目 录

[1 高斯列主元法解线性方程组算法](#_Toc29303_WPSOffice_Level1) [1](#_Toc29303_WPSOffice_Level1)

[1.1 算法说明](#_Toc11603_WPSOffice_Level2) [1](#_Toc11603_WPSOffice_Level2)

[1.1.1 列主元](#_Toc11603_WPSOffice_Level3) [1](#_Toc11603_WPSOffice_Level3)

[1.1.2 高斯消去法](#_Toc19345_WPSOffice_Level3) [1](#_Toc19345_WPSOffice_Level3)

[1.2 算法流程图](#_Toc19345_WPSOffice_Level2) [1](#_Toc19345_WPSOffice_Level2)

[1.3 算法程序调试](#_Toc2214_WPSOffice_Level2) [2](#_Toc2214_WPSOffice_Level2)

[1.3.1 C++程序运行](#_Toc2214_WPSOffice_Level3) [2](#_Toc2214_WPSOffice_Level3)

[1.3.2 Matlab程序运行](#_Toc6245_WPSOffice_Level3) [2](#_Toc6245_WPSOffice_Level3)

[2 牛顿法解非线性方程组算法](#_Toc11603_WPSOffice_Level1) [3](#_Toc11603_WPSOffice_Level1)

[2.1 算法说明](#_Toc6245_WPSOffice_Level2) [3](#_Toc6245_WPSOffice_Level2)

[2.2 算法流程图](#_Toc10465_WPSOffice_Level2) [3](#_Toc10465_WPSOffice_Level2)

[2.3 算法程序调试](#_Toc14903_WPSOffice_Level2) [4](#_Toc14903_WPSOffice_Level2)

[2.3.1 C++程序运行](#_Toc19206_WPSOffice_Level3) [4](#_Toc19206_WPSOffice_Level3)

[2.3.2 Matlab程序运行](#_Toc28341_WPSOffice_Level3) [5](#_Toc28341_WPSOffice_Level3)

[3 经典四阶龙哥库塔法解一阶微分方程组算法](#_Toc19345_WPSOffice_Level1) [6](#_Toc19345_WPSOffice_Level1)

[3.1 算法说明](#_Toc30730_WPSOffice_Level2) [6](#_Toc30730_WPSOffice_Level2)

[3.2 算法流程图](#_Toc31358_WPSOffice_Level2) [6](#_Toc31358_WPSOffice_Level2)

[3.3 算法程序调试](#_Toc19206_WPSOffice_Level2) [7](#_Toc19206_WPSOffice_Level2)

[3.3.1 C++程序运行](#_Toc11520_WPSOffice_Level3) [7](#_Toc11520_WPSOffice_Level3)

[3.3.2 Matlab程序运行](#_Toc31371_WPSOffice_Level3) [8](#_Toc31371_WPSOffice_Level3)

[4 三次样条插值算法](#_Toc2214_WPSOffice_Level1) [9](#_Toc2214_WPSOffice_Level1)

[4.1 算法说明](#_Toc28341_WPSOffice_Level2) [9](#_Toc28341_WPSOffice_Level2)

[4.2 算法流程图](#_Toc11520_WPSOffice_Level2) [9](#_Toc11520_WPSOffice_Level2)

[4.3 算法程序调试](#_Toc31371_WPSOffice_Level2) [10](#_Toc31371_WPSOffice_Level2)

[4.3.1 C++程序运行](#_Toc2755_WPSOffice_Level3) [10](#_Toc2755_WPSOffice_Level3)

[4.3.2 Matlab程序运行](#_Toc1206_WPSOffice_Level3) [11](#_Toc1206_WPSOffice_Level3)

[4.3.3 三次样条曲线图像](#_Toc9898_WPSOffice_Level3) [11](#_Toc9898_WPSOffice_Level3)

[5 龙贝格求积分算法](#_Toc6245_WPSOffice_Level1) [13](#_Toc6245_WPSOffice_Level1)

[5.1 算法说明](#_Toc11093_WPSOffice_Level2) [13](#_Toc11093_WPSOffice_Level2)

[5.2 算法流程图](#_Toc29867_WPSOffice_Level2) [13](#_Toc29867_WPSOffice_Level2)

[5.3 算法程序调试](#_Toc19899_WPSOffice_Level2) [14](#_Toc19899_WPSOffice_Level2)

[5.3.1 C++程序运行](#_Toc26777_WPSOffice_Level3) [14](#_Toc26777_WPSOffice_Level3)

[5.3.2 Matlab程序运行](#_Toc26073_WPSOffice_Level3) [15](#_Toc26073_WPSOffice_Level3)

[6 M次多项式曲线拟合算法](#_Toc10465_WPSOffice_Level1) [16](#_Toc10465_WPSOffice_Level1)

[6.1 算法说明](#_Toc2755_WPSOffice_Level2) [16](#_Toc2755_WPSOffice_Level2)

[6.2 算法流程图](#_Toc1206_WPSOffice_Level2) [16](#_Toc1206_WPSOffice_Level2)

[6.3 算法程序调试](#_Toc9898_WPSOffice_Level2) [16](#_Toc9898_WPSOffice_Level2)

[6.3.1 C++程序运行](#_Toc13730_WPSOffice_Level3) [16](#_Toc13730_WPSOffice_Level3)

[6.3.2 Matlab程序运行](#_Toc32151_WPSOffice_Level3) [17](#_Toc32151_WPSOffice_Level3)

[7 复化辛普森公式求积分算法](#_Toc14903_WPSOffice_Level1) [18](#_Toc14903_WPSOffice_Level1)

[7.1 算法说明](#_Toc26777_WPSOffice_Level2) [18](#_Toc26777_WPSOffice_Level2)

[7.2 算法流程图](#_Toc26073_WPSOffice_Level2) [18](#_Toc26073_WPSOffice_Level2)

[7.3 算法程序调试](#_Toc13730_WPSOffice_Level2) [18](#_Toc13730_WPSOffice_Level2)

[7.3.1 C++程序运行](#_Toc5626_WPSOffice_Level3) [18](#_Toc5626_WPSOffice_Level3)

[7.3.2 Matlab程序运行](#_Toc31103_WPSOffice_Level3) [19](#_Toc31103_WPSOffice_Level3)

[8 拉格朗日插值算法](#_Toc30730_WPSOffice_Level1) [20](#_Toc30730_WPSOffice_Level1)

[8.1 算法说明](#_Toc32151_WPSOffice_Level2) [20](#_Toc32151_WPSOffice_Level2)

[8.2 算法流程图](#_Toc5626_WPSOffice_Level2) [20](#_Toc5626_WPSOffice_Level2)

[8.3 算法程序调试](#_Toc31103_WPSOffice_Level2) [20](#_Toc31103_WPSOffice_Level2)

[8.3.1 C++程序运行](#_Toc4246_WPSOffice_Level3) [21](#_Toc4246_WPSOffice_Level3)

[8.3.2 Matlab程序运行](#_Toc28496_WPSOffice_Level3) [21](#_Toc28496_WPSOffice_Level3)

[9 二分法解非线性方程算法](#_Toc31358_WPSOffice_Level1) [22](#_Toc31358_WPSOffice_Level1)

[9.1 算法说明](#_Toc4246_WPSOffice_Level2) [22](#_Toc4246_WPSOffice_Level2)

[9.2 算法流程图](#_Toc28496_WPSOffice_Level2) [22](#_Toc28496_WPSOffice_Level2)

[9.3 算法程序调试](#_Toc27730_WPSOffice_Level2) [22](#_Toc27730_WPSOffice_Level2)

[9.3.1 C++程序运行](#_Toc27730_WPSOffice_Level3) [22](#_Toc27730_WPSOffice_Level3)

