**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 1](#_Toc92649912)

[2拟合 1](#_Toc92649913)

[3微分方程 2](#_Toc92649914)

[3.1欧拉法 2](#_Toc92649915)

[3.2改进欧拉公式 2](#_Toc92649916)

[3.3隐式阿当姆斯 3](#_Toc92649917)

[4结果与讨论 3](#_Toc92649918)

[4.1拟合结果与讨论 3](#_Toc92649919)

[4.2常微分方程结果与讨论 4](#_Toc92649920)

[附录 5](#_Toc92649921)

# 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。



# 2拟合

已知一组（二维）数据，即平面上个点,寻求一个函数（曲线），使在某种准则下与所有数据点最为接近，即曲线拟合得最好。对于拟合函数使得达到最小。

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。令得到方程组



该方程组称为超定方程组，记为。我们定义残差向量为。最小二乘问题就转换成求得其极值。最终转换成正规方程组求其系数，即。

# 3微分方程

一阶常微分方程初值问题是



其中是未知函数，是初值条件，而是给定的二元函数。

## 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：



## 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中右端的积分，即



再以代替，代替，就得到求解初值问题的另一种数值解法，梯形法。



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

校正

## 3.3隐式阿当姆斯

如果计算时，除用的值，还用到的值，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：



其中，为的近似，，，，为常数，，不全为0，则称为线性步法。

考虑形如，其中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。

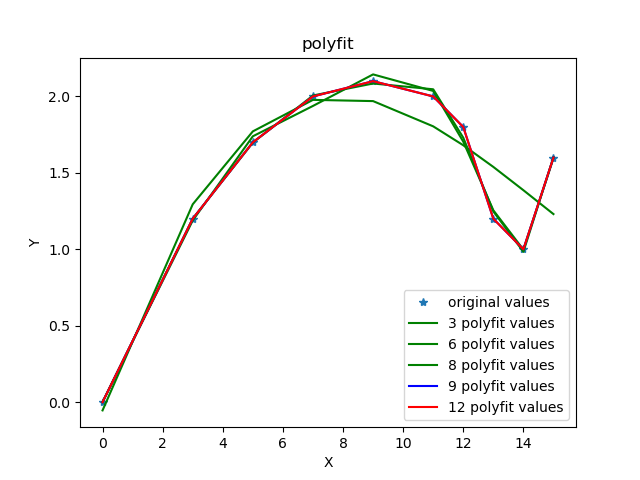


图1拟合曲线

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 | 0.001187x3-0.05168x2+0.5939x-0.05407 |
| 6 | 3.343e-05x6-0.001436x5+0.02325x4-0.1756x3+0.5837x2-0.2947x+0.0008352 |
| 8 | -1.233e-06x8+8.532e-05x7-0.002417x6+0.03625x5-0.3098x4  +1.503x3-3.836x2+4.341x-1.395e-05 |
| 9 | -2.081e-06x9+0.000159x8-0.005179x7+0.09356x6-1.022x5+6.893x4-27.85x3+61.26x2-55.29x-6.997e-12 |
| 12 | -1.204e-10x12+4.958e-09x11-3.779e-08x10-7.622e-07x9+5.84e-06x8+0.0001643x7-0.001094x6-0.03213x5+0.5277x4-3.254x3+9.18x2-9.359x |

## 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() |