高等数值算法与应用（十）

Advanced Numerical Algorithms & Applications

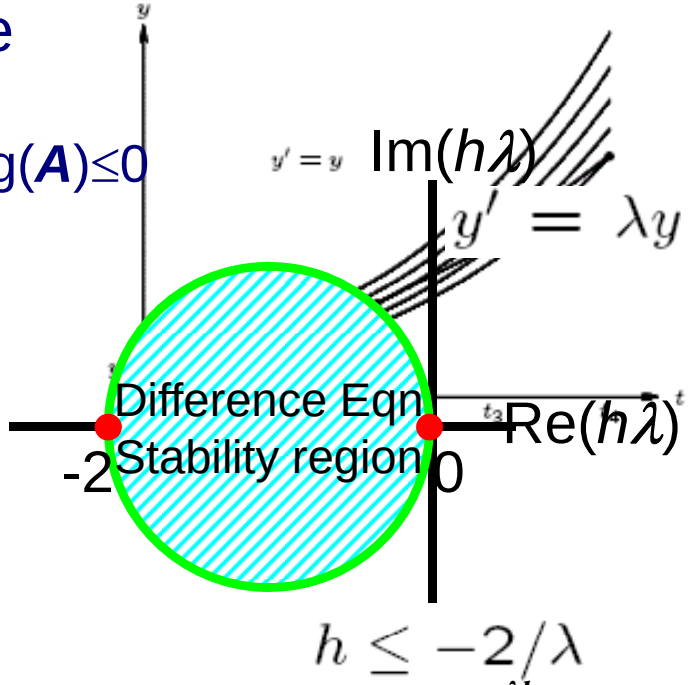
计算机科学与技术系 喻文健

Summary - Ordinary differential equations

- 求解未知函数（单自变量t）
- 阶数——方程中未知函数求导阶数 $y' = f(t, y)$
- 高阶**ODE** → 一阶**ODE**形成的方程组（一阶向量**ODE**）
- 初值问题（IVP） $y(t_0) = y_0$
- Stability of problem （初值的扰动对准确解的影响）
  - Stable, Asymptotically stable, Unstable
  - 简单的例子： $y' = \lambda y,\ \ \boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y}$
  - 一般的非线性问题 $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$

Eig($\boldsymbol{A}$)≤0

局部稳定：Eig($\boldsymbol{J}_f$) ≤0



$h \le -2/\lambda$

- Euler's method
  - 整体截断误差 $e_k = y_k - y(t_k)$
  - 局部截断误差 $\ell_k = y_k - u_{k-1}(t_k)$
  - Order p accuracy: $\ell_k = \mathcal{O}(h_k^{p+1})$
  - Stability, stability region
  - 1阶准确度（一般性） $\ell_k = y_k - u_{k\text{-}1}(t_k) = (1 + h\lambda)y_{k\text{-}1} - y_{k\text{-}1}e^{\lambda h}$

Ordinary Differential Equations (II)

Wenjian Yu

内容概要

- 常微分方程数值解法—初值问题
  - 向后欧拉法，梯形法
    - 稳定性、准确度 (考察模型问题, 一般问题)
  - **Runge-Kutta法**
    - **2级方法的推导, 经典4阶方法， 自动R-K算法**
  - 刚性**ODE-IVP**问题
    - 两个例子、一般概念
  - 多步法
    - 待定系数法推导, **Adams**公式, **BDF**公式思想
  - **Matlab**与应用
    - **Matlab**命令, **Lorenz**吸引子

Euler's Method $y_{k+1} = y_k + h_k f(t_k, y_k)$

优点：显格式，因此计算简单
缺点：要保证算法稳定，步长往往很小，因此整体效率低

考虑从时间点t_k到t_{k+1}

$$y_{k+1} = \int_{t_k}^{t_k+h_k} f(s, y(s))ds + y_k$$

$$\approx h_k f(t_k, y(t_k)) + y_k \qquad \approx h_k f(t_{k+1}, y(t_{k+1})) + y_k$$

y_{k+1}代替准确的$y(t_{k+1})$，得到向后(Backward)欧拉法:

$$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$$

单步、隐格式

y_{k+1}“隐藏”在f 函数中, 要通过求解方程计算

- 一般是非线性方程，采用牛顿法等迭代求解技术
- 迭代初值可取显格式公式结果，使收敛较快

Example: Backward Euler Method

Consider nonlinear scalar ODE

$$y' = -y^3 \qquad f(t,y) = -y^3$$

with initial condition $y(0) = 1$

Using backward Euler method with step size $h = 0.5$, we obtain equation

$$y_1 = y_0 + hf(t_1, y_1) = 1 - 0.5y_1^3$$

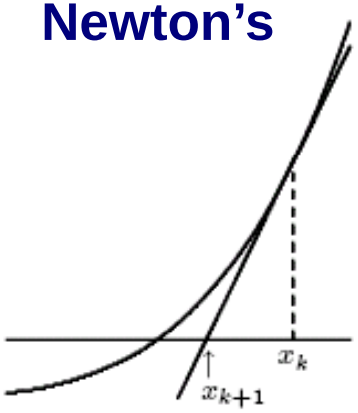
for solution value at next step

This nonlinear equation for y_1 could be solved by fixed-point iteration or Newton's method

To obtain starting guess for y_1, we could use previous solution value, $y_0 = 1$, or we could use an explicit method. Euler's method gives $y_1 = y_0 - 0.5y_0^3 = 0.5$

Iterations eventually converge to final value $y_1 \approx 0.7709$

Newton's



$$x_{k+1} = x_k - f(x_k)/f'(x_k)$$

此处 $f(x) = 0.5x^3 + x - 1$

To determine stability of backward Euler, we apply it to scalar ODE $y' = \lambda y$, obtaining

$$y_{k+1} = y_k + h\lambda y_{k+1},$$

or

$$(1 - h\lambda)y_{k+1} = y_k,$$

so that

$$y_k = \left(\frac{1}{1 - h\lambda}\right)^k y_0$$

隐格式方法每步计算量更大，好处在哪里？

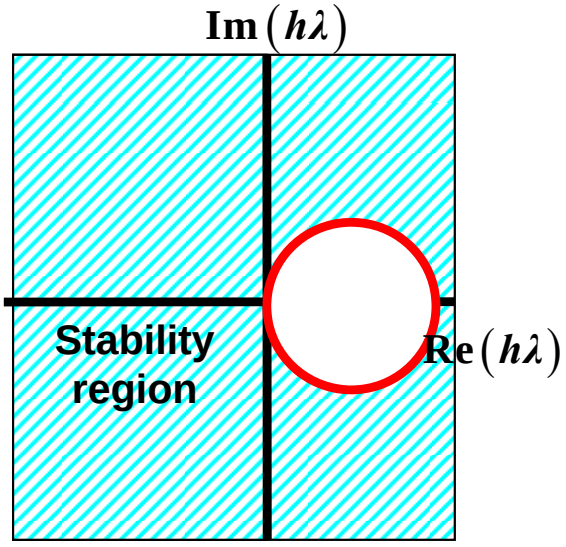
若 $\frac{1}{1-h\lambda} > 1$，y_k的值发散

Thus, for backward Euler to be stable we must have

$$\left|\frac{1}{1 - h\lambda}\right| \leq 1,$$

which holds for *any* $h > 0$ when $\mathrm{Re}(\lambda) < 0$

So stability region for backward Euler method includes entire left half of complex plane, or interval $(-\infty, 0)$ if λ is real



向后欧拉法的准确度

考虑局部误差：$\ell_k = y_k - u_{k-1}(t_k) = (1-h\lambda)^{-1} y_{k-1} - y_{k-1} e^{\lambda h}$

$$\frac{1}{1-h\lambda} = 1 + h\lambda + (h\lambda)^2 + \cdots$$

$$e^{h\lambda} = 1 + h\lambda + \frac{(h\lambda)^2}{2} + \frac{(h\lambda)^3}{6} + \cdots$$

所以 $\ell_k \sim \mathcal{O}(h^2)$，为1阶准确度

对模型问题 $y' = \lambda y$ 分析的结论可推广到一般问题!