[9.3.2 Matlab程序运行](#_Toc23081_WPSOffice_Level3) [23](#_Toc23081_WPSOffice_Level3)

[10 雅可比迭代解线性方程组算法](#_Toc19206_WPSOffice_Level1) [24](#_Toc19206_WPSOffice_Level1)

[10.1 算法说明](#_Toc23081_WPSOffice_Level2) [24](#_Toc23081_WPSOffice_Level2)

[10.2 算法流程图](#_Toc4289_WPSOffice_Level2) [24](#_Toc4289_WPSOffice_Level2)

[10.3 算法程序调试](#_Toc8338_WPSOffice_Level2) [24](#_Toc8338_WPSOffice_Level2)

[10.3.1 C++程序运行](#_Toc4289_WPSOffice_Level3) [25](#_Toc4289_WPSOffice_Level3)

[10.3.2 Matlab程序运行](#_Toc8338_WPSOffice_Level3) [25](#_Toc8338_WPSOffice_Level3)

[设计体会及改进意见](#_Toc28341_WPSOffice_Level1) [26](#_Toc28341_WPSOffice_Level1)

[参 考 文 献](#_Toc11520_WPSOffice_Level1) [27](#_Toc11520_WPSOffice_Level1)

[附 录](#_Toc31371_WPSOffice_Level1) [28](#_Toc31371_WPSOffice_Level1)

1 高斯列主元法解线性方程组算法

* 1. 算法说明
     1. 列主元

记，在Gauss消元过程的第一步，取的第一列中绝对值 最大的元素，即

 (1-1)

作为主元素，若，交换第行和第1行。

一般地，在Gauss消元过程的第步，取

 (1-2)

作为主元素。若，交换第行和第行。

* + 1. 高斯消去法

首先使用初等行变换将方程组转化为一个同解的上三角形方程组，在通过回代法求解该三角形方程组。

* 1. 算法流程图

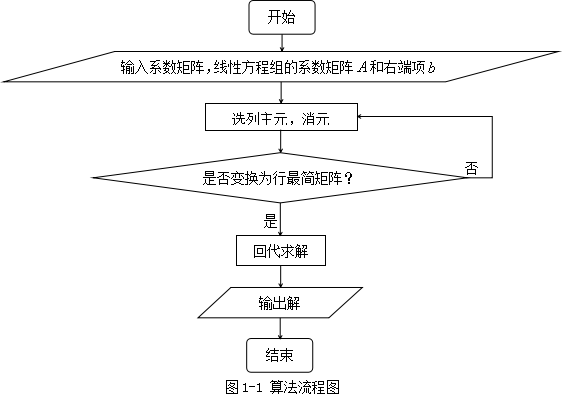


图1-1 高斯列主元法解方程组算法流程图

* 1. 算法程序调试

求解方程组

1.3.1 C++程序运行

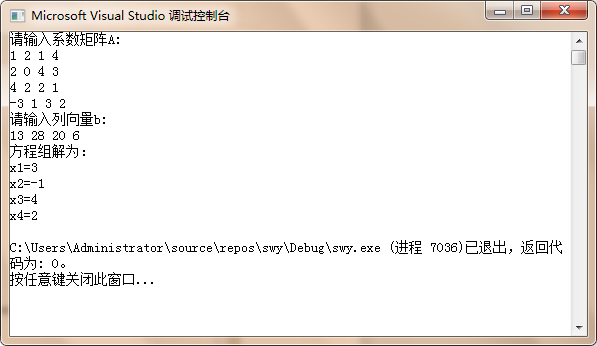


图1-2 高斯列主元法解方程组算法C++程序运行图

1.3.2 Matlab程序运行

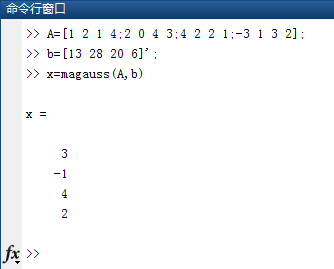


图1-3 高斯列主元法解方程组算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果一致。

2 牛顿法解非线性方程组算法

2.1 算法说明

牛顿法是一种特殊形式的迭代法，它是求解非线性方程组的最有效的方法，根据导数的几何意义，牛顿法的几何表现为：x是函数f(x)在点x处的切线与x轴的交点。根据牛顿法解非线性方程的原理，可知其求解的过程。第一步，线性化。第二步，确定初始值x，要计算近似解，必须计算，从而进行迭代。第三步，确定结束条件，要求小于精度。

公式说明：

设已知

第1步：计算函数

 (2-1)

第2步：计算雅可比矩阵

 (2-2)

第3步：求线性方程组

 (2-3)

的解。

第4步：计算下一点

 (2-4)

