

$$|f_y(t, y)| \le L \quad \text{for all } (t, y) \in R, \tag{9}$$

then f satisfies a Lipschitz condition in the variable y with Lipschitz constant L over the rectangle R.

数值解法：就是寻求解 $y(x)$ 在一系列离散节点

$$x_1 < x_2 < \cdots < x_n < x_{n+1} < \cdots$$

上的近似值 $y_1, y_2, \cdots, y_n, y_{n+1}, \cdots$。相邻两节点间的间距 $h_n = x_{n+1} - x_n$ 称为步长。假定 $h_i = h$ (其中$i = 0,1,\cdots$) 为常数，这时节点为 $x_n = x_0 + nh,$ 其中$n = 0,1,2,\cdots$

初值问题数值解法的基本特点：它们都采用 “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，给出已知信息 $y_n, y_{n-1}, y_{n-2}, \cdots$ 和计算 y_{n+1} 的递推公式。

初值问题数值解法中一类是计算 y_{n+1} 时只用到前一点的值 y_n，称为单步法。另一类是用到 y_{n+1} 前面k 点的值 $y_n, y_{n-1}, \cdots, y_{n-k+1}$，称为 k 步法。

为了考察数值方法提供的数值解，是否有实用价值，需要知道如下3个结论：

① 步长充分小时，所得到的数值解能否逼近问题的真解；即收敛性问题

② 误差估计

③ 产生的舍入误差，在以后的各步计算中，是否会无限制扩大；即稳定性问题

单步法：在计算y_{i+1}时只利用y_i

多步法：在计算y_{i+1}时不仅利用y_i，还要利用 $y_{i-1}, y_{i-2}, \cdots,$

k步法：在计算y_{i+1}时要用到y_i，y_{i-1}，…，y_{i-k+1}

显式计算公式可写成: $y_{k+1}=y_k+h\Phi_f(x_k, y_k; h)$

隐式格式: $y_{k+1}=y_k+h\Phi_f(x_k, y_k, y_{k+1}; h)$

它每步求解y_{k+1}需要解一个隐式（非线性）方程

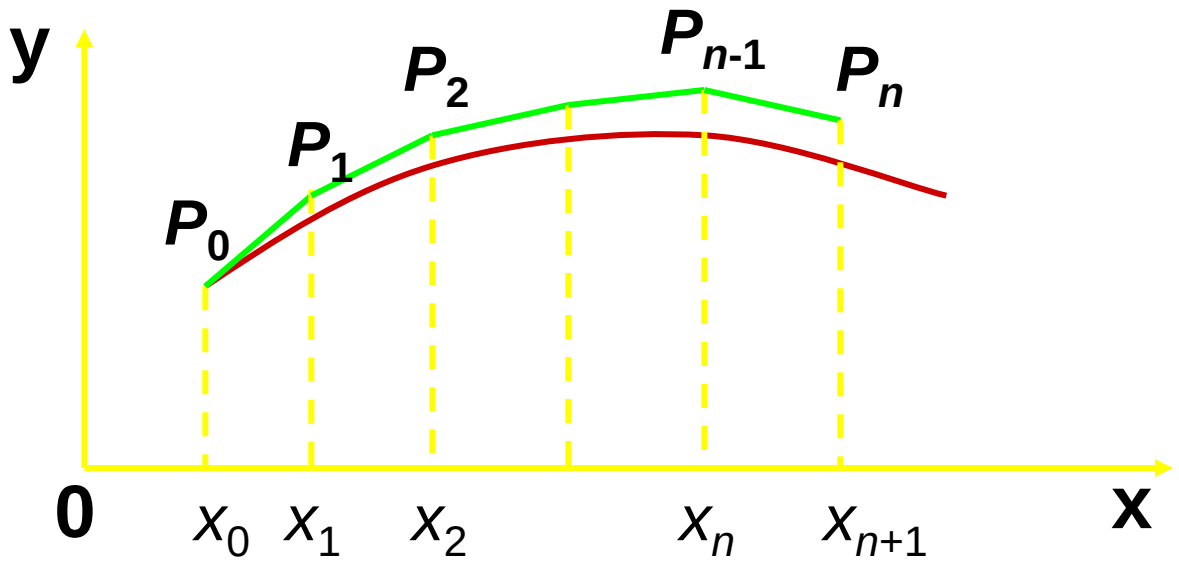
Newton法

Euler 1755, 1768, 1770

9.2 简单的数值方法

一、四种简单的数值方法

1、欧拉方法



Euler前向差商公式
（数值微分/求导）

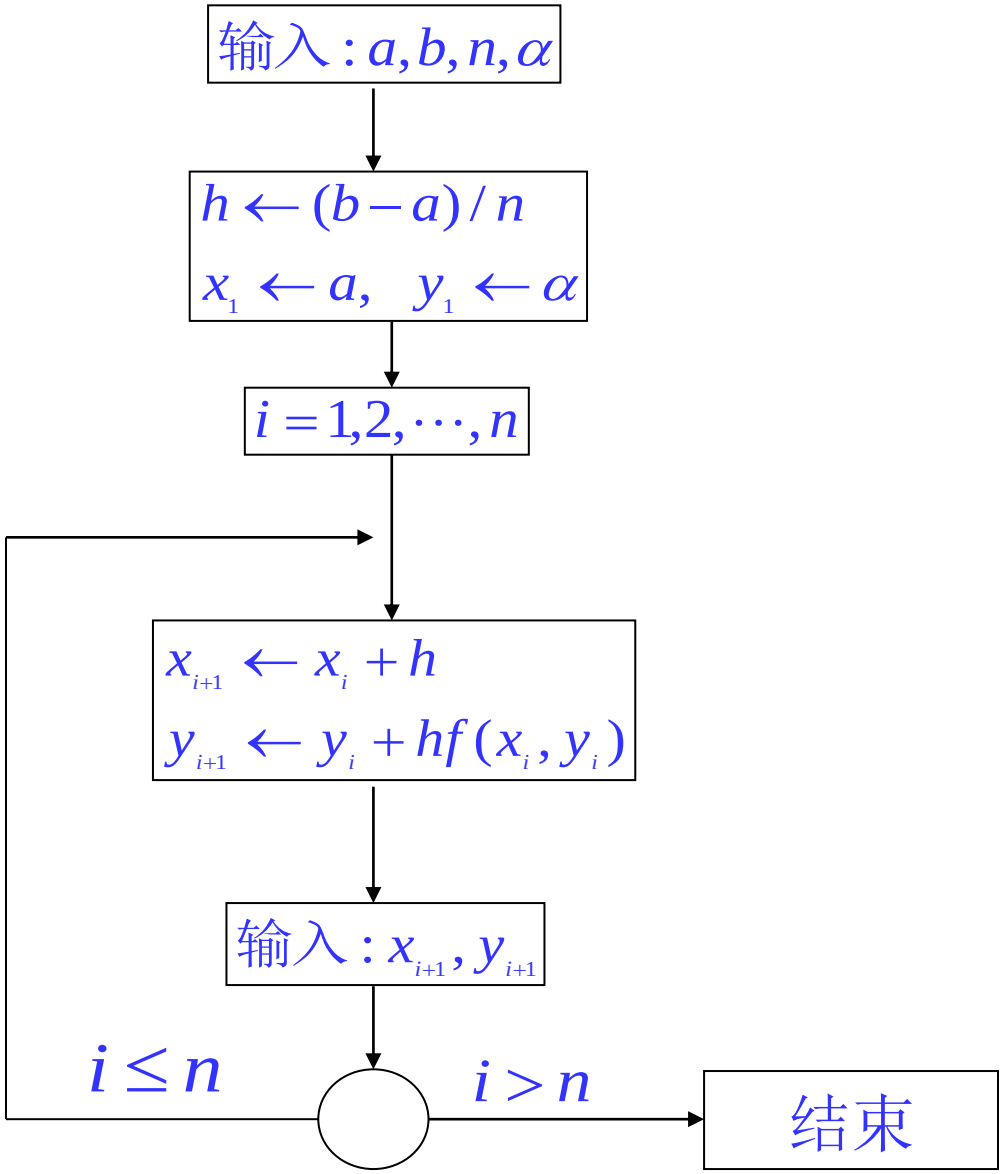
一般地，设已做出该折线的顶点P_n，过$P_n(x_n, y_n)$依方向场的方向再推进$P_{n+1}(x_{n+1}, y_{n+1})$，显然两个顶点$P_n$，$P_{n+1}$的坐标有关系

$$\frac{y_{n+1}-y_n}{x_{n+1}-x_n}=f(x_n,y_n), \qquad \text{即 } y_{n+1}=y_n+hf(x_n,y_n)$$

推导

这就是欧拉(Euler)公式，这种求解问题的方法称为欧拉方法。

1）特式
2）通式
3）算法描述或框图
4）（细化的）流程图
5）伪代码
6）MATLAB等实际编码



Euler法流程图

Euler法 (MATLAB的M文件)

```
function [t, y] = Eulode(dydt, tspan, y0, h)
% [t, y] = Eulode(dydt, tspan, y0, h)
%   uses Euler's method to integrate an ODE
% input
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf] where ti and tf = initial and final
%           values of independent variable
% output
%   t = vector of independent variable
%   y = vector of solution of dependent variable

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n) < tf
    t(n+1) = tf;
    n = n + 1;
end
y = y0*ones(n,1); % preallocate y to improve efficiency
for i = 1: n-1    % implement Euler's method
    y(i+1) = y(i) + feval(dydt,t(i),y(i))*(t(i+1)-t(i));
a = tspan(1); b = tspan(2); h = (b - a) / n;
end
disp('   step         t                y')
k = 1:length(t); out = [k; t'; y'];
fprintf('%5d  %15.10f %15.10f\n',out)
```

