《数值分析》实验考核报告

211240021 田铭扬

§1 问题

应用经典的四阶 Runge-Kutta 方法求解如下问题：

$$\begin{cases} y''+\dfrac{2}{x}y'-\dfrac{6}{x^2}y=5-6x+7x^2, & 1<x<2 \\ y(1)=\dfrac{1}{2}, \quad y(2)=4+4\ln 2 \end{cases}$$

该问题的真解为 $y=x^2-x^3+\frac{1}{2}x^4+x^2\ln x$. 分别取步长 h=1/200,1/100,1/50,1/25,1/10,1/5,1/4 进行计算，并根据结果给出误差收敛阶，对计算结果进行分析。

§2 算法实现

由于题目中已给出真解，故可以根据真解求出 y'(1) = 2，计算相应的二阶常微分方程初值问题，并将计算得到的 y(2) 近似值与真值 4+ln4 进行比较。进一步，若要解此二阶 ODE，我们可以将其转化为如下的二元初值问题：

$$\begin{cases} y_1'=y_2 \\ y_2'=\dfrac{6}{x^2}y_1-\dfrac{2}{x}y_2+5-x(6-7x) \end{cases} \quad y_1(1)=\frac{1}{2},\ \ y_2(1)=2$$

只要利用数值方法得到 $y_1(2)$的值即可。

在算法的实现上，选择使用 C++语言进行。C++中有 pair 类型变量（定义在 utility 头文件中），正好能满足我们的需要：只需将经典四阶 Runge-Kutta 实现中的相关变量（y, k1, k2, k3, k4）均换为 pair<double,double>类型，并相应构造的数乘、加法运算即可。

此外，题目还要求给出误差收敛阶的后验估计。设此方法局部收敛阶为 q，且在步长为 1/N 时，y(2) 近似值为 y_N，误差为 ε_N，则有：

$$\varepsilon_N=|y_N-y|=C(\frac{1}{N^q})+O(\frac{1}{N^q})\approx C(\frac{1}{N^q})\Rightarrow \ln(\varepsilon_N)\approx q\ln(\frac{1}{N})+\ln C$$

其中 C 只与微分方程本身有关。进而，对于不同的步长 $1/N_1$ 与 $1/N_2$，我们有：

$$q\approx\frac{\ln\varepsilon_{N2}-\ln\varepsilon_{N1}}{\ln(1/N_2)-\ln(1/N_1)}=\frac{\ln(\varepsilon_{N2}/\varepsilon_{N1})}{\ln(N_1/N_2)}$$

对于题目中给出的 7 种步长，观察由每相邻两种计算出的 q 值，即可给出对收敛阶的估计。

§3 结果、分析与结论

步长	1/4	1/5	1/10	1/25	1/50	1/100	1/200
绝对误差	-1.12×10^{-3}	-4.89×10^{-4}	-3.48×10^{-5}	-9.64×10^{-7}	-6.20×10^{-8}	-3.92×10^{-9}	-2.47×10^{-10}
相对误差	1.66×10^{-4}	7.22×10^{-5}	5.13×10^{-6}	1.42×10^{-7}	9.14×10^{-9}	5.79×10^{-10}	3.64×10^{-11}
收敛阶		3.72	3.81	3.91	3.96	3.98	3.99

表 1 运行结果

运行结果如上表。从表格中我们可以看到，经典的四阶 Runge-Kuta 方法在解决上述问题的表现较好。当步长较小时，能够表现出 4 阶收敛性，这与理论是一致的。

上述方法是建立在题目已给出真解的基础上的。若是题目没给出真解，对于这一常微分方程边值问题，我们应当考虑“打靶”法：即假设初值 y'(1)，使用“某种”数值方法计算得

到 y(2)的值，然后通过和题目中边值 4+4ln2 的比较，改变 y'(1)的值重新计算，如此迭代计算，直到得到原边值问题的数值解。这是教材后面的内容。

而本次实验的结果表明们，经典的四阶 Runge-Kuta 方法可以很好地解决二元常微分方程组初值问题，因而能够胜任上一段提到的“某种”方法，用来解决二阶边值问题。

附录一 完整代码

```
#include <cstdio>
#include <cmath>
#include <utility>
#define M(a,b) make_pair(a,b)
#define PDD pair<double,double>
using namespace std;
PDD func(double _x,PDD _y){
   double _y2=6*_y.first/_x/_x-2*_y.second/_x+5-_x*(6-7*_x);
   return M(_y.second,_y2);
}
PDD sum(PDD _a, PDD _b){
   return M(_a.first+_b.first,_a.second+_b.second);
}
PDD mul(double _a, PDD _b){
   return M(_a*_b.first,_a*_b.second);
}
PDD integ_R(int _N,double _s,double _e,PDD _0){
   PDD _y=_0,_k1,_k2,_k3,_k4;
   double _h=(_e-_s)/_N,_t=_s;
   double _h2=_h/2,_h6=_h/6;
   for(int i=0;i<_N;i++){
     _k1=func(_t,_y);
     _k2=func(_t+_h2,sum(_y,mul(_h2,_k1)));
     _k3=func(_t+_h2,sum(_y,mul(_h2,_k2)));
     _t+=_h;
     _k4=func(_t,sum(_y,mul(_h,_k3)));
     _y=sum(mul(_h6,(sum(sum(_k1,mul(2,_k2)),sum(mul(2,_k3),_k4)))),_y);
   }
   return _y;
}
int main() {
   double Real=4+4*log(2),_r,e[7];
   int N[7]={4,5,10,25,50,100,200};
   for(int i=0;i<7;i++){
     _r=integ_R(N[i],1,2,M(0.5,2)).first;
     e[i]=_r-Real;
     printf("Stepsize=1/%d\n",N[i]);
     printf("Result:%.15f\n",_r);
     printf("AError:%.15f\n",e[i]);
     printf("RError:%.15f\n\n",e[i]/Real);
   }
   printf("Order:\n");
   for(int i=0;i<6;i++)
     printf("%.2f ",log(e[i+1]/e[i])/log(double(N[i])/N[i+1]));
   getchar(); getchar();
   return 0;
}
```