## 最后判断与之间的“距离”是否小于。如果是，结束；如果不是，回到第1步并以为起始点。

2.2 算法流程图

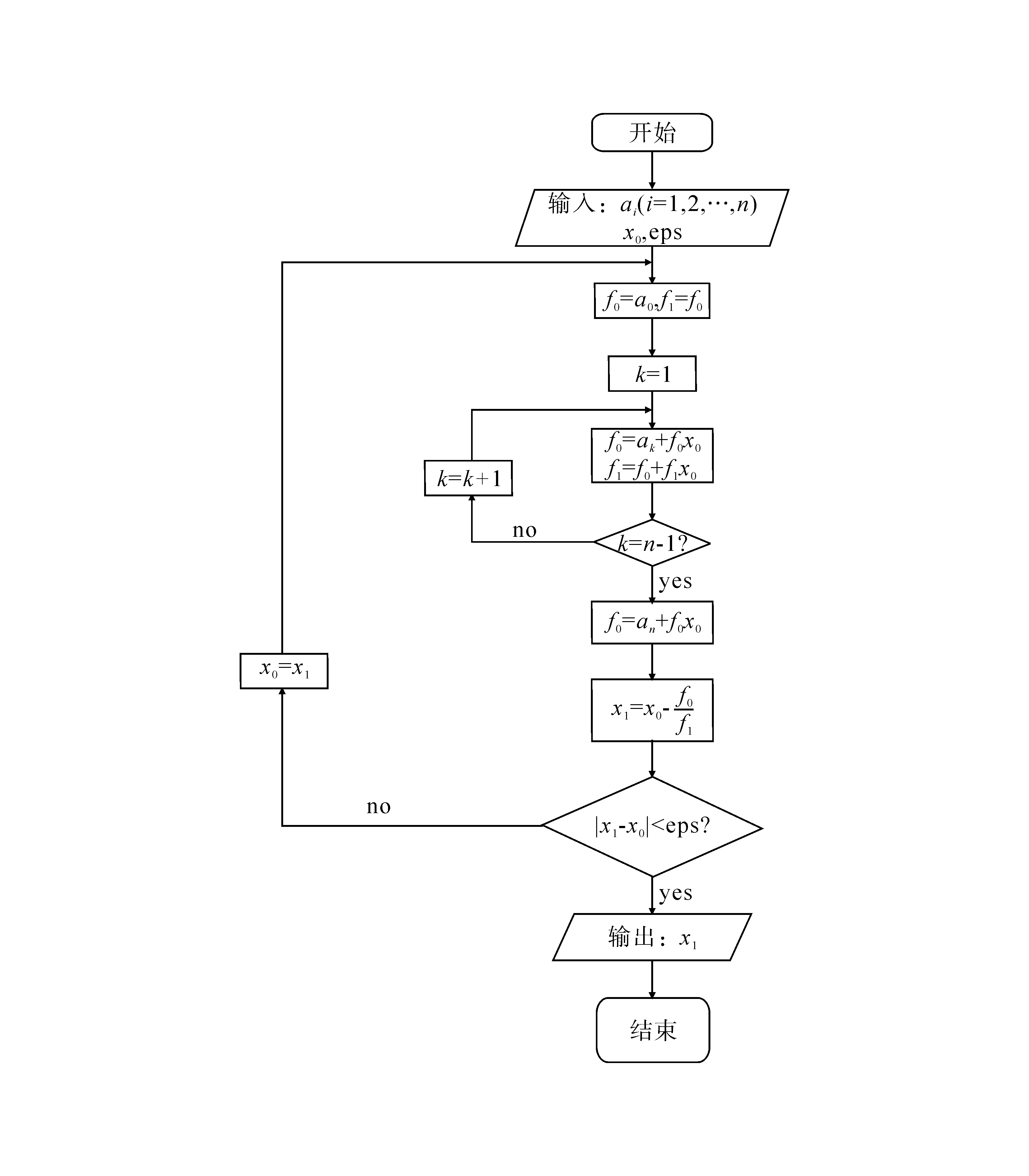


图2-1 牛顿法解非线性方程组算法流程图

2.3 算法程序调试

用牛顿法解非线性方程组：



初始近似值分别取2.00和0.25，进行三次迭代求出和。

2.3.1 C++程序运行

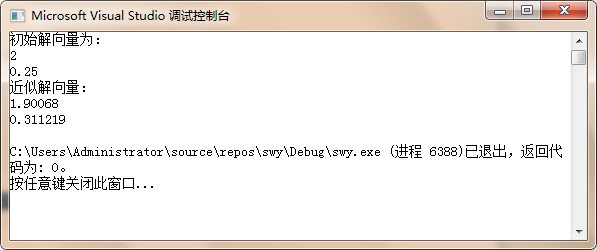


图2-2 牛顿法解非线性方程组算法C++程序运行图

2.3.2 Matlab程序运行

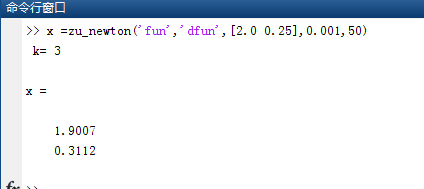


图2-3 牛顿法解非线性方程组算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果一致。

3 经典四阶龙哥库塔法解一阶微分方程组算法

3.1 算法说明

龙格-库塔（Runge-Kutta）方法由泰勒方法推导而来，使其最终全局误差为。一种折中的方法就是每步进行若干次函数求值，从而省去高阶导数计算。这种方法可构造任意N阶精度的近似公式。最长用的是N=4的龙格-库塔法，它适用于一般的应用，因为它非常精确、稳定，且易于编程。如果需要更高的精度，则应该使用更小的步长或某种自适应方法。

公式说明：

4阶龙格-库塔方法（RK4）可模拟N=4的泰勒方法的精度。标准的阶龙格-库塔方法，其描述如下。自初始点开始，利用

 (3-1)

生成近似值序列，其中

 (3-2)

为函数的入口地址，为初值，为所求点，step为计算次数。

3.2 算法流程图

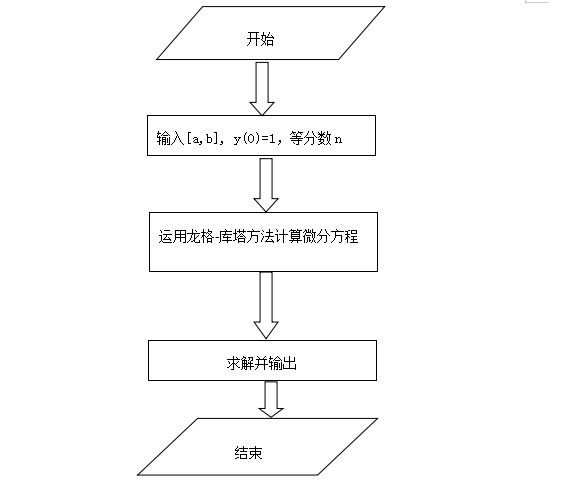


图 3-1 经典四阶龙格库塔法解一阶微分方程组算法流程图

3.3 算法程序调试

求解初值问题：



取初值，区间，。

3.3.1 C++程序运行

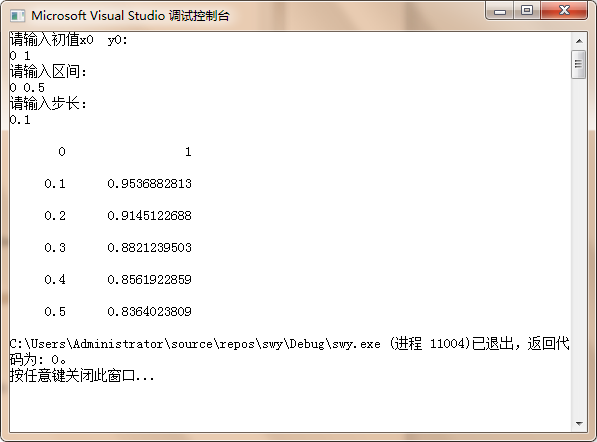


图3-2 经典四阶龙格库塔法解一阶微分方程组算法C++程序运行图

3.3.2 Matlab程序运行

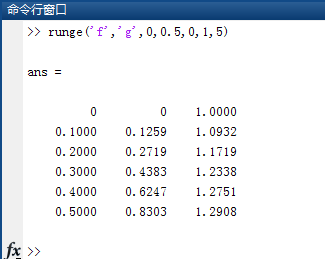


图3-3 经典四阶龙格库塔法解一阶微分方程组算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果存在误差。

4 三次样条插值算法

4.1 算法说明

设函数是区间上的二次连续可微函数，在区间上给出一个划分

## 

## 如果函数满足条件

## （1）；

## （2）在每个小区间上是不超过三次的多项式；

## （3）在开区间上有连续的二阶的导数，则称为区间对应于划分的三次样条函数。

## 对于待定系数，即个未知系数，而插值条件为个，还缺两个，因此须给出两个条件成为边界条件，有一下三类：

## 第一类：已知两端点的一阶导数：

 (4-1)

## 第二类：已知两端点二阶导数

 (4-2)

## 第三类：周期边界条件

 (4-3)

4.2 算法流程图

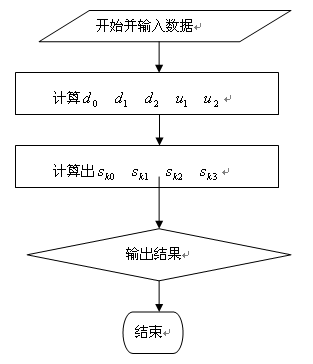


图4-1 三次样条插值算法流程图

4.3 算法程序调试

求满足下列数据的三阶样条插值，三阶样条插值算法数据表如表4-1所示：

表4-1 三次样条插值算法数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | -1 | 0 | 1 | 2 |
|  | -1 | 0 | 1 | 0 |
|  | 0 |  |  | 1 |