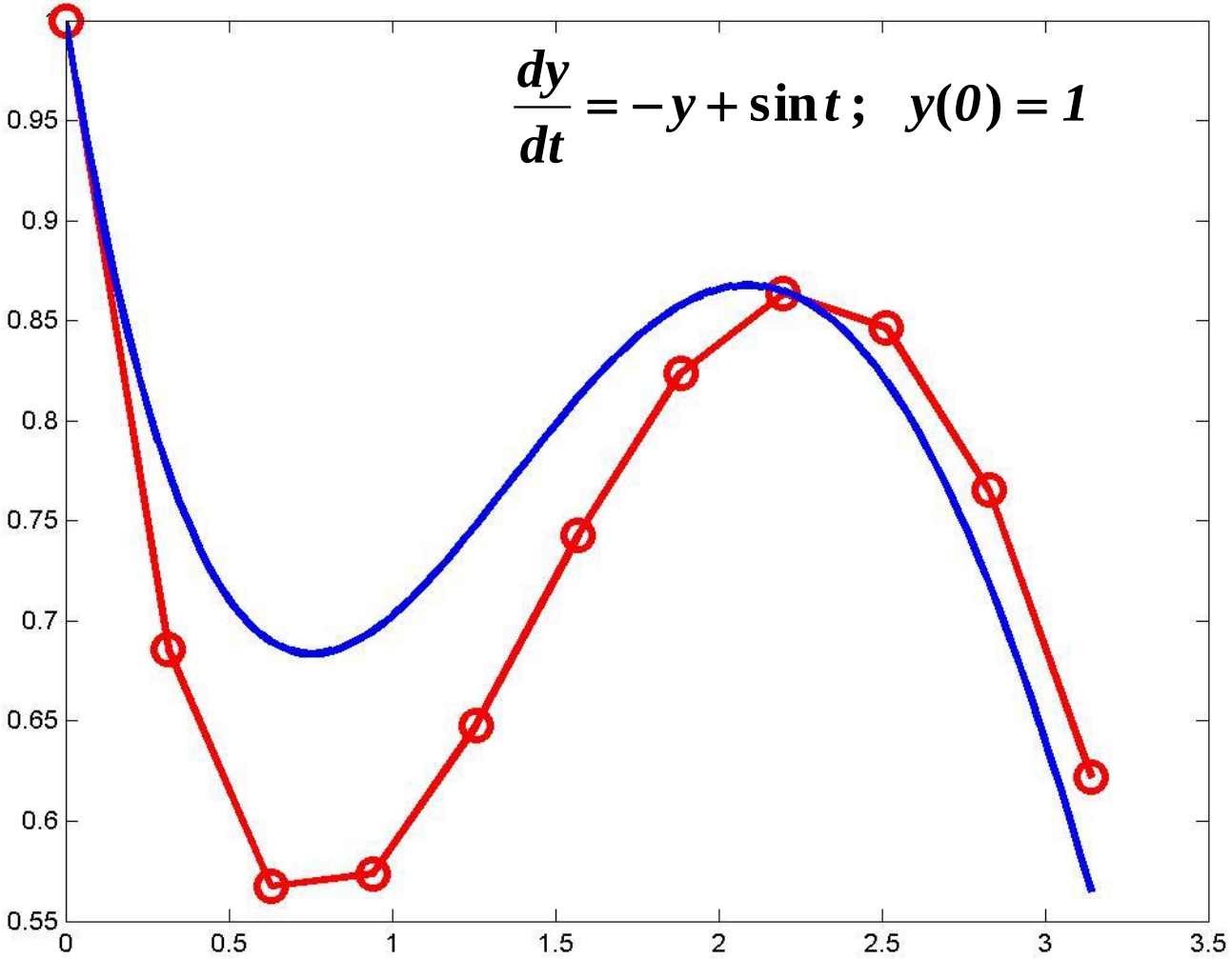
cf. fever

Euler法

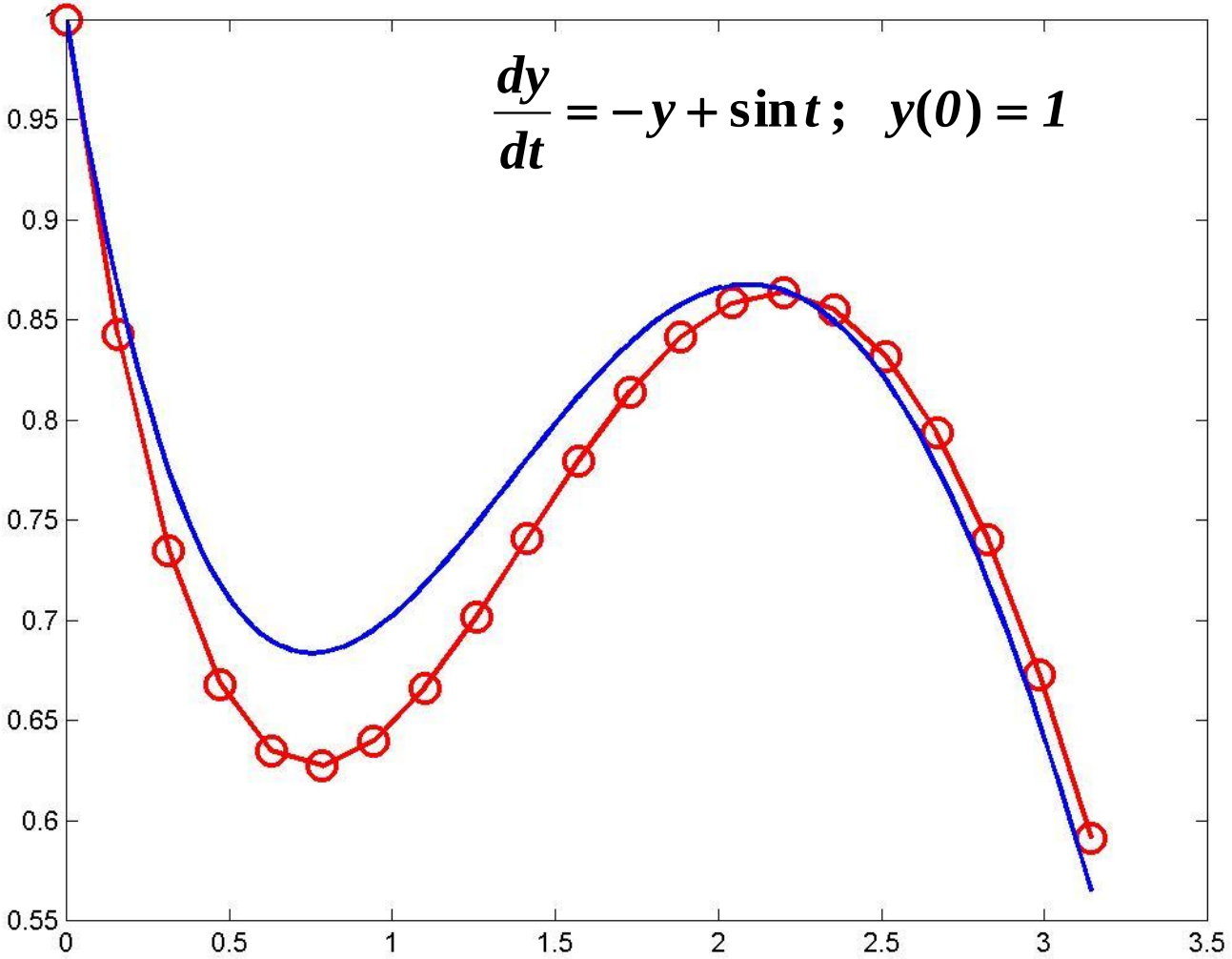
```
>> tt=0:0.01*pi:pi;
>> f=inline('-y+sin(t)','t','y')
>> ye=1.5*exp(-tt)+0.5*sin(tt)-0.5*cos(tt);
>> [t,y]=Eulode(f,[0 pi],1,0.1*pi);
   step          t                    y
    1      0.0000000000     1.0000000000
    2      0.3141592654     0.6858407346
    3      0.6283185307     0.5674580652
    4      0.9424777961     0.5738440394
    5      1.2566370614     0.6477258022
    6      1.5707963268     0.7430199565
    7      1.8849555922     0.8237526182
    8      2.1991148575     0.8637463173
    9      2.5132741229     0.8465525934
   10      2.8274333882     0.7652584356
   11      3.1415926536     0.6219259596
>> plot(t,y,'r-o',tt,ye,'LineWidth',3,'MarkerSize',12);
>> print -djpeg ode01.jpg
```

$$\frac{dy}{dt} = -y + \sin t\,; \quad y(0) = 1$$

Euler法 ($h = 0.1\pi$)



Euler法 ($h = 0.05\pi$)



计算过程：

若初值y_0已知，则依公式可逐步算出

$$y_1 = y_0 + hf(x_0, y_0)$$

$$y_2 = y_1 + hf(x_1, y_1)$$

$$\cdots$$

例9-1　利用欧拉方法求解初值问题

$$\begin{cases} y' = y - \dfrac{2x}{y} & (0 \le x \le 1) \\ y(0) = 1 \end{cases}$$

步长 $h = 0.1$

解：求解本题的欧拉公式为

$$y_{n+1} = y_n + h\left(y_n - \frac{2x_n}{y_n}\right)$$

展开计算

取步长$h = 0.1$, 计算结果见下表

x_n	y_n	$y(x_n)$		x_n	y_n	$y(x_n)$
0.1	1.1000	1.0954		0.6	1.5090	1.4832
0.2	1.1918	1.1832		0.7	1.5803	1.5492
0.3	1.2774	1.2649		0.8	1.6498	1.6125
0.4	1.3582	1.3416		0.9	1.7178	1.6733
0.5	1.4351	1.4142		1.0	1.7848	1.7321

为对比计算精度，利用解析解得求解结果也在表中给出。通过与解析解对比可知欧拉方法的精度较差。

为了分析计算公式的精度，通常可用泰勒展开将$y(x_{n+1})$在x_n处展开，则有

$$y(x_{n+1}) = y(x_n + h)$$

$$= y(x_n) + y'(x_n)h + \frac{h^2}{2} y''(\xi_n), \quad \xi_n \in (x_n, x_{n+1})$$

在 $y_n = y(x_n)$ 的前提下，$f(x_n, y_n) = f(x_n, y(x_n)) = y'(x_n)$。

得欧拉法的误差

$$y(x_{n+1}) - y_{n+1} = \frac{h^2}{2} y''(\xi_n) \approx \frac{h^2}{2} y''(x_n)$$

称为此方法的局部截断误差。

2、后退的欧拉法(也称为隐式欧拉法，后向欧拉法)

数值积分也可

如果对方程(9.1) 从x_n 到x_{n+1} 积分，得

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(t, y(t))\mathrm{d}t \qquad (9.2)$$

右端积分用左矩形公式 $hf(x_n, y(x_n))$ 近似，再以 y_n 代替 $y(x_n)$ y_{n+1} 代替 $y(x_{n+1})$ 也得到欧拉公式。

如果上式右端积分用右矩形公式 $hf(x_{n+1}, y(x_{n+1}))$ 近似，则得另一个公式

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

称为后退（后向）的欧拉公式。

Euler后向差商公式
（数值微分/求导）

欧拉方法是显式的，后退的欧拉方法是隐式的。

隐式公式可以通过迭代法求解，因此隐式欧拉方法一般是应用迭代法求解。迭代过程的实质是逐步显化过程。具体方法如下。

设用欧拉公式

$$y_{n+1}^{(0)} = y_n + hf(x_n, y_n)$$

给出迭代初值 $y_{n+1}^{(0)}$，用它代入隐式右端，使之转化为显式，直接计算得

$$y_{n+1}^{(1)} = y_n + hf(x_{n+1}, y_{n+1}^{(0)})$$

如此反复进行，得

$$y_{n+1}^{(k+1)} = y_n + hf(x_{n+1}, y_{n+1}^{(k)}) \qquad (k = 0,1,\cdots) \qquad (9.3)$$

由于 $f(x,y)$ 对 y 满足利普希茨条件。于是得

$$\left|y_{n+1}^{(k+1)} - y_{n+1}\right| = h\left|f(x_{n+1}, y_{n+1}^{(k)}) - f(x_{n+1}, y_{n+1})\right| \le hL\left|y_{n+1}^{(k)} - y_{n+1}\right| \le \cdots \le (hL)^{k+1}\left|y_{n+1}^{(0)} - y_{n+1}\right|$$

只要 $hL < 1$ 迭代法(9.3) 就收敛到解 y_{n+1} 。

课堂推导

步长h要足够小

3、梯形方法

等式(9.2)右端积分中若用梯形公式近似。

并用 y_n 代替 $y(x_n)$，用 y_{n+1} 代替 $y(x_{n+1})$，则得到梯形方法

$$y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1}) \right]$$

梯形方法是隐式单步法，可以利用迭代法求解。迭代公式为

$$\begin{cases} y_{n+1}^{(0)} = y_n + hf(x_n, y_n) \\ y_{n+1}^{(k+1)} = y_n + \dfrac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(k)}) \right] \\ \qquad\qquad (k = 0, 1, 2, \cdots) \end{cases} \tag{9.4}$$

接下来给出收敛性分析

$$y_{n+1} - y_{n+1}^{(k+1)} = \frac{h}{2}\left[f(x_{n+1}, y_{n+1}) - f(x_{n+1}, y_{n+1}^{(k)}) \right]$$

于是有

$$\left|y_{n+1}-y_{n+1}^{(k+1)}\right| \le \frac{hL}{2}\left|y_{n+1}-y_{n+1}^{(k)}\right| \le \cdots \le \left(\frac{hL}{2}\right)^{k+1}\left|y_{n+1}-y_{n+1}^{(0)}\right|$$

课堂推导

L 为 $f(x,y)$ 关于 y 的利普希茨常数。若 h 充分小使得 $\frac{hL}{2}<1$,

则当 $k\to\infty$ 时有 $y_{n+1}^{(k)}\to y_{n+1}$，迭代过程收敛。

此处h范围比后向Euler法大些

例9-2　用梯形方法求解初值问题

$$\begin{cases} y'=x+y, & 0\le x\le 0.5 \\ y(0)=1 \end{cases}$$

取步长$h=0.1$。

解：梯形方法计算公式为

$$y_{n+1}=y_n+\frac{1}{2}h\left[x_n+y_n+x_{n+1}+y_{n+1}\right]$$

解得

$$y_{n+1}=\frac{1}{1-h/2}\left[\left(1+\frac{h}{2}\right)y_n+\frac{h(x_n+x_{n+1})}{2}\right], \qquad n=0,1,\cdots,4$$

x_n	y_n	$\lvert y(x_n)-y_n\rvert$
0.1	1.110526	0.000184479
0.2	1.243213	0.000407779
0.3	1.400393	0.000676027
0.4	1.584645	0.000996210
0.5	1.798818	0.001376285

为对比计算精度，利用解析解 $y=-x-1+2e^{x}$ 求解结果也在表中给出。通过与解析解对比可知梯形方法的精度较好。

另外说明，此题函数关系比较简单，没有采用迭代法求解，而是将直接将公式整理成显格式求解。

例9-3 用梯形方法求解初值问题

$$\begin{cases} y'=\dfrac{2}{y-x}+1, & 0\le x\le 1 \\ y(0)=1 \end{cases}$$

取$h=0.2$，要求$\left|y_{n+1}^{[k+1]}-y_{n+1}^{[k]}\right|<10^{-4}$

解：用梯形方法的迭代计算公式为

$$\begin{cases} y_{n+1}^{[0]} = y_n + hf(x_n, y_n) \\ y_{n+1}^{[k+1]} = y_n + \dfrac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{[k]})] \end{cases}$$

计算结果如下：

$y_1^{[0]} = 1.6$　　$y_1^{[1]} = 1.542857$

$y_1^{[2]} = 1.548936$　　$y_1^{[3]} = 1.548265$

$y_1^{[4]} = 1.548339$

课堂演算

此时 $\left|y_{n+1}^{[4]} - y_{n+1}^{[3]}\right| < 10^{-4}$，所以$y(0.2) \approx y_1^{[4]} = 1.548339$

同样方法，解得

$y(0.4) \approx 2.020118,$　　$y(0.6) \approx 2.451578$

$y(0.8) \approx 2.8565830,$　　$y(1.0) \approx 3.243224$

4、改进的欧拉公式（Heun法）

先用欧拉公式求得一个初步的近似值 $\bar{y}_{n+1}$ 称之为预测值。再用梯形公式校正一次，迭代一次得 y_{n+1} 称校正值，这样建立的“预测-校正系统”通常称为改进的欧拉公式。

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

校正：$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \bar{y}_{n+1})]$

为了便于编写程序通常表示为平均化的形式

$$\begin{cases} y_{\mathrm{p}} = y_n + hf(x_n, y_n) \\ y_{\mathrm{c}} = y_n + hf(x_{n+1}, y_{\mathrm{p}}) \\ y_{n+1} = (y_{\mathrm{p}} + y_{\mathrm{c}})/2 \end{cases}$$

例9-4　用改进的欧拉方法求解初值问题例9-1

解：改进的欧拉公式为

$$\begin{cases} y_{\mathrm{p}} = y_n + h\left(y_n - \dfrac{2x_n}{y_n}\right) \\ y_{\mathrm{c}} = y_n + h\left(y_{\mathrm{p}} - \dfrac{2x_{n+1}}{y_{\mathrm{p}}}\right) \\ y_{n+1} = \dfrac{1}{2}(y_{\mathrm{p}} + y_{\mathrm{c}}) \end{cases}$$

x_n	y_n	$y(x_n)$		x_n	y_n	$y(x_n)$
0.1	1.0959	1.0954		0.6	1.4860	1.4832
0.2	1.1841	1.1832		0.7	1.5525	1.5492
0.3	1.2662	1.2649		0.8	1.6153	1.6125
0.4	1.3434	1.3416		0.9	1.6782	1.6733
0.5	1.4164	1.4142		1.0	1.7379	1.7321

可以看出：改进的欧拉方法比欧拉方法精度高。

FGE（final global error）

泰勒展开

$$y(x_{n+1})=y(x_n+h)=y(x_n)+y'(x_n)h+\frac{y''(x_n)}{2!}h^2+\frac{y'''(x_n)}{3!}h^3+\cdots$$

$$f(x_n+h,y_n+k)=f(x_n,y_n)+\frac{\partial f(x_n,y_n)}{\partial x}h+\frac{\partial f(x_n,y_n)}{\partial y}k$$
$$+\frac{1}{2!}\left[\frac{\partial^2 f(x_n,y_n)}{\partial x^2}h^2+2\frac{\partial^2 f(x_n,y_n)}{\partial x\partial y}hk+\frac{\partial^2 f(x_n,y_n)}{\partial y^2}k^2\right]+\cdots$$

