学号：20002462　姓名：刘子言

实验三 常微分方程的差分方法实验

一. 实验目的

（1）深入理解常微分方程的差分方法的原理，学会用差分方法解决某些实际的常微分方程问题，比较这些方法解题的不同之处。

（2）熟悉 Matlab 编程环境，利用 Matlab 实现具体的常微分方程。

二. 实验要求

用 Matlab 软件实现欧拉方法、改进的欧拉方法、龙格-库塔方法和亚当姆斯方法，并用实例在计算机上计算。

三. 实验内容

3-1：用欧拉方法、改进的欧拉方法、四阶龙格-库塔方法求解初值问题：

$$\begin{cases} y' = \dfrac{2x}{3y^2} \\ y(0) = 1 \end{cases}$$

在区间[0,1]上取 h=0.1 时的数值解，并与精确解$y = \sqrt[3]{1+x^2}$进行比较。

1、设计思想

（1）欧拉方法的设计思想

对于一阶常微分方程的初值问题，设在区间$[x_n, x_{n+1}]$的左端点x_n列出方程：

$$y'(x_n) = f(x_n, y(x_n))$$

并用差商$\dfrac{y(x_{n+1}) - y(x_n)}{h}$代替其中的导数项$y'(x_n)$，则有近似关系式：

$$y(x_{n+1}) \approx y(x_n) + hf(x_n, y(x_n))$$

若用$y(x_n)$的近似值y_n代入上式右端，并记所得结果为y_{n+1}，由此设计得到 Euler 格式：

$$y_{n+1} = y_n + hf(x_n, y_n)$$

（2）改进的欧拉方法的设计思想

先用 Euler 格式（显式格式）求得一个初步近似值作为预报值$\bar{y}_{n+1}$，再用梯形格式（隐式格式）迭代得出较高精度的校正值y_{n+1}，这样可以得出改进的 Euler 格式，再将其改写为平均化形式，便于编程实现：

$$\begin{cases} y_p = y_n + hf(x_n, y_n) \\ y_c = y_n + hf(x_{n+1}, y_p) \\ y_{n+1} = \dfrac{1}{2}(y_p, y_c) \end{cases}$$

（3）四阶龙格-库塔方法的设计思想

设法在区间$[x_n,x_{n+1}]$上预报四个点$x_n,x_{n+\frac{1}{2}},x_{n+\frac{1}{2}},x_{n+1}$的斜率，然后将它们加权平均作为平均斜率，即可设计出更高精度的四阶 Runge-Kutta 格式：

$$\begin{cases} y_{n+1}=y_n+\dfrac{h}{6}(K_1+2K_2+2K_3+K_4) \\ K_1=f(x_n,y_n) \\ K_2=f\left(x_{n+\frac{1}{2}},y_n+\dfrac{h}{2}K_1\right) \\ K_3=f\left(x_{n+\frac{1}{2}},y_n+\dfrac{h}{2}K_2\right) \\ K_4=f(x_{n+1},y_n+hK_3) \end{cases}$$

其中K_1直接求出，然后依次预报出K_2，K_3和K_4。可以看出，这一格式每一步需 4 次计算 f 的函数值 y。

2、对应程序

（1）常微分方程的函数代码：

```
function z = liuziyan_3_1_dEquation( x,y )
z = 2*x./(3*y.^2);
end
```

（2）求准确解的函数代码：

```
function y = liuziyan_3_1_solvef( x )
y = power(1+x.^2,1/3);
end
```

（3）欧拉方法：

```
function E = liuziyan_3_1_1_Euler( f,a,b,N,ya )
% f——微分方程右端函数；N——区间等分的个数；
% a,b——自变量的取值区间[a,b]的端点；ya——表初值 y(a)
% E=[x',y']是自变量 X 和解 Y 所组成的矩阵
h = (b-a)/N;
y = zeros(1,N+1);
x = zeros(1,N+1);
y(1) = ya;
x = a:h:b;
for i = 1:N
    y(i+1) = y(i) + h*feval(f,x(i),y(i));
```

```
end
E = [x',y'];
```

