电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

数值分析第三次作业

The third assignment of numerical analysis

课程题目：	数值分析第三次作业
专　　业：	电子信息
姓　　名：	郭元洪
学　　号：	202122140307

目录

1 问题

作业 1：利用合适的方法找一个函数，使之能很好地描述下面数据的关系：

x=[0,3,5,7,9,11,12,13,14,15];

y=[0,1.2,1.7,2.0,2.1,2.0,1.8,1.2,1.0,1.6];

作业 2：利用欧拉法、修正欧拉法、隐式阿当姆斯构求解下面的微分方程，讨论计算的稳定性。

$$\begin{cases} \dfrac{dy}{dx} = \arctan(x+y)e^{-(x^2+y^2)}, \\ y(0)=1. \end{cases}$$

2 拟合

已知一组（二维）数据，即平面上 n 个点 $(x_i, y_i) i=1,...,n$,寻求一个函数（曲线）$y = f\left(x\right)$，使 $f\left(x\right)$ 在某种准则下与所有数据点最为接近，即曲线拟合得最好。对于拟合函数 $\varphi(x)=\sum_{j=1}^{n} a_j \varphi_j(x)$ 使得 $\sum_{i=1}^{n}[\varphi(x_i)-y_i]^2$ 达到最小。

在实际问题中常用幂函数，指数函数，三角函数作为拟合函数。曲线拟合质量与曲线逼近数据点的准则有关，这里拟合条件直观上可解释为：要求拟合曲线与各数据点在 y 方向的误差平方和最小。这一准则不仅满足了数值逼近上的要求，而且满足了计算上的可实现性。

根据拟合条件确定拟合函数中的系数的方法称作最小二乘法。令 $\sum_{k=1}^{n} a_k \varphi_k(x_j) = y_j (j=1,2,...,m)$ 得到方程组

$$\begin{pmatrix} \varphi_0(x_1) & \varphi_1(x_1) & \cdots & \varphi_n(x_1) \\ \varphi_0(x_2) & \varphi_1(x_2) & \cdots & \varphi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_0(x_m) & \varphi_1(x_m) & \cdots & \varphi_n(x_m) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} (m > n+1)$$

该方程组称为超定方程组，记为 $GX = F$ 。我们定义残差向量为 $r = GX - F$ 。最小二乘问题就转换成 $\min \| GX - F \|_2$ 求得其极值。最终转换成正规方程组求其系数，即 $G^T GX = G^T F$ 。

3 微分方程

一阶常微分方程初值问题是

$$\frac{dy}{dx} = f(x, y)$$
$$y(x_0) = y_0$$

其中 $y = y(x)$ 是未知函数，$y(x_0) = y_0$ 是初值条件，而 $f(x, y)$ 是给定的二元函数。

3.1 欧拉法

欧拉法是求解一阶微分方程初值问题较为简单的数值方法。我们用差商代替导数计算，并将微分方程离散化，得到一般递推式：

$$y_{n+1} = y_n + hf(x_n, y_n)(n = 0,1,2,...)$$

3.2 改进欧拉公式

为了得到比欧拉方法更精准的计算公式，用梯形求积公式近似 $y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(t, y(t))dt$ 中右端的积分，即

$$\int_{x_n}^{x_{n+1}} f(t, y(t))dt \approx \frac{h}{2}[f(x_n, y(x_n)) + f(x_{n+1}, y(x_{n+1}))]$$

再以 y_n 代替 $y(x_n)$， y_{n+1} 代替 $y(x_{n+1})$，就得到求解初值问题的另一种数值解法，梯形法。

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})]$$

梯形法相对于欧拉方法虽然精度提高了，但是每一步都要用迭代法解方程，每迭代一次，都要重新计算函数 $f(x)$ 的值，而迭代又要反复进行若干次，计算量很大，且事先难以估计迭代次数。实际计算时，一种有效措施是构造所谓预测-校正法。当 h 较小时，只迭代一二次就可转入下一步的计算，这就简化了算法。

