数值分析第5次作业

211240021 田铭扬

§1 问题

常微分方程初值问题：

$$\begin{cases} y'=-\dfrac{1}{x^2}-\dfrac{y}{x}-y^2, \quad 1\leqslant x\leqslant 2 \\ y(1)=-1 \end{cases}$$

1.分别用Euler方法、改进的Euler方法、Heun公式、中点方法和四阶Runge-Kutta方法求解上述初值问题，列表、画图比较他们的计算结果。

2.用经典的Runge-Kutta方法提供初始出发值，与四阶Adams预测-矫正方法的PECE模式结合起来求解上述初值问题，并于四阶Runge-Kutta方法进行比较。

§2 问题分析与实现

由于各数值积分方法均在课上进行过详细的推导，可以按部就班地编程实现，故此处不再过多分析。

值得注意的是，各种求解方法的“结构”差异并不大，因而可以考虑使用同一个程序，通过向求解的函数传递参数来选择要使用的方法（用C++中的switch语句实现），这样全部数据可以在一次运行中得到，节省不少时间。

由于本题还要求了图像绘制，因此还用JavaScript为每种方法制作了计算+绘制的程序。出于排版考虑，图像会放在附录中。

此外，关于每种方法的误差，我选择通过将比较数值计算得到的y(2)值与真解进行比较来反映。对于每一步的误差情况，则可以通过图像比较直观地呈现。（PS.利用Mathmatica可以求得真解为 $y=-\frac{1}{x}\tan(\ln x+\frac{\pi}{4})$ ）

§3 运行结果与分析

以下是第1小题的运行结果。由于程序的运行时间都很短，使用chrono得到的时长都是“0”（小于0.5μs），故不在表中列出。

步长		1/250	1/500	1/1000	1/2000	1/4000	1/8000	1/16000
绝对误差	Euler	1.63×10^{-1}	8.40×10^{-2}	4.26×10^{-2}	2.15×10^{-3}	1.08×10^{-3}	5.40×10^{-4}	2.70×10^{-4}
	改进Euler	7.58×10^{-4}	1.91×10^{-4}	4.78×10^{-5}	1.20×10^{-5}	3.00×10^{-6}	7.50×10^{-7}	1.87×10^{-7}
	Heun	1.15×10^{-3}	2.82×10^{-4}	7.07×10^{-5}	1.77×10^{-5}	4.43×10^{-6}	1.11×10^{-6}	2.78×10^{-7}
	中点	1.30×10^{-3}	3.27×10^{-4}	8.22×10^{-5}	2.06×10^{-5}	5.16×10^{-6}	1.29×10^{-6}	3.22×10^{-7}
	4阶R-K	1.17×10^{-8}	7.32×10^{-10}	4.58×10^{-11}	3.00×10^{-12}	4.09×10^{-14}	3.75×10^{-13}	4.71×10^{-13}

表1 Euler方法、改进的Euler方法、Heun公式、中点方法和四阶Runge-Kutta方法比较

从表中可以看到，在五种方法中，经典的四阶Runge-Kutta方法表现最好，而最“朴素”的Euler法也是表现最差的。其余三种方法的表现介于它们之间且相差不大，改进的Euler方法表现略好于另外两个。

这与理论是符合的：Euler法是1阶方法，4阶Runge-Kutta方法——顾名思义——是4阶方法，而其余三种均为2阶方法。事实上，我特意设定了每次的步长均为上次的1/2，进而可以根据后验的阶数估计公式

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

$$\Rightarrow \quad q\approx\frac{\ln\varepsilon_{N2}-\ln\varepsilon_{N1}}{\ln(h_2)-\ln(h_1)}=\frac{\ln(\varepsilon_{N2}/\varepsilon_{N1})}{\ln(N_1/N_2)}$$

通过运行数据来“目视”估计阶数。即若阶数为q，则相邻两次的误差差距大约是$1/2^q$。如此，所有方法的实际阶数均与理论相符。不过特别注意到，四阶Runge-Kutta方法只有前2项符合此估计——这是因为此方法收敛速度快，很快就到达的double类型精度导致的“阈值”，使舍入误差占了主导，甚至出现了步长缩小、误差却增大的现象。

