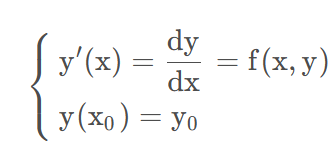
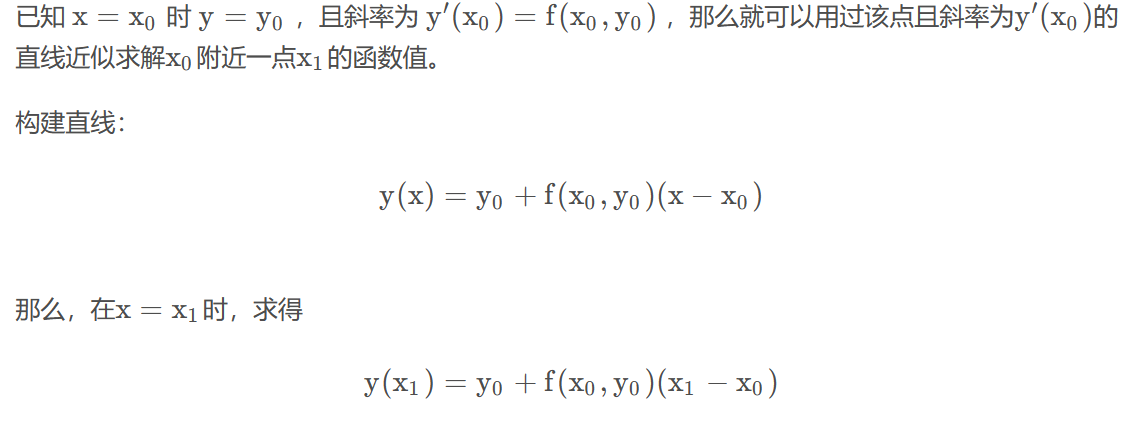
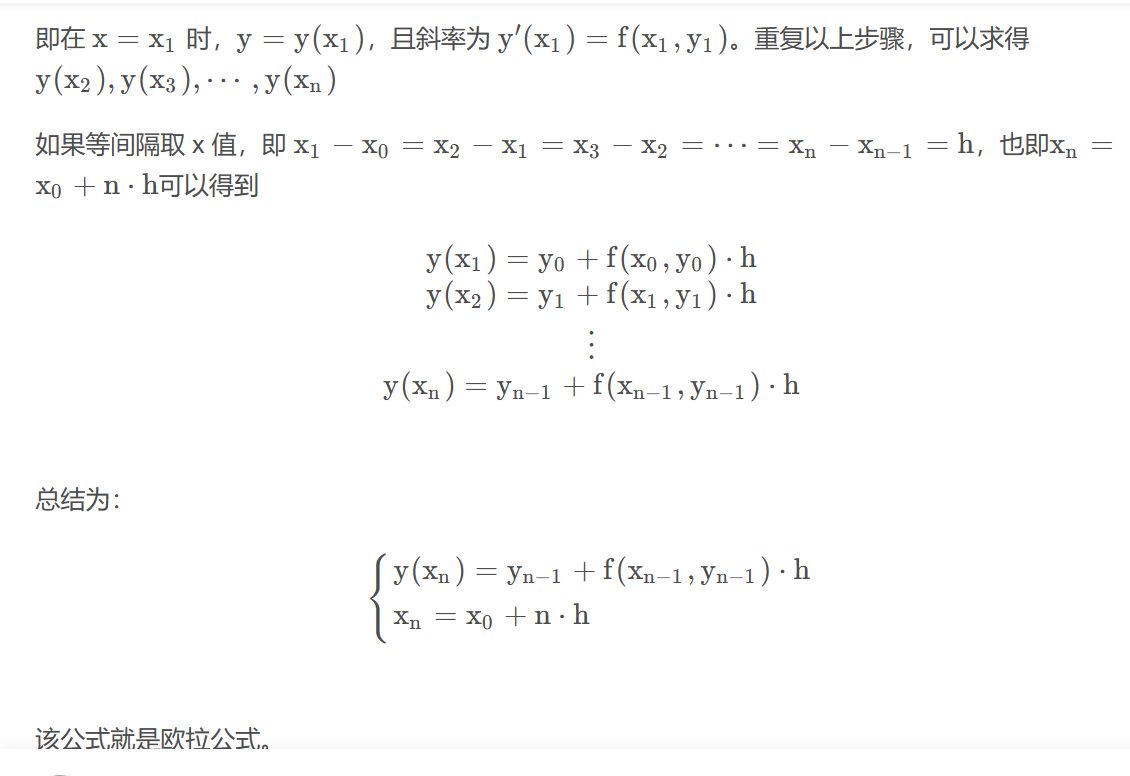
# 欧拉法

