**ODEs差分解法**

**一．前向Euler法**

一阶ODE一般形式为：



前向Euler法是求解ODE最简单最常用的方法，利用每一点处的斜率代入ODE计算，再外推找到下一个点。



故（显式方法）



其中，.

**例1** 用前向Euler法求解微分方程：



**代码**：

% 微分方程系数: a x' = b x + c t

a = 1.0; b = -2.0; c = 1.0;

% 初始和终止时间

tinit = 0; tmax = 5.0;

% 迭代步数和步长

maxt = 3000;

dt = (tmax - tinit) / maxt;

% 初始条件

x(1) = 1.0; t(1) = tinit;

% 循环迭代求解

for j = 1:maxt

x(j+1) = x(j) + dt \* ((b \* x(j) + c \* (j) \* dt) / a);

t(j+1) = tinit + j \* dt; % t = tinit + j \* dt

end

% 绘图

plot(t, x)

title('前向Euler法')

xlabel('t'), ylabel('x(t)')



**二．后向Euler法**

后向Euler法是在新位置做计算（隐式迭代）：



**三．中点法**

对称的方法计算导数，用只涉及奇数次幂的泰勒展式（更精确）：



但是需要特殊的初始化生成额外的过去值：



**例2** 用中点法求解微分方程：



**代码**：

% 微分方程系数: a x' = b x + c t

a = 1.0; b = -2.0; c = 1.0;

% 初始和终止时间

tinit = 0; tmax = 5.0;

% 迭代次数和步长

maxt = 3000;

dt = (tmax - tinit) / maxt;

% 初始条件

x(2) = 1.0;

x(1) = 1.0 - dt \* ((b \* x(2) + c \* (2) \* dt) / a);

t(2) = tinit;

% 循环迭代

for j = 2:(maxt+1) % t = tinit + j \* dt

x(j+1) = x(j-1) + 2.0 \* dt \* ((b \* x(j) + c \* (j) \* dt) / a);

t(j+1) = tinit + (j-1) \* dt;

end

% 绘图

plot(t, x)

title('中点法')

xlabel('t'), ylabel('x(t)')

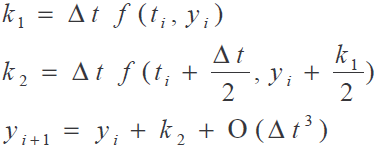


**三. 二阶****Runge-Kutta法（梯形法）**

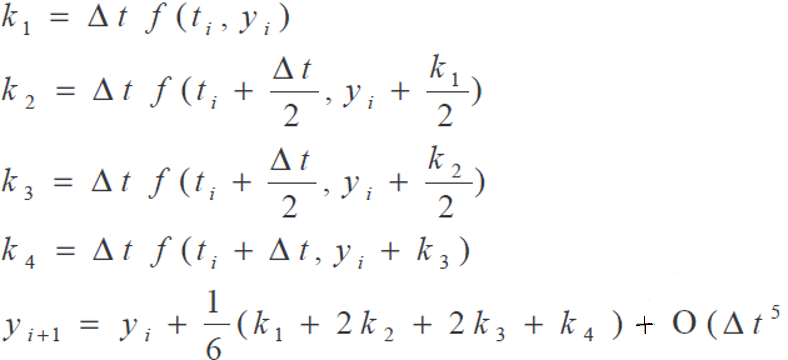
中点法可改写为：

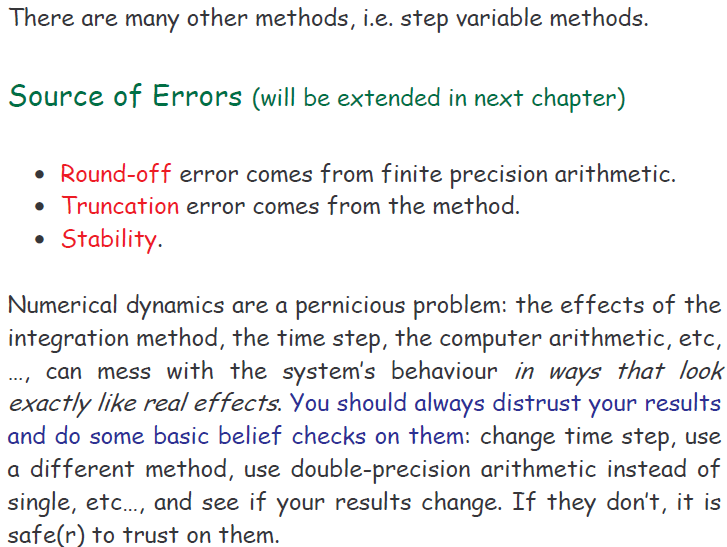


并且不用再做特殊的初始化，利用直接外推得到二阶Runge-Kutta法：



**四．四阶Runge-Kutta法**





**五．N阶LODEs**