另外, 在$y_n=y(x_n)$的条件下, 考虑到$y'(x)=f(x,y(x))$, 则有

$y'(x_n)=f(x_n,y(x_n))=f(x_n,y_n)=f_n$

$$y''(x_n)=\frac{\mathrm{d}}{\mathrm{d}x}[f(x_n,y(x_n))]=\frac{\partial f_n}{\partial x}+\frac{\partial f_n}{\partial y}f_n$$

$$y'''(x_n)=\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial f_n}{\partial x}+\frac{\partial f_n}{\partial y}f_n\right)$$

$$=\frac{\partial^2 f_n}{\partial x^2}+2\frac{\partial^2 f_n}{\partial x\partial y}f_n+\frac{\partial^2 f_n}{\partial y^2}f_n^2+\frac{\partial f_n}{\partial x}\frac{\partial f_n}{\partial y}+\left(\frac{\partial f_n}{\partial y}\right)^2 f_n$$

推导练习

先看，后面有用

二、单步法的局部截断误差与阶

单步法的一般表示形式

$$y_{n+1} = y_n + h\phi(x_n, y_n, y_{n+1}, h)$$

其中ϕ与$f(x,y)$有关，称为增量函数，当ϕ含有y_{n+1}时，方法是隐式的，若不含y_{n+1}则为显式方法。

显式单步法可以表示为

$$y_{n+1} = y_n + h\phi(x_n, y_n, h) \qquad (9.5)$$

格式

定义9-1 设$y(x)$是初值问题（9.1）的准确解，称

$$T_{n+1} = y(x_{n+1}) - y(x_n) - h\phi(x_n, y(x_n), h)$$

为显式单步法(9.5)的局部截断误差。

T_{n+1}之所以称为局部的，是假设在x_n前各步没有误差。局部截断误差可以理解为用方法计算一步的误差，根据定义可以得到欧拉方法的局部截断误差。

$$T_{n+1} = y(x_{n+1}) - y(x_n) - hf(x_n, y(x_n))$$
$$= y(x_n + h) - y(x_n) - hy'(x_n)$$
$$= \frac{h^2}{2} y''(x_n) + O(h^3)$$

数值微分也有误差主项与次项探讨

$\frac{h^2}{2} y''(x_n)$称为局部截断误差的主项，显然$T_{n+1} = O(h^2)$，故欧拉方法是一阶的。

定义9-2 设 $y(x)$ 是初值问题(9.1)的准确解，若存在最大整数 p 使显式单步法(9.5)的局部截断误差满足

$$T_{n+1} = y(x+h) - y(x) - h\phi(x, y, h)$$
$$= O(h^{p+1})$$

则称方法(9.5)具有 p 阶精度。

以上定义对隐式也适用，例如后退欧拉法局部截断误差为

课堂推导

$$T_{n+1} = y(x_{n+1}) - y(x_n) - hf(x_{n+1}, y(x_{n+1}))$$

$$= hy'(x_n) + \frac{h^2}{2} y''(x_n) + O(h^3) - h[y'(x_n) + hy''(x_n) + O(h^2)]$$

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

这里 p=1，后退的欧拉方法是一阶方法，$-\frac{h^2}{2} y''(x_n)$ 为局部截断误差的主项。

再如梯形方法局部截断误差为

课堂推导

$$T_{n+1} = y(x_{n+1}) - y(x_n) - \frac{h}{2}[y'(x_n) + y'(x_{n+1})]$$

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

$$+ hy''(x_n) + \frac{h^2}{2} y'''(x_n)] + O(h^4) = -\frac{h^3}{12} y'''(x_n) + O(h^4)$$

这里 $p = 2$, 梯形方法为二阶方法。$-\frac{h^3}{12} y'''(x_n)$ 为局部截断误差的主项。

Runge 1895, Kutta 1905

9.3 龙格-库塔方法

1、显式龙格-库塔法的一般形式

若要使得到的公式阶数 p 更大，ϕ 就必须包含更多的f 值。为此积分形式的公式表示为

$$y(x_{n+1})-y(x_n)=\int_{x_n}^{x_{n+1}} f(x,y(x))\mathrm{d}x \approx h\sum_{i=1}^{r} c_i f(x_n+\lambda_i h, y(x_n+\lambda_i h))$$

数值积分

或表示为 $$y_{n+1}=y_n+h\phi(x_n,y_n,h)$$

其中 $$\phi(x_n,y_n,h)=\sum_{i=1}^{r} c_i K_i$$

$$K_1=f(x_n,y_n)$$

$$K_i=f\left(x_n+\lambda_i h, y_n+h\sum_{j=1}^{i-1}\mu_{ij}K_j\right) \qquad i=2,\cdots,r$$

这里c_i,λ_i和μ_{ij}均为常数。此方法称为 r 级显式龙格 - 库塔(Runge-Kutta) 法，简称 R-K 方法。特别当 $r=1$ 时，就是欧拉方法。

2、二阶显式R-K方法

$r=2$ 的R-K方法,计算公式如下:

$$\begin{cases} y_{n+1}=y_n+h(c_1K_1+c_2K_2) \\ K_1=f(x_n,y_n) \\ K_2=f(x_n+\lambda_2 h, y_n+\mu_{21}hK_1) \end{cases}$$

这里$c_1,c_2,\lambda_2,\mu_{21}$均为待定常数。我们希望适当选取系数，使公式阶数 p 尽可能高。

利用泰勒展开时可知要使公式具有$p=2$ 阶，必须使

$$\begin{cases} c_2\lambda_2=1/2 \\ c_2\mu_{21}=1/2 \\ c_1+c_2=1 \end{cases} \qquad (9.6)$$

4未知量3方程

此方程组解不唯一。可令$c_2=a\neq 0$, 则得 $c_1=1-a,\ \lambda_2=\mu_{21}=\dfrac{1}{2a}$ 。

这样得到的公式称为二阶R-K 方法。特别$a=1/2$, 则$c_1=c_2=1/2$ ，$\lambda_2=\mu_{21}=1$, 就是改进的欧拉公式（Heun 法）。

若取$a=1$,则$c_2=1, c_1=0, \lambda_2=\mu_{21}=\frac{1}{2}$,得计算公式

$$\begin{cases} y_{n+1}=y_n+hK_2 \\ K_1=f(x_n,y_n) \\ K_2=f(x_n+h/2,y_n+hK_1/2) \end{cases}$$

称为中点公式, 也可写为

$$y_{n+1}=y_n+hf\left(x_n+\frac{h}{2},y_n+\frac{h}{2}f(x_n,y_n)\right)。$$

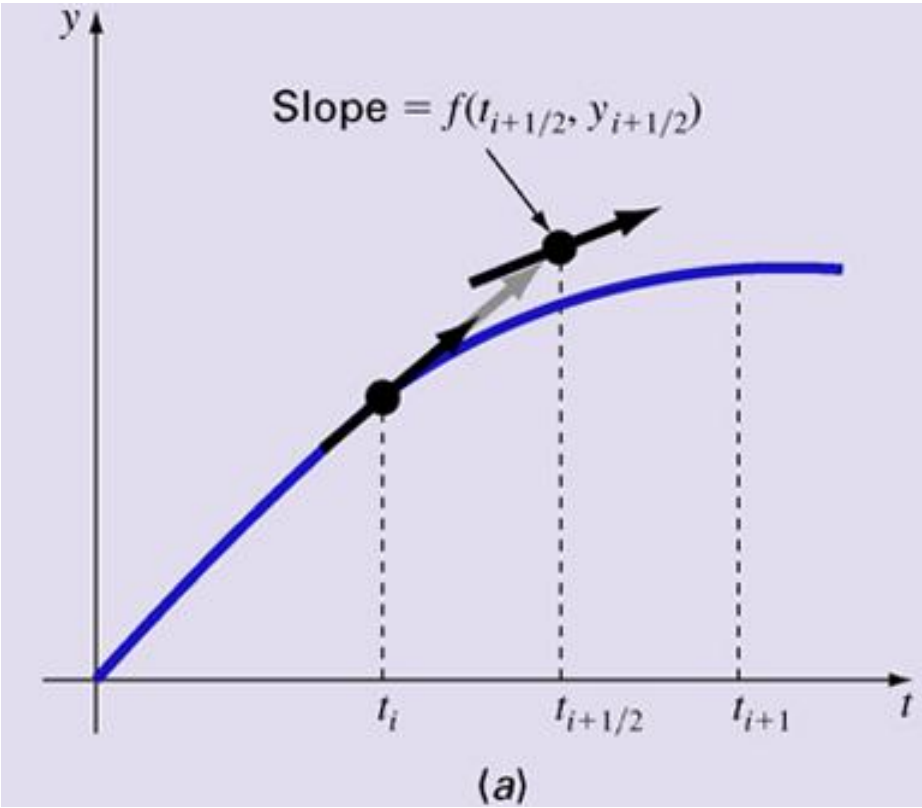
中矩形

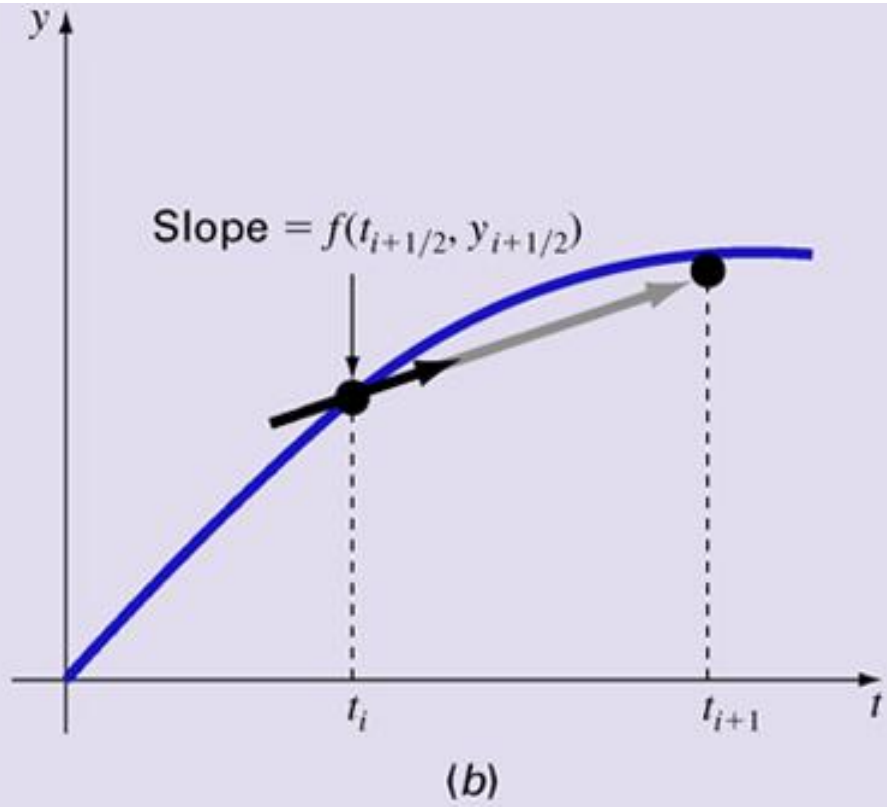
中点法 (Midpoint Method)

- **Use the slope at midpoint to represent the average slope**

$$y_{i+1/2} = y_i + f(t_i, y_i)\frac{h}{2}; \qquad y'_{i+1/2} = f(t_{i+1/2}, y_{i+1/2})$$

$$y_{i+1} = y_i + f(t_{i+1/2}, y_{i+1/2})h$$





中点 (Midpoint, 2阶RK) 法

```
function [t, y] = midpoint(f, tspan, y0, h)
% function [t, y] = midpoint(f, tspan, y0, h)
% solve y' = f(t, y)
% with initial condition y(a) = y0
% using n steps of the midpoint (RK2) method;

a = tspan(1); b = tspan(2); n = (b-a) / h;
t = (a+h : h: b);
k1 = feval(f,a,y0) ;
k2 = feval(f, a + h/2, y0 + k1/2*h);
y(1) = y0 + k2*h;
for i = 1 : n-1
    k1 = feval(f, t(i), y(i));
    k2 = feval(f, t(i) + h/2, y(i) + k1/2*h);
    y(i+1) = y(i) + k2*h;
end
t = [ a   t ]; y = [ y0  y ];
disp('   step          t                    y')
k = 1:length(t); out = [k; t; y];
fprintf('%5d  %15.10f %15.10f\n',out)
```