以下是第2小题的运行结果。在题目的基础上，我还加入了不使用预估-矫正方法的运行结果，以进一步地比较。

步长		1/250	1/500	1/1000	1/2000	1/4000
绝对误差	4阶Adams	1.14×10^{-6}	7.65×10^{-8}	4.95×10^{-8}	3.15×10^{-9}	1.98×10^{-10}
	4阶Adams PECE	7.84×10^{-7}	5.52×10^{-8}	3.66×10^{-9}	2.35×10^{-10}	1.51×10^{-11}
	4阶Runge-Kutta	1.17×10^{-8}	7.32×10^{-10}	4.58×10^{-11}	3.00×10^{-12}	4.09×10^{-14}

表2 四阶Adams方法、四阶Adams PECE方法和四阶Runge-Kutta方法比较

可以看到，使用预估矫正模式的Adams方法，相对于不使用，在误差的表现上有一定（常数级别）的提升，这也和理论相符。但是与同阶数的Runge-Kutta方法相比，它的表现并不好。那么Adams方法的优势在哪里呢？通过观察代码（见附录）发现，Adams-PECE方法在每一步需要两次调用求f(t,y)值的函数，而R-K方法需要4次。因而我推测，当所求常微分方程的表达式比较复杂时（比如涉及到大量三角函数、对数运算等），Adams会体现出时间的优势。

此外，本表中的Adams方法和Adams-PECE方法的数据并没有表现出“目视”的4阶收敛，这是由于上文所述的“阈值”导致的。事实上，若取步长为1/500k (k=1,2...)，编程用上文中的公式进行后验阶数估计，可以看到在步长较大时，两种方法确实是4阶的。程序见附录，数据从略。