4.3.1 C++程序运行

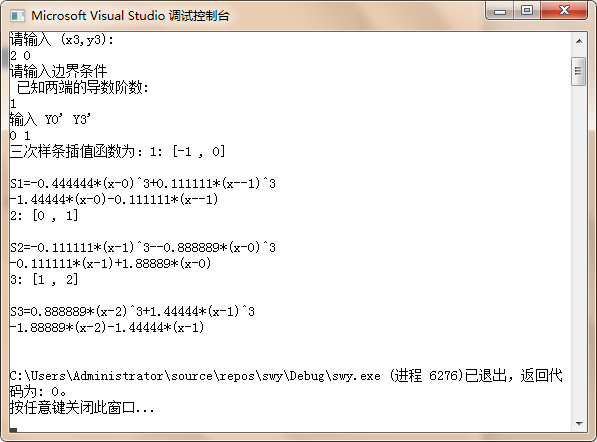


图4-2 三次样条插值算法C++程序运行图

4.3.2 Matlab程序运行

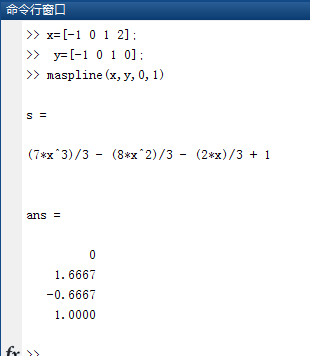


图4-3 三次样条插值算法Matlab程序运行图

4.3.3 三次样条曲线图像

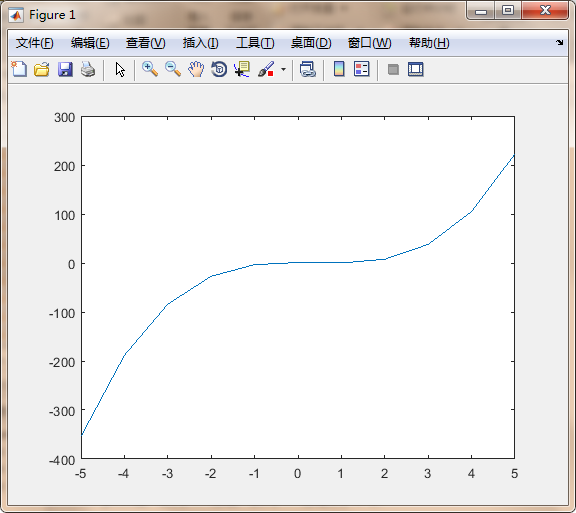


图4-4 三次样条插值算法计算所得曲线图

经检验知：在C++中的运行结果与在MATLAB中的运行存在误差。

5 龙贝格求积分算法

5.1 算法说明

在梯形公式、辛普森公式和柯特斯公式之间的关系的基础上，构造出一种加速计算积分的方法。 作为一种外推算法，它在不增加计算量的前提下提高了误差的精度。在等距基点的情况下，用计算机计算积分值通常都采用把[区间](https://baike.so.com/doc/6615804-6829597.html" \t "_blank)逐次分半的方法进行。

在整个区间上，令构造梯形值序列，求出

 (5-1)

把区间等分，每个小区间长度为，再应用复化梯形公式求出

 (5-2)

把区间等分，每个小区间长度为，则

 (5-3)

类似地再将每个小区间分半。一般地，每次总是在前一次的基础上再将小区间分半，然后利用递推公式进行计算，直至相邻两个值之差小于允许误差为止。

5.2 算法流程图

输入a,b,ep

计算

是

输出结果

图5-1 龙贝格求积分算法流程图

5.3 算法程序调试

计算定积分,上下限为[0,1]的近似值。

5.3.1 C++程序运行



图5-2 龙贝格求积分算法C++程序运行图

5.3.2 Matlab程序运行

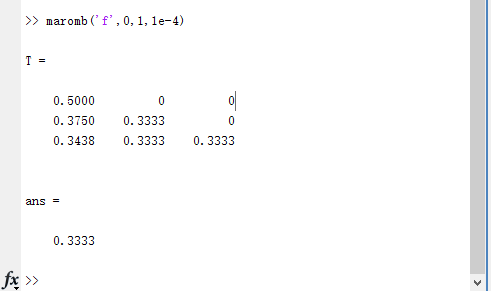


图5-3 龙贝格求积分算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果一致。

6 M次多项式曲线拟合算法

6.1 算法说明

根据给出点，找到：，计算

 (6-1)

列出方程组：

 (6-2)

求解。

6.2 算法流程图

输入点位置

高斯消去法求系数

输出系数

图6-1 M次多项式曲线拟合算法流程图

6.3 算法程序调试

已知：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | -3 | -1 | 1 | 3 |
|  | 15 | 5 | 1 | 5 |

表6-2 M次多项式拟合数据表

求拟合曲线。

6.3.1 C++程序运行

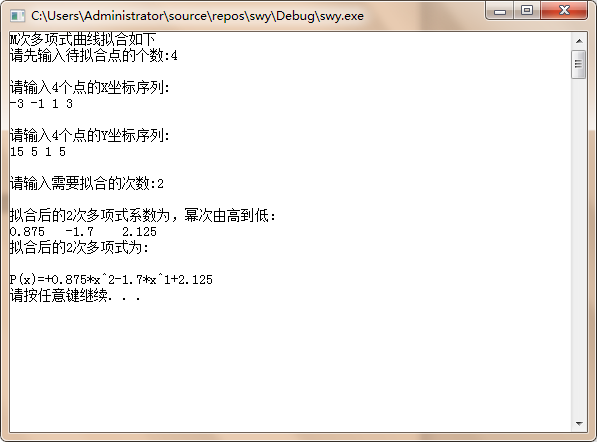


图6-2 M次多项式曲线拟合算法C++程序运行图

6.3.2 Matlab程序运行

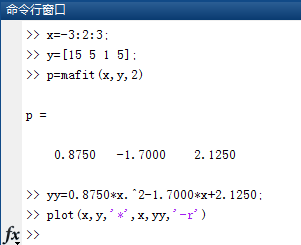


图6-3 M次多项式曲线拟合Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果一致。

7 复化辛普森公式求积分算法

7.1 算法说明

将积分区间分成等分，分点,其中，在每个小区间上用辛普森公式，则可得到复化辛普森公式

 (7-1)

7.2 算法流程图

定义f(x)

计算：

计算：



图7-1 复化辛普森公式求积分算法流程图

7.3 算法程序调试

计算函数，其中。

7.3.1 C++程序运行

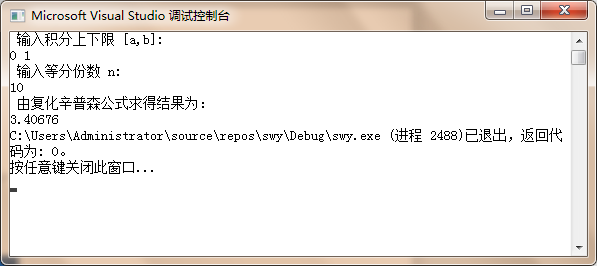


图7-2 复化辛普森公式求积分算法C++程序运行图

7.3.2 Matlab程序运行

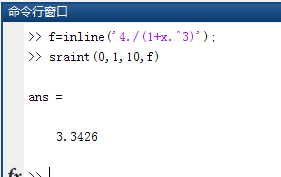


图7-3 复化辛普森公式求积分算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果存在误差。

8 拉格朗日插值算法

8.1 算法说明

有个点的次数最高为的多项式的构造方法，它具有的形式，其中是基于节点

 (8-1)

的拉格朗日系数多项式。

对于每个固定的，拉格朗日系数多项式具有性质为：

 (8-2)

8.2 算法流程图

输入点的个数与坐标

代入求解

输出结果

图8-1 拉格朗日插值算法流程图

8.3 算法程序调试

已知：

表8-1 拉格朗日插值算法数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 |
| y | 3 | 5 | 10 | 15 |