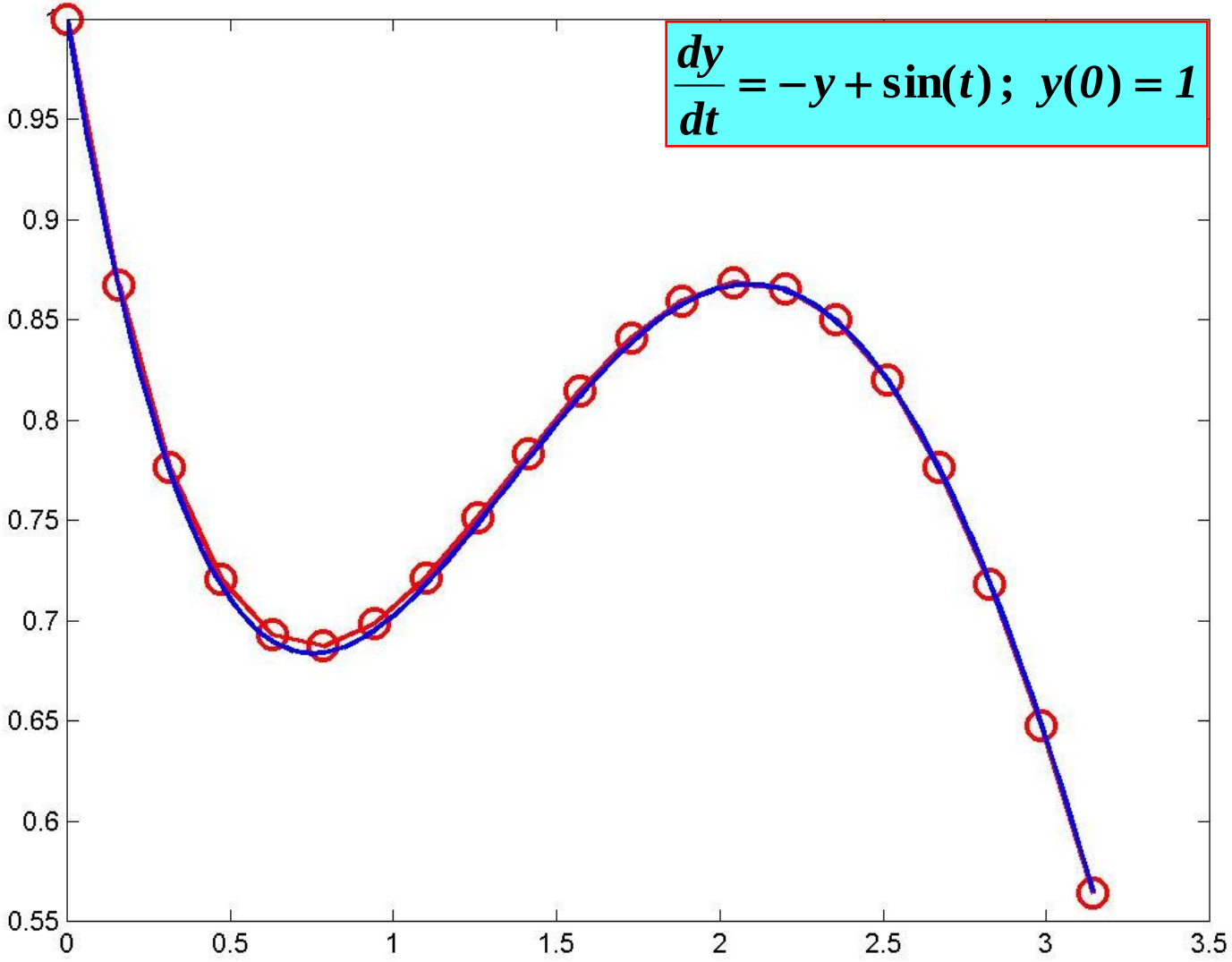
中点 (Midpoint, 2阶RK) 法

```
>> [t,y] = midpoint('example2_f',[0 pi],1,0.05*pi);
   step          t                    y
    1      0.0000000000     1.0000000000
    2      0.1570796327     0.8675816988
    3      0.3141592654     0.7767452235
    4      0.4712388980     0.7206165079
    5      0.6283185307     0.6927855807
    6      0.7853981634     0.6872735266
    7      0.9424777961     0.6985164603
    8      1.0995574288     0.7213629094
    9      1.2566370614     0.7510812520
   10      1.4137166941     0.7833740988
   11      1.5707963268     0.8143967658
   12      1.7278759595     0.8407772414
   13      1.8849555922     0.8596353281
   14      2.0420352248     0.8685989204
   15      2.1991148575     0.8658156821
   16      2.3561944902     0.8499586883
   17      2.5132741229     0.8202249154
   18      2.6703537556     0.7763257790
   19      2.8274333882     0.7184692359
   20      2.9845130209     0.6473332788
   21      3.1415926536     0.5640309524
>> tt = 0:0.01*pi:pi; yy = example2_e(tt);
>> H = plot(t,y,'r-o',tt,yy);
>> set(H,'LineWidth',3,'MarkerSize',12);
```

$$\frac{dy}{dt} = -y + \sin(t)$$

$$y(0) = 1$$

中点 (Midpoint, 2阶RK) 法



推导：

不做要求

利用泰勒展开式 $y(x_{n+1}) = y_n + hy'_n + \frac{h^2}{2}y''_n + \frac{h^3}{3!}y'''_n + O(h^4)$

$$\begin{cases} y'_n = f(x_n, y_n) = f_n \\ y''_n = f'_x(x_n, y_n) + f'_y(x_n, y_n) \cdot f_n \\ y'''_n = f''_{xx}(x_n, y_n) + 2f_n f'_{xy}(x_n, y_n) + f_n^2 f''_{yy}(x_n, y_n) + f'_y(x_n, y_n)[f'_x(x_n, y_n) + f_n f'_y(x_n, y_n)] \end{cases}$$

局部截断误差为 $T_{n+1} = y(x_{n+1}) - y(x_n) - h[c_1 f(x_n, y_n) + c_2 f(x_n + \lambda_2 h, y_n + \mu_{21} h f_n)]$

$$f(x_n + \lambda_2 h, y_n + \mu_{21} h f_n) = f_n + f'_x(x_n, y_n)\lambda_2 h + f'_y(x_n, y_n)\mu_{21} h f_n + O(h^2)$$

代入局部截断误差公式，有

$$T_{n+1} = hf_n + \frac{h^2}{2}[f'_x(x_n, y_n) + f'_y(x_n, y_n) f_n] - h[c_1 f_n + c_2(f_n + \lambda_2 f'_x(x_n, y_n)h + \mu_{21} f'_y(x_n, y_n) f_n h)] + O(h^3)$$

$$= (1 - c_1 - c_2) f_n h + \left(\frac{1}{2} - c_2\lambda_2\right) f'_x(x_n, y_n)h^2 + \left(\frac{1}{2} - c_2\mu_{21}\right) f'_y(x_n, y_n) f_n h^2 + O(h^3)$$

要使公式具有$p = 2$ 阶，必须满足 (9.6) 式。

3、三阶显式R-K方法

$$\begin{cases} y_{i+1} = y_i + h(\lambda_1 K_1 + \lambda_2 K_2 + \lambda_3 K_3) \\ K_1 = f(x_i, y_i) \\ K_2 = f(x_i + ph, y_i + phK_1) \\ K_3 = f(x_i + qh, y_i + qh(\alpha_1 K_1 + \alpha_2 K_2)) \end{cases} \quad \text{(RK3.6)}$$

类似于二阶龙格-库塔格式的导出过程，运用泰勒展开的方法，可找出格式（RK3.6）的局部截断误差为 $O(h^4)$ ，从而具有3阶精度所必须满足的条件为右式：

$$\begin{cases} \alpha_2 + \alpha_1 = 1 \\ \lambda_1 + \lambda_2 + \lambda_3 = 1 \\ \lambda_2 p + \lambda_3 q = \dfrac{1}{2} \\ \lambda_2 p^2 + \lambda_3 q^2 = \dfrac{1}{3} \\ \lambda_3 pq\alpha_2 = \dfrac{1}{6} \end{cases} \quad \text{(RK3.7)}$$

7未知量5方程

其中共有七个待定系数：$\lambda_1,\lambda_2,\lambda_3,p,q,\alpha_1,\alpha_2$ ，但只有五个方程式，因此还有两个自由度。凡满足条件（RK3.7）的一族格式（RK3.6）统称为三阶龙格-库塔格式。

当待定系数取为 $\lambda_1=\lambda_3=1/6,\quad \lambda_2=2/3,\quad p=1/2$

$q=1,\quad \alpha_1=-1,\quad \alpha_2=2$

时的三阶龙格-库塔格式称为库塔格式，其具体形式为

$$\begin{cases} y_{i+1}=y_i+\dfrac{h}{6}(K_1+4K_2+K_3) \\ K_1=f(x_i,y_i) \\ K_2=f\left(x_i+\dfrac{h}{2},y_i+\dfrac{h}{2}K_1\right) \\ K_3=f\left(x_i+h,y_i-hK_1+2hK_2\right) \end{cases}$$

(RK3.8)

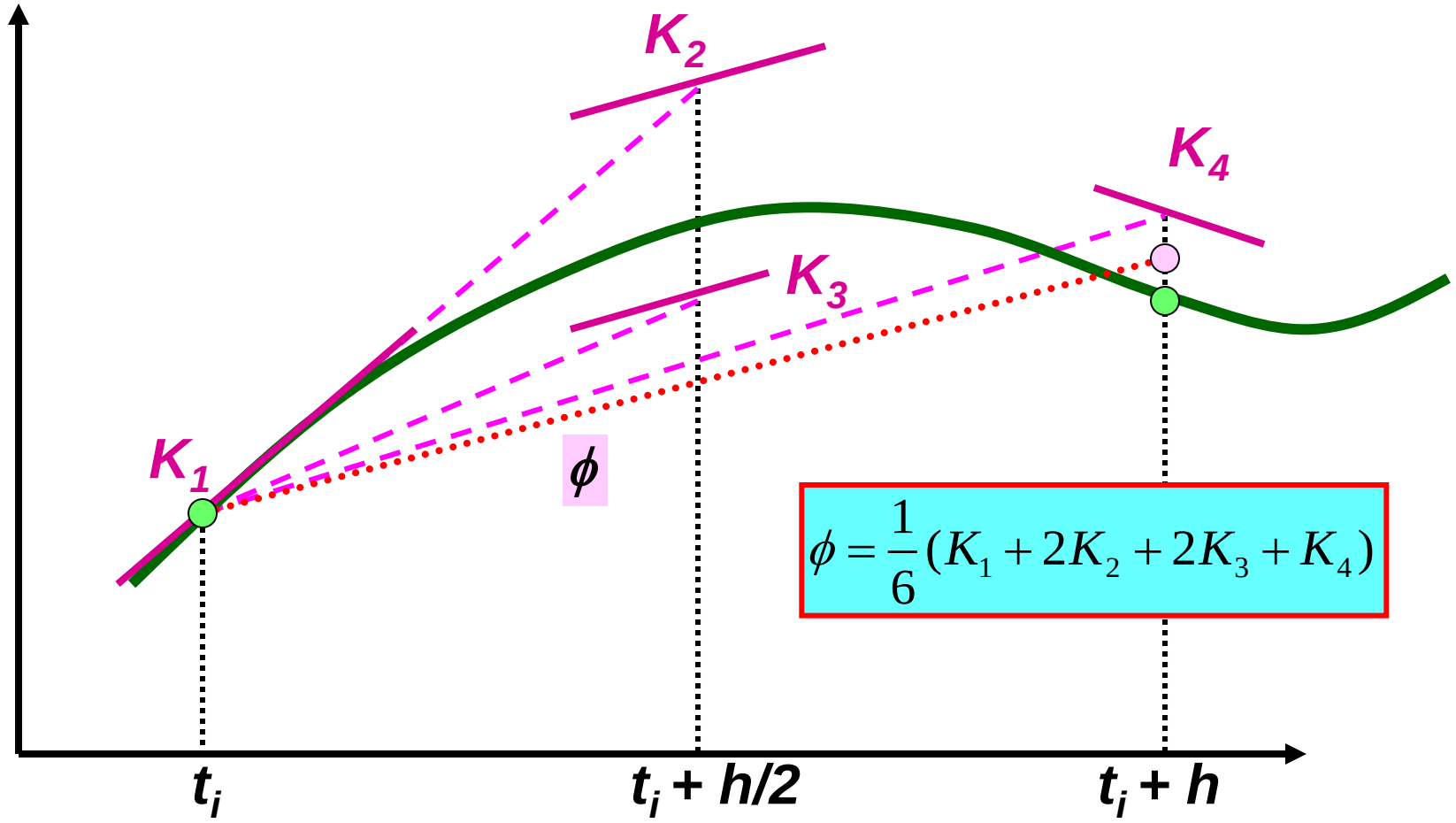
对比141Simpson数值积分

4、四阶显式R-K方法

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

对比1221复合梯形公式：非也

几何解释



$$\phi = \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

1）特式

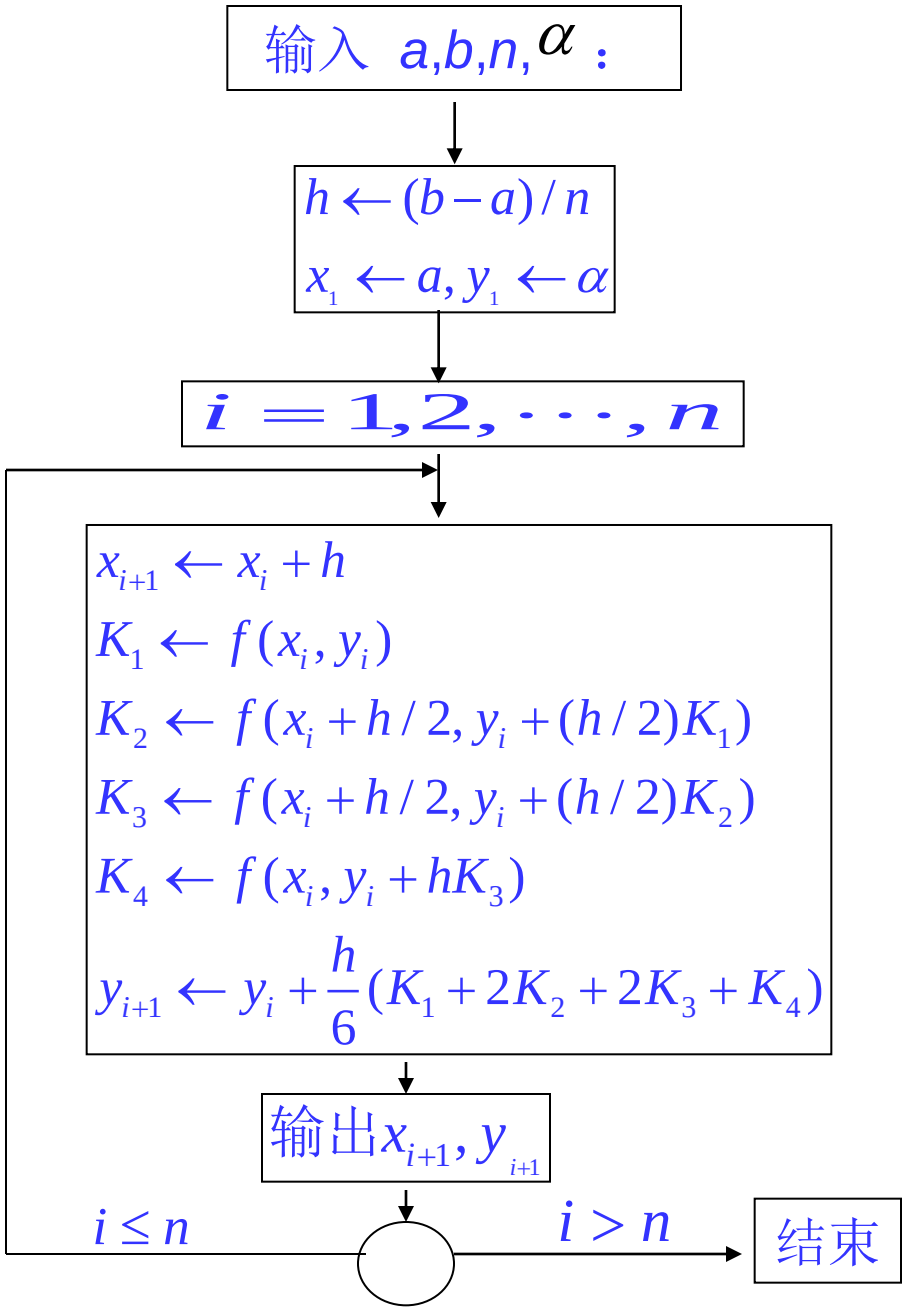
2）通式

3）算法描述或框图

4）（细化的）流程图

5）伪代码

6）MATLAB等实际编码



4阶Runge-Kutta法

```
function [t, y] = RK4(f, tspan, y0, h)
% function [t, y] = RK4(f, tspan, y0, h)
% solve y' = f(t,y) with initial condition y(a) = y0 using
% n steps of the classical 4th order Runge Kutta method;

a = tspan(1); b = tspan(2); n = (b-a) / h;
t = (a+h : h: b);
k1 = feval(f, a, y0);
k2 = feval(f, a + h/2, y0 + k1/2*h);
k3 = feval(f, a + h/2, y0 + k2/2*h);
k4 = feval(f, a + h, y0 + k3*h);
y(1) = y0 + (k1/6 + k2/3 + k3/3 + k4/6)*h;
for i = 1 : n-1
    k1 = feval(f, t(i), y(i));
    k2 = feval(f, t(i) + h/2, y(i) + k1/2*h);
    k3 = feval(f, t(i) + h/2, y(i) + k2/2*h);
    k4 = feval(f, t(i) + h, y(i) + k3*h);
    y(i+1) = y(i) + (k1/6 + k2/3 + k3/3 + k4/6)*h;
end
t = [ a   t ]; y = [ y0  y ];
disp('   step          t                    y')
k = 1:length(t); out = [k; t; y];
fprintf('%5d  %15.10f %15.10f\n',out)
```