附录一 图像

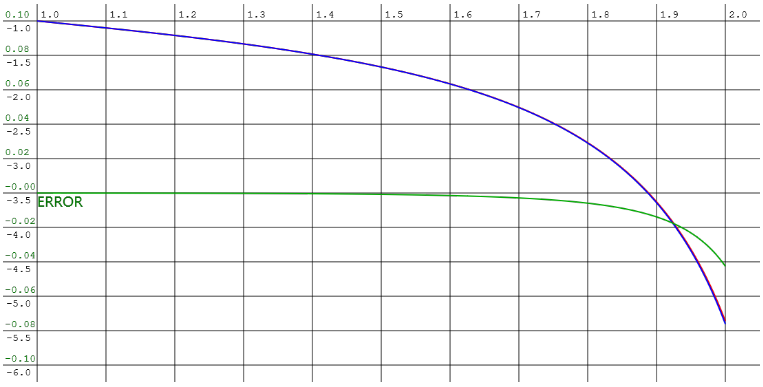

图 1 Euler 方法（步长 1/1000，下同）

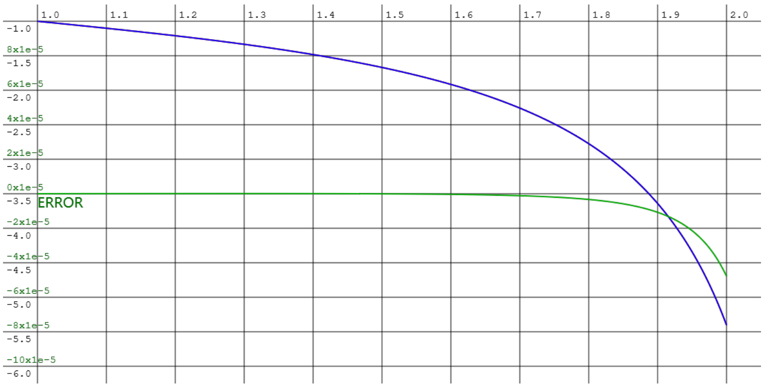

图 2 改进的 Euler 方法

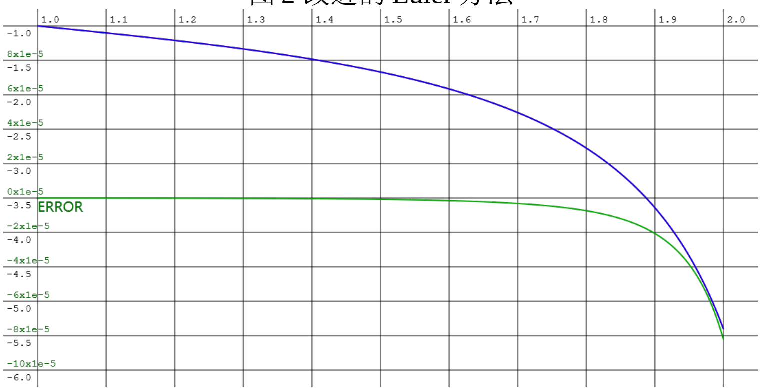

图 3 中点方法

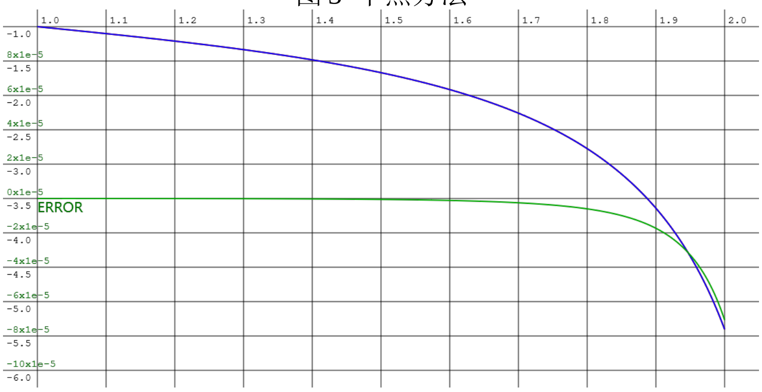

图 4 Heun 方法

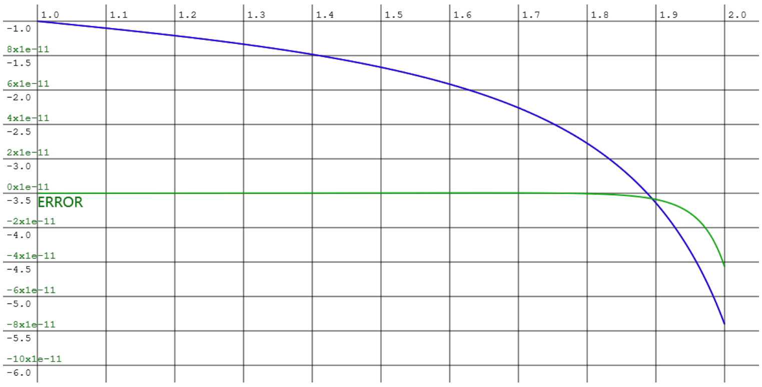

图 5 经典的四阶 Runge-Kutta 方法

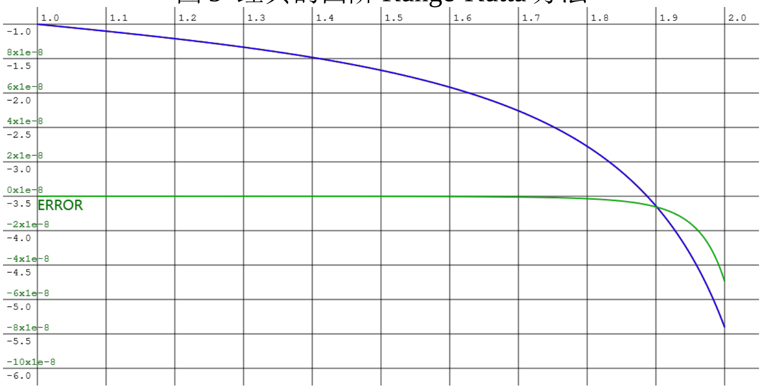

图 6 四阶 Adams 方法

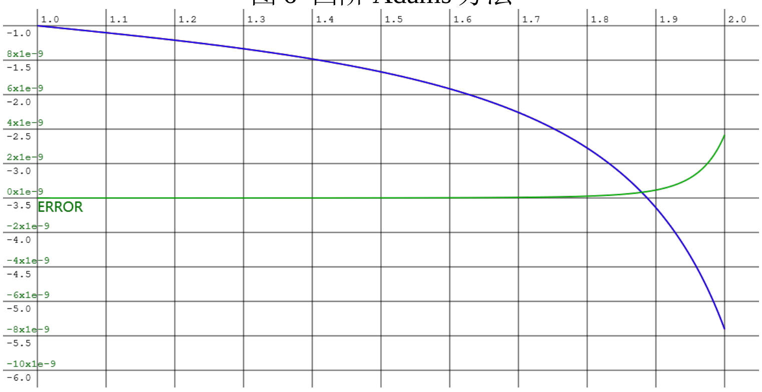

图 7 四阶 Adams 预估-矫正方法 PECE 模式

附录二 完整代码

1.求解问题的 C++程序

```
#include <cstdio>
#define _USE_MATH_DEFINES //Use const M_PI_4 for pi/4
#include <cmath>
#include <chrono>
using namespace std;
using namespace std::chrono;
double func(double _t,double _y){
     return -(_y+1.0/_t)/_t-_y*_y;
}
double integ(int _m,int _N,double _s,double _e,double _0){
     double _y=_0,_h=(_e-_s)/_N,_t=_s;
     double _h2=_h/2,_h6=_h/6,_h4=_h/4,_h2_3=_h*2/3,_h24=_h/24;
     double _f,_k1,_k2,_k3,_k4;
     switch(_m){
          case 1: //Euler方法
               for(int i=0;i<_N;i++,_t+=_h) _y=_y+_h*func(_t,_y);
               break;
          case 2: //改进的Euler方法
               for(int i=0;i<_N;i++){
                    _f=func(_t,_y); _k1=_y+_h*_f;
                    _t+=_h; _y=_y+_h2*(_f+func(_t,_k1));
               }
               break;
          case 3: //Heun公式
               for(int i=0;i<_N;i++,_t+=_h){
               _f=func(_t,_y); _k1=_y+_h2_3*_f;
                    _y=_y+_h4*(_f+3*func(_t+_h2_3,_k1));
               }
               break;
          case 4: //中点方法（变形的Euler方法）
               for(int i=0;i<_N;i++,_t+=_h){
               _f=func(_t,_y); _k1=_y+_h2*_f;
                    _y=_y+_h*func(_t+_h2,_k1);
               }
               break;
          case 5:  //四阶经典Runge-Kuta方法
               for(int i=0;i<_N;i++){
               _k1=func(_t,_y); _k2=func(_t+_h2,_y+_h2*_k1);
                    _k3=func(_t+_h2,_y+_h2*_k2); _t+=_h;
                    _k4=func(_t,_y+_h*_k3);
                    _y=_h6*(_k1+2*_k2+2*_k3+_k4)+_y;
               }
               break;
          default: //四阶Adams PECE方法