求解一般问题时的稳定性 $y' = f(t, y)$

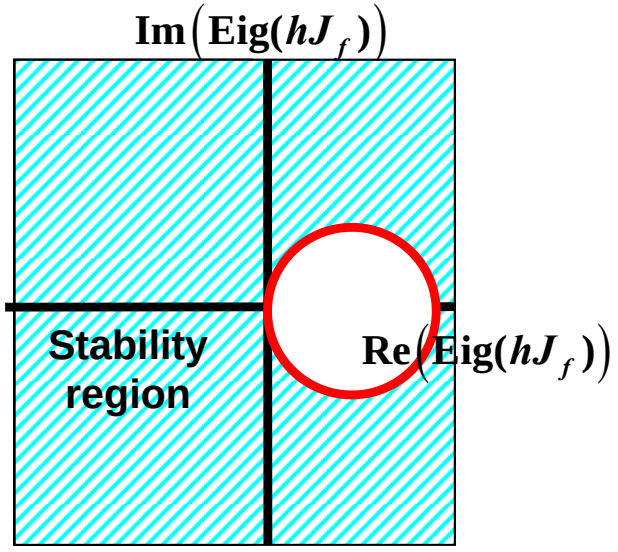
$$e_{k+1} = y_{k+1} - y(t_{k+1})$$

$$= y_k + h_k f(t_{k+1}, y_{k+1}) - (y(t_k) + h_k f(t_{k+1}, y(t_{k+1})) - O(h_k^2))$$

$$f(t_{k+1}, y_{k+1}) - f(t_{k+1}, y(t_{k+1})) = J_f(t_{k+1}, \xi)(y_{k+1} - y(t_{k+1}))$$

$$\Rightarrow e_{k+1} = e_k + h_k J_f(\ldots) e_{k+1} + O(h_k^2)$$

$$\Rightarrow e_{k+1} = (I - h_k J_f)^{-1} e_k + \ldots$$



误差不发散的算法稳定性要求：

$$\left|\mathrm{Eig}((I - h_k J_f)^{-1})\right| \le 1$$

对稳定的ODE问题，无条件稳定!

$$\mathrm{Eig}(J_f) \le 0$$

Trapezoid Method

欧拉、向后欧拉法都只有**1**阶准确度

Higher-order accuracy can be achieved by averaging Euler and backward Euler methods to obtain implicit *trapezoid rule*

$$y_{k+1} = y_k + h_k \left(f(t_k, y_k) + f(t_{k+1}, y_{k+1}) \right) /2$$

$$y_{k+1} = \int_{t_k}^{t_k+h_k} f(s, y(s)) ds + y_k$$

对模型问题分析稳定性和准确度

To determine its stability and accuracy, we apply it to scalar ODE $y' = \lambda y$, obtaining

$$y_{k+1} = y_k + h \left(\lambda y_k + \lambda y_{k+1} \right) /2,$$

which implies that

$$y_k = \left(\frac{1 + h\lambda/2}{1 - h\lambda/2} \right)^k y_0$$

Method is stable if

$$\left| \frac{1 + h\lambda/2}{1 - h\lambda/2} \right| < 1, \quad \Longleftrightarrow \quad \mathbf{Re}(h\lambda)<0$$

which holds for any $h > 0$ when $\mathrm{Re}(\lambda) < 0$, so trapezoid rule is unconditionally stable

Trapezoid Method, continued

考虑局部误差：$\ell_k = y_k - u_{k-1}(t_k) = \dfrac{1+h\lambda/2}{1-h\lambda/2} y_{k-1} - y_{k-1}e^{\lambda h}$

$$\begin{aligned}\frac{1+h\lambda/2}{1-h\lambda/2} &= \left(1+\frac{h\lambda}{2}\right)\left(1+\frac{h\lambda}{2}+\left(\frac{h\lambda}{2}\right)^2\right. \\ &\quad \left.+\left(\frac{h\lambda}{2}\right)^3+\cdots\right) \\ &= 1+h\lambda+\frac{(h\lambda)^2}{2}+\frac{(h\lambda)^3}{4}+\cdots\end{aligned}$$

agrees with expansion of $e^{h\lambda}$ through terms of order h^2, so trapezoid method is second-order accurate

~$O(h^3)$，即二阶准确度

More generally, trapezoid method has growth factor $(I+\frac{1}{2}hJ_f)(I-\frac{1}{2}hJ_f)^{-1}$, whose spectral radius is less than 1 provided eigenvalues of hJ_f lie in left half of complex plane

无条件稳定：对稳定的**ODE**问题，步长无限制

无条件稳定的方法最高阶数为**2**，但各种隐格式方法比显格式方法有更大的稳定区间

Runge-Kutta Methods

更普遍的单步法(可具有更高阶准确度, 主要讨论显格式)

构造思想 $y_{k+1}=\int_{t_k}^{t_k+h_k} f(s,y(s))ds+y_k$

机械求积公式 $y_{k+1}=y_k+h_k\sum_{i=1}^{r}c_i f(t_k+\lambda_i h_k,\ y(t_k+\lambda_i h_k))$

$\lambda_i\in[0,\ 1]$, c_i,λ_i 为待定系数　　待定系数越多，精度阶越高

一般 $\lambda_1=0$，怎么算其他积分点对应的 $y(t_k+\lambda_i h_k)$？

时间值	y函数的近似值	f函数近似值
t_k	y_k	$f(t_k,y_k)\equiv k_1$
$t_k+\lambda_2 h$	$y_k+\lambda_2 h k_1$	$f(t_k+\lambda_2 h,y_k+\lambda_2 h k_1)\equiv k_2$
$t_k+\lambda_3 h$	$y_k+h\sum_{j=1}^{2}\mu_{3,j}k_j$	$f\left(t_k+\lambda_3 h,y_k+h\sum_{j=1}^{2}\mu_{3,j}k_j\right)\equiv k_3$
...	...	...