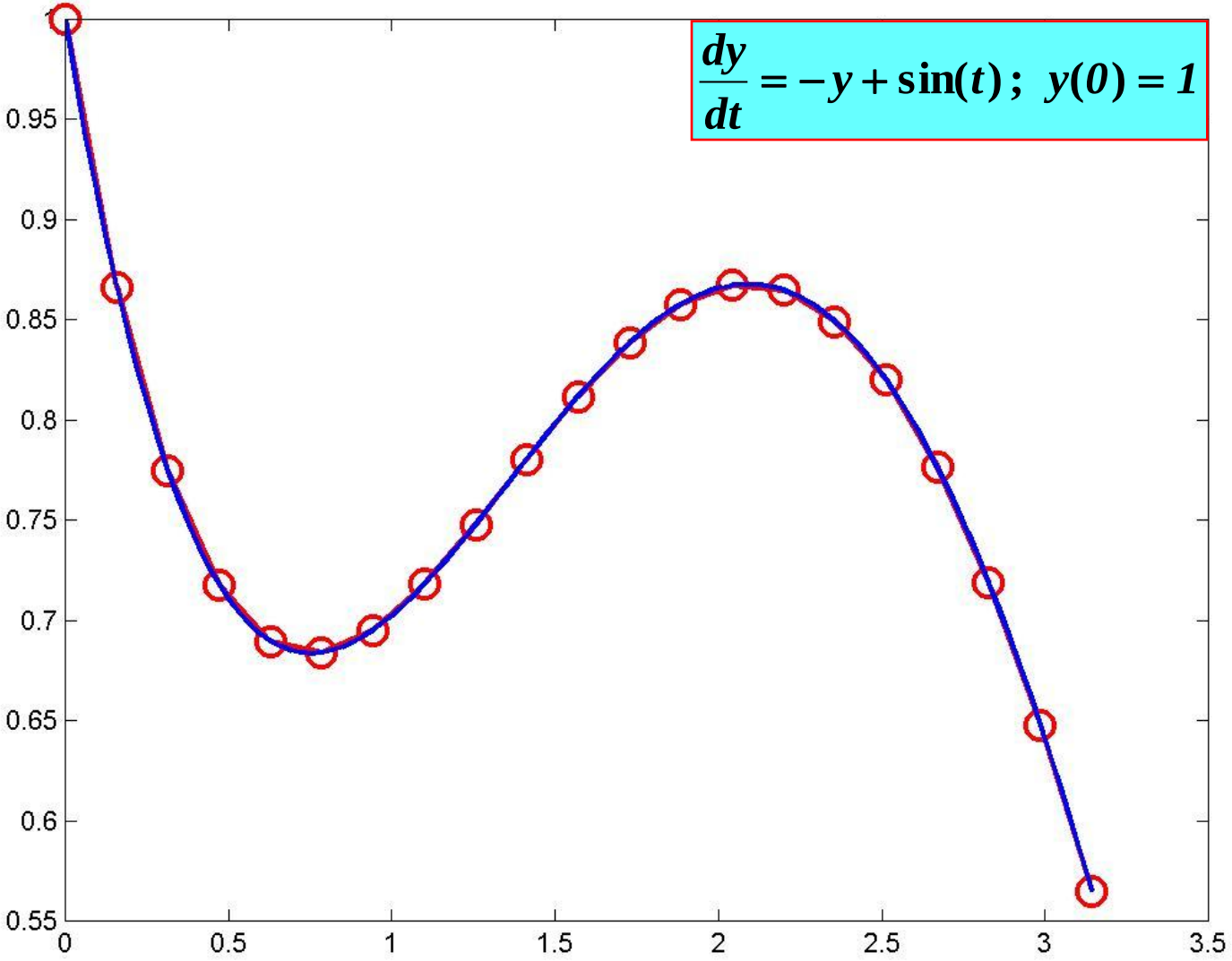
4阶Runge-Kutta法

```
>> tt=0:0.01*pi:pi; yy=example2_e(tt);
>> [t,y]=RK4('example2_f',[0 pi],1,0.05*pi);
   step          t                    y
    1      0.0000000000     1.0000000000
    2      0.1570796327     0.8663284784
    3      0.3141592654     0.7745866433
    4      0.4712388980     0.7178375776
    5      0.6283185307     0.6896194725
    6      0.7853981634     0.6839104249
    7      0.9424777961     0.6951106492
    8      1.0995574288     0.7180384347
    9      1.2566370614     0.7479364401
   10      1.4137166941     0.7804851708
   11      1.5707963268     0.8118207434
   12      1.7278759595     0.8385543106
   13      1.8849555922     0.8577907953
   14      2.0420352248     0.8671448731
   15      2.1991148575     0.8647524420
   16      2.3561944902     0.8492761286
   17      2.5132741229     0.8199036965
   18      2.6703537556     0.7763385417
   19      2.8274333882     0.7187817811
   20      2.9845130209     0.6479057500
   21      3.1415926536     0.5648190301
>> H=plot(x,y,'r-o',xx,yy);
>> set(H,'LineWidth',3,'MarkerSize',12);
```

$$\frac{dy}{dt} = -y + \sin(t)$$

$$y(0) = 1$$

4阶Runge-Kutta法



例 用四阶标准R-K方法求初值问题

$y'=y-2x/y$, $0\le x\le 1$ 其中$y(0)=1$

的数值解, 取步长$h=0.2$。

解 四阶标准R-K公式为

课堂推算

$$\begin{cases} y_{n+1}=y_n+\frac{1}{6}h(K_1+2K_2+2K_3+K_4) \\ K_1=y_n-2x_n/y_n \\ K_2=y_n+\frac{1}{2}hK_1-(2x_n+h)/(y_n+\frac{1}{2}hK_1) \\ K_3=y_n+\frac{1}{2}hK_2-(2x_n+h)/(y_n+\frac{1}{2}hK_2) \\ K_4=y_n+hK_3-2(x_n+h)/(y_n+hK_3) \end{cases}$$

计算结果如下:

n	x_n	y_n	$y(x_n)$		n	x_n	y_n	$y(x_n)$
0	0.0	1.00	1.00		3	0.6	1.4833	1.4832
1	0.2	1.1832	1.1832		4	0.8	1.6125	1.6125
2	0.4	1.3417	1.3416		5	1.0	1.7321	1.7321

FGE（final global error）

最终全局误差

注：

- 龙格-库塔法的主要运算在于计算 K_i 的值，即计算 f 的值。Butcher 于1965年给出了计算量与可达到的最高精度阶数的关系：

每步须算K_i的个数	2	3	4	5	6	7	$n \geq 8$
可达到的最高精度	$O(h^2)$	$O(h^3)$	$O(h^4)$	$O(h^4)$	$O(h^5)$	$O(h^6)$	$O(h^{n-2})$

- 由于龙格-库塔法的导出基于泰勒展开，故精度主要受解函数的光滑性影响。对于光滑性不太好的解，最好采用低阶算法而将步长h 取小。

Butcher's sixth-order Runge-Kutta Method ?

$$y_{i+1} = y_i + \frac{1}{90}(7k_1 + 32k_2 + 12k_4 + 32k_5 + 7k_6)$$

$$\begin{cases} k_1 = f(t_i, y_i) \\ k_2 = f(t_i + \frac{1}{4}h, y_i + \frac{1}{4}k_1 h) \\ k_3 = f(t_i + \frac{1}{4}h, y_i + \frac{1}{8}k_1 h + \frac{1}{8}k_2 h) \\ k_4 = f(t_i + \frac{1}{2}h, y_i - \frac{1}{2}k_2 h + k_3 h) \\ k_5 = f(t_i + \frac{3}{4}h, y_i + \frac{3}{16}k_1 h + \frac{9}{8}k_4 h) \\ k_6 = f(t_i + h, y_i - \frac{3}{7}k_1 h + \frac{2}{7}k_2 h + \frac{12}{7}k_3 h - \frac{12}{7}k_4 h + \frac{8}{7}k_5 h) \end{cases}$$

9.4 单步法的收敛性

对单步法 $y_{n+1}=y_n+h\phi(x_n,y_n,h)$ 而言，如下。

递推索引

定义9-3 若一种数值方法（如单步法(9.5)）对于固定的$x_n=x_0+nh$，当$h\to 0$时有$y_n\to y(x_n)$，其中$y(x)$是(9.1)的准确解，则称该方法是收敛的。

可见，若方法如(9.5)是收敛的，则当$h\to 0$时，整体截断误差$e_n=y(x_n)-y_n$将趋于零.

定理9-1 假设单步法(9.5)具有 p 阶精度，且增量函数$\phi(x,y,h)$关于 y 满足利普希茨条件

$$\left|\phi(x,y,h)-\phi(x,\bar{y},h)\right|\le L_\phi\left|y-\bar{y}\right|$$

又设初值 y_0 是准确的，即 $y_0=y(x_0)$，则其整体截断误差

$$y(x_n)-y_n=O(h^p)$$

例9-5 设 $f(x,y)$ 关于 y 满足利普希茨条件，试证明欧拉法、改进的欧拉法都是收敛的。

分析：依据定理9-1，判断单步法的收敛性，归结为验证增量函数 ϕ 能否满足利普希茨条件。

解： 欧拉方法增量函数 ϕ 就是 $f(x,y)$, 故当 $f(x,y)$ 关于 y 满足利普希茨条件它就收敛。

因为 $f(x,y)$ 关于 y 满足利普希茨条件，记利普希茨常数为 L 。

改进的欧拉方法（Heun法）的增量函数为

$$\phi(x,y,h)=\frac{1}{2}[f(x,y)+f(x+h,y+hf(x,y))]$$

由于

$$\begin{aligned}|\phi(x,y,h)-\phi(x,\bar{y},h)| &\le \frac{1}{2}[|f(x,y)-f(x,\bar{y})| \\ &\quad +|f(x+h,y+hf(x,y))-f(x+h,\bar{y}+hf(x,\bar{y}))|] \\ &\le \frac{1}{2}L|y-\bar{y}|+\frac{1}{2}L|y+hf(x,y)-\bar{y}-hf(x,\bar{y})| \\ &\le \frac{1}{2}L|y-\bar{y}|+\frac{1}{2}L|y-\bar{y}|+\frac{h}{2}L^2|y-\bar{y}| = L\left(1+\frac{h}{2}L\right)|y-\bar{y}|\end{aligned}$$

推导

设限定 $h \le h_0$, 上式表明 ϕ 关于 y 的普希茨常数为

$$L_\phi = L\left(1+\frac{h_0}{2}L\right)$$

因此改进的欧拉方法（Heun法）收敛。

9.4A 单步方法的稳定性

定义5 对于初值ODE问题, 取定步长h, 用某个差分方法进行计算时, 假设只在一个节点值y_n上产生计算误差δ, 即计算值 $\bar{y}_n=y_n+\delta$, 如果这个误差引起以后各节点值y_m (其中$m>n$)的变化均不超过δ, 则称此差分方法是**稳定**的.

讨论数值方法的稳定性, 通常仅限于典型的试验方程 $y'=\lambda y$

其中λ是复数且real(λ)<0.

在复平面上, 当方法稳定时要求变量λh的取值范围称为方法的**绝对稳定域**, 它与实轴的交集称为**绝对稳定区间**.

将Euler方法应用于方程$y'=\lambda y$, 得到 $y_{n+1}=(1+\lambda h)y_n$

设计算y_n时产生误差δ_n, 计算值 $\bar{y}_n=y_n+\delta_n$, 则δ_n将对以后各节点值计算产生影响. 记 $\bar{y}_m=y_m+\delta_m$, 其中$m\geq n$, 由上式可知误差δ_m满足方程

$$\delta_m=(1+\lambda h)\delta_{m-1}=\ldots=(1+\lambda h)^{m-n}\delta_n \text{ , 其中} m\geq n.$$

可见, 若要$|\delta_m|<|\delta_n|$, 必须且只须$|1+\lambda h|<1$, 因此Euler法的绝对稳定域为$|1+\lambda h|<1$, 绝对稳定区间是$-2<\text{real}(\lambda)h<0$.

对隐式单步方法也可类似讨论. 如将**梯形公式**用于方程$y'=\lambda y$, 则有

$$y_{n+1}=y_n+(h/2)*\lambda(y_n+y_{n+1})$$

解出y_{n+1}得

$$y_{n+1}=\frac{1+\frac{1}{2}\lambda h}{1-\frac{1}{2}\lambda h}y_n$$

推导

类似前面分析, 可知绝对稳定区域为

$$\left|\frac{1+\frac{1}{2}\lambda h}{1-\frac{1}{2}\lambda h}\right|<1$$

因real(λ)<0, 故此不等式对任意步长h恒成立, 这是隐式公式的优点.