               double *_ff=(double*)malloc((_N+1)*sizeof(double));
               _ff[0]=func(_s,_0);
               for(int i=0;i<3;i++){
               _k1=func(_t,_y); _k2=func(_t+_h2,_y+_h2*_k1);
                    _k3=func(_t+_h2,_y+_h2*_k2); _t+=_h;
                    _k4=func(_t,_y+_h*_k3);
                    _y=_h6*(_k1+2*_k2+2*_k3+_k4)+_y;
                    _ff[i+1]=func(_t,_y);
               }
               if(_m==7) goto X;
               for(int i=3;i<_N;){
                    _k1=_y+_h24*(55*_ff[i]-59*_ff[i-1]+37*_ff[i-2]-9*_ff[i-3]);
               _t+=_h;
                    _y=_y+_h24*(9*func(_t,_k1)+19*_ff[i]-5*_ff[i-1]+_ff[i-2]);
               i++; _ff[i]=(func(_t,_y));
          } break;
          X:
```

```
            for(int i=3;i<_N;){
                _y=_y+_h24*(55*_ff[i]-59*_ff[i-1]+37*_ff[i-2]-9*_ff[i-3]);
            _t+=_h; i++; _ff[i]=(func(_t,_y));
        }
    }
    return _y;
}
void output(int _m,int _N,double _t,double _e){
    switch(_m){
        case 1: printf("1 Euler Method\n"); break;
        case 2: printf("2 Improved Euler Method\n"); break;
        case 3: printf("3 Heun's Method \n"); break;
        case 4: printf("4 Midpoint Method\n"); break;
        case 5:  printf("5 Runge-Kutta 4th Order Method\n"); break;
        case 7:  printf("-1 4th Order Adams' Method\n"); break;
        default: printf("0 4th Order Adams' PECE Method\n");
    }
    printf("Stepsize=1/N N: %d\n",_N);
    if(_t>1e-4)
        printf("Time(μs): %.3f\n",_t);
    printf("A.Error: %.18f\n",abs(_e));
    printf("   digit 0.123456789012345678\n\n");
    return ;
}
double ans(double _x){
    return -tan(log(_x)+M_PI_4)/_x;
}
int main() {
    double Real=ans(2),r;
    time_point<steady_clock> s,e;
    for(int m=1;m<8;m++){
        for(int N=250;N<64001;N*=2){
            s=steady_clock::now();
            r=integ(m,N,1,2,-1);
            e=steady_clock::now();
            output(m,N,(e-s).count()/1000.0,r-Real);
        }
        printf("--------\n\n");
    }
    getchar(); getchar();
    return 0;
}
```

