《数值分析》上机期末考试报告

学号：221840189，姓名：王晨光

§1 问题

应用 Runge-Kutta-Fehlberg 算法解初值问题：

$$\begin{aligned} y' &= -y + t^2 + 2, \quad 0 \le t \le 1, \\ y(0) &= 1 \end{aligned}$$

取 $TOL = 10^{-4}$, $h_{\max} = 0.2$, $h_{\min} = 0.0001$.

分别用表和图给出计算结果.

§2 算法思路

在微分方程数值解法中，步长 h 的选择是很重要的. 但是, 要做到合理选择也是比较困难的，因为它与问题本身和所选用的数值方法都有关系. 前面介绍的 Runge-Kutta 方法，实际上，往往并不直接应用它们. 一般地，使用者要求数值解与精确解之差不超过某一误差容限. 由这个容限来确定数值方法中应取多大的步长. 为了选取比较合适的步长，一方面必须对局部 (每一步) 误差有一个估计; 另一方面在整个区间不取固定的步长 h，在区间的某些段上取相对小的 h，而在另一些段上则取相当大的 h.

以三阶 Runge-Kutta 方法为例，记

$$\tau_{n+1} = (y(t_{n+1}) - y_{n+1})/h$$

则存在常数 K 使得

$$\tau_{n+1} \backsimeq Kh^2$$

可以写成

$$Kh^2 \backsimeq \frac{1}{h}(u_{n+1} - y_{n+1})$$

由给定的误差容限度 TOL，我们可以选取 q 使得：

$$q \leqslant \left[\frac{TOL}{\frac{|u_{n+1}-y_{n+1}|}{h}}\right]^{\frac{1}{2}}$$

假定 $u_n = y_n$ 可以得到：

$$\frac{u_{n+1} - y_{n+1}}{h} = \frac{2K_1 + 4K_3 - 6K_2}{9}$$

若

$$\frac{|u_{n+1} - y_{n+1}|}{h} \leqslant TOL$$

则取 $q = 1$，且下一步仍然使用 h；若：

$$\frac{|u_{n+1} - y_{n+1}|}{h} > TOL$$

则将步长缩小，重新计算为了确保步长缩小不至无限循环下去，我们给出一个步长的下限，如 $h_{\min}$. 若 $h < h_{\min}$，则终止计算；若 $\dfrac{|u_{n+1} - y_{n+1}|}{h}$ 比 TOL 小到一定程度，则增大步长.

Fehlberg 于 1970 年提出用五阶 Runge-Kutta 方法：

$$u_{n+1}=u_n+\frac{16}{135}K_1+\frac{6656}{12825}K_3+\frac{28561}{56430}K_4-\frac{9}{50}K_5+\frac{2}{55}K_6$$

去估计四阶 Runge-Kutta 方法：

$$y_{n+1}=y_n+\frac{25}{216}+\frac{1408}{2565}K_3+\frac{2197}{4104}K_4-\frac{1}{5}K_5$$

其中：

$$\begin{cases} K_1=hf\left(t_n,y_n\right)\\ K_2=hf\left(t_n+\frac{h}{4},y_n+\frac{1}{4}K_1\right)\\ K_3=hf\left(t_n+\frac{3}{8}h,y_n+\frac{3}{32}K_1+\frac{9}{32}K_2\right)\\ K_4=hf\left(t_n+\frac{12}{12}h,y_n+\frac{1932}{2197}K_1-\frac{7200}{2197}K_2+\frac{7296}{2197}K_3\right)\\ K_5=hf\left(t_n+h,y_n+\frac{439}{216}K_1-8K_2+\frac{3680}{513}K_3-\frac{845}{4104}K_4\right)\\ K_6=hf\left(t_n+\frac{h}{2},y_n-\frac{8}{27}K_1+2K_2-\frac{3544}{2565}K_3+\frac{1859}{4104}K_4-\frac{11}{40}K_5\right)\end{cases}$$

我们称它为 **Runge-Kutta-Fehlberg** 方法，简称 **R-K-F** 方法. 显然，这个方法的优点是每一步只要计算 6 个函数值. 而其他四阶和五阶 Runge-Kutta 方法一起使用时，每一步将要求计算 10 个函数值.