一些常用方法的绝对稳定区间如下。

方　　法	方法的阶数	稳 定 区 间
Euler方法	1	(-2, 0)
梯形方法	2	(-∞, 0)
改进Euler方法(H)	2	(-2, 0)
二阶R-K方法	2	(-2, 0)
三阶R-K方法	3	(-2.51, 0)
四阶R-K方法	4	(-2.78, 0)

中点计算公式：

$$\begin{cases} y_{n+1} = y_n + hk_2 \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + h/2, y_n + hK_1/2) \end{cases}$$

也可写为

$$y_{n+1} = y_n + hf\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\right) 。$$

其绝对稳定区间为何？

课堂练习

例　考虑初值问题 $y'=-30y$ ，$0\le x\le 1$ 其中$y(0)=1$

取步长$h=0.1$, 利用Euler方法计算y_{10}也即$y(1)$.　[$y(x)=e^{-30x}$]

解　因$y_0=1$, 计算得$y_{10}=1024$, 而$y(1)=9.357623\times10^{-14}$.

这是因为$\lambda h=-3$不属于Euler方法的绝对稳定区间.

若取$h=0.01$, 计算得$y_{100}=3.234477\times10^{-16}$.

若取$h=0.001$, 计算得$y_{1000}=5.911998\times10^{-14}$.

若取$h=0.0001$, 计算得$y_{10000}=8.945057\times10^{-14}$.

若取$h=0.00001$, 计算得$y_{100000}=9.3156\times10^{-14}$.

步长h要足够小

此题步长h=0.1

例：考察初值问题 $\begin{cases} y'(x) = -30y(x) \\ y(0) = 1 \end{cases}$ 在区间$[0, 0.5]$上的解。

分别用欧拉法、后退的欧拉法和改进的欧拉法计算数值解。

$y_{n+1} = -2y_n$　　$y_{n+1} = 0.25y_n$　　$y_{n+1} = 02.5y_n$　　推算

节点 x_i	欧拉显式	后退的欧拉法	改进的欧拉法	精确解 $y = e^{-30x}$
0.0	1.0000	1.0000	1.0000	1.0000
0.1	−2.0000	2.5000×10^{-1}	2.5000	4.9787×10^{-2}
0.2	4.0000	6.2500×10^{-2}	6.2500	2.4788×10^{-3}
0.3	−8.0000	1.5625×10^{-2}	1.5626×10^{1}	1.2341×10^{-4}
0.4	1.6000×10^{1}	3.9063×10^{-3}	3.9063×10^{1}	6.1442×10^{-6}
0.5	-3.2000×10^{1}	9.7656×10^{-4}	9.7656×10^{1}	3.0590×10^{-7}

也即，前题答案细节

What is wrong?!

单步显式方法的**稳定性**与**步长**密切相关, 在一种步长下是稳定的差分公式, 取大一点步长就可能是不稳定的.

1）**收敛性**是反映差分公式本身的**截断误差**对数值解的影响;

2）**稳定性**是反映计算过程中**舍入误差**对数值解的影响.

3）只有即**收敛又稳定**的差分公式才有实用价值.

9.5　线性多步方法

由于在计算y_{n+1}时，已经知道y_n ,y_{n-1}，…，及$f(x_n,y_n)$, $f(x_{n-1},y_{n-1})$, …，利用这些值构造出精度高、计算量小的差分公式就是线性多步法.

9.5.1　利用待定参数法构造线性多步方法

r+1步线性多步方法的一般形式为 $y_{n+1}=\sum_{i=0}^{r}\alpha_i y_{n-i}+h\sum_{i=-1}^{r}\beta_i f_{n-i}$

当$\beta_{-1}\neq 0$时, 公式为隐式公式; 反之, 为显式公式. 参数$\{\alpha_i,\beta_i\}$的选择原则是使方法的局部截断误差为 $y(x_{n+1})-y_{n+1}=O(h^{p+1})$ 其中p尽可能大。

这里, 局部截断误差是指: 在$y_{n-i}=y(x_{n-i})$, i=0,1, … , r的前提下, 误差 $y(x_{n+1})-y_{n+1}$.

例 选取四个参数α, β_0, β_1, β_2, 使三步方法

$$y_{n+1}=\alpha y_n+h(\beta_0 f_n+\beta_1 f_{n-1}+\beta_2 f_{n-2})$$

为三阶方法.

解 设$y_n=y(x_n)$, $y_{n-1}=y(x_{n-1})$, $y_{n-2}=y(x_{n-2})$, 则有

$$f_n=f(x_n,y(x_n))=y'(x_n)$$

$$f_{n-1}=f(x_{n-1},y(x_{n-1}))=y'(x_{n-1})=y'(x_n-h)$$

$$=y'(x_n)-hy''(x_n)+1/2h^2y'''(x_n)-1/6h^3y^{(4)}(x_n)+O(h^4)$$

$$f_{n-2}=y'(x_n)-2hy''(x_n)+2h^2y'''(x_n)-4/3h^3y^{(4)}(x_n)+O(h^4)$$

于是有

$$y_{n+1}=\alpha y(x_n)+h(\beta_0+\beta_1+\beta_2)y'(x_n)-h^2(\beta_1+2\beta_2)y''(x_n)$$

$$+h^3(1/2\beta_1+2\beta_2)y'''(x_n)-h^4/6(\beta_1+8\beta_2)y^{(4)}(x_n)+O(h^5)$$

$$y(x_{n+1})=y(x_n)+hy'(x_n)+1/2h^2y''(x_n)+1/6h^3y'''(x_n)$$

$$+1/24h^4y^{(4)}(x_n)+O(h^5)$$

若使$y(x_{n+1})-y_{n+1}=O(h^4)$，只要$\alpha$, β_0, β_1和β_2满足

$$\alpha=1,\ \beta_0+\beta_1+\beta_2=1,\ \beta_1+2\beta_2=-1/2\ ,\ \beta_1+4\beta_2=1/3$$

解之得 $\alpha=1,\quad \beta_0=\dfrac{23}{12},\quad \beta_1=-\dfrac{4}{3},\quad \beta_2=\dfrac{5}{12}$

于是有三步三阶显式差分公式

$$y_{n+1}=y_n+h(23f_n-16f_{n-1}+5f_{n-2})/12$$

Adams-Bashforth三步法

不做要求

于是得到四步显式公式

$$y_{n+4}=y_n+\frac{4h}{3}(2f_{n+3}-f_{n+2}+2f_{n+1}),\qquad (5.11)$$

称为**米尔尼**(Milne）**方法**.

其局部截断误差为 $T_{n+4}=\frac{14}{45}h^5y^{(5)}(x_n)+O(h^6).\qquad (5.12)$

REPORT1： 试用Milne法求解右边初值问题或其它初值问题（尽可能多尝试与报告）

$$\begin{cases}\dfrac{\mathrm{d}y}{\mathrm{d}x}=-y,\ x\in[0,1]\\ y(0)=1\end{cases}$$

REPORT2：Milne法的绝对稳定区间为何？

再如，用中心差商表示的三点数值微分公式

$$y'(x_i)=\frac{1}{2h}[y(x_{i+1})-y(x_{i-1})]-\frac{h^2}{6}y^{(3)}(\xi_i)$$
$$(i=0,1,2,\ldots,n-1)$$

代入

$$y'(x_i)=f(x_i,y(x_i))\quad(i=0,1,2,\ldots,n)$$

可得

$$y(x_{i+1})=y(x_{i-1})+2hf(x_i,y(x_i))+\frac{h^3}{3}y^{(3)}(\xi_i)$$
$$\approx y(x_{i-1})+2hf(x_i,y(x_i))$$
$$(i=0,1,2,\ldots,n-1)$$

其绝对稳定区间为何？

REPORT

§ 9.7 一阶ODE方程组及高阶方程

9.7.1 一阶ODE方程组

对于一阶微分方程组的初值问题：

$$\begin{cases} y' = f(x, y, z), & y(x_0) = y_0 \\ z' = g(x, y, z), & z(x_0) = z_0 \end{cases} \tag{9.7.1}$$

可以将单个方程 $y' = f(x, y, z)$ 中的 f 和 y 看作向量来处理，这样就可把前面介绍的各种差分算法推广到一阶方程组中应用。

设 $x_i = x_0 + i\,h\ (i = 1, 2, 3, \ldots\ldots)$，$y_i$，$z_i$ 为节点 x_i 上的近似解，则有改进的欧拉格式（Heun法格式）如下：

$$
\begin{cases}
\text{预测：} \\
\bar{y}_{i+1} = y_i + hf(x_i, y_i, z_i) \\
\bar{z}_{i+1} = z_i + hg(x_i, y_i, z_i) \\
\text{校正：} \\
y_{i+1} = y_i + \dfrac{h}{2}[f(x_i, y_i, z_i) + f(x_{i+1}, \bar{y}_{i+1}, \bar{z}_{+1i})] \\
z_{i+1} = z_i + \dfrac{h}{2}[g(x_i, y_i, z_i) + g(x_{i+1}, \bar{y}_{i+1}, \bar{z}_{i+1})]
\end{cases}
\qquad (9.7.2)
$$

再如经典R-K格式如下：

$$\begin{cases} y_{i+1} = y_i + \dfrac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ z_{i+1} = z_i + \dfrac{h}{6}(L_1 + 2L_2 + 2L_3 + L_4) \end{cases} \tag{9.7.3}$$

其中

$$\begin{cases} K_1 = f(x_i, y_i, z_i) \\ L_1 = g(x_i, y_i, z_i) \\ K_2 = f\left(x_i + \dfrac{h}{2},\ y_i + \dfrac{h}{2}K_1,\ z_i + \dfrac{h}{2}L_1\right) \\ L_2 = g\left(x_i + \dfrac{h}{2},\ y_i + \dfrac{h}{2}K_1,\ z_i + \dfrac{h}{2}L_1\right) \end{cases} \tag{9.7.4}$$

$$
\begin{cases}
K_3 = f\left(x_i + \dfrac{h}{2},\ y_i + \dfrac{h}{2}K_2,\ z_i + \dfrac{h}{2}L_2\right) \\
L_3 = g\left(x_i + \dfrac{h}{2},\ y_i + \dfrac{h}{2}K_2,\ z_i + \dfrac{h}{2}L_2\right) \\
K_4 = f(x_i + h,\ y_i + hK_3,\ z_i + hL_3) \\
L_4 = g(x_i + h,\ y_i + hK_3,\ z_i + hL_3)
\end{cases}
\qquad (9.7.4[续])
$$

将节点 x_i 上的 y_i 和 z_i 值代入（9.7.4），依次算出K_1，L_1，K_2，L_2，K_3，L_3，K_4和 L_4，然后代入（9.7.3）算出节点 x_{i+1} 上的 y_{i+1} 和 z_{i+1} 值。

对于具有三个或三个以上方程的方程组的初值问题，也可用类似方法处理。此外，多步方法也同样可以应用于求解方程组的初值问题。

例
$$\begin{cases} \dfrac{\mathrm{d}u}{\mathrm{d}t} = 0.05u\left(1-\dfrac{u}{20}\right)-0.002uv \\ \dfrac{\mathrm{d}v}{\mathrm{d}t} = 0.09v\left(1-\dfrac{v}{15}\right)-0.15uv \\ u(0)=0.193 \\ v(0)=0.083 \end{cases}$$

1、
$$\phi=\begin{pmatrix} f(u,v,t) \\ g(u,v,t) \end{pmatrix}=\begin{pmatrix} 0.05u(1-u/20)-0.002uv \\ 0.09v(1-v/15)-0.15uv \end{pmatrix}$$

2、确定方法，然后求解

4阶Runge-Kutta法，h=1

（0.202760　0.0881157）
（0.213007　0.0934037）
（0.223763　0.0988499）
（0.235052　0.1044370）
（0.246902　0.1101460）

验算

9.7.2 高阶方程

对于高阶微分方程（或方程组）的初值问题，可以化为一阶方程组来求解。例如，有二阶微分方程的初值问题

$$\begin{cases} y'' = f(x, y, y') \\ y(x_0) = y_0,\ \ y'(x_0) = y_0' \end{cases} \tag{9.7.5}$$

令新变量 $z = y'$，代入（9.7.5）得

$$\begin{cases} y' = z,\ \ y(x_0) = y_0 \\ z' = f(x, y, z),\ \ z(x_0) = y_0' \end{cases} \tag{9.7.6}$$

Euler做法
1727, 1750

（9.7.6）为一个一阶微分方程组的初值问题，对此可用前述方法来求解。例如用经典R-K方法：

$$\begin{cases} y_{i+1} = y_i + \dfrac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ z_{i+1} = z_i + \dfrac{h}{6}(L_1 + 2L_2 + 2L_3 + L_4) \end{cases} \tag{9.7.7}$$

可进一步细化算法公式，进而做法巧妙、更高效：

请对照看书311页公式（7.3）和313页，对照易懂。

课堂阅读

从二阶到更高阶方程（通式情况）：

$$\begin{cases} \dfrac{\mathrm{d}^m y}{\mathrm{d}x^m} = f\left(x, y, y', \cdots, y^{(m-1)}\right); \\ y(a) = \eta_1, \\ y'(a) = \eta_2, \\ \vdots \\ y^{(m-1)}(a) = \eta_m; \end{cases}$$

其中$a \le x \le b$

令

$$\begin{cases} y = y_1 \\ \dfrac{\mathrm{d}y_1}{\mathrm{d}x} = y_2 \\ \vdots \\ \dfrac{\mathrm{d}y_{m-1}}{\mathrm{d}x} = y_m \end{cases}$$

则有

$$\begin{cases} \dfrac{\mathrm{d}y_1}{\mathrm{d}x} = y_2, \\ \vdots \\ \dfrac{\mathrm{d}y_m}{\mathrm{d}x} = f(x, y_1, \cdots, y_m); \\ y_1(a) = \eta_1, \\ \vdots \\ y_m(a) = \eta_m; \end{cases}$$