2.四阶 Adams-PECE 与 R-K 后验阶数估计的 C++程序（其他方法同理）

```
#include <cstdio>
#define _USE_MATH_DEFINES //Use const M_PI_4 for pi/4
#include <cmath>
using namespace std;
double func(double _t,double _y){
    return -(_y+1.0/_t)/_t-_y*_y;
}
double integ_R(int _N,double _s,double _e,double _0){
    double _y=_0,_h=(_e-_s)/_N,_t=_s;
    double _h2=_h/2,_h6=_h/6,_k1,_k2,_k3,_k4;
    for(int i=0;i<_N;i++){
       _k1=func(_t,_y); _k2=func(_t+_h2,_y+_h2*_k1);
        _k3=func(_t+_h2,_y+_h2*_k2); _t+=_h;
        _k4=func(_t,_y+_h*_k3);
        _y=_h6*(_k1+2*_k2+2*_k3+_k4)+_y;
    }
    return _y;
}
double integ_A(int _N,double _s,double _e,double _0){
    double _y=_0,_h=(_e-_s)/_N,_t=_s;
```

```
    double _h2=_h/2,_h6=_h/6,_h24=_h/24;
    double _k1,_k2,_k3,_k4;
    double *_ff=(double*)malloc((_N+1)*sizeof(double));
    _ff[0]=func(_s,_0);
    for(int i=0;i<3;i++){
       _k1=func(_t,_y); _k2=func(_t+_h2,_y+_h2*_k1);
        _k3=func(_t+_h2,_y+_h2*_k2); _t+=_h;
        _k4=func(_t,_y+_h*_k3);
        _y=_h6*(_k1+2*_k2+2*_k3+_k4)+_y;
        _ff[i+1]=func(_t,_y);
    }
    for(int i=3;i<_N;){
        _k1=_y+_h24*(55*_ff[i]-59*_ff[i-1]+37*_ff[i-2]-9*_ff[i-3]);
       _t+=_h;
        _y=_y+_h24*(9*func(_t,_k1)+19*_ff[i]-5*_ff[i-1]+_ff[i-2]);
       i++; _ff[i]=(func(_t,_y));
    }
    return _y;
}
int main() {
    double Real=-tan(log(2)+M_PI_4)/2;
    double eA[20],eR[20],h[20],th;
    for(int i=0,N=500;i<20;i++,N+=500){
        h[i]=1.0/N;
        eA[i]=abs(Real-integ_A(N,1,2,-1));
        eR[i]=abs(Real-integ_R(N,1,2,-1));
    }
    printf("Admas R-K\n");
    for(int i=0;i<19;i++){
        th=1.0/log(h[i+1]/h[i]);
        printf("%.2f ",log(eA[i+1]/eA[i])*th);
        printf("%.2f\n",log(eR[i+1]/eR[i])*th);
    }
    getchar(); getchar();
    return 0;
}
```