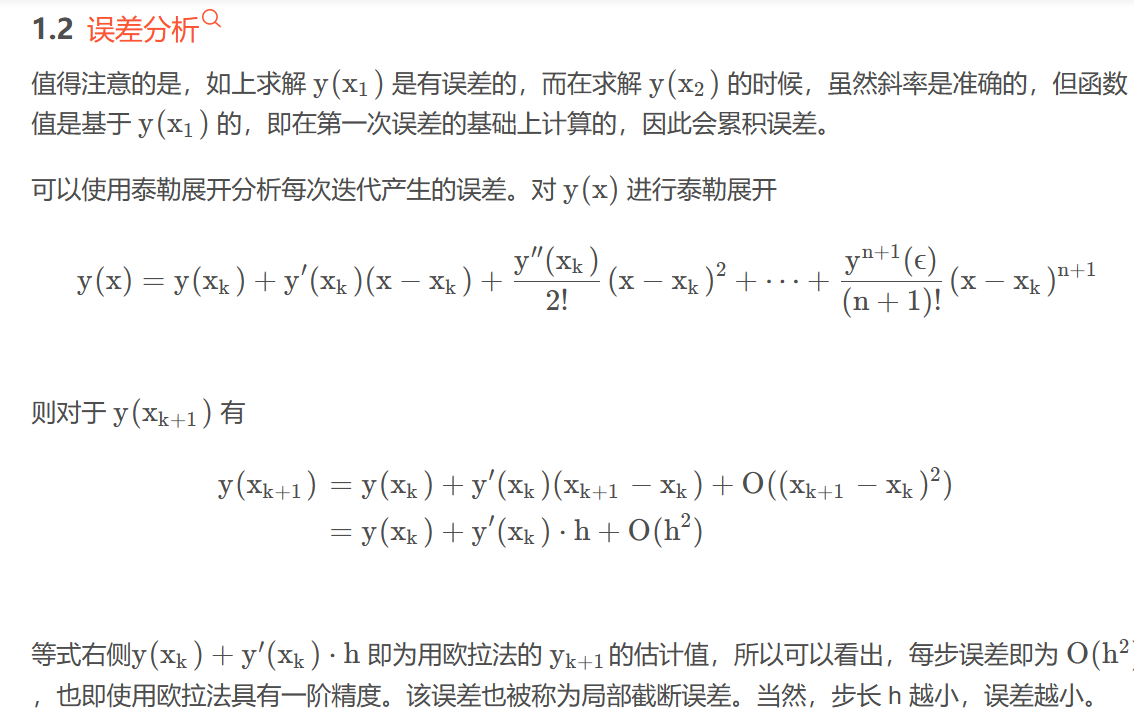
已知某点的坐标，以及该点的斜率值,即



欧拉法是对该方程组是进行迭代求解，

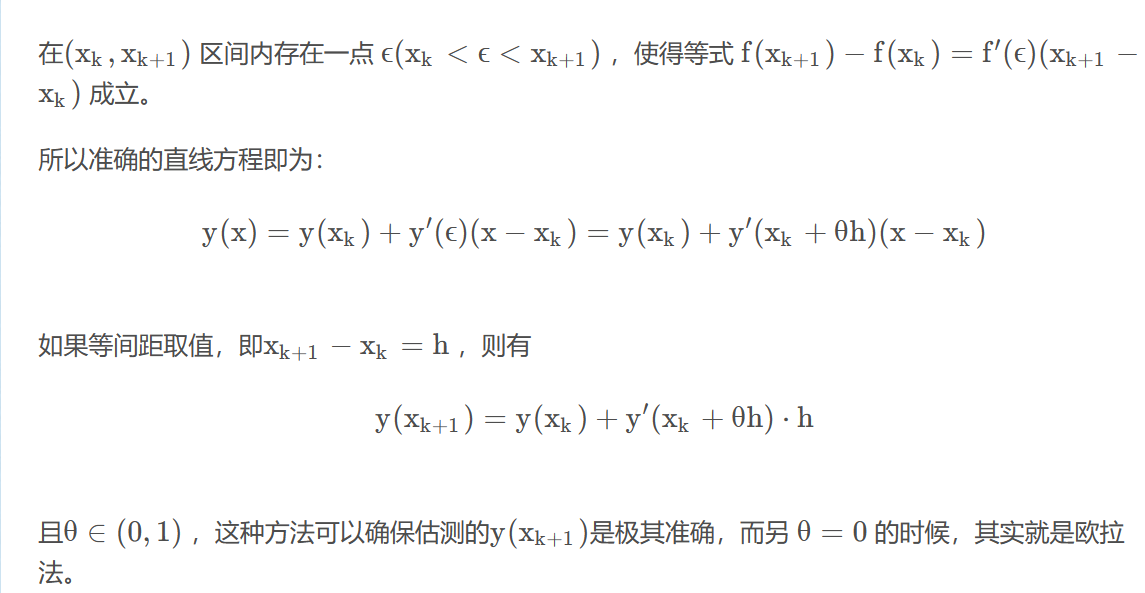




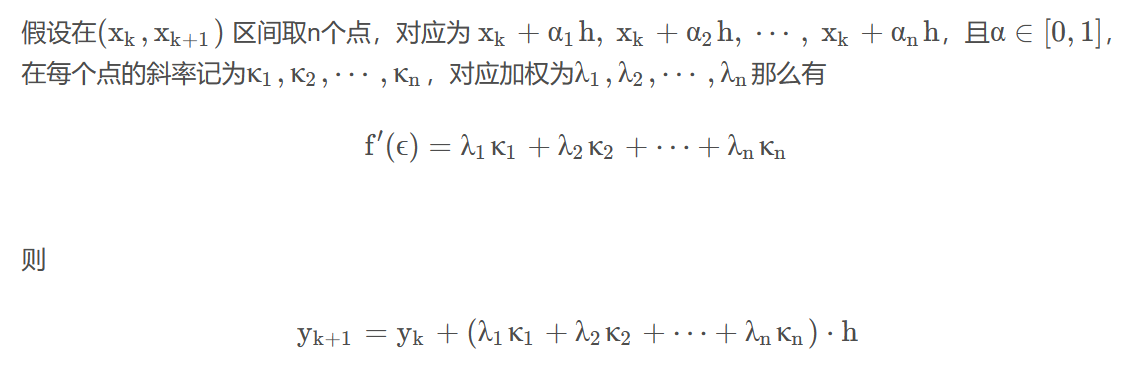


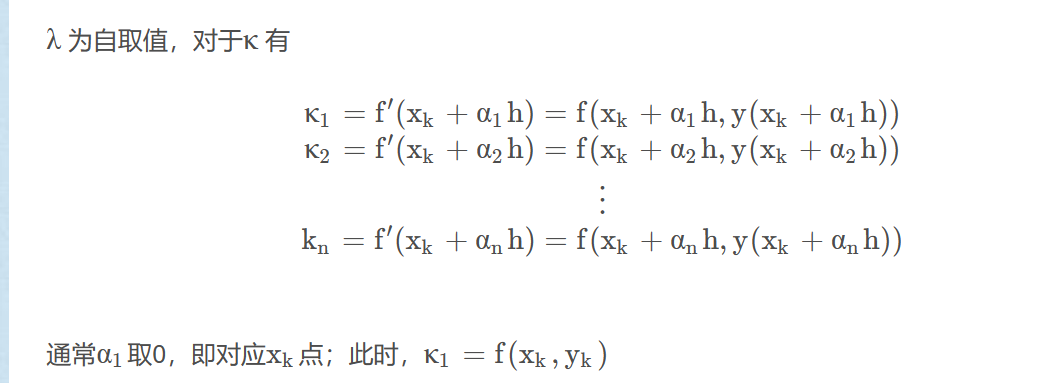
# 龙格库塔算法

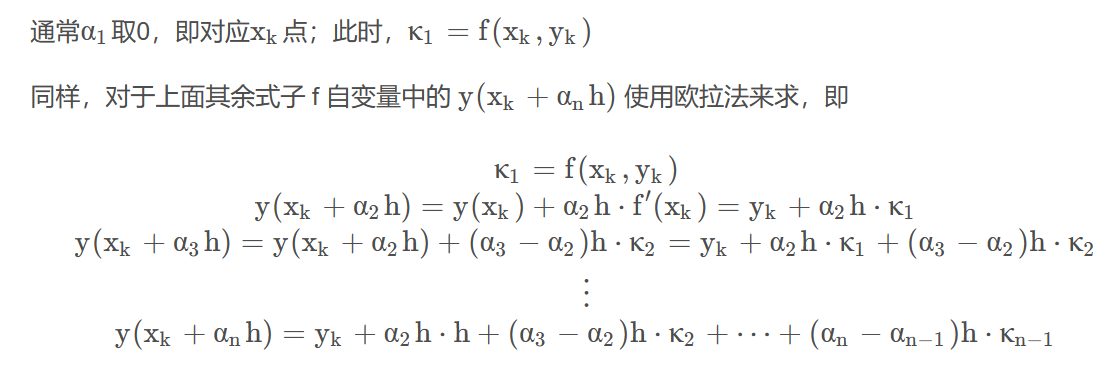