（4）改进的欧拉方法：

```
function E = liuziyan_3_1_2_MendEuler( f,a,b,N,ya )
% f——微分方程右端函数；N——区间等分的个数；
% a,b——自变量的取值区间[a,b]的端点；ya——表初值 y(a)
% E=[x',y']是自变量 X 和解 Y 所组成的矩阵
h = (b-a)/N;
y = zeros(1,N+1);
x = zeros(1,N+1);
y(1) = ya;
x = a:h:b;
for i = 1:N
    y1 = y(i) + h*feval(f,x(i),y(i));
    y2 = y(i) + h*feval(f,x(i+1),y1);
    y(i+1) = (y1+y2)/2;
end
E = [x',y'];
```

（5）四阶龙格-库塔方法：

```
function R = liuziyan_3_1_3_Rungkuta4( f,a,b,N,ya )
% R = [x',y']是自变量 X 和解 Y 所组成的矩阵
h = (b-a)/N;
y = zeros(1,N+1);
x = zeros(1,N+1);
y(1) = ya;
x = a:h:b;
for i = 1:N
    k1 = feval(f,x(i),y(i));
    k2 = feval(f,x(i)+h/2,y(i)+(h/2)*k1);
    k3 = feval(f,x(i)+h/2,y(i)+(h/2)*k2);
    k4 = feval(f,x(i)+h,  y(i)+ h   *k3);
    y(i+1) = y(i) + (h/6) * (k1 + 2*k2 + 2*k3 + k4);
end
R = [x',y'];
```

3、实验结果

实验 3-1 的运行结果如下：

（1）欧拉方法：

```
>> f=@liuziyan_3_1_dEquation;a=0;b=1;N=10;ya=1;
>> E = liuziyan_3_1_1_Euler( f,a,b,N,ya );
>> y = liuziyan_3_1_solvef( a:(b-a)/N:b );
>> m = [E,y']

m =

                   0   1.000000000000000   1.000000000000000
   0.100000000000000   1.000000000000000   1.003322283542089
   0.200000000000000   1.006666666666667   1.013159403820177
   0.300000000000000   1.019823984328173   1.029142466571507
   0.400000000000000   1.039053996210607   1.050717574498580
   0.500000000000000   1.063753742840065   1.077217345015942
   0.600000000000000   1.093211287375390   1.107931651350893
   0.700000000000000   1.126680984094971   1.142164759185383
   0.800000000000000   1.163443468397965   1.179273707994073
   0.900000000000000   1.202844551131971   1.218688907741976
   1.000000000000000   1.244314379696825   1.259921049894873
```

（2）改进的欧拉方法：

```
>> f=@liuziyan_3_1_dEquation;a=0;b=1;N=10;ya=1;
>> E = liuziyan_3_1_2_MendEuler( f,a,b,N,ya );
>> y = liuziyan_3_1_solvef( a:(b-a)/N:b );
>> m = [E,y']

m =

                   0   1.000000000000000   1.000000000000000
   0.100000000000000   1.003333333333333   1.003322283542089
   0.200000000000000   1.013180434398852   1.013159403820177
   0.300000000000000   1.029171244550309   1.029142466571507
   0.400000000000000   1.050751079998023   1.050717574498580
   0.500000000000000   1.077252310612832   1.077217345015942
   0.600000000000000   1.107965053358377   1.107931651350893
   0.700000000000000   1.142194135689444   1.142164759185383
   0.800000000000000   1.179297284217600   1.179273707994073
   0.900000000000000   1.218705575564524   1.218688907741976
   1.000000000000000   1.259930265862033   1.259921049894873
```

（3）四阶龙格-库塔方法：

```
>> f=@liuziyan_3_1_dEquation;a=0;b=1;N=10;ya=1;
>> R = liuziyan_3_1_3_Rungkuta4( f,a,b,N,ya );
>> y = liuziyan_3_1_solvef( a:(b-a)/N:b );
>> m = [R,y']

m =

                   0   1.000000000000000   1.000000000000000
   0.100000000000000   1.003322292719565   1.003322283542089
   0.200000000000000   1.013159438200695   1.013159403820177
   0.300000000000000   1.029142535439115   1.029142466571507
   0.400000000000000   1.050717679021904   1.050717574498580
   0.500000000000000   1.077217479999272   1.077217345015942
   0.600000000000000   1.107931808368870   1.107931651350893
   0.700000000000000   1.142164929384162   1.142164759185383
   0.800000000000000   1.179273883780467   1.179273707994073
   0.900000000000000   1.218689083410464   1.218688907741976
   1.000000000000000   1.259921221582087   1.259921049894873
```