3.图像绘制的 JavaScript 程序

（步长 1/1000，四阶 Adams-PECE 方法，其它方法同理）

```
<html>
<head>
    <title>四阶 Adams PECE 方法</title>
</head>
<body>
    <canvas id="canvas" width="1100" height="550"></canvas>
    <script>
        function func(_t,_y){
            if(Math.abs(_t-1)<1e-7) return -1.0;
             return -(_y+1.0/_t)/_t-_y*_y;
        }
        function solu(_x){
            return -Math.tan(Math.log(_x)+Math.PI/4)/_x
        }

        var canvas=document.getElementById("canvas");
        var ctx=canvas.getContext("2d");
        var solution_appr=[],solution_real=[],inter_f=[];
        var N=1000;

        var h=1/N,t=1,y=-1,y0,h2=h/2,h6=h/6,h24=h/24,k1,k2,k3,k4;
        solution_appr.push(-1); solution_real.push(-1); inter_f.push(-1);
        for(var i=0;i<3;i++){
            k1=func(t,y); k2=func(t+h2,y+h2*k1); k3=func(t+h2,y+h2*k2);
            t+=h; k4=func(t,y+h*k3); y=h6*(k1+2*k2+2*k3+k4)+y;
```

```
            solution_appr.push(y); solution_real.push(solu(t)); inter_f.push(func(t,y));
        }
        for(var i=3;i<N;i++){
            y0=y+h24*(55*inter_f[i]-59*inter_f[i-1]+37*inter_f[i-2]-9*inter_f[i-3]);
            t+=h; y=y+h24*(9*func(t,y0)+19*inter_f[i]-5*inter_f[i-1]+inter_f[i-2]);
            solution_appr.push(y); solution_real.push(solu(t)); inter_f.push(func(t,y));
        }

        ctx.lineWidth=1; ctx.strokeStyle="#000000";
        ctx.beginPath();
        for(var i=50;i<1100;i+=100){
            ctx.moveTo(i,0); ctx.lineTo(i,650);}
        for(var i=25;i<650;i+=50){
            ctx.moveTo(0,i); ctx.lineTo(1100,i);}
        ctx.stroke();
        ctx.fillStyle="#000000"; ctx.font="15px courier";
        for(var i=1;i<2.01;i+=0.1) ctx.fillText(i.toFixed(1),55+(i-1)*1000,20);
        for(var i=-1;i>-6.01;i-=0.5) ctx.fillText(i.toFixed(1),5,40+(-i-1)*100);
        ctx.fillStyle="#007700"; ctx.font="15px courier";
        for(var i=8;i>-11;i-=2) ctx.fillText(i+"x1e-9",5,(-i*25)+270);
        //for(var i=0.1;i>-0.11;i-=0.02) ctx.fillText(i.toFixed(2),3,(-i*2500)+270);
        ctx.font="20px courier bold"; ctx.fillText("ERROR",50,295);

        ctx.lineWidth=2; ctx.strokeStyle="#ff0000";
        ctx.beginPath(); ctx.moveTo(50,25); t=1;
        for(var i=1;i<=N;i++){
            t+=h; ctx.lineTo(50+(t-1)*1000,25+(-solution_appr[i]-1)*100);}
        ctx.stroke();
        ctx.strokeStyle="#0000ff";
        ctx.beginPath(); ctx.moveTo(50,25); t=1;
        for(var i=1;i<=N;i++){
            t+=h; ctx.lineTo(50+(t-1)*1000,25+(-solution_real[i]-1)*100);}
        ctx.stroke();
        ctx.strokeStyle="#00aa00";
        ctx.beginPath(); ctx.moveTo(50,275); t=1;
        for(var i=1;i<=N;i++){
            t+=h; ctx.lineTo(50+(t-1)*1000,275+(solution_appr[i]-
solution_real[i])*25000000000);
            console.log(solution_appr[i]-solution_real[i]);}
        ctx.stroke();
    </script>
</body>
</html>
```