*r*级Runge-Kutta方法，*r*级表示需计算*r*次*f*()函数

2级Runge-Kutta方法的推导

$$\begin{cases} k_1 = f(t_k, y_k) \\ k_2 = f(t_k + \lambda_2 h_k, y_k + \lambda_2 h_k k_1) \\ y_{k+1} = y_k + h_k(c_1 k_1 + c_2 k_2) \end{cases}$$

通过局部截断误差确定 c_1, c_2, λ_2

其中 $y_k = y(t_k)$

$f_k \equiv f(t_k, y_k) = y'(t_k)$

$$\begin{aligned} l_{k+1} &= y_{k+1} - y(t_k + h) \\ &= y_k + h(c_1 f(t_k, y_k) + c_2 f(t_k + \lambda_2 h, y_k + \lambda_2 h k_1)) \\ &\quad - y(t_k + h) \end{aligned}$$

$$y(t_k + h) = y_k + h f_k + \frac{h^2}{2}\frac{df(t_k, y_k)}{dt} + O(h^3)$$

$$f(t_k + \lambda_2 h, y_k + \lambda_2 h f_k) = f_k + \lambda_2 h \frac{\partial f}{\partial t}(t_k, y_k) + \lambda_2 h f_k \frac{\partial f}{\partial y}(t_k, y_k) + O(h^2)$$

$$\begin{aligned} l_{k+1} &= y_k + h f_k (c_1 + c_2) + h c_2 \lambda_2 h f_t'(t_k, y_k) + h c_2 \lambda_2 h f_k f_y'(t_k, y_k) + O(h^3) \\ &\quad - y_k - h f_k - 0.5 h^2 f_t'(t_k, y_k) - 0.5 h^2 f_y'(t_k, y_k) y_k' - O(h^3) \end{aligned}$$

2阶精度

$$l_{k+1} = O(h^3) \Rightarrow \begin{cases} c_1 + c_2 = 1 \\ c_2 \lambda_2 = 0.5 \end{cases}$$

对模型问题 $y' = \lambda y$
分析得到相同的结论

解不唯一，常见的是 *Heun's method*

$$\begin{aligned} y_{k+1} &= y_k + \frac{h_k}{2}(k_1 + k_2), \\ k_1 &= f(t_k, y_k) \\ k_2 &= f(t_k + h_k, y_k + h_k k_1) \end{aligned}$$

也叫”改进Euler”法

2级最多是2阶

Runge-Kutta Methods

2级Runge-Kutta方法的稳定性

考察模型问题 $y' = \lambda y$

$$y_{k+1} = y_k + 0.5h(\lambda y_k + \lambda(y_k + h\lambda y_k))$$
$$= y_k(1 + h\lambda + \frac{(h\lambda)^2}{2})$$

$$\rightarrow \left|1 + h\lambda + \frac{(h\lambda)^2}{2}\right| < 1 \rightarrow -2 < h\lambda < 0$$

步长h不能太大！

Heun's method

$$y_{k+1} = y_k + \frac{h_k}{2}(k_1 + k_2),$$
$$k_1 = f(t_k, y_k)$$
$$k_2 = f(t_k + h_k, y_k + h_k k_1)$$

经典的4阶Runge-Kutta方法

$$y_{k+1} = y_k + \frac{h_k}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$
$$k_1 = f(t_k, y_k)$$
$$k_2 = f(t_k + h_k/2, y_k + (h_k/2)k_1)$$
$$k_3 = f(t_k + h_k/2, y_k + (h_k/2)k_2)$$
$$k_4 = f(t_k + h_k, y_k + h_k k_3)$$

阶数超过4的R-K公式，其级数一定比阶数大，因此很少用

与2级公式一样，稳定性对步长h有限制

自动的Runge-Kutta方法

- 基本思想
  - 两个公式分别为**p, p+1**阶准确度，它们的差为误差估计

$$\hat{y}_{k+1} - u_k(t_k + h) = O(h^{p+1}) = c_{p+1}h^{p+1} + O(h^{p+2})$$

$$y_{k+1} - u_k(t_k + h) = O(h^{p+2})$$

$$\Rightarrow \quad l_{k+1} \approx c_{p+1}h^{p+1} \approx \hat{y}_{k+1} - y_{k+1}$$

  - 取高阶准确度的值 y_{k+1} 作为这一步的结果
- BS23算法
  - Bogachi & Shampine [1989]提出, (2, 3阶R-K方法)

3阶公式：

$$y_{k+1} = y_k + \frac{h}{9}(2k_1 + 3k_2 + 4k_3)$$

$$k_1 = f(t_k, y_k)$$

$$k_2 = f(t_k + h/2,\ y_k + (h/2)k_1)$$

$$k_3 = f(t_k + 3h/4,\ y_k + (3h/4)k_2)$$

2阶公式：

$$k_4 = f(t_{k+1},\ y_{k+1})$$

$$\hat{y}_{k+1} = y_k + \frac{h}{24}(7k_1 + 6k_2 + 8k_3 + 3k_4)$$

误差估计式：

$$y_{k+1} - \hat{y}_{k+1} = \frac{h}{72}(-5k_1 + 6k_2 + 8k_3 - 9k_4)$$

自动的Runge-Kutta方法

- BS23算法
  - Matlab中的ode23函数
  - 每步计算3次f()函数
  - 若e_{k+1}小于阈值，则进入下一步(k_4得到复用)，否则减小步长重算当前步

$$k_1 = f(t_k, y_k),\ k_2 = f(t_k + h/2,\ y_k + (h/2)k_1)$$

$$k_3 = f(t_k + 3h/4,\ y_k + (3h/4)k_2)$$

$$y_{k+1} = y_k + \frac{h}{9}(2k_1 + 3k_2 + 4k_3)$$

$$k_4 = f(t_{k+1},\ y_{k+1})$$

$$e_{k+1} = \frac{h}{72}(-5k_1 + 6k_2 + 8k_3 - 9k_4)$$

- ode45的内部算法
  - 每步计算6次f()函数
  - 形成4阶和5阶的R-K公式，它们的差为局部误差估计
- R-K方法小结
  - 也可构造隐格式R-K方法，更稳定　　对刚性问题很重要
  - R-K方法是单步法，易于改变步长，便于编程实现