由计算结果可知，改进的 Euler 方法比 Euler 方法计算结果更准确一些，但精度依旧有待提高；四阶的 Runge-Kutta 方法与前两种方法相比，精度要更高一些。

3-2：分别用四阶亚当姆斯方法、改进的四阶亚当姆斯预估校正系统求解题 3-1 中的初值问题。提示：可用四阶龙格-库塔方法求出开头三步的值。

1、设计思想

（1）四阶亚当姆斯方法的设计思想

将四阶显式和隐式的 Adams 格式匹配在一起，即可生成四阶 Adams 预报校正系统：

$$\begin{cases} \bar{y}_{n+1} = y_n + \dfrac{h}{24}\left(55y'_n - 59y'_{n-1} + 37y'_{n-2} - 9y'_{n-3}\right) \\ \bar{y}'_{n+1} = f\left(x_{n+1}, \bar{y}_{n+1}\right) \\ y_{n+1} = y_n + \dfrac{h}{24}\left(9\bar{y}'_{n+1} + 19y'_n - 5y'_{n-1} + y'_{n-2}\right) \\ y'_{n+1} = f\left(x_{n+1}, y_{n+1}\right) \end{cases}$$

由于这种方法是 4 步法，在计算时要用到前三步的信息，因此它不能自行启动，实际计算时，可利用四阶 Runge-Kutta 方法为其提供开头 3 步的值。

（2）改进的四阶亚当姆斯方法的设计思想

利用误差估计式修改四阶 Adams 预报校正系统，即可导出下列改进的四阶 Adams 预

报校正系统：

$$\begin{cases} p_{n+1}=y_n+\dfrac{h}{24}\left(55y'_n-59y'_{n-1}+37y'_{n-2}-9y'_{n-3}\right) \\ m_{n+1}=p_{n+1}-\dfrac{251}{270}\left(p_n-c_n\right) \\ m'_{n+1}=f\left(x_{n+1},m_{n+1}\right) \\ c_{n+1}=y_n+\dfrac{h}{24}\left(9m'_{n+1}+19y'_n-5y'_{n-1}+y'_{n-2}\right) \\ y_{n+1}=c_{n+1}-\dfrac{19}{270}\left(p_{n+1}-c_{n+1}\right) \\ y'_{n+1}=f\left(x_{n+1},y_{n+1}\right) \end{cases}$$

2、对应程序

（1）对应的常微分方程和求准确解的函数代码与实验 3-1 的相同：

```
function z = liuziyan_3_1_dEquation( x,y )
z = 2*x./(3*y.^2);
end
function y = liuziyan_3_1_solvef( x )
y = power(1+x.^2,1/3);
end
```

（2）四阶亚当姆斯方法：

```
function A1 = liuziyan_3_2_1_Adams4PC( f,a,b,N,ya )
% A1 = [x',y']是自变量 X 和解 Y 所组成的矩阵
if N < 4
    return;
end
h = (b-a)/N;
y = zeros(1,N+1);
x = zeros(1,N+1);
y(1) = ya;
x = a:h:b;
F = zeros(1,4);
for i = 1:N
    if i < 4          %用四阶 Runge-Kutta 法求前 3 个初始解
        k1 = feval(f,x(i),y(i));
        k2 = feval(f,x(i)+h/2,y(i)+(h/2)*k1);
```

```
        k3 = feval(f,x(i)+h/2,y(i)+(h/2)*k2);
        k4 = feval(f,x(i)+h,  y(i)+ h   *k3);
        y(i+1) = y(i) + (h/6)*(k1+2*k2+2*k3+k4);
    else
        F = feval(f, x(i-3:i), y(i-3:i));
        py = y(i) + (h/24)*( F*[-9, 37, -59, 55]'); %预报
        p = feval(f, x(i+1), py);
        F = [F(2) F(3) F(4) p];
        y(i+1) = y(i) + (h/24)*( F*[1, -5, 19, 9]'); %校正
    end
end
A1 = [x',y'];
```