利用拉格朗日中值定理作为斜率k解决欧拉法直接使用导数作为斜率k的误差，

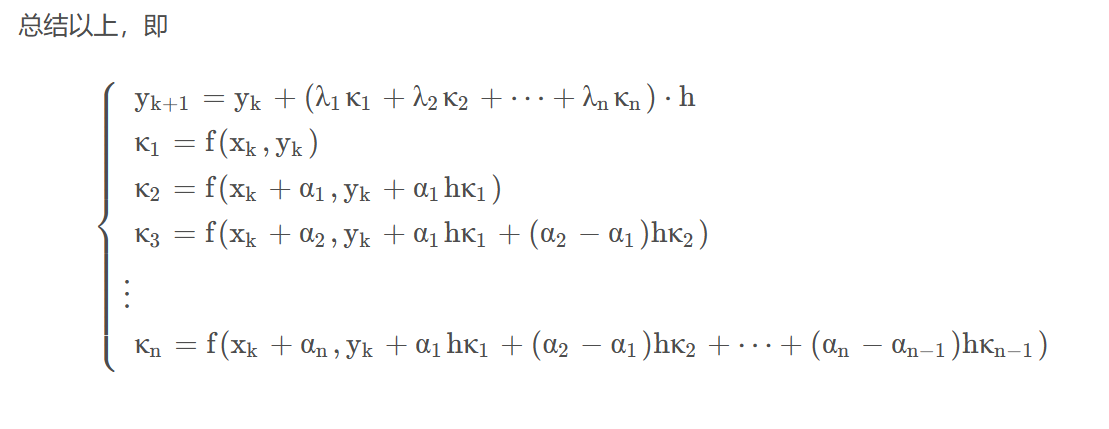


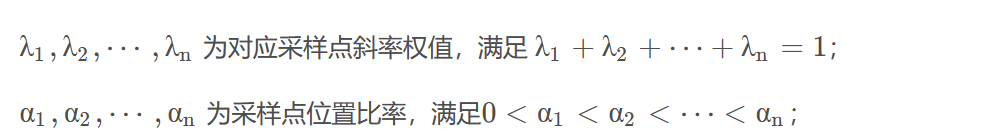
高阶算法是用更多点加权来估算f ′ ( ϵ ) f 的值



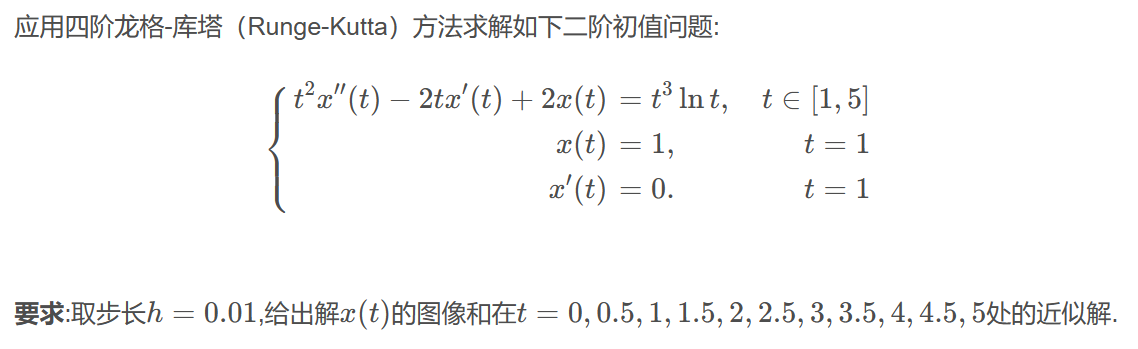




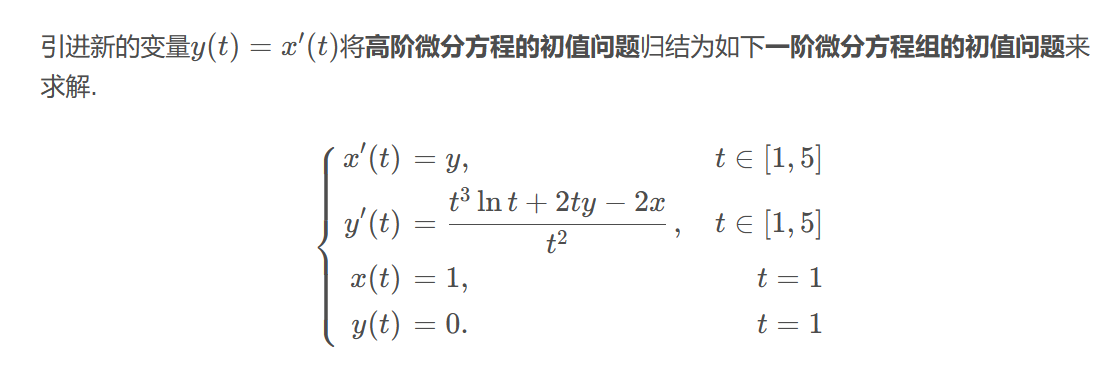