Stiff Differential Equations

“刚性”常微分方程
“困难”常微分方程

一个例子

Consider scalar ODE

$$y' = -100y + 100t + 101$$

with initial condition $y(0) = 1$

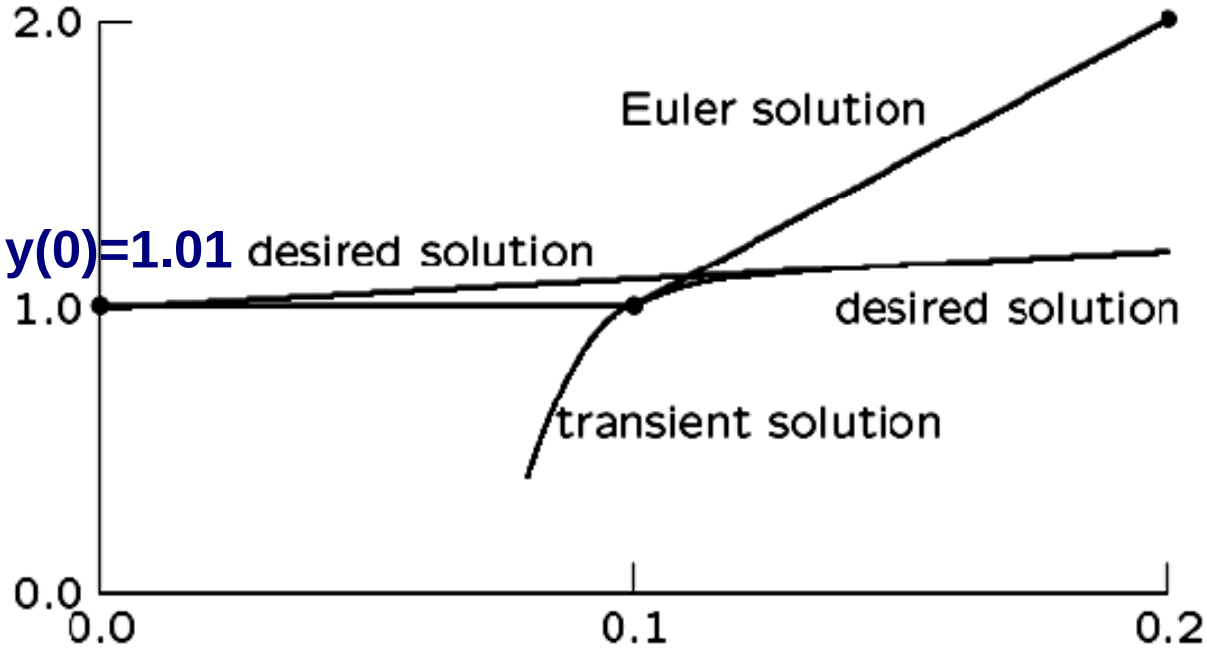
General solution is $y(t) = 1 + t + ce^{-100t}$, and particular solution satisfying initial condition is $y(t) = 1 + t$ (i.e., $c = 0$)

h=0.1

初始值准，**Euler**准确
初值扰动，**Euler**不准

t	0.0	0.1	0.2	0.3	0.4
exact sol.	1.00	1.10	1.20	1.30	1.40
Euler sol.	0.99	1.19	0.39	8.59	−64.2
Euler sol.	1.01	1.01	2.01	−5.99	67.0

Computed solution is incredibly sensitive to initial value, as each tiny perturbation results in wildly different solution



常识：对平缓变化的函数，求解步长可大一些；
对变化剧烈的函数，求解步长要小一些。

这个例子的现象有点违背常识

分析：

解函数本身变化缓慢，但其周围的解变化快 $y(t)=1+t+c\cdot e^{-100t}$

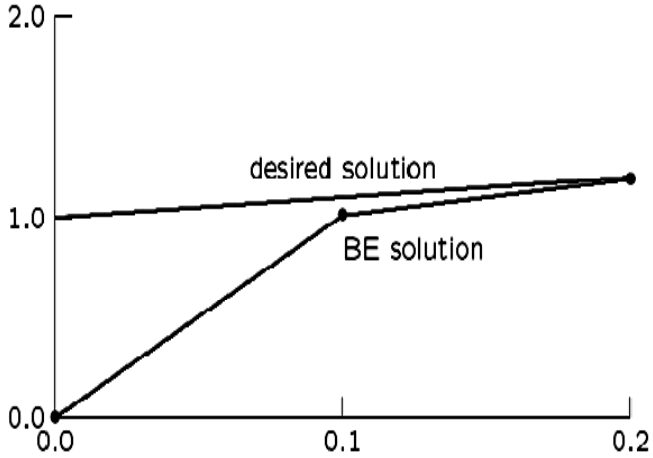
欧拉法的步长要求： $h\le -2/\lambda = 0.02$ 取**h=0.1**违背了稳定性要求

用向后Euler法求解

用隐格式计算不受步长约束

Backward Euler method has no trouble solving this problem and is extremely *insensitive* to initial value:

t	0.0	0.1	0.2	0.3	0.4
exact sol.	1.00	1.10	1.20	1.30	1.40
BE sol.	0.00	1.01	1.19	1.30	1.40
BE sol.	2.00	1.19	1.21	1.30	1.40



Even with very large perturbation in initial value, by using derivative at next point rather than current point, transient is quickly damped out and backward Euler solution converges to desired solution after only few steps

直观地说，若用欧拉法等显式方法求解某个**ODE**问题，需采用很小的步长才能保证精度，则这个问题就是**stiff**问题。

对刚性问题应采用稳定区间更大的隐格式方法，它们也被称为**stiff ODE solver**

例2：火焰燃烧方程

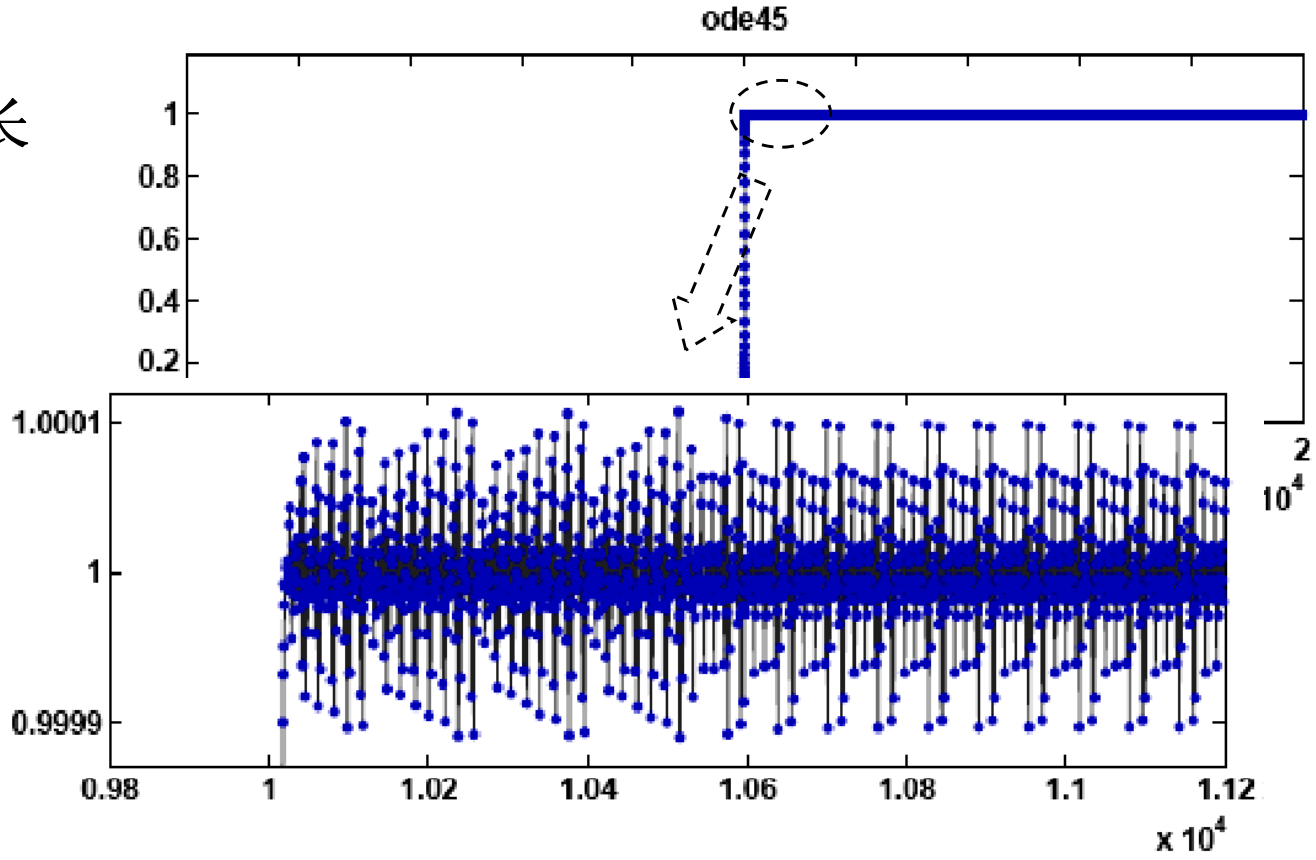
“当点燃一根火柴时，火焰迅速增大直到一个临界体积，然后维持这一体积不变，此时火焰内部燃烧耗费的氧气和其表面现存的氧气达到了一种平衡。” 火球半径**y(t)**满足：