求f(1.5)。

8.3.1 C++程序运行

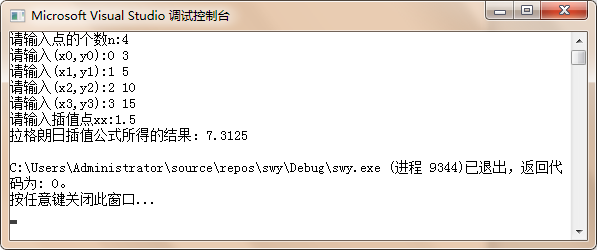


图8-2 拉格朗日插值算法C++程序运行图

8.3.2 Matlab程序运行

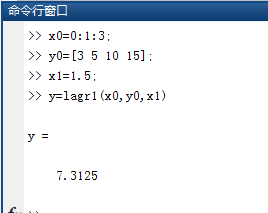


图8-3 拉格朗日插值算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果一致。

9 二分法解非线性方程算法

9.1 算法说明

是起始区间，是中点。

是第二个区间，它包含零点r，同时是中点，区间的宽度范围是的一半。

得到第n个区间（包含r，并有中点）后，可构造出，它也包括r，宽度范围是的一半。

9.2 算法流程图

输入隔根区间[a,b]，设定精度ep

置x:=(a+b)/2

f(x)=0

若f(a)·f(x)<0,置b:=x；

否则，置a:=x

置x=(a+b)/2

|b-a|<ep

输出x

否

是

是

是

图9-1 二分法解非线性方程算法流程图

9.3 算法程序调试

求解非线性方程的根（取）。

9.3.1 C++程序运行

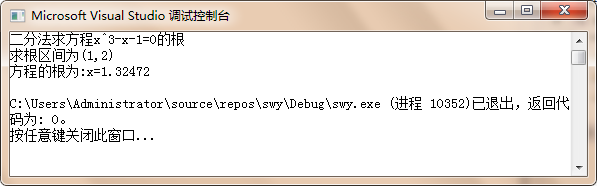


图9-2 二分法解非线性方程算法C++程序运行图

9.3.2 Matlab程序运行

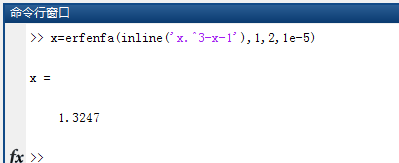


图9-3 二分法解非线性方程算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果一致。

10 雅可比迭代解线性方程组算法

10.1 算法说明

对线性方程组，设，可以得到

 (10-1)

即

 (10-2)

该公式即为雅克比迭代法。

10.2 算法流程图

输入初始数据

输出结果

是

图10-1 雅可比迭代解线性方程组算法流程图

10.3 算法程序调试

解线性方程组，取初值性，精度要求。

****

10.3.1 C++程序运行

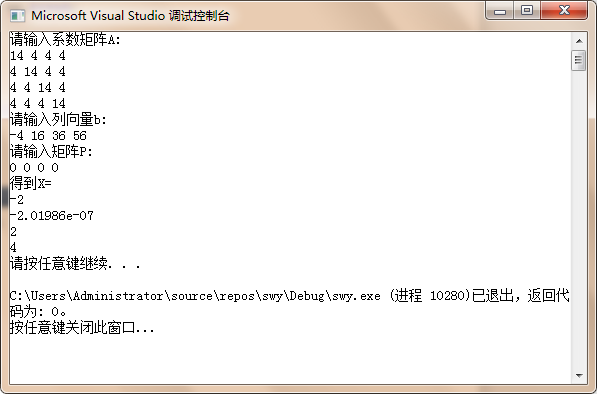


图10-2 雅可比迭代解线性方程组算法C++程序运行图

10.3.2 Matlab程序运行

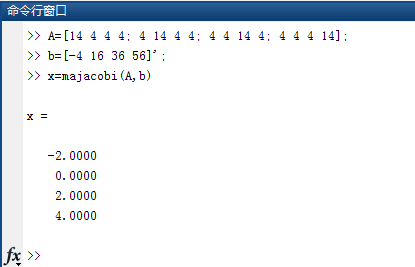


图10-3 雅可比迭代解线性方程组算法Matlab程序运行图

经检验知：在C++中的运行结果与在MATLAB中的运行结果基本一致。

# 设计体会及改进意见

在这次课程设计试验中，学习了数值计算中许多经典算法。并且要求我们用C++的知识进行求解，通过不断地学习，我顺利的完成了此次实验，期间我发现课本上的内容只有在运用时才能知道自己究竟在哪有问题，知道了自己努力的方向。

此次的课设我深深的感受到了自己的问题，在c++的很多知识我都有所欠缺，我需要不断地努力，完善自己的知识体系，让自己不断地进步，保持着一颗求学的心，有不懂的地方就要及时的请教。

在此次的一周的时间内，我和同学们，舍友们都进行过讨论，在函数的调用，定义，递归等方面都有所进步，明显感觉到自己受益匪浅，在这10个程序中，要考虑的不仅仅是后端的代码，也要考虑用户友好度的问题，将界面美化才能有更好的推广性和可使用性。

希望在今后的日子里，我能不断地学习，像在这次的时间内一样，将自己的思维不断地拓宽，不断的进步。

# 参 考 文 献

[1] 谭浩强编.C++程序设计[M],北京:清华大学出版社,2004.5

[2] 马昌凤,林伟川编著.现代数值计算方法(MATLAB版)[M]，北京:科学出版社,2008.6

[3] 林成森编著.数值计算方法[M],北京：科学出版社

[4] 丁丽娟，程杞元编著.数值计算方法[M],北京：北京理工大学出版社

# 附 录

附录1 高斯列主元法解线性方程组

C++ 源码

#include<iostream>

using namespace std;

const int n = 4;

void guass(double a[n][n],double b[n]) {

double m;

for (int i = 0;i<n ; i++){

for (int j = 1; j < n-i ; j++){

m = a[i + j][i] / a[i][i];

for (int k = 0; k < n; k++){

a[i + j][k] = a[i + j][k] - m \* a[i][k];

}

b[i + j] = b[i + j] - m \* b[i];

}

}

double x[n] = { 0,0,0,0 };

double k = 0;

for (int i = 4; i > 0; i --) {

k = b[i - 1];

for (int j = 0; j <n-i; j++) {

k = k - x[n-j-1] \* a[i-1][n-j-1];

}

x[i-1] = k / a[i-1][i-1];

}

cout<<"方程组解为："<<endl;

for (int i = 0; i < 4; i++) {

cout << "x" << i + 1 << "=" << x[i] << endl;

}

}

int main() {

double a[n][n], b[n];

cout << "请输入系数矩阵A:" << endl;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

cin >> a[i][j];

cout << "请输入列向量b:" << endl;

for (int i = 0; i < n; i++)

cin >> b[i];

guass(a, b);

return 0;

}

Mtalab 代码

function x = magauss(A,b,flag)

if nargin<3,flag = 0;end

n = length(b);

for k=1:(n-1)

[ap,p] = max(abs(A(k:n,k)));

p = p+k-1;

if p > k

t = A(k,:); A(k,:)=A(p,:); A(p,:) = t;

t = b(k);b(k) = b(p);b(p) = t;

end

m = A(k+1:n,k)/A(k,k);

A(k+1:n,k+1:n)= A(k+1:n,k+1:n)-m\*A(k,k+1:n);

b(k+1:n)=b(k+1:n)-m\*b(k);

A(k+1:n,k)=zeros(n-k,1);

if flag~=0,Ab=[A,b];end

end

x = zeros(n,1);

x(n) = b(n)/A(n,n);

for k = n-1:-1:1

x(k)=(b(k)-A(k,k+1:n)\*x(k+1:n))/A(k,k);

end

附录2 牛顿法解非线性方程组

C++ 源码

#include<iostream>

#include<cmath>

#define N 2

#define ep 0.0001

#define max 100

using namespace std;

const int N2 = 2 \* N;

int main(){

void ff(double xx[N], double yy[N]);

void ffjacobi(double xx[N], double yy[N][N]);

void inv\_jacobi(double yy[N][N], double inv[N][N]);

void newdun(double x0[N], double inv[N][N], double y0[N], double x1[N]);

double x0[N] = { 2.0,0.25 }, y0[N], jacobi[N][N], invjacobi[N][N], x1[N], fanshu;

int i, nargin = 0;

cout << "初始解向量为：" << endl;

for (i = 0; i < N; i++)

cout << x0[i] << " "<<endl;

do{

nargin = nargin + 1;

ff(x0, y0);

ffjacobi(x0, jacobi);

inv\_jacobi(jacobi, invjacobi);

newdun(x0, invjacobi, y0, x1);

fanshu = 0;

for (i = 0; i < N; i++)

fanshu = fanshu + fabs(x1[i] - x0[i]);

if (fanshu < ep)

break;

for (i = 0; i < N; i++)

x0[i] = x1[i];