由于两式的局部离散误差分别为 $O(h^6)$ 和 $O(h^5)$，因此我们有：

$$\tau_{n+1}\backsimeq Kh^4$$

和

$$Kh^4\backsimeq\frac{1}{h}(u_{n+1}-y_{n+1})$$

从而得到了：

$$q\leqslant\left[\frac{TOL}{\frac{|u_{n+1}-y_{n+1}|}{h}}\right]^{\frac{1}{4}}$$

实际使用时，通常取

$$q=\left[\frac{TOL}{\frac{2|u_{n+1}-y_{n+1}|}{h}}\right]^{\frac{1}{4}}=0.84\left[\frac{TOL}{\frac{|u_{n+1}-y_{n+1}|}{h}}\right]^{\frac{1}{4}}$$

现在进一步修改得到：

$$\frac{|u_{n+1}-y_{n+1}|}{h}=\left|\frac{1}{360}K_1-\frac{128}{4275}K_3-\frac{2197}{75240}K_4+\frac{1}{50}K_5+\frac{2}{55}K_6\right|/h$$

此后继续沿用上述的自适应法则.

Algorithm 1 Runge-Kutta-Fehlberg 方法

Require: 区间 $[a, b]$; 初值 η; 积分函数 f; 最大步长 h_{max}; 最小步长 h_{min}; 误差容限 TOL.

Ensure: t, y, 步长 h

function RUNGE-KUTTA-FEHLBERG$(a, b, \eta, f, h_{max}, h_{min}, TOL)$

 $t \leftarrow a$

 $y \leftarrow \eta$

 $h \leftarrow h_{max}$

 for $t < b$ **do**

  $K_1 \leftarrow hf\left(t, y\right)$

  $K_2 \leftarrow hf\left(t + \frac{h}{4}, y + \frac{1}{4}K_1\right)$

  $K_3 \leftarrow hf\left(t + \frac{3}{8}h, y + \frac{3}{32}K_1 + \frac{9}{32}K_2\right)$

  $K_4 \leftarrow hf\left(t + \frac{12}{12}h, y + \frac{1932}{2197}K_1 - \frac{7200}{2197}K_2 + \frac{7296}{2197}K_3\right)$

  $K_5 \leftarrow hf\left(t + h, y + \frac{439}{216}K_1 - 8K_2 + \frac{3680}{513}K_3 - \frac{845}{4104}K_4\right)$

  $K_6 \leftarrow hf\left(t + \frac{h}{2}, y - \frac{8}{27}K_1 + 2K_2 - \frac{3544}{2565}K_3 + \frac{1859}{4104}K_4 - \frac{11}{40}K_5\right)$

  $R \leftarrow \left|\frac{1}{360}K_1 - \frac{128}{4275}K_3 - \frac{2197}{75240}K_4 + \frac{1}{50}K_5 + \frac{2}{55}K_6\right| / h$

  $\delta \leftarrow 0.84(TOL/R)^{\frac{1}{4}}$

  if $R \leqslant TOL$ **then**

   $t \leftarrow t + h$

   $y \leftarrow y + \frac{25}{216} + \frac{1408}{2565}K_3 + \frac{2197}{4104}K_4 - \frac{1}{5}K_5$

   输出 t, y, h

  end if

  if $\delta \leqslant 0.1$ **then**

   $h \leftarrow 0.1h$

  else

   if $\delta \geqslant 4$ **then**

    $h \leftarrow 4h$

   else

    $h \leftarrow \delta h$

   end if

  end if

  if $h \geqslant h_{max}$ **then**

   $h \leftarrow h_{max}$

  end if

  if $h < h_{min}$ **then**

   break

  end if

 end for

end function

§3 结果分析

我们将使用 python 中 scipy 库中的 integrate.solve_ivp 函数求解得到的结果作为该初值问题的精确解，并与通过 **R-K-F** 方法得到的结果进行比较.