$$\begin{cases} y' = y^2 - y^3 \\ y(0) = \eta \end{cases}$$ 设初始半径η=0.0001，求解 $y(t)$

用**R-K(4,5)**法求解，步长很小，效率很低

分析：非线性**ODE**
在**y(t)=1**处**Eig(J_f)=–1**

若用**Euler**法，步长上限为**2**, 但区间长**~10^4**



后面给出刚性求解算法求解的结果

刚性常微分方程(组)

$$h < \frac{-2}{\lambda}$$

- 刚性常微分方程：$\partial f/\partial y$ 实部的绝对值很大 (其倒数远小于求解区间)或虚部绝对值非常大(震荡型)
- 由于求解区间大小取决于解函数趋于稳定的时间，则在刚性ODE问题中：解函数变化缓慢，但其周围的解变化快(特征值的绝对值较大)
- 刚性常微分方程组: Jacobi矩阵特征值大小相差悬殊.
  解函数趋于稳定的时间由最小特征值决定，而保持稳定的步长上限由最大特征值决定

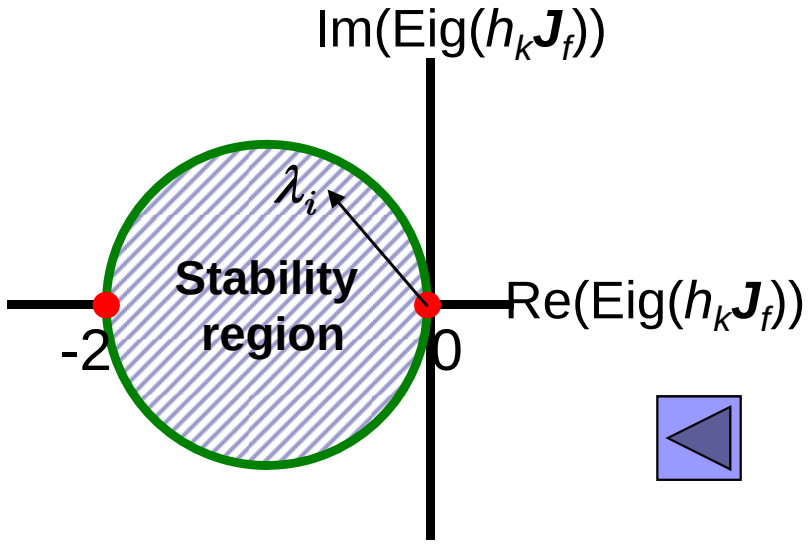
$y' = Ay$ 的通解：

$$y(t) = \sum_{i=1}^{n} \alpha_i v_i e^{\lambda_i t}$$

稳定性要求：

$$\left|1 + hEig(J_f)\right| < 1$$

$$\left|1 + h\lambda_i\right| < 1$$



有”记忆” 的公式

✓利用已有节点处函数值，而不像**R-K**法使用额外中间节点；

✓一般是等间距步长的，易于构造、计算效率高；

Multistep Methods

Multistep methods use information at more than one previous point to estimate solution at next point

线性多步法**(m**步法**)**

Linear multistep methods have form

$$y_{k+1} = \sum_{i=1}^{m} \alpha_i y_{k+1-i} + h \sum_{i=0}^{m} \beta_i f(t_{k+1-i}, y_{k+1-i})$$

If $\beta_0 = 0$, method is explicit, but if $\beta_0 \neq 0$, method is implicit

常根据多项式插值构造，便于提供非节点处函数值

如何确定一个 p 阶多步法中的系数？

多步法的计算公式为：

$y_{k+1} = \sum_{i=1}^{m} \alpha_i y_{k+1-i} + h \sum_{i=0}^{m} \beta_i f(t_{k+1-i}, y_{k+1-i})$,

在计算局部误差、评价算法准确度时，假设公式等号右边的参数都等于准确解函数。则，

$y_{k+1} = \sum_{i=1}^{m} \alpha_i y(t_{k+1-i}) + h \sum_{i=0}^{m} \beta_i y'(t_{k+1-i})$ ， (1)

对$y(t_{k+1})$在t_k点进行 Taylor 展开，有

$$y(t_{k+1}) = y(t_k + h) = y(t_k) + hy'(t_k) + \frac{h^2}{2} y''(t_k) + \cdots, \quad (2)$$

同样，我们可以对公式(1)中的$y(t_{k+1-i}), i = 1, \ldots, m$，以及$y'(t_{k+1-i}), i = 0, \ldots, m$在$t_k$点进行 Taylor 展开，然后合并同类项得到：

$$y_{k+1} = c_0 y(t_k) + c_1 hy'(t_k) + c_2 \frac{h^2}{2} y''(t_k) + \cdots, \quad (3)$$

其中c_i为与α_i, β_i有关的系数。

要有 p 阶准确度，则$l_k = y_{k+1} - y(t_{k+1}) = O(h^{p+1})$。那么，公式(2)和(3)的前 p+1 项必须对应相等，且$y_{k+1} - y(t_{k+1}) = c'_{p+1} h^{p+1} y^{(p+1)}(t_k) + \cdots$。

由于 p 次多项式的 p+1 阶导数为零，$l_k = O(h^{p+1})$等价于：对函数$y(t) = t^k, k = 0, 1, \ldots, p$有$y_{k+1} - y(t_{k+1}) \equiv 0$。即

$$\sum_{i=1}^{m} \alpha_i y(t_{k+1-i}) + h \sum_{i=0}^{m} \beta_i y'(t_{k+1-i}) \equiv y(t_{k+1})$$