（3）改进的四阶亚当姆斯方法：

```
function A2 = liuziyan_3_2_2_CAdams4PC( f,a,b,N,ya )
% A2 = [x',y']是自变量 X 和解 Y 所组成的矩阵
if N < 4
    return;
end
h = (b-a)/N;
y = zeros(1,N+1);
x = zeros(1,N+1);
y(1) = ya;
x = a:h:b;
F = zeros(1,4);
for i = 1:N
    if i < 4        %用四阶 Runge-Kutta 法求前 3 个初始解
        k1 = feval(f,x(i),y(i));
        k2 = feval(f,x(i)+h/2,y(i)+(h/2)*k1);
        k3 = feval(f,x(i)+h/2,y(i)+(h/2)*k2);
        k4 = feval(f,x(i)+h,  y(i)+ h   *k3);
        y(i+1) = y(i) + (h/6)*(k1+2*k2+2*k3+k4);
    elseif i==4
            F = feval(f, x(i-3:i), y(i-3:i));
            py = y(i) + (h/24)*( F*[-9, 37, -59, 55]');  %预报
            p = feval(f, x(i+1), py);
            F = [ F(2) F(3) F(4) p ];
            y(i+1) = y(i) + (h/24)*( F*[1, -5, 19, 9]');  %校正
            p = py;
```

```
            c = y(i+1);
    else
            F = feval(f, x(i-3:i), y(i-3:i));
            py = y(i) + (h/24)*( F*[-9, 37, -59, 55]');  %预报
            my = py - 251*(p-c)/270;                     %改进
            m = feval(f, x(i+1), my);
            F = [ F(2) F(3) F(4) m ];
            cy = y(i) + (h/24)*( F*[1, -5, 19, 9]');  %校正
            y(i+1) = cy + 19*(py-cy)/270;             %改进
            p = py;
            c = cy;
    end
end
A2 = [x',y'];
```

3、实验结果

实验 3-2 的运行结果如下：

将四阶亚当姆斯方法、改进的四阶亚当姆斯方法结果与准确值放在一起显示：

```
>> f=@liuziyan_3_1_dEquation;a=0;b=1;N=10;ya=1;
>> A1 = liuziyan_3_2_1_Adams4PC( f,a,b,N,ya );
>> A2 = liuziyan_3_2_2_CAdams4PC( f,a,b,N,ya );
>> y = liuziyan_3_1_solvef(a:(b-a)/N:b);
>> m=[A1,A2(:,2),y']

m =

                   0   1.000000000000000   1.000000000000000   1.000000000000000
   0.100000000000000   1.003322292719565   1.003322292719565   1.003322283542089
   0.200000000000000   1.013159438200695   1.013159438200695   1.013159403820177
   0.300000000000000   1.029142535439115   1.029142535439115   1.029142466571507
   0.400000000000000   1.050720005701320   1.050720005701320   1.050717574498580
   0.500000000000000   1.077222006226289   1.077219577281977   1.077217345015942
   0.600000000000000   1.107937969725546   1.107933404741466   1.107931651350893
   0.700000000000000   1.142171994240436   1.142165924289030   1.142164759185383
   0.800000000000000   1.179281183806477   1.179274343902799   1.179273707994073
   0.900000000000000   1.218696130520936   1.218689154161893   1.218688907741976
   1.000000000000000   1.259927725124502   1.259921053872529   1.259921049894873
```

以上 4 列从左到右依次为：离散节点值，四阶 Adams 预报校正系统所求解，改进的四阶 Adams 预报校正系统所求解，准确解。通过计算结果的比较分析可知，改进的四阶 Adams 预报校正系统效果比四阶 Adams 预报校正系统效果要好。

四. 实验体会

对实验过程进行分析总结，对比求解常微分方程的不同方法，指出每种算法的设计要点及应注意的事项，以及自己通过实验所获得的对常微分方程的差分方法的理解。

答：

（1）Euler 法是一种显式算法，其计算量小，但精度很低；

（2）改进的 Euler 法也是一种显式格式，是将 Euler 格式与梯形格式相结合建立的预报校正系统，保证了计算量不会过大的同时还明显地改善了精度；

（3）Runge-Kutta 方法阶数不高，但是可以达到较高的精度，它在设计时用 f(x,y)在某点上的值的线性组合来计算 y_{n+1}，避免计算函数 f(x,y)的偏导数，从而提高了精度；

（4）Adams 预报校正系统相比同阶的单步法如 Runge-Kutta 法计算量大大减小，同时也能获得不错的精度；

（5）对于显式法和隐式法，一般来说同阶的隐式法比显式法精确，而且数值稳定性也较好，但在隐式公式中需要用迭代法求解，计算量较大。所以在实际计算中，很少单独使用显式公式或者隐式公式，而是将它们联合使用，先用显式公式求预测值，再用隐式公式对预测值进行校正，从而得出近似值。