} while (nargin < max);

cout << "近似解向量：" << endl;

for (i = 0; i < N; i++)

cout << x1[i] << " " << endl;

return 0;

}

void ff(double xx[N], double yy[N]) {

double x, y;

int i;

x = xx[0];

y = xx[1];

yy[0] = x \* x - 2 \* x - y + 0.5;

yy[1] = x \* x + 4 \* y \* y - 4;

}

void ffjacobi(double xx[N], double yy[N][N]){

double x, y;

int i, j;

x = xx[0];

y = xx[1];

yy[0][0] = 2 \* x - 2;

yy[0][1] = -1;

yy[1][0] = 2 \* x;

yy[1][1] = 8 \* y;

}

void inv\_jacobi(double yy[N][N], double inv[N][N]){

float aug[N][N2], L;

int i, j, k;

for (i = 0; i < N; i++){

for (j = 0; j < N; j++)

aug[i][j] = yy[i][j];

for (j = N; j < N2; j++)

if (j == i + N) aug[i][j] = 1;

else aug[i][j] = 0;

}

for (i = 0; i < N; i++){

for (k = i + 1; k < N; k++){

L = -aug[k][i] / aug[i][i];

for (j = i; j < N2; j++)

aug[k][j] = aug[k][j] + L \* aug[i][j];

}

}

for (i = N - 1; i > 0; i--){

for (k = i - 1; k >= 0; k--){

L = -aug[k][i] / aug[i][i];

for (j = N2 - 1; j >= 0; j--)

aug[k][j] = aug[k][j] + L \* aug[i][j];

}

}

for (i = N - 1; i >= 0; i--)

for (j = N2 - 1; j >= 0; j--)

aug[i][j] = aug[i][j] / aug[i][i];

for (i = 0; i < N; i++){

for (j = N; j < N2; j++)

inv[i][j - N] = aug[i][j];

}

}

void newdun(double x0[N], double inv[N][N], double y0[N], double x1[N]){

int i, j;

double sum = 0;

for (i = 0; i < N; i++){

sum = 0;

for (j = 0; j < N; j++)

sum = sum + inv[i][j] \* y0[j];

x1[i] = x0[i] - sum;

}

}

Mtalab 代码

function X =zu\_newton(fun,dfun,x0,ep,N )

k=0;

while k<N

F=feval(fun,x0);

J=feval(dfun,x0);

deltaX=-J \ F';

X=x0'+deltaX;

k=k+1;

if norm(deltaX,inf)<ep

disp([' k= ',num2str(k)]);

break;

elseif k==N

warning('\*\*\*\*\*\*\*\*\*\*\*\*');

else

x0=X';

end

end

end

附录3 经典四阶龙哥库塔法解一阶微分方程组

C++ 源码

#include <iostream>

#include <iomanip>

using namespace std;

double Runge\_Kuta(double(\*f)(double x, double y), double x0, double y0, double xn, int step){

double k1, k2, k3, k4, result;

double h = (xn - x0) / step;

if (step <= 0)

return(y0);

if (step == 1){

k1 = f(x0, y0);

k2 = f(x0 + h / 2, y0 + h \* k1 / 2);

k3 = f(x0 + h / 2, y0 + h \* k2 / 2);

k4 = f(x0 + h, y0 + h \* k3);

result = y0 + h \* (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;

}else{

double x1, y1;

x1 = xn - h;

y1 = Runge\_Kuta(f, x0, y0, xn - h, step - 1);

k1 = f(x1, y1);

k2 = f(x1 + h / 2, y1 + h \* k1 / 2);

k3 = f(x1 + h / 2, y1 + h \* k2 / 2);

k4 = f(x1 + h, y1 + h \* k3);

result = y1 + h \* (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;

}

return(result);

}

int main(){

double f(double x, double y); double x0, y0;

double a, b;

cout << "请输入初值x0 y0: "<<endl;

cin >> x0 >> y0;

cout << "请输入区间："<<endl;

cin >> a >> b;

double x, step;

cout << "请输入步长："<<endl;

cin >> step;

cout.precision(10);

for (int i = 0; i <= (b - a) / step; i++){

x = x0 + i \* step;

cout << endl << setw(8) << x << setw(18) << Runge\_Kuta(f, x0, y0, x, i) << endl;

}

return(0);

}

double f(double x, double y){

double r;

r = (x - y) / 2;

return (r);

}

Mtalab 代码

function R = runge(f,g,a,b,xa,ya,N)

f=@(t,x,y)(x+y);

g=@(t,x,y)(y-2\*x/y);

h=(b-a)/N;

T=zeros(1,N+1);

X=zeros(1,N+1);

Y=zeros(1,N+1);

T=a:h:b;

X(1)=xa;

Y(1)=ya;

for j=1:N

f1=feval(f,T(j),X(j),Y(j));

g1=feval(g,T(j),X(j),Y(j));

f2=feval(f,T(j)+h/2,X(j)+h/2\*f1,Y(j)+g1/2);

g2=feval(g,T(j)+h/2,X(j)+h/2\*f1,Y(j)+h/2\*g1);

f3=feval(f,T(j)+h/2,X(j)+h/2\*f2,Y(j)+h\*g2/2);

g3=feval(g,T(j)+h/2,X(j)+h/2\*f2,Y(j)+h/2\*g2);

f4=feval(f,T(j)+h,X(j)+h\*f3,Y(j)+h\*g3);

g4=feval(g,T(j)+h,X(j)+h\*f3,Y(j)+h\*g3);

X(j+1)=X(j)+h\*(f1+2\*f2+2\*f3+f4)/6;

Y(j+1)=Y(j)+h\*(g1+2\*g2+2\*g3+g4)/6;

R=[T' X' Y'];

end

end

附录4 三次样条插值算法

C++ 源码

#include<iostream>

#include<iomanip>

using namespace std;

const int max = 50;

float x[max], y[max], h[max];

float c[max], a[max], fxym[max];

float f(int x1, int x2, int x3){

float a=(y[x3]-y[x2])/(x[x3]-x[x2]);

float b=(y[x2]-y[x1])/(x[x2]-x[x1]);

return (a-b)/(x[x3]-x[x1]);

}

void cal\_m(int n){

float B[max];

B[0]=c[0]/2;

for(int i=1;i<n;i++)

B[i]=c[i]/(2-a[i]\*B[i-1]);

fxym[0]=fxym[0]/2;

for(int i=1;i<=n;i++)

fxym[i]=(fxym[i]-a[i]\*fxym[i-1])/(2-a[i]\*B[i-1]);

for(int i=n-1;i>=0;i--)

fxym[i]=fxym[i]-B[i]\*fxym[i+1];