因此，将 $1, t, t^2, \ldots$等单项式函数代入多步法公式，即可确定待定系数、得出其准确度阶数。

待定系数法推多步法公式

pp. 350例9.13

- 对如下显式两步法，确定系数使其有最高阶准确度

$$y_{k+1} = \alpha_1 y_k + h(\beta_1 y'_k + \beta_2 y'_{k-1})$$

注意: 将$f(t_k, y_k)$简记为y'_k

- 3个待定参数，根据 $y(t)=1,\ t,\ t^2$列方程, 可达2阶准确度

$$y(t)=1 \xrightarrow{y_{k+1}=y(t_{k+1})} 1=\alpha_1$$

$$y(t)=t \longrightarrow t_k+h=\alpha_1 t_k+h(\beta_1+\beta_2)$$

$$y(t)=t^2 \longrightarrow (t_k+h)^2=\alpha_1 t_k^2+h(2\beta_1 t_k+2\beta_2(t_k-h))$$

$$\begin{cases}\alpha_1=1\\ \beta_1=1/2\\ \beta_2=1/2\end{cases}$$

梯形法

- 进一步说明
  - 算局部误差需把 y_{k+1}和 $y(t_{k+1})$都表示为$\mathbf{t_k}$点处函数值和各阶导数值的求和形式
  - $y(t)=t^p$推出的方程表明两个展开式中 $y_k^{(p)}$的系数相同

Adams方法

公式中只有前一个函数值

$$y_{k+1} = y_k + \int_{t_k}^{t_{k+1}} f(t, y(t))dt \approx y_k + \int_{t_k}^{t_{k+1}} p(t)dt$$

$p(t)$为通过前m个值点的插值多项式，近似$f(t,y(t))$

一般公式：$y_{k+1} = y_k + h\sum_{i=0}^{m} \beta_i f(t_{k+1-i}, y_{k+1-i})$

例如：显式4步公式 (利用 $y'_k, y'_{k-1}, y'_{k-2}, y'_{k-3}$ 构造插值多项式)

$$p(t) = y'_k \frac{(t-t_{k-3})(t-t_{k-2})(t-t_{k-1})}{3h\cdot 2h\cdot h} + y'_{k-1} \frac{(t-t_{k-3})(t-t_{k-2})(t-t_k)}{2h\cdot h\cdot(-h)}$$

$$+ y'_{k-2} \frac{(t-t_{k-3})(t-t_{k-1})(t-t_k)}{h\cdot(-h)\cdot(-2h)} + y'_{k-3} \frac{(t-t_{k-2})(t-t_{k-1})(t-t_k)}{(-h)\cdot(-2h)\cdot(-3h)}$$

公式系数：$h\beta_1 = \int_{t_k}^{t_k+h} \frac{(t-t_{k-3})(t-t_{k-2})(t-t_{k-1})}{3h\cdot 2h\cdot h} dt = \frac{55}{24}h$

。。。

为何是4阶精度？

4阶Adams-Bashforth: $y_{k+1} = y_k + \frac{h}{24}(55y'_k - 59y'_{k-1} + 37y'_{k-2} - 9y'_{k-3})$

Adams方法

Stability and accuracy of some popular multi-step Adams methods are summarized below

仅α_1=1, 其他α_i=0

$h\lambda$ > ? for $y' = \lambda y,$

Explicit Methods

	Order	Stability threshold	Error constant
Euler, single step	1	-2	$1/2$
two step	2	-1	$5/12$
	3	$-6/11$	$3/8$
Adams-Bashforth	4	$-3/10$	$251/720$

局部误差：$\ell_k = ?h^{p+1}$

Implicit Methods

	Order	Stability threshold	Error constant
Backward Euler	1	$-\infty$	$-1/2$
Trapezoid	2	$-\infty$	$-1/12$
	3	-6	$-1/24$
Adams-Moulton	4	-3	$-19/720$

可证明**m**步显格式**Adams**法有**m**阶准确度；**m**步隐格式**Adams**法有**m+1**阶准确度

Implicit methods are both more stable and more accurate than corresponding explicit methods of same order

可形成一对公式，构造估计误差的自动方法

BDF隐格式方法

Backward differentiation formulas form another important family of implicit multistep methods

基于：$y'_{k+1} \approx p'(t_{k+1})$

根据y_{k+1}, y_k, …构造插值多项式p(t)近似y(t)，然后求导得到 y'_{k+1} 的近似值

$$f(t_{k+1}, y_{k+1}) = \frac{dp}{dt}(t_{k+1}) = \sum_{i=0}^{m} \alpha'_i y_{k+1-i}$$

$$y_{k+1} = \sum_{i=1}^{m} \alpha_i y_{k+1-i} + h\beta_0 f(t_{k+1}, y_{k+1})$$

仅β_0不为0，其他β_i都为0

BDF methods, typified by formula

$$y_{k+1} = \frac{1}{11}(18y_k - 9y_{k-1} + 2y_{k-2}) + \frac{6h}{11}y'_{k+1},$$

are effective for solving stiff ODEs

三阶方法

稳定区间？涉及特征多项式的求解

Properties of Multistep Methods

They are not self-starting, since several previous values of y_k are needed initially

一般用同阶**R-K**法得到前几个点

Changing step size is complicated, since interpolation formulas are most conveniently based on equally spaced intervals for several consecutive points

注意：等步长不是必需的

变步长的推导、实现较复杂

Being based on interpolation, they can efficiently provide solution values at output points other than integration points

易于得到非节点函数值。另外，计算效率比**R-K**法高

Properly designed implicit multistep method can be very effective for solving stiff ODEs

如**BDF**方法

多值法(课本**9.3.9**)是多步法的发展，更易于变步长

Variable-Order/Variable-Step Solvers

Such routines are analogous to adaptive quadrature routines: they automatically adapt to given problem, varying order and step size of integration method as necessary to meet user-supplied error tolerance efficiently

Such routines often have options for solving either stiff or nonstiff problems, and some even detect stiffness automatically and select appropriate method accordingly

Ability to change order easily also obviates need for special starting procedures: one can simply start with first-order method, which requires no additional starting values, and let automatic order/step size selection procedure increase order as needed for required accuracy

构成现在的主要软件

Matlab issues

- Matlab commands for ODE
  - ode45 (R-K(4,5))：一般情况下，对大多数问题先尝试使用**ode45**求解
  - ode23 (R-K(2,3))：对误差要求不高的非刚性问题，比**ode45**效率更高
  - ode113 (Adams-Bashforth-Moulton PECE)：多步法，适合函数***f***求值代价高的情况，计算精度也较高
  - ode23t (梯形公式)：精度要求不高的适度刚性问题
  - ode15s (BDF)：隐式多步法，求解刚性问题
  - ode23s (**2**阶单步法)：精度要求不高时比**ode15s**有效
  - ode23tb (梯形**+**BDF)：实质上为**2**级隐式**R-K**
- Inline命令定义函数　**f=inline('-10*y*t', 't','y');**

Solving IVP in Matlab using ode23 on

$$\begin{cases} y' = f(t,y) \\ y(t_0) = y_0 \end{cases}$$

- Syntax:
  - `[T,Y,TE,YE,IE] =` **ODE23**`(ODEFUN,TSPAN,Y0,OPTIONS,P1,P2,...)`

    optional

    order 2 and order 3 RK methods of Bogacki and Shampine (1989)
  - **ODE23**`(ODEFUN,[T0 TFINAL],Y0...)` gives graphical output

    计算的步长由**ODE**解法决定，并输出
- Key inputs:
  - `ODEFUN`: function *f*(*t*,*y*) as *column* vector
  - `TSPAN = [T0 T1 ... TFINAL]`: time steps including starting and final time values