其中$a \le x \le b$

课堂推导

Euler做法

1727, 1750

求解1阶ODE方程组的Euler法M程序

```
function [t, y] = Euler_sys(f, tspan, y0, h)

% Solve a system of ODE-IVP using Euler method
% y' = f(t,y)   a <= t <= b
% y = [y1, y2, ..., yn]
% interval of interest is given as tspan = [a, b]
% function f(t, y) returns a column vector of values

a = tspan(1); b = tspan(2);
n = (b-a)/h;
t = (a+h : h : b);
k = feval(f, a, y0)';
y(1, :) = y0 + h*k;
for i = 1 : n-1
    k = feval(f, t(i), y(i, :) )';
    y(i+1, :) = y(i, :) + h*k;
end
t = [a t]; y = [y0; y]; out = [t' y];
disp('     t              y1                y2                y3   ...');
fprintf('%8.3f  %15.10f %15.10f \n',out')
```

y is a column vector with *n* variables

求解1阶ODE方程组的Euler法

```
function f = example5(t,y)
% dy1/dt = f1 = -0.5 y1
% dy2/dt = f2 = 4 - 0.1*y1 - 0.3*y2
% let y(1) = y1, y(2) = y2
% tspan = [0 1]
% initial conditions y0 = [4, 6]
f1 = -0.5*y(1);
f2 = 4 - 0.1*y(1) - 0.3*y(2);
f = [f1, f2]';
```

```
>> [t,y]=Euler_sys('example5',[0 1],[4 6],0.5);
     t              y1                y2                y3   ...
   0.000     4.0000000000     6.0000000000
   0.500     3.0000000000     6.9000000000
   1.000     2.2500000000     7.7150000000

>> [t,y]=Euler_sys('example5',[0 1],[4 6],0.2);
     t              y1                y2                y3   ...
   0.000     4.0000000000     6.0000000000
   0.200     3.6000000000     6.3600000000
   0.400     3.2400000000     6.7064000000
   0.600     2.9160000000     7.0392160000
   0.800     2.6244000000     7.3585430400
   1.000     2.3619600000     7.6645424576
```

(*h* = 0.5)

(*h* = 0.2)

FGE（final global error）

最终全局误差

求解2阶ODE的Euler法

$$\frac{d^2y}{dt^2} + 0.3\frac{dy}{dt} = -\sin y$$

非线性摆 (Nonlinear Pendulum)

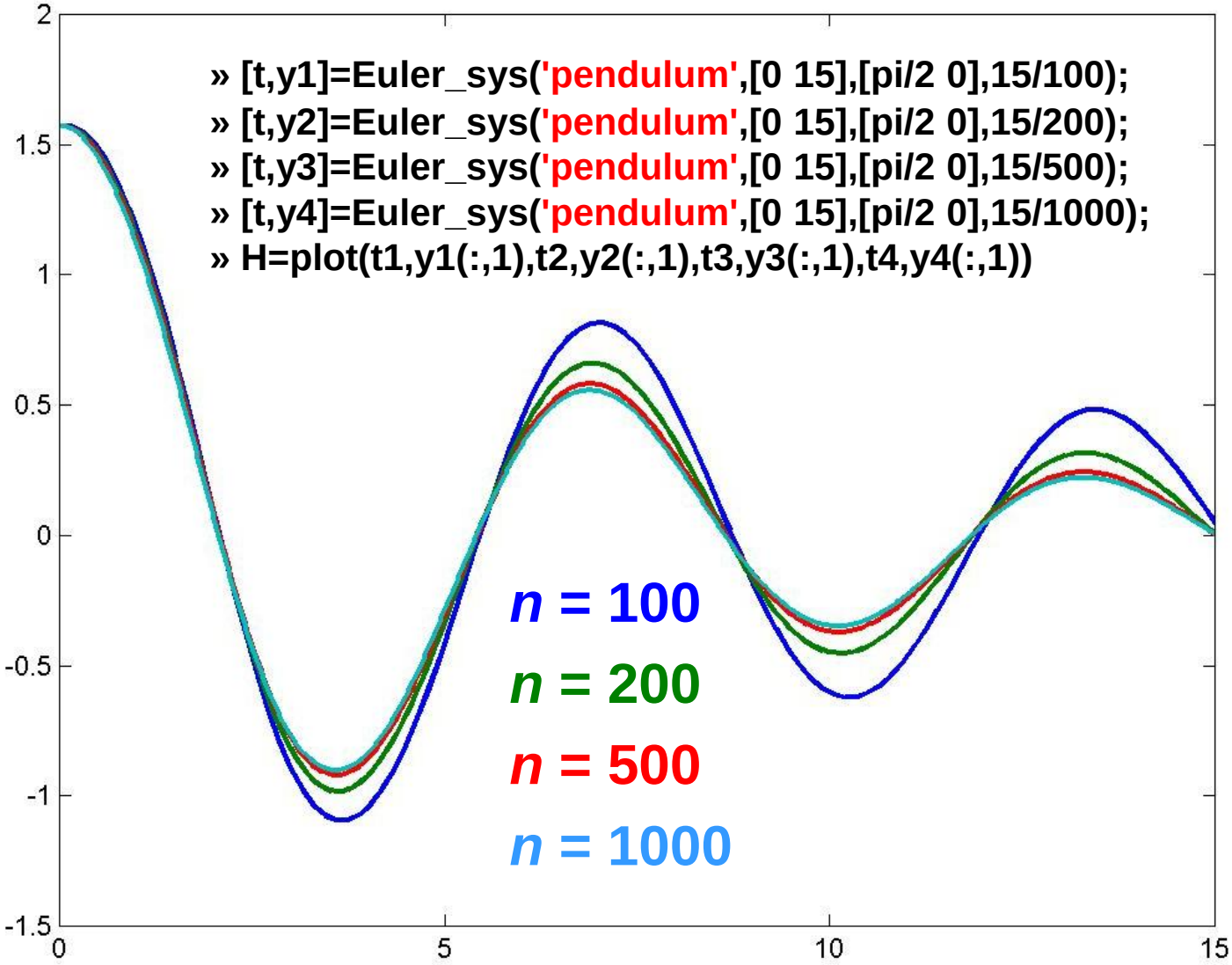
$$\text{let} \begin{cases} y_1 = y \\ y_2 = \dfrac{dy}{dt} \end{cases} \Rightarrow \begin{cases} \dfrac{dy_1}{dt} = \dfrac{dy}{dt} = y_2 \\ \dfrac{dy_2}{dt} = \dfrac{d^2y}{dt^2} = -0.3\dfrac{dy}{dt} - \sin y = -0.3y_2 - \sin y_1 \end{cases}$$

```
function f = pendulum(t,y)
% nonlinear pendulum  d^2y/dt^2 + 0.3dy/dt = -sin(y)
% convert to two first-order ODEs
% dy1/dt = f1 = y2
% dy2/dt = f2 = -0.1*y2 - sin(y1)
% let y(1) = y1, y(2) = y2
% tspan = [0 15]
% initial conditions y0 = [pi/2, 0]
f1 = y(2);
f2 = -0.3*y(2) - sin(y(1));
f = [f1, f2]';
```

求解2阶ODE的Euler法

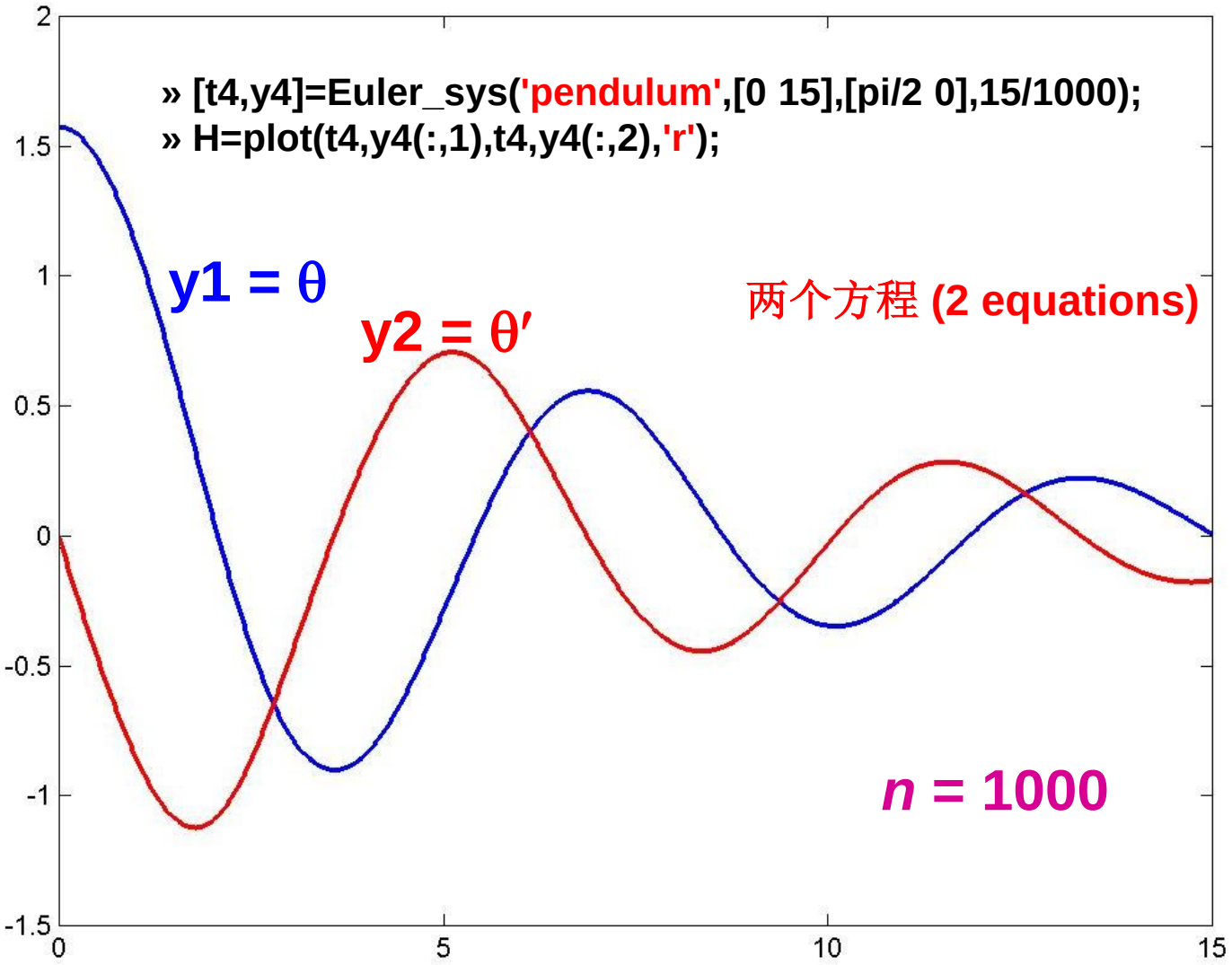
非线性摆 (Nonlinear Pendulum)

```
» [t,y1]=Euler_sys('pendulum',[0 15],[pi/2 0],15/100);
» [t,y2]=Euler_sys('pendulum',[0 15],[pi/2 0],15/200);
» [t,y3]=Euler_sys('pendulum',[0 15],[pi/2 0],15/500);
» [t,y4]=Euler_sys('pendulum',[0 15],[pi/2 0],15/1000);
» H=plot(t1,y1(:,1),t2,y2(:,1),t3,y3(:,1),t4,y4(:,1))
```



求解2阶ODE的Euler法

非线性摆 (Nonlinear Pendulum)



求解1阶ODE方程组的4阶RK法M程序

```
function [t, y] = RK4_sys(f, tspan, y0, h)
% Solve a system of ODEs using 4th-order RK method
% input:  column vector t and row vector y
% return: column vector of values for y'

a = tspan(1); b = tspan(2); n=(b-a)/h;
t = (a+h : h : b)';
k1 = feval(f, a, y0)';
k2 = feval(f, a + h/2, y0 + k1/2*h)';
k3 = feval(f, a + h/2, y0 + k2/2*h)';
k4 = feval(f, a + h,   y0 + k3*h)';
y(1, :) = y0 + (k1+2*k2+2*k3+k4)/6*h;
for i = 1 : n-1
    k1 = feval(f, t(i),       y(i,:))';
    k2 = feval(f, t(i) + h/2, y(i,:) + k1/2*h)';
    k3 = feval(f, t(i) + h/2, y(i,:) + k2/2*h)';
    k4 = feval(f, t(i) + h  , y(i,:) + k3*h)';
    y(i+1, :) = y(i, :) + (k1+2*k2+2*k3+k4)/6*h;
end
t = [a; t]; y = [y0; y]; out = [t y];
disp('     t              y1                y2                y3   ...');
fprintf('%8.3f  %15.10f %15.10f \n',out')
```