}

void printout(int n){

cout<<setprecision(6);

for(int i=0; i<n; i++){

cout<<i+1<<": ["<<x[i]<<" , "<<x[i+1]<<"]\n"<<endl;

cout<<"S"<<i+1<<"=";

float t=fxym[i]/(6\*h[i]);

if(t>0)

cout<<-t<<"\*(x-"<<x[i+1]<<")^3";

else

cout<<-t<<"\*(x-"<<x[i+1]<<")^3";

t=fxym[i+1]/(6\*h[i]);

if(t > 0)

cout<<"+"<<t<<"\*(x-"<<x[i]<<")^3";

else

cout<<"-"<<t<<"\*(x-"<<x[i]<<")^3";

cout<<endl;

t=(y[i]-fxym[i]\*h[i]\*h[i]/6)/h[i];

if(t>0)

cout<<"-"<<t<<"\*(x-"<<x[i+1]<<")";

else

cout<<"-"<<-t<<"\*(x-"<<x[i+1]<<")";

t=(y[i+1]-fxym[i+1]\*h[i]\*h[i]/6)/h[i];

if(t>0)

cout<<"+"<<t<<"\*(x-"<<x[i]<<")";

else

cout<<"-"<<-t<<"\*(x-"<<x[i]<<")";

cout<<endl;

}

cout<<endl;

}

int main(){

int n=3,i;

char ch;

for(i=0;i<=n;i++){

cout<<"请输入 (x"<<i<<",y"<<i<<"):"<<endl;

cin>>x[i]>>y[i];

}

for(i=0;i<n;i++)

h[i]=x[i+1]-x[i];

cout<<"请输入边界条件\n 已知两端的导数阶数: "<<endl;

int t;

float f0, f1;

cin>>t;

switch(t){

case 1:cout<<"输入 Y0\' Y"<<n<<"\'"<<endl;

cin>>f0>>f1;

c[0]=1;a[n]=1;

fxym[0]=6\*((y[1]-y[0])/(x[1]-x[0])-f0)/h[0];

fxym[n]=6\*(f1-(y[n]-y[n-1])/(x[n]-x[n-1]))/h[n-1];

break;

case 2:cout<<"输入 Y0\" Y"<<n<<endl;

cin>>f0>>f1;

c[0]=a[n]=0;

fxym[0]=2\*f0;

fxym[n]=2\*f1;

break;

default:cout<<"不可用\n";

};

for(i=1;i<n;i++)

fxym[i]=6\*f(i-1,i,i+1);

for(i=1;i<n;i++){

a[i]=h[i-1]/(h[i]+h[i-1]);

c[i]=1-a[i];

}

a[n]=h[n-1]/(h[n-1]+h[n]);

cal\_m(n);

cout<<"三次样条插值函数为：";

printout(n);

return 0;

}

Mtalab 代码

function m=maspline(x,y,dy0,dyn)

n=length(x)-1;

h=diff(x);

lambda=h(2:n)./(h(1:n-1)+h(2:n));

mu=1-lambda;

theta=3\*(lambda.\*diff(y(1:n))./h(1:n-1)+mu.\*diff(y(2:n+1))./h(2:n));

theta(1)=theta(1)-lambda(1)\*dy0;

theta(n-1)=theta(n-1)-lambda(n-1)\*dyn;

dy=machase(lambda,2\*ones(1:n-1),mu,theta);

m=[dy0;dy;dyn];

s=zeros(length(x));

syms x

for i=1:n

s=alpha0(x)\*y(i)+alpha1(x)\*y(i+1)...

+h(i)\*beta0(x)\*m(i,:)+h(i)\*beta1(x)\*m(i+1,:);

end

s

end

function x = machase(a,b,c,d)

n = length(a);

for k = 2:n

b(k) = b(k)-a(k)/b(k-1)\*c(k-1);

d(k) = d(k)-a(k)/b(k-1)\*d(k-1);

end

x(n) = d(n)/b(n);

for k = n-1:-1:1

x(k) = (d(k)-c(k)\*x(k+1))/b(k);

end

x = x(:);

end

function y = alpha0(x)

y = 2\*x.^3 - 3\*x.^2+1;

end

function y = alpha1(x)

y = 2\*x.^3 - 3\*x.^2+1;

end

function y = beta0(x)

y = x.^3 -2\*x.^2+x;

end

function y = beta1(x)

y = x.^3 - x.^2;

end

附录5 龙贝格求积分算法

C++ 源码

#include<iostream>

#include<stdio.h>

#include<math.h>

double f(double x){

return x \* x;

}

int main(){

int M = 1, n = 0, p = 0, K = 0, i = 0, j = 0, J = 0;

double h = 0.0, a = 0.0, b = 0.0, err = 1.0, quad = 0.0, s = 0.0, x = 0.0, tol = 0.0;

double R[30][30] = { 0 };

a = 0;

b = 1;

h = b - a;

n = 4;

tol = 0.00001;

printf("求解函数y=x^2在(0,1)上的龙贝格矩阵\n");

printf("龙贝格矩阵最大行数为%d\n误差限为%lf\n", n, tol);

R[0][0] = h \* (f(a) + f(b)) / 2;

while (((err > tol) && (J < n)) || (J < 4)){

J = J + 1;

h = h / 2;

s = 0;

for (p = 1; p <= M; p++){

x = a + h \* (2 \* p - 1);

s = s + f(x);

}

R[J][0] = R[J - 1][0] / 2 + h \* s;

M = 2 \* M;

for (K = 1; K <= J; K++){

R[J][K] = R[J][K - 1] + (R[J][K - 1] - R[J - 1][K - 1]) / (4 \* K - 1);

}

err = fabs(R[J - 1][J - 1] - R[J][K]);

}

quad = R[J][J];

printf("\n龙贝格矩阵为:\n");

for (i = 0; i < (J + 1); i++){

for (j = 0; j < (J + 1); j++){

printf("%.5lf ", R[i][j]);

}

printf("\n");

}

printf("\n积分值为:quad=%lf", quad);

printf("\n误差估计为:err=%lf", err);

printf("\n使用过的最小步长h=%lf\n", h);

system("pause:");

return 0;

}

Mtalab 代码

function s=maromb(fun,a,b,tol)

if nargin<4,tol=1e-4;end

i=1; j=1; h=b-a;

T(1,1)=h\*(feval(fun,a)+feval(fun,b))/2;

T(i+1,j)=T(i,j)/2+sum(feval(fun,a+h/2:h:b-h/2))\*h/2;

T(i+1,j+1)=(4^j\*T(i+1,j)-T(i,j))/(4^j-1);

while(abs(T(i+1,i+1)-T(i,i))>tol)

i=i+1; h=h/2;

T(i+1,i)=T(i,1)/2+sum(feval(fun,a+h/2:h:b-h/2))\*h/2;

for j=1:i

T(i+1,j+1)=(4^j\*T(i+1,j)-T(i,j))/(4^j-1);

end

end

T

s=T(i+1,j+1);

end

附录6 M次多项式曲线拟合

C++ 源码

#include<iostream>

#include<cmath>

#define N 20

using namespace std;

void inv(double X[N][N], int n, double E[N][N]){

int i, j, k;

double temp = 0;

for (i = 0; i < N; i++){

for (j = 0; j < N; j++)

if (i == j)

E[i][j] = 1;

}

for (i = 0; i < n - 1; i++){

temp = X[i][i];

for (j = 0; j < n; j++){

X[i][j] = X[i][j] / temp;

E[i][j] = E[i][j] / temp;

}

for (k = 0; k < n; k++){

if (k == i)

continue;

temp = -X[i][i] \* X[k][i];

for (j = 0; j < n; j++){

X[k][j] = X[k][j] + temp \* X[i][j];

E[k][j] = E[k][j] + temp \* E[i][j];

}

}

}

}

int main(){

int n, M, i, j, k;

double X[N] = { 0 }, Y[N] = { 0 }, F[N][N] = { 0 }, B[N] = { 0 };

double A[N][N] = { 0 }, BF[N][N] = { 0 }, C[N] = { 0 }, E[N][N] = { 0 };

cout << "M次多项式曲线拟合如下" << endl;

cout << "请先输入待拟合点的个数:";

cin >> n;

cout << "\n请输入" << n << "个点的X坐标序列:\n";

for (i = 0; i < n; i++)

cin >> X[i];

cout << "\n请输入" << n << "个点的Y坐标序列:\n";

for (i = 0; i < n; i++)

cin >> Y[i];

cout << "\n请输入需要拟合的次数:";

cin >> M;

for (i = 0; i < n; i++)

for (k = 1; k <= M + 1; k++)