    计算中就采用这里给定的步长和节点
  - `Y0`

Matlab demos

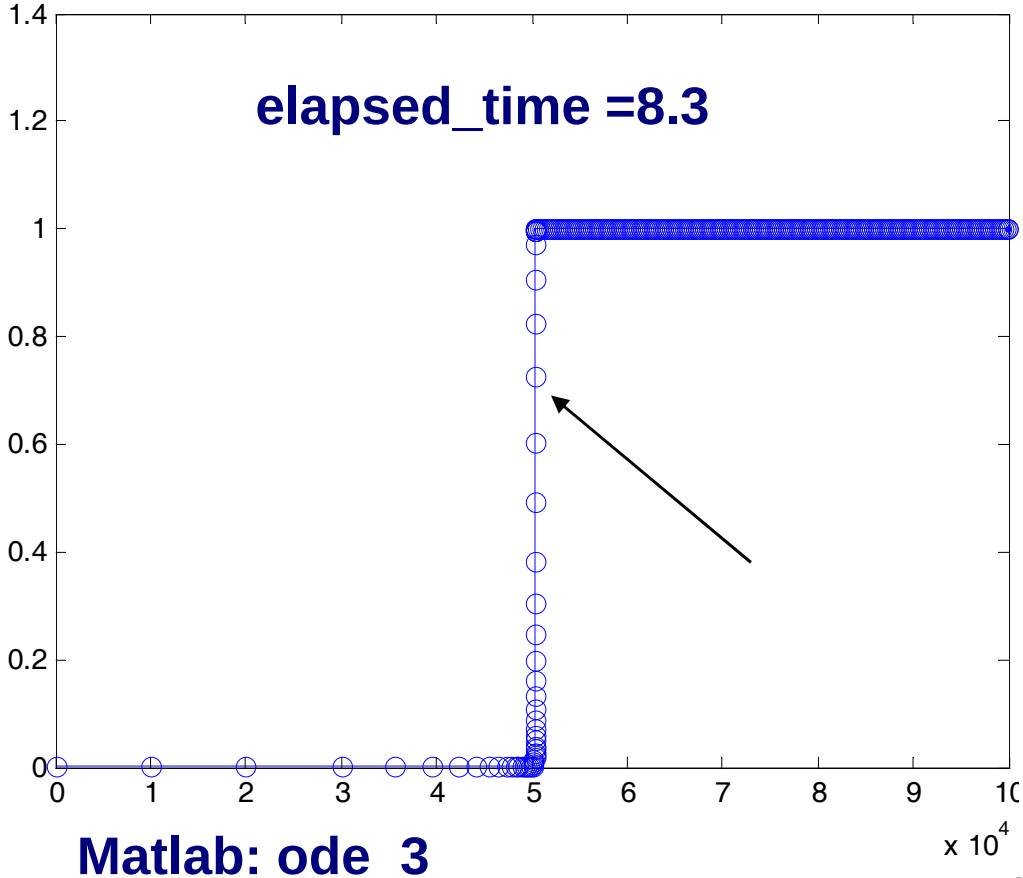
- “火焰燃烧”的例子

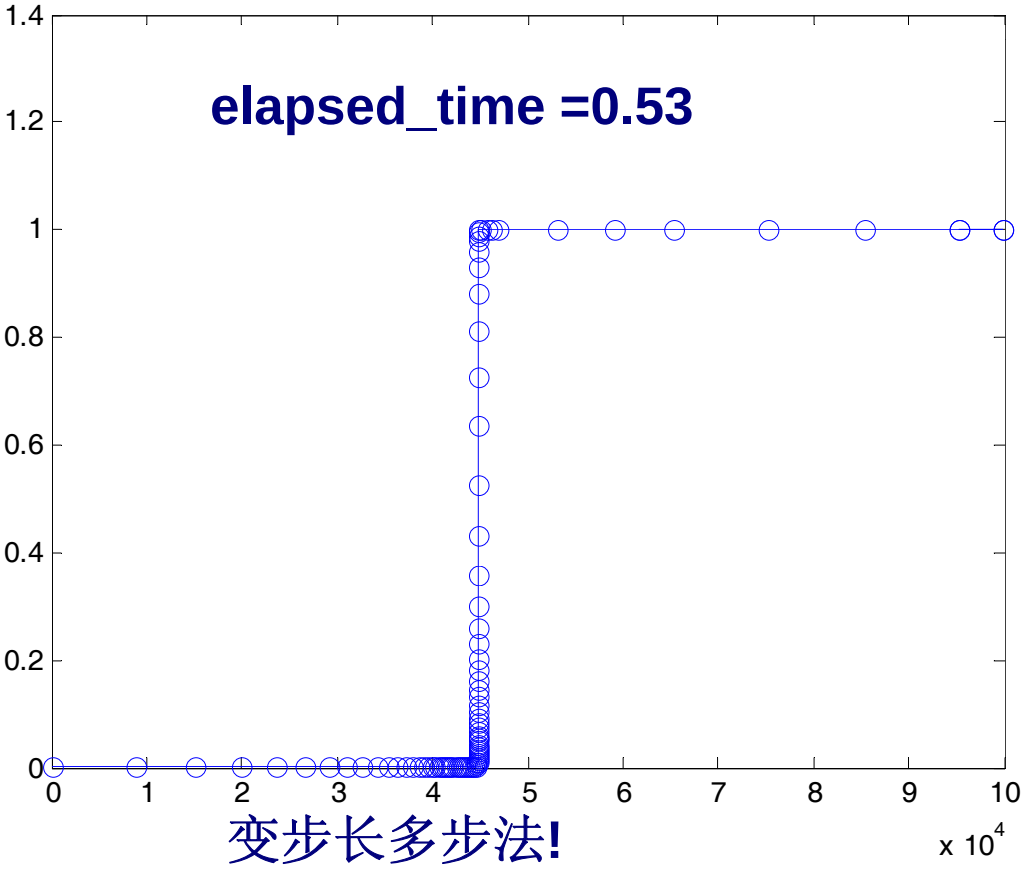
$$\begin{cases} y' = y^2 - y^3 \\ y(0) = \eta \\ 0 \le t \le 2/\eta \end{cases}$$

Try η=0.00002

f= inline('y^2-y^3', 't', 'y');
ode23(f, [0, 2/eta], eta)

ode15s(f, [0, 2/eta], eta);