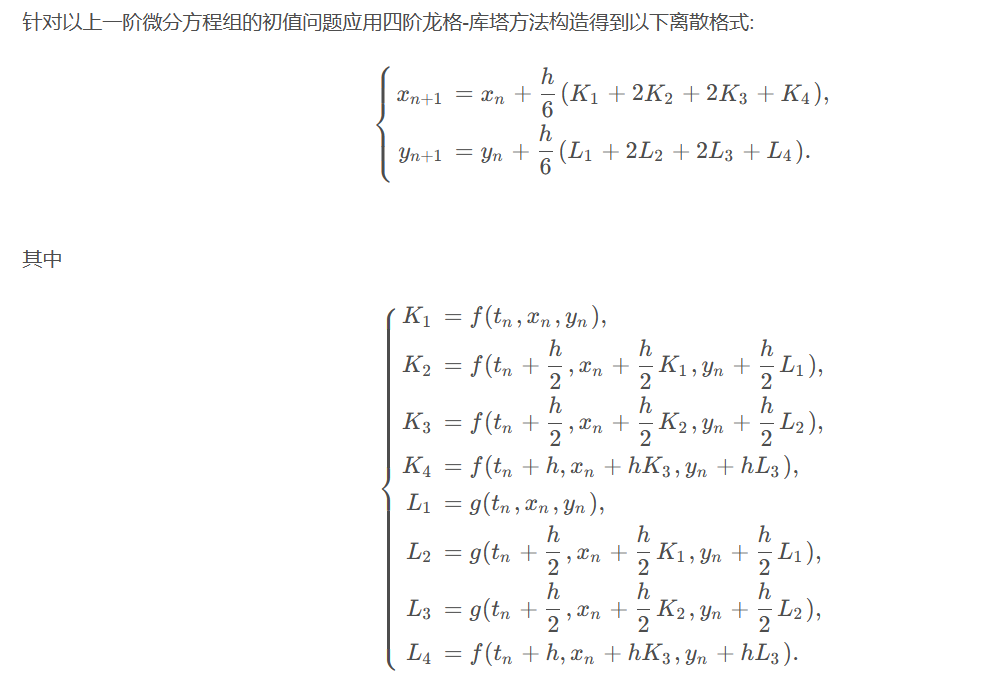


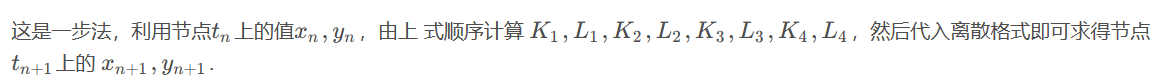
下面引入样例：



Step1：将原方程组化为一阶微分方程组：



Step2：



代码样例如下：

***'''***

***Author: gggaaallleee 1293587368@qq.com***

***Date: 2023-04-11 10:25:28***

***LastEditors: gggaaallleee 1293587368@qq.com***

***LastEditTime: 2023-04-12 08:13:14***

***FilePath: \数据库\1.py***

***Description: 这是默认设置,请设置`customMade`, 打开koroFileHeader查看配置 进行设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%85%8D%E7%BD%AE***