F[i][k - 1] = pow(X[i], k - 1);

for (i = 0; i < n; i++){

for (j = 0; j < M + 1; j++){

BF[j][i] = F[i][j];

}

}

for (i = 0; i < M + 1; i++)

for (j = 0; j < M + 1; j++)

for (k = 0; k < n; k++)

A[i][j] += BF[i][k] \* F[k][j];

for (i = 0; i < M + 1; i++)

for (j = 0; j < n; j++)

B[i] += BF[i][j] \* Y[j];

inv(A, n, E);

for (i = 0; i < M + 1; i++)

for (j = 0; j < n; j++)

C[i] += E[i][j] \* B[j];

cout << "\n拟合后的" << M << "次多项式系数为，幂次由高到低：\n";

for (i = M; i >= 0; i--)

cout << C[i] << "\t";

cout << "\n拟合后的" << M << "次多项式为:\n";

cout << "\nP(x)=";

for (i = M; i >= 0; i--){

if (i == 0){

if (C[i] >= 0)

cout << "+" << C[i];

else

cout << C[i];

}else{

if (C[i] >= 0)

cout << "+" << C[i] << "\*x^" << i;

else

cout << C[i] << "\*x^" << i;

}

}

cout << endl;

system("pause");

return 0;

}

Mtalab 代码

function p=mafit(x,y,m)

format short;

A=zeros(m+1,m+1);

for i=0:m

for j=0:m

A(i+1,j+1)=sum(x.^(i+j));

end

b(i+1)=sum(x.^i.\*y);

end

a=A\b';

p=fliplr(a');

end

附录7 复化辛普森公式求积分

C++ 源码

#include <iostream>

#include <cmath>

using namespace std;

double f(double x);

double simpse(double a,double b,double n);

int main(){

double a,b,n,s;

cout << " 输入积分上下限 [a,b]:" << endl;

cin >> a >> b;

cout << " 输入等分份数 n:" << endl;

cin >> n;

cout << " 由复化辛普森公式求得结果为： "<<endl;

s = simpse(a, b, n);

cout << s;

return 0;

}

double f(double x){

double y;

y = 4 / (1 + x \* x \* x);

return (y);

}

double simpse(double a, double b, double n){

double h = (b - a) / n;

double x, s1 = 0, s2 = 0;

for (int k = 1; k < n; k++){

x = a + h \* k;

s1 = s1 + f(x);

}

for (int k = 0; k < n; k++){

x = a + h \* (k + 1 / 2);

s2 = s2 + f(x);

}

double s = (f(a) + 2 \* s1 + 4 \* s2 + f(b)) \* h / 6;

return s;

}

Mtalab 代码

function y = sraint(a,b,n,f)

h = (b - a) / n;

x = linspace(a,b,2\*n+1);

y1 = feval(f,x);

y1(2:2:2\*n) = 4 \* y1(2:2:2\*n);

y1(3:2:2\*n-1) = 2 \* y1(3:2:2\*n-1);

y = h / 6 \* sum(y1);

end

附录8 拉格朗日插值法

C++ 源码

#include<iostream>

using namespace std;

int main(){

double f(double xx, int k, double x[], double y[], int n);

double sum = 0;

int n;

cout << "请输入点的个数n:";

cin >> n;

double\* x = (double\*)malloc(n \* sizeof(double));

double\* y = (double\*)malloc(n \* sizeof(double));

double xx;

int i;

for (i = 0; i < n; i++){

cout << "请输入(x" << i << ",y" << i << "):";

cin >> x[i] >> y[i];

}

cout << "请输入插值点xx:";

cin >> xx;

for (i = 0; i < n; i++){

sum = sum + f(xx, i, x, y, n) \* y[i];

}

cout << "拉格朗日插值公式所得的结果：" << sum << endl;

return 0;

}

double f(double xx, int k, double x[], double y[], int n) {

int i;

double t = 1;

double s;

for (i = 0; i < n; i++){

if (i == k) continue;

else s = (xx - x[i]) / (x[k] - x[i]);

t = t \* s;

}

return t;

}

Mtalab 代码

function y = lagr1(x0,y0,x)

n = length(x0);

m = length(x);

for i = 1 : m

z = x(i);

s = 0;

for k = 1 : n

p = 1;

for j = 1 : n

if j~=k

p = p \* (z - x0(j))/(x0(k) - x0(j));

end

end

s = p \* y0(k) + s;

end

y(i) = s;

end

end

附录9 二分法解非线性方程

C++ 源码

#include<iostream>

using namespace std;

#define ep 1e-5

int main(){

double f(double x);

double f1, f2, f3, a, b, c;

cout << "二分法求方程x^3-x-1=0的根" << endl;

cout << "求根区间为(1,2)" << endl;

a = 0, b = 5;

f1 = f(a);f2 = f(b);

while ((b - a) > ep){

c = (a + b) / 2;

f3 = f(c);

if (f3 == 0)

break;

else if (f3 \* f2 < 0){

a = c;f1 = f3;

}else{

b = c;f2 = f3;

}

}

cout << "方程的根为:x=" << c << endl;

}

double f(double x) {

return x \* x \* x - x - 1;

}

Mtalab 代码

function x=erfenfa(fun,a,b,ep)

x=(a+b)/2.0;

k=0;

while abs(feval(fun,x))>ep || (b-a>ep)

if feval(fun,x)\*feval(fun,a)<0

b=x;

else

a=x;

end

x=(a+b)/2.0;

k=k+1;

end

附录10 雅可比迭代解线性方程组

C++ 源码

#include <iostream>

#include <cmath>

using namespace std;

#define e 0.000001

#define n 4

#define M 100

int main(){

int i, j, k;

double err, norm, A[n][n], b[n], P[n], X[n];

cout << "请输入系数矩阵A:" << endl;

for (i = 0; i < n; i++)

for (j = 0; j < n; j++)

cin >> A[i][j];

cout << "请输入列向量b:" << endl;

for (i = 0; i < n; i++)

cin >> b[i];

cout << "请输入矩阵P:" << endl;

for (i = 0; i < n; i++)

cin >> P[i];

for (k = 0; k < M; k++){

for (j = 0; j < n; j++){

X[j] = b[j];

for (i = 0; i < n; i++)

X[j] = X[j] - A[j][i] \* P[i];

X[j] = (X[j] + A[j][j] \* P[j]) / A[j][j];

}

norm = 0;

for (i = 0; i < n; i++)

norm = norm + pow((X[i] - P[i]), 2);

norm = pow(norm, 0.5);

err = fabs(norm);

for (i = 0; i < n; i++)

P[i] = X[i];

if (err < e)

goto loop;

}

loop:cout << "得到X=" << endl;

for (i = 0; i < n; i++)

cout << X[i] << endl;

system("pause");

return 0;

}

Mtalab 代码

function x = majacobi(A,b,x0,ep,N)

n = length(b);

if nargin < 5, N = 500;

end

if nargin < 4, ep = 1e-6;

end

if nargin < 3, x0 = zeros(n,1);

end

x = zeros(n,1);

k = 0;

while k < N

for i = 1 : n

x(i) = (b(i) - A(i,[1 :(i - 1),(i + 1):n]) \* x0([1:(i - 1),(i + 1):n]))/A(i,i);

end

if norm(x - x0,inf) < ep,break;

end

k = k + 1;

x0 = x;

end

if k == N , Warning('已到达迭代次数上限');

end

end