Matlab: ode_3

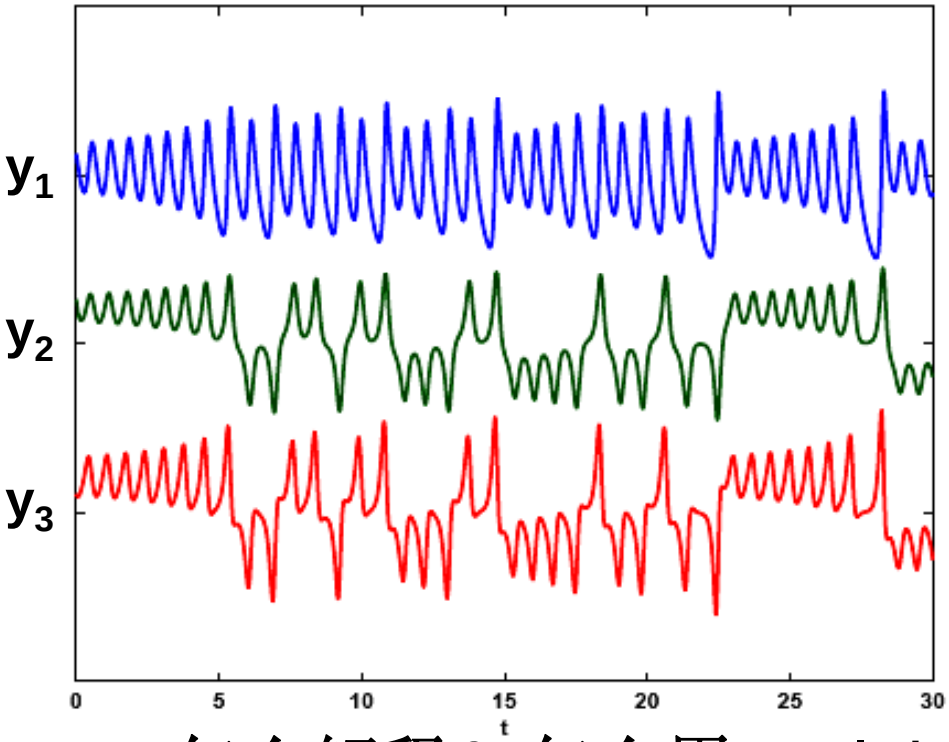
变步长多步法!

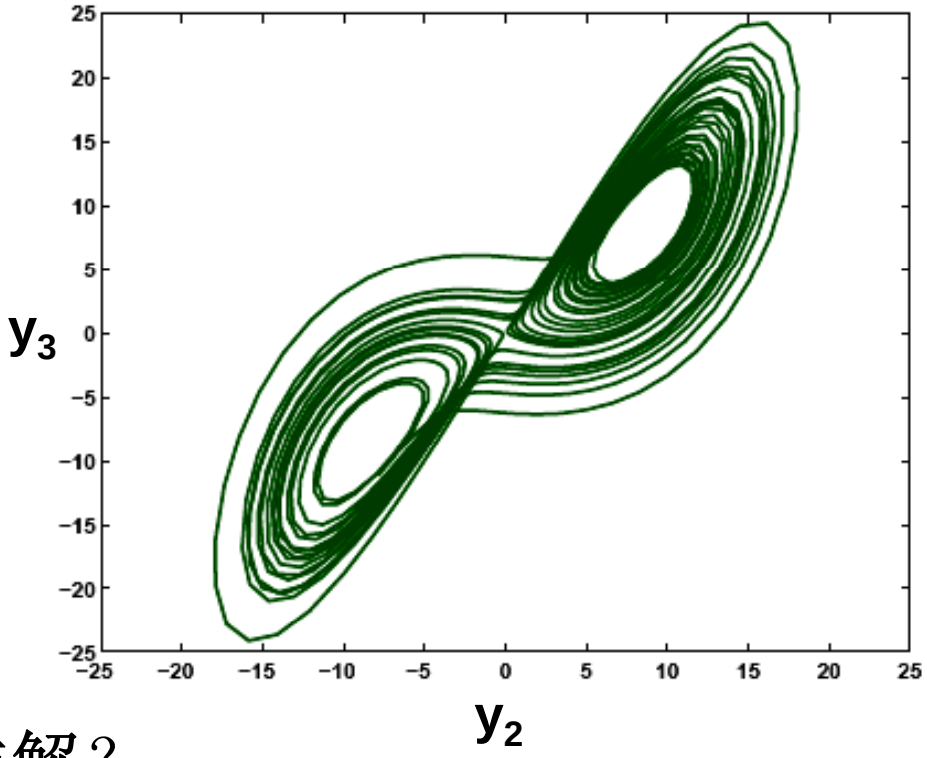
实例与Matlab demo

- Lorenz吸引子
  - 1963年，MIT的Edward Lorenz研究地球大气的流体模型时发现的现象
  - n=3的常微分方程组 $\dot{y} = Ay$ ， $A = \begin{bmatrix} -\beta & 0 & y_2 \\ 0 & -\sigma & \sigma \\ -y_2 & \rho & -1 \end{bmatrix}$
  - $y_1(t)$与大气对流有关, $y_2(t)$和$y_3(t)$与水平和垂直的温度变化有关
  - σ: Prandtl数, β: 与区域的几何有关, ρ: 规范化的Rayleigh数
  - 看似简单的非线性使系统表现非常复杂，虽然参数是确定的，但会产生“混沌”现象
  - 三维空间中**y**(t)的轨迹混乱地在两个点（吸引子）之间往返，有界但无周期，不收敛也不自交

实例与Matlab demo

- Lorenz吸引子





- 怎么解释？怎么用Matlab求解？
- 矩阵**A**有可能奇异，若有非零的**y**(0)使**Ay=0**，则**y**(0)为ODE的不动点
- 实验表明这些不动点不稳定，接近它们时会被排斥

实例与Matlab demo

- Lorenz吸引子

$$A=\begin{bmatrix} -\beta & 0 & y_2 \\ 0 & -\sigma & \sigma \\ -y_2 & \rho & -1 \end{bmatrix} \Rightarrow y_2 = \pm\sqrt{\beta(\rho-1)} \text{ 时奇异}$$

  - 要使**Ay(0)=0**，求**A**的化零向量，并使其第2个分量为y_2
  - 求解得到：$\begin{bmatrix} -\beta & 0 & y_2 \\ 0 & -\sigma & \sigma \\ -y_2 & \rho & -1 \end{bmatrix}\begin{bmatrix} \rho-1 \\ y_2 \\ y_2 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
  - 两个吸引子: $\left[\rho-1, \sqrt{\beta(\rho-1)}, \sqrt{\beta(\rho-1)}\right]^T$, $\left[\rho-1, -\sqrt{\beta(\rho-1)}, -\sqrt{\beta(\rho-1)}\right]^T$
  - 若参数σ=10, β=8/3, ρ=28, 吸引子为**(27, ±8.5, ±8.5)**
  - 求解程序，以及演示程序

实例与Matlab demo

- Lorenz吸引子

演示程序：**lorenzgui**

comp9_6.m

comp9_1.m

```
rho = 28;
sigma = 10;
beta = 8/3;
eta = sqrt(beta*(rho-1));      % √(β(ρ−1))
A = [ -beta    0      eta
         0  -sigma    sigma
       -eta   rho     -1  ];   % 形成矩阵A

yc = [rho-1; eta; eta];        % 吸引子位置
y0 = yc + [0; 0; 3];           % 初始值

tspan = [0 Inf];
opts = odeset('reltol',1.e-6,'outputfcn',@lorenzplot);
ode45(@lorenzeqn, tspan, y0, opts, A);     % A为额外参数

function ydot = lorenzeqn(t,y,A)
A(1,3) = y(2);
A(3,1) = -y(2);
ydot = A*y;
```

设置输出处理函数

相位显示：**opts= odeset('outputfcn', @odephas2, 'outputSel', [1, 2]);**