***'''***

***# ----------------------------------------------------------------------------------------------------------***

import math

import matplotlib.pyplot as plt

def f(t, x, y):

    a = y

    return a

def g(t, x, y):

    a = (t \*\* 3 \* math.log(t) + 2 \* t \* y - 2 \* x) / t \*\* 2

    return a

def RK4(t, x, y, h):

***"""***

***:param t: t 的初始值***

***:param x: x 的初始值***

***:param y: y 的初始值***

***:param h: 时间步长***

***:return: 迭代新解***

***"""***

    tarray, xarray, yarray = [], [], []

    while t <= 5:

        tarray.append(t)

        xarray.append(x)

        yarray.append(y)

        t += h

        K\_1 = f(t, x, y)

        L\_1 = g(t, x, y)

        K\_2 = f(t + h / 2, x + h / 2 \* K\_1, y + h / 2 \* L\_1)

        L\_2 = g(t + h / 2, x + h / 2 \* K\_1, y + h / 2 \* L\_1)

        K\_3 = f(t + h / 2, x + h / 2 \* K\_2, y + h / 2 \* L\_2)

        L\_3 = g(t + h / 2, x + h / 2 \* K\_2, y + h / 2 \* L\_2)

        K\_4 = f(t + h, x + h \* K\_3, y + h \* L\_3)

        L\_4 = g(t + h, x + h \* K\_3, y + h \* L\_3)

        x = x + (K\_1 + 2 \* K\_2 + 2 \* K\_3 + K\_4) \* h / 6

        y = y + (L\_1 + 2 \* L\_2 + 2 \* L\_3 + L\_4) \* h / 6

    return tarray, xarray, yarray

def main():

    tarray, xarray, yarray = RK4(1, 1, 0, 0.01)

    print(***"***龙格-库塔 数值结果***"***.center(168))

    print(***'***-***'*** \* 178)

    print(***"***对象\\时刻\t***"***, ***"***t=0\t\t***"***, ***"***  t=0.5\t\t***"***, ***"***  t=1\t\t\t***"***, ***"***  t=1.5\t\t***"***, ***"***  t=2\t\t\t***"***, ***"*** t=2.5\t\t\t***"***,

***"***  t=3\t\t\t***"***, ***"***  t=3.5\t\t***"***, ***"***  t=4\t\t\t***"***, ***"*** t=4.5\t\t\t***"***, ***"***  t=5***"***)

    print(***'***-***'*** \* 178)

    print(***"***x:***"***, end=***''***)

    for i in range(len(xarray)):

        if i % 40 == 0:

            print(***"***\t\t***"***, ***"***%8.4f***"*** % xarray[i], end=***''***)

    print(***'***\n***'***, ***'***-***'*** \* 177)

    print(***"***y:***"***, end=***''***)

    for i in range(len(yarray)):

        if i % 40 == 0:

            print(***"***\t\t***"***, ***"***%8.4f***"*** % yarray[i], end=***''***)

    print(***'***\n***'***, ***'***-***'*** \* 177)

    plt.figure(***'***龙格-库塔 数值结果***'***)

    plt.subplot(221)

***# plt.plot(tarray, xarray, label='x\_runge\_kutta')***

    plt.scatter(tarray, xarray, label=***'***x\_scatter***'***, s=1, c=***'***#DC143C***'***, alpha=0.6)

***# plt.xlabel('t')***

    plt.legend()

    plt.subplot(222)

***# plt.plot(tarray, yarray, label='y\_runge\_kutta')***

    plt.scatter(tarray, yarray, label=***'***y\_scatter***'***, s=1, c=***'***#DC143C***'***, alpha=0.6)

***# plt.xlabel('t')***

    plt.legend()

    plt.subplot(212)

***# plt.plot(tarray, xarray, label='runge\_kutta')***

    plt.scatter(tarray, xarray, label=***'***Numerical solution scatter***'***, s=1, c=***'***#DC143C***'***, alpha=0.6)

    plt.xlabel(***'***t***'***)

    plt.legend()

    plt.show()

if \_\_name\_\_ == ***"***\_\_main\_\_***"***:

    main()

matlab：ode45函数 4阶五级Runge-Kutta算法。

# 求积分

Trapezoidal Rule（梯形法则，），把积分分成一小块梯形面积相加