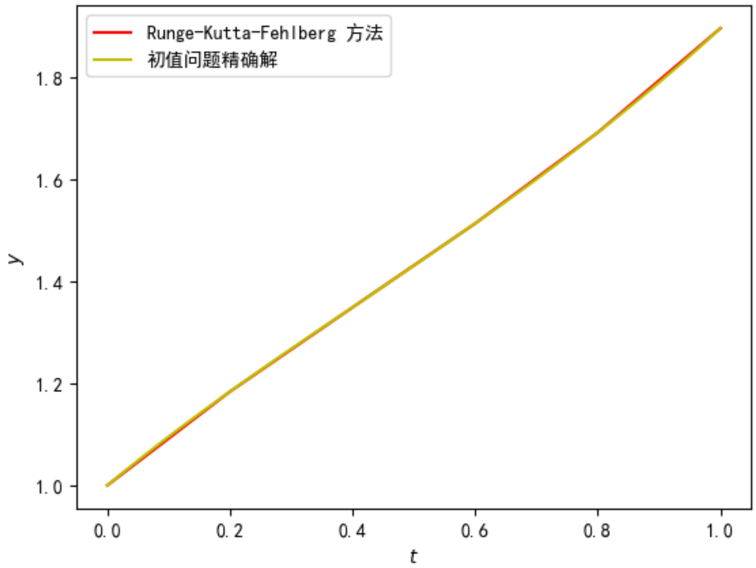

图 1: 使用 **R-K-F** 方法得到的结果与精确结果的比较图

t	y	Runge-Kutta-Fehlberg
0	1.0	1
0.2	1.1838953507690455	1.1838083076923076
0.4	1.3493627038060534	1.3490406228872582
0.6000000000000001	1.5137912922810752	1.5135657904689523
0.8	1.6919546132170773	1.6920135743911044
1.0	1.896184383974114	1.896361805046761

表 1: 使用 **R-K-F** 方法得到的结果与精确结果的比较表

从图、表中可以看出，**R-K-F** 方法用于求解该微分方程初值问题有非常精确的效果.

§4 结论

Runge-Kutta-Fehlberg 方法巧妙地将“自适应”的思想引入到微分方程数值解的经典方法五阶 **Runge-Kutta** 方法中，有着很不俗的表现. 可以被我们广泛地用于求微分方程的精确数值解.

§5 附录: 程序代码

```python
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 初始条件和步长
x0, y0 = 0, 1
x1 = 1
h_min = 0.0001
h_max = 0.2
tol = 1e-4
# 定义常微分方程
def f(x, y):
    return -y + x**2 + 2

def runge_kutta_4(f, x0, x1, y0, tol, hmax, hmin):
    y1=[]
    x=[]
    t = x0
    y = y0
    y1.append(y0)
    x.append(x0)
    h = hmax
    while t < x1:
        k1=h*f(t,y)
        k2=h*f(t+0.25*h,y+0.25*k1)
        k3=h*f(t+(3/8)*h,y+(3/32)*k1+(9/32)*k2)
        k4=h*f(t+(12/13)*h,y+(1932/2197)*k1-(7200/2197)*k2+(7296/2197)*k3)
        k5=h*f(t+h,y+(439/216)*k1-8*k2+(3680/513)*k3-(845/4104)*k4)
        k6=h*f(t+(1/2)*h,y-(8/27)*k1+2*k2-(3544/2565)*k3+(1859/4104)*k4-(11/40)*k5
        R=np.abs((1/360)*k1-(128/4275)*k3-(2197/75240)*k4+(1/50)*k5+(2/55)*k6)
        delta=0.84*((tol/R)**(1/4))
        if R<=tol:
            t=t+h
            y=y+(25/216)*k1+(1408/2565)*k3+(2197/4104)*k4-(1/5)*k5
            x.append(t)
            y1.append(y)
        if delta<=0.1:
            h=0.1*h
        else:
            if delta>=4:
                h=4*h
            else:
                h=delta*h
```

```
        if h>hmax:
            h=hmax
        if h<hmin:
            print('stop')
            break
    return x, y1
x_rk4, y_rk4 = runge_kutta_4(f, x0, x1, y0, tol, h_max, h_min)
for i in range(len(y_rk4)):
    print(y_rk4[i])
    # print(sol.sol(x_rk4)[0][i])
    # print(x_rk4[i])
from scipy.integrate import solve_ivp
x_span=(0, 1)
y_0=np.array([1])
sol=solve_ivp(f, x_span, y_0, method='RK45', dense_output=True)
x_values = np.linspace(x_span[0], x_span[1], 1000)
y_values = sol.sol(x_values)
print("Solution: ", sol.sol(x_rk4)[0])
print("Runge-Kutta 4 Method:", y_rk4)
plt.plot(x_rk4, y_rk4, label='Runge-Kutta-Fehlberg 方法', color='r')
plt.plot(x_values, y_values[0], label='初值问题精确解', color='y')
plt.xlabel('$t$')
plt.ylabel('$y$')
plt.legend()
plt.show()
```