Valid for any number of coupled ODEs

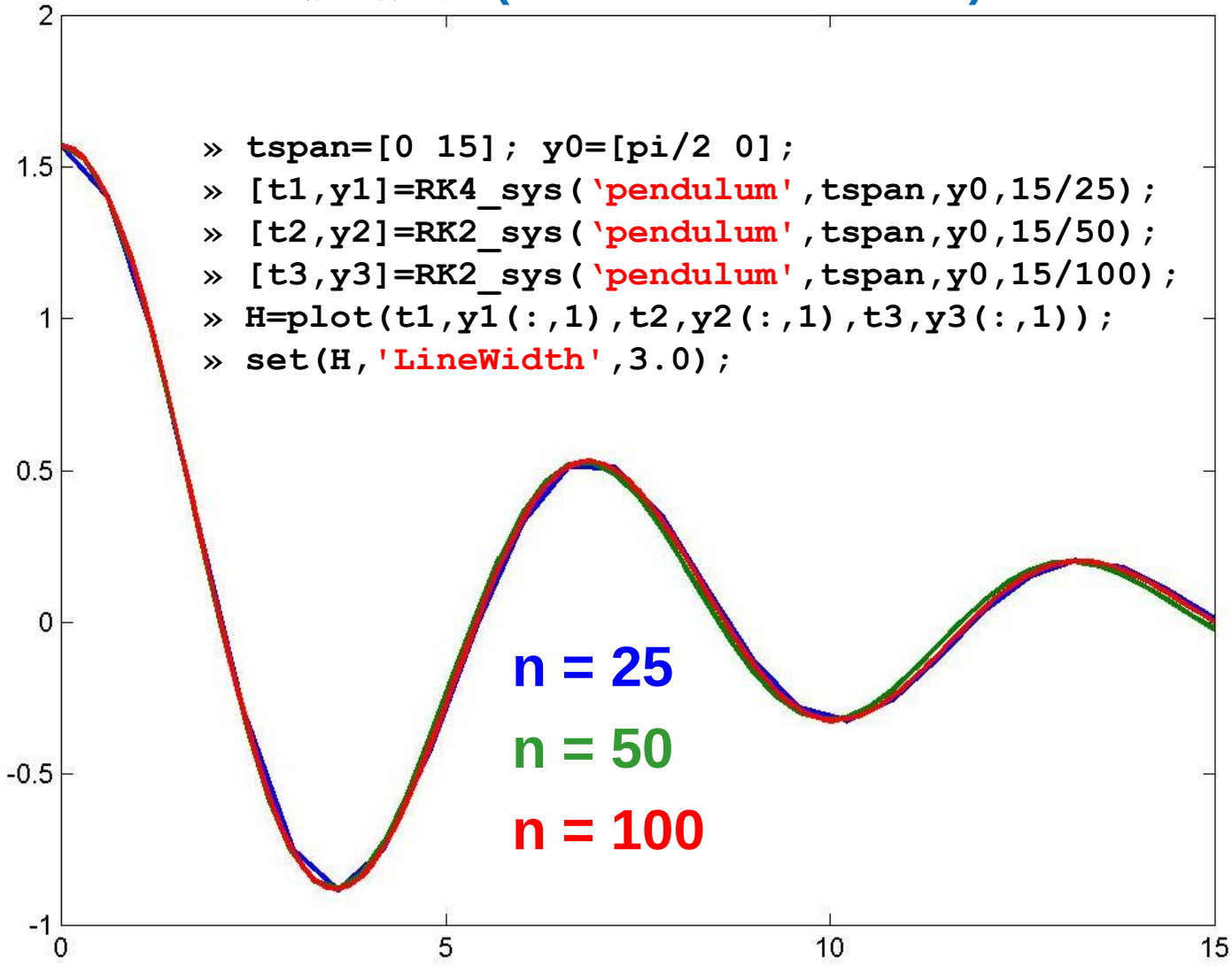
求解1阶ODE方程组的4阶RK法

```
>> [t,y]=RK4_sys('example5',[0 10],[4 6],0.5);
     t              y1                y2              y3   ...
   0.000      4.0000000000      6.0000000000
   0.500      3.1152343750      6.8576703125
   1.000      2.4261713028      7.6321056734
   1.500      1.8895230605      8.3268859767
   2.000      1.4715767976      8.9468651000
   2.500      1.1460766564      9.4976013588
   3.000      0.8925743491      9.9849540205
   3.500      0.6951445736     10.4148035640
   4.000      0.5413845678     10.7928635095
   4.500      0.4216349539     11.1245594257
   5.000      0.3283729256     11.4149566980
   5.500      0.2557396564     11.6687232060
   6.000      0.1991722422     11.8901165525
   6.500      0.1551170538     12.0829881442
   7.000      0.1208064946     12.2507984405
   7.500      0.0940851361     12.3966392221
   8.000      0.0732743126     12.5232598757
   8.500      0.0570666643     12.6330955637
   9.000      0.0444440086     12.7282957874
   9.500      0.0346133758     12.8107523359
  10.000      0.0269571946     12.8821259602
```

求解1阶ODE方程组的4阶RK法

非线性摆 (Nonlinear Pendulum)

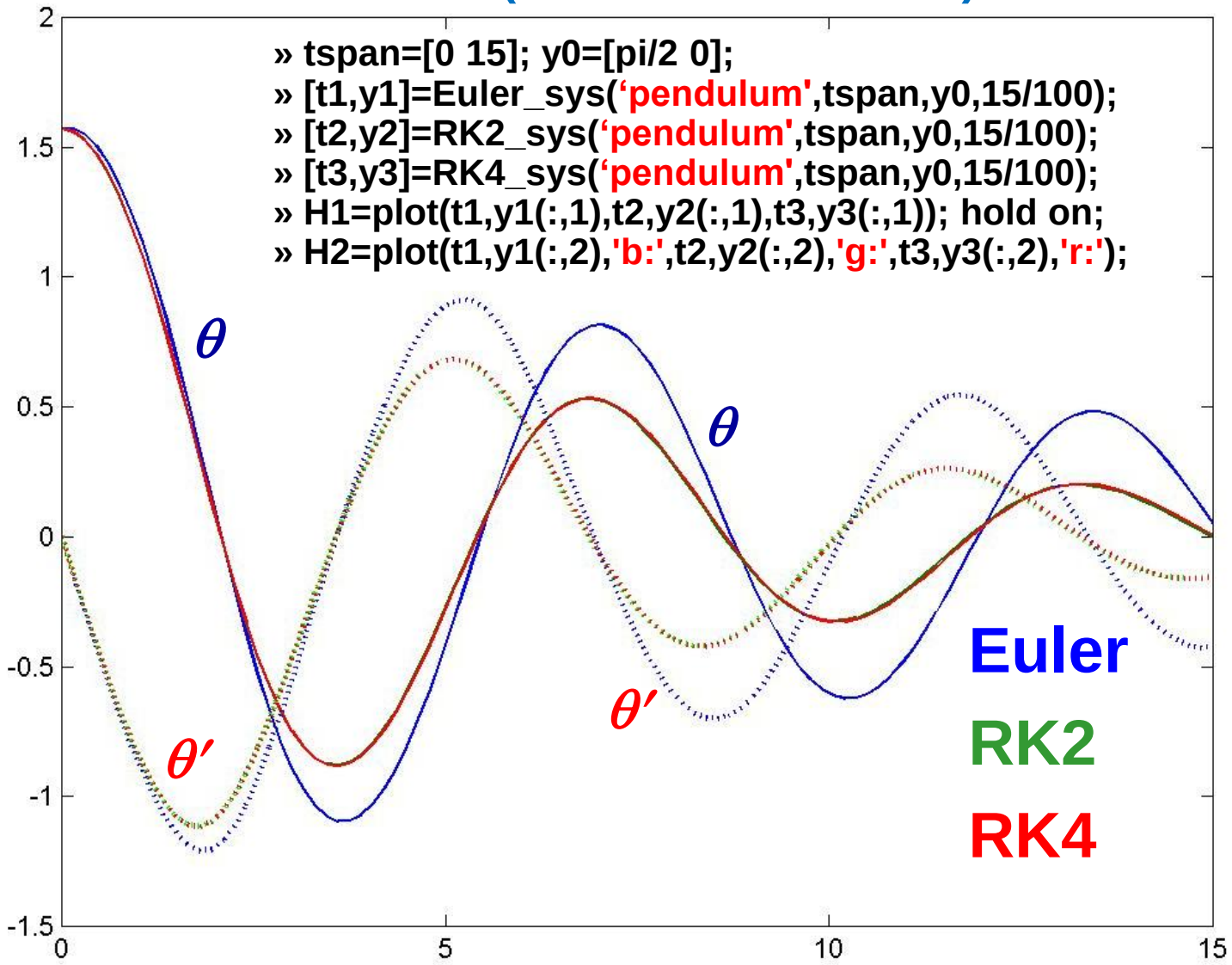
```
» tspan=[0 15]; y0=[pi/2 0];
» [t1,y1]=RK4_sys('pendulum',tspan,y0,15/25);
» [t2,y2]=RK2_sys('pendulum',tspan,y0,15/50);
» [t3,y3]=RK2_sys('pendulum',tspan,y0,15/100);
» H=plot(t1,y1(:,1),t2,y2(:,1),t3,y3(:,1));
» set(H,'LineWidth',3.0);
```



数值精度对比

非线性摆 (Nonlinear Pendulum)

```
» tspan=[0 15]; y0=[pi/2 0];
» [t1,y1]=Euler_sys('pendulum',tspan,y0,15/100);
» [t2,y2]=RK2_sys('pendulum',tspan,y0,15/100);
» [t3,y3]=RK4_sys('pendulum',tspan,y0,15/100);
» H1=plot(t1,y1(:,1),t2,y2(:,1),t3,y3(:,1)); hold on;
» H2=plot(t1,y1(:,2),'b:',t2,y2(:,2),'g:',t3,y3(:,2),'r:');
```



另1示例 (4维ODE方程组)

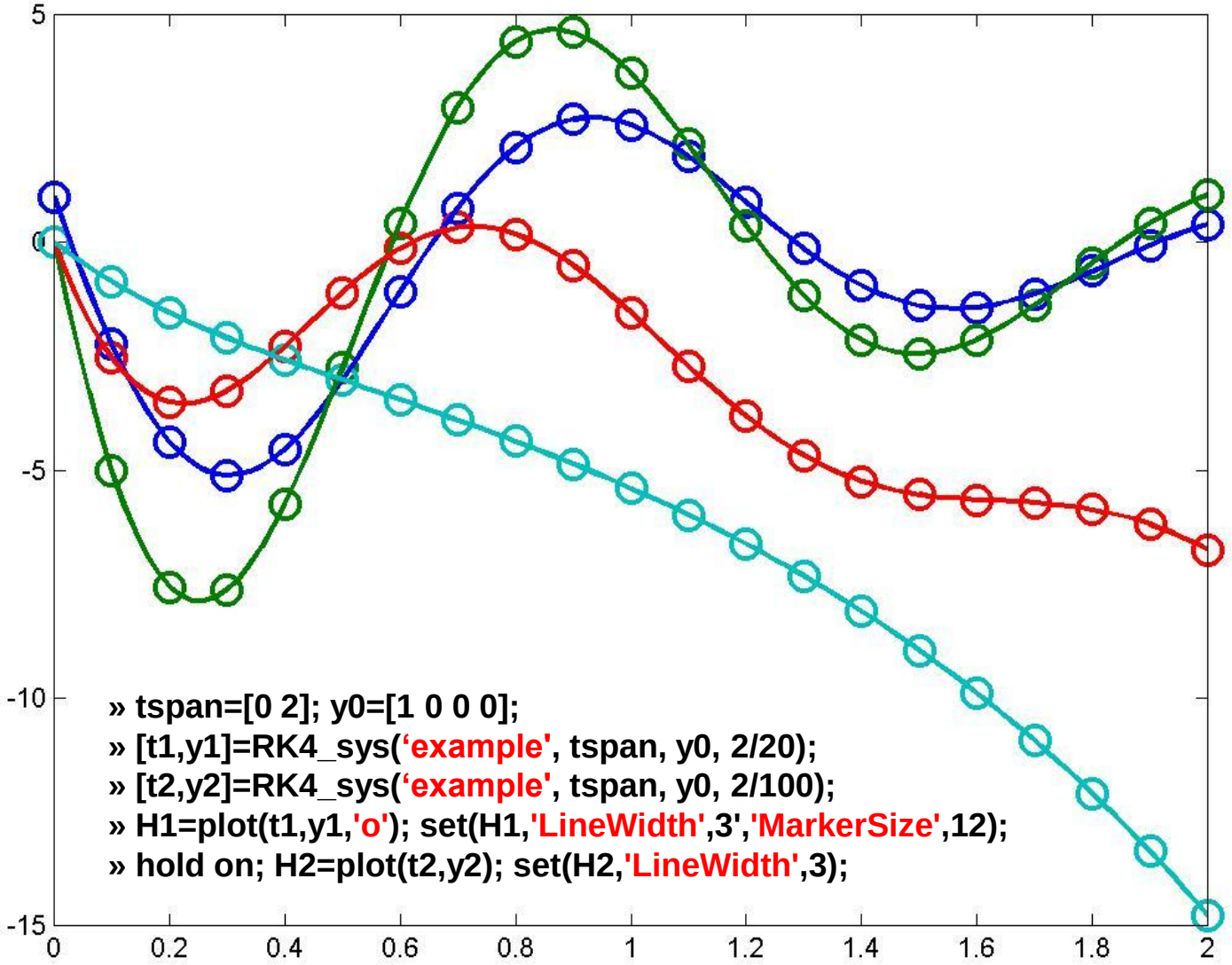
$$\begin{cases} \dfrac{dy_1}{dt} = f_1 = -36y_1 + 30y_2 - 20y_3 + 10y_4 \; ; \; y_1(0) = 1 \\ \dfrac{dy_2}{dt} = f_2 = -61y_1 + 50y_2 - 36y_3 + 18y_4 ; \; y_2(0) = 0 \\ \dfrac{dy_3}{dt} = f_3 = -34y_1 + 29y_2 - 25y_3 + 13y_4 ; \; y_3(0) = 0 \\ \dfrac{dy_4}{dt} = f_4 = -10y_1 + 10y_2 - 10y_3 + 6y_4 ; \; y_4(0) = 0 \end{cases}$$

$$A = \begin{bmatrix} -36 & 30 & -20 & 10 \\ -61 & 50 & -36 & 18 \\ -34 & 29 & -25 & 13 \\ -10 & 10 & -10 & 6 \end{bmatrix}$$

$$t0 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}'$$

```
function f = example(t, y)
% solve y' = Ay = f, y0 = [1 0 0 0]'
% four first-order ODE-IVPs
 A = [ -36  30 -20  10
       -61  50 -36  18
       -34  29 -25  13
       -10  10 -10   6];
y=[ y(1) y(2) y(3) y(4)]';
f = A*y;
```

另1示例 (4维ODE方程组)



知识结构图

常微分方程初值问题数值解法

- 单步法
  - 重要概念（截断误差、方法精度、收敛性、稳定性、绝对稳定性等）
  - 主要方法
    - 欧拉方法
    - 梯形方法
    - 龙格-库塔法（包括改进的欧拉法）
- 绝对稳定空间
- 转一阶方程组
- 线性多步法
  - 构造方法（数值积分法、泰勒展开法）
  - 主要方法
    - 阿达姆斯显式与隐式方法
    - 米尔尼方法与辛普森方法
    - 汉明方法
    - 预测-校正方法
- 方程组与高阶方程

复习与思考题 (无需提交)

P315: 1, 2, 3, 4, 5, 7

习题 (如愿可做，无需提交)

P316: 5, 8

习题 (需提交)

P316: 1, 2, 14

在具体求解微分方程时，需要具备某种定解条件，微分方程和定解条件合在一起组成定解问题。

定解条件有两种：

1）一种是给出积分曲线在初始点的状态，称为初始条件，相应的定解问题称为初值问题；

2）另一种是给出积分曲线首尾两端的状态，称为边界条件，相应的定解问题则称为边值问题。

终值问题，稳态问题