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

校正 $y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \overline{y_{n+1}})]$

3.3 隐式阿当姆斯

如果计算 y_{n+k} 时，除用 $y_{n+k\text{-}1}$ 的值，还用到 $y_{n+i}\left(i=0,1,2,...,k-2\right)$ 的值，则称此方法为线性多步法。一般的线性多步法公式可表示为：

$$y_{n+k}=\sum_{i=0}^{k-1}\alpha_i y_{n+i}+h\sum_{i=0}^{k}\beta_i f_{n+i}$$

其中， y_{n+i} 为 $y(x_{n+i})$ 的近似， $f_{n+i}=f(x_{n+i},y_{n+i})$， $x_{n+i}=x_0+ih$， α_i， β_i 为常数， α_0， β_0 不全为 0，则称为线性 k 步法。

考虑形如 $y_{n+k}=y_{n+k-1}+h\sum_{i=0}^{k}\beta_i f_{n+i}$，其中 $\beta_k\neq 0$ 为隐式阿当姆斯方法。

4 结果与讨论

4.1 拟合结果与讨论

我们分别采用 3 次，6 次，8,次，9 次和 12 次的多项式对数据进行拟合。从图 1 中我们可以看出，3 次多项式对数据拟合最差，随着次数越高，对所有数据的拟合程度就更好，在使用 9 次多项式和 12 次多项式时两个曲线接近重合，拟合效果最好。

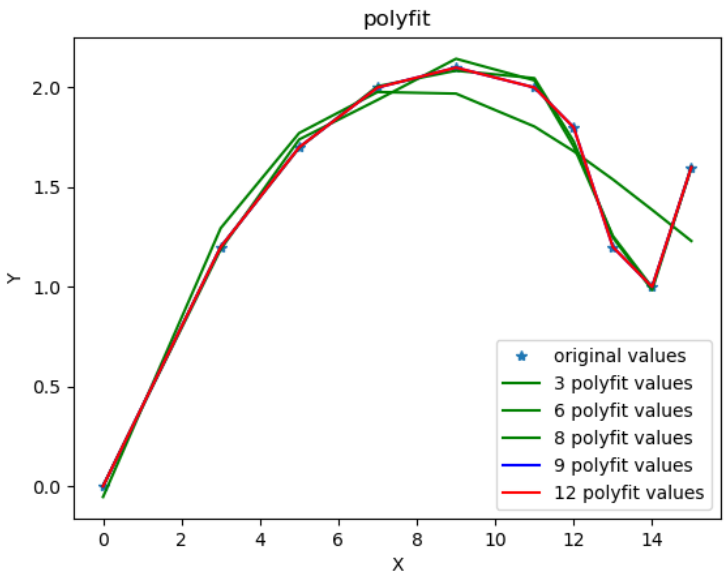


图 1 拟合曲线

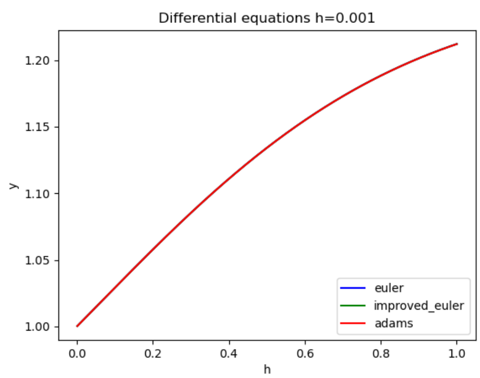
我们将上述多项式的表达式列在表 1 中。

表 1 多项式表达式

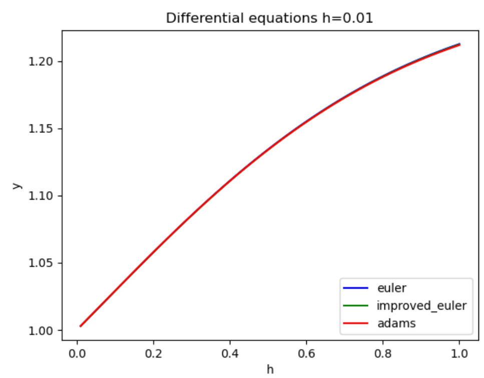
次数	多项式表达式
3	$0.001187x^3-0.05168x^2+0.5939x-0.05407$
6	$3.343e^{-05}x^6-0.001436x^5+0.02325x^4-0.1756x^3+0.5837x^2-0.2947x+0.0008352$
8	$-1.233e^{-06}x^8+8.532e^{-05}x^7-0.002417x^6+0.03625x^5-0.3098x^4+1.503x^3-3.836x^2+4.341x-1.395e^{-05}$
9	$-2.081e^{-06}x^9+0.000159x^8-0.005179x^7+0.09356x^6-1.022x^5+6.893x^4-27.85x^3+61.26x^2-55.29x-6.997e^{-12}$
12	$-1.204e^{-10}x^{12}+4.958e^{-09}x^{11}-3.779e^{-08}x^{10}-7.622e^{-07}x^9+5.84e^{-06}x^8+0.0001643x^7-0.001094x^6-0.03213x^5+0.5277x^4-3.254x^3+9.18x^2-9.359x$

4.2 常微分方程结果与讨论

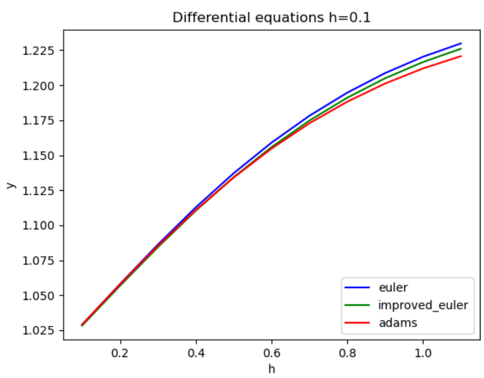
我们设定在[0,1]区间上，选取步长为 0.001，0.01，0.1 三种步长对欧拉法，改进欧拉法，隐式阿当姆斯方法进行实验，结果如图 2 所示。



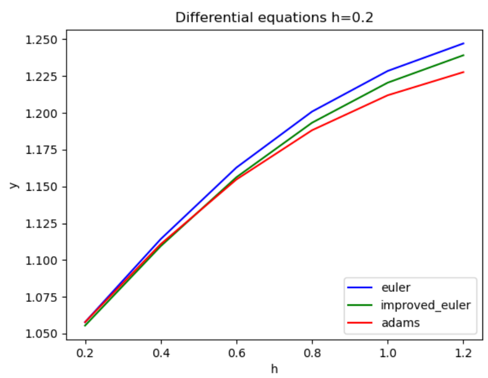
A 步长 0.001



B 步长 0.01



C 步长 0.1



A 步长 0.2

图 2 微分方程数值解结果

从图中我们可以看出，步长取 0.001 时，三种方法都能很好的表示精确解，稳定性都很好。当步长取 0.01 时，欧拉法不能很好的表示精确解，稳定性不好。当步长取 0.1 与 0.2 时，三种方法都偏离了精确解，缺乏稳定性。从中我们看出，步长越小，计算结果越好。

综上所述，其原因在于微分方程初值问题的数值方法是用差分格式进行计算的，而在差分方程的求解过程中，存在着各种计算误差，这些计算误差如舍入误差等引起的扰动，在传播过程中，可能会大量积累，对计算结果的准确性将产生影响，这就涉及到算法稳定性问题。

附录

作业 1 代码

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0,3,5,7,9,11,12,13,14,15])
y = np.array([0,1.2,1.7,2.0,2.1,2.0,1.8,1.2,1.0,1.6])

z1 = np.polyfit(x,y,3) #用 3 次多项式拟合　返回三次多项式系数
z2 = np.polyfit(x,y,6)
z3 = np.polyfit(x,y,8)
z4 = np.polyfit(x,y,9)
z5 = np.polyfit(x,y,12)

p1= np.poly1d(z1)
p2= np.poly1d(z2)
p3= np.poly1d(z3)
p4= np.poly1d(z4)
p5= np.poly1d(z5)

print(p1) #在屏幕上打印拟合多项式
print(p2) #在屏幕上打印拟合多项式
print(p3) #在屏幕上打印拟合多项式
print(p4) #在屏幕上打印拟合多项式
print(p5) #在屏幕上打印拟合多项式

yvals1 = p1(x)#也可以使用 yvals=np.polyval(z1,x)
yvals2 = p2(x)
yvals3 = p3(x)
yvals4 = p4(x)
yvals5 = p5(x)

plot0 = plt.plot(x,y,'*',label='original values')
plot1 = plt.plot(x,yvals1,'g',label='3 polyfit values')
plot2 = plt.plot(x,yvals2,'g',label='6 polyfit values')
plot3 = plt.plot(x,yvals3,'g',label='8 polyfit values')
plot4 = plt.plot(x,yvals4,'b',label='9 polyfit values')
```

```
plot5 = plt.plot(x,yvals5,'r',label='12 polyfit values')

plt.xlabel('X')
plt.ylabel('Y')
plt.legend(loc=4)    #指定 legend 的位置
plt.title('polyfit')
plt.show()
```

```
from numpy import arctan
from pylab import *
import warnings
import math
warnings.filterwarnings('ignore')

def f(t,y):
    return arctan(t+y)*math.exp(-(t**2+y**2))

def euler_forward(f,a=0,b=1,ya=1,h=0.1,verbose=True):
    res = []
    xi = a
    yi = ya
    j = 0
    x = []
    while xi<=b: # 在求解区间范围
        y = yi + h*f(xi,yi)
        if verbose:
            print('x[{}]:{:.2f}, y[{}]:{:.6f}'.format(j,xi,j,yi))
        res.append(y)
        xi, yi = xi+h, y
        x.append(xi)
        j=j+1
    return res,x


def improved_euler(f,a=0,b=1,ya=1,h=0.1,verbose=True):
    res = []
    xi = a
    yi = ya
    j = 0
    x = []
    while xi <= b: # 在求解区间范围
        yp = yi + h*f(xi, yi)
        y = yi + h/2 * (f(xi, yi) + f(xi, yp))
        if verbose:
            print('x[{}]:{:.2f}, y[{}]:{:.6f}'.format(j,xi,j,yi))
```

```
            res.append(y)
            xi, yi = xi+h, y
            x.append(xi)
            j=j+1
        return res,x

def adams(f,a,b,ya,h):
        res = []
        xi = a
        yi = ya
        j = 0
        x = []
        while xi <= b:
            x1 = xi + h
            k1 = f(xi, yi)
            k2 = f(xi+h/2, yi+h*k1/2)
            k3 = f(xi+h/2, yi+h*k2/2)
            k4 = f(x1, yi+h*k3)
            y1 = yi + h * (k1 + 2* k2 + 2 * k3 + k4) / 6
            print("%.2f, %.6f" %(x1, y1))
            res.append(y1)
            xi = x1
            yi = y1
            x.append(xi)
            j=j+1
        return res,x

res1,x1 = euler_forward(f,a=0,b=1,ya=1,h=0.3,verbose=True)
res2,x2 = improved_euler(f,a=0,b=1,ya=1,h=0.3,verbose=True)
res3,x3 = adams(f,a=0,b=1,ya=1,h=0.3)

plot1 = plt.plot(x1,res1,'b',label='euler')
plot2 = plt.plot(x2,res2,'g',label='improved_euler')
plot3 = plt.plot(x3,res3,'r',label='adams')

plt.xlabel('h')
plt.ylabel('y')
plt.legend(loc=4)    #指定 legend 的位置
plt.title('Differential equations h=0.3')
plt.show()
```