****

中国地质大学（北京）

科学与工程计算

实验报告

学 院：信息工程学院

专 业：计算机科学与技术

班 级：10041912

学 号：1004192118

姓 名：陈达

联系方式：18269849385

邮 箱：2229859481@qq.com

指导老师：王玉柱

日 期：2021年5月23日

**摘 要**

**关键词：非线性方程，线性方程组，迭代，插值，拟合，积分，等距节点，复杂度分析**

实验一涉及到采用二分法、牛顿法、牛顿下山法、弦截法求解非线性方程组的形式，同时对精度以及算法优劣提出了对比要求，设计算法时需要注意统一结果以及比较方式。

实验二涉及到如何使用改进的Cholesky算法求解线性方程组，并要求设计计算\_n的有效位数变化情况，需要注意采用简便的形式进行位数求解。

实验三涉及到采用雅可比迭代法、SOR法、共轭梯度法求解线性方程组的问题，主要涉及到是否收敛以及判定条件方面的问题。

实验四涉及到应用拉格朗日插值法、牛顿插值法、三次样条插值法对函数进行插值的问题，同时对这几种方法的优略进行了对比，需要注意断点处的跳跃式波动现象。

实验五涉及到使用最小二乘拟合对函数进行拟合的问题，同时将其与拟合方程本身次数以及与插值法做对比，需要注意其原理，从原理出发进行对比分析。

实验六涉及到采用龙贝格积分法对给定函数进行数值积分的问题，并于其他的方法一同显示出来有所对比，值得注意循环套用变量时的定义方式。

最后对写出了我个人对本次实验经历的一些心得体会，以及我对这门课的一些看法和感激。

**目 录**

**[实 验 正 文 4](#_Toc6969)**

**[实验一、非线性方程求根 4](#_Toc30878)**

[一、实验内容： 4](#_Toc25538)

[二、 算法基本思想及复杂度分析： 4](#_Toc3278)

[三、 源程序及注释： 5](#_Toc13553)

[四、 运行输出结果： 7](#_Toc27303)

[五、 调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训： 9](#_Toc14165)

**[实验二、线性方程组的直接解法 10](#_Toc24397)**

[一、实验内容： 10](#_Toc30478)

[二、 算法基本思想及复杂度分析： 10](#_Toc5057)

[三、 源程序及注释： 11](#_Toc6384)

[四、 运行输出结果： 14](#_Toc8701)

[五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训： 14](#_Toc32189)

**[实验三、线性方程组的迭代解法 16](#_Toc951)**

[一、 实验内容： 16](#_Toc27718)

[二、 算法基本思想及复杂度分析： 16](#_Toc12435)

[三、 源程序及注释： 18](#_Toc24941)

[四、 运行输出结果： 25](#_Toc11915)

[五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训： 27](#_Toc23903)

**[实验四、函数插值 29](#_Toc11977)**

[一、 实验内容： 29](#_Toc32644)

[二、 算法基本思想及复杂度分析： 29](#_Toc25181)

[三、 源程序及注释： 30](#_Toc3820)

[四、 运行输出结果： 33](#_Toc19807)

[五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训： 34](#_Toc15184)

**[实验五、最小二乘拟合 35](#_Toc21645)**

[一、 实验内容： 35](#_Toc728)

[二、 算法基本思想及复杂度分析： 35](#_Toc24672)

[三、 源程序及注释： 36](#_Toc19264)

[四、 运行输出结果： 39](#_Toc2596)

[五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训： 40](#_Toc28959)

**[实验六、数值积分 41](#_Toc10919)**

[一、 实验内容： 41](#_Toc26196)

[二、 算法基本思想及复杂度分析： 41](#_Toc8048)

[三、 源程序及注释： 42](#_Toc227)

[四、 运行输出结果： 44](#_Toc673)

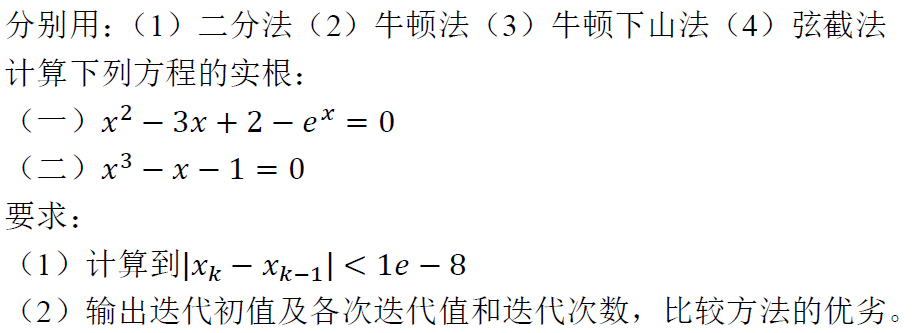
[五、 调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训： 44](#_Toc26066)

**[心 得 体 会 45](#_Toc13039)**

**实 验 正 文**

# 实验一、非线性方程求根

## 一、实验内容：



## 算法基本思想及复杂度分析：

按照要求，首先声明常量精度precise，之后定义func1和func2函数保存两个方程，之后按照题目要求，依次定义四个函数分别代表四种算法，首先根据计算，基本确定了两个方程解的大致区间，之后分别以该区间作为参数传入到四个函数中进行计算。

**二分法：**定义变量middle为x和y的中间值，循环带入函数，同时将f(middle)和f(x)之积与0比较，从而确定下一次循环middle应该赋给x还是y，循环直到||小于精度为止，输出迭代初值以及每次迭代值和次数。复杂度为O(logN)。

**牛顿法：**依次由于||即|f(x)/f’(x)|，则比较|f(x)/f’(x)|与精度的大小，每次让x减去f(x)/f’(x)，再次循环，输出迭代初值以及每次迭代值和次数，直到满足条件为止。复杂度为O(logN)

**牛顿下山法：**比较u\*|f(x)/f’(x)|与精度的大小，初始u=1，每次比较||与||,若前者大于等于后者，不满足下降条件，就让u减半，直到满足条件为止，让x减去u\*f(x)/f’(x)，再次循环，输出迭代初值以及每次迭代值和次数，直到满足条件为止。复杂度要优于牛顿法。

**弦截法：**运用变量tmp和x1存储和，从而达到使用来判断条件的目的，每次比较该式与精度的关系，大于精度则继续循环，同时按顺序将tmp和x1和x依次赋值达到传递的效果，输出迭代初值以及每次迭代值和次数，直到满足条件为止。速度在此次运算中略慢于牛顿迭代法。

## 源程序及注释：

#include <cstdio>

#include <cmath>

#include<iostream>

using namespace std;

const double precise=1e-8;//按照题目要求，确定精度为1e-8

double func1(double v,bool id){//题目给定的方程

if(id) return v\*v-3\*v+2-exp(v);//根据id来判断计算那个方程，方程1 x^2-3\*x+2-e^x=0

else return v\*v\*v-v-1;//方程2 x^3-x-1=0

}

double func2(double v,bool id){//给定方程的求导形式

if(id) return 2\*v-3-exp(v);//方程1的导数

else return 3\*v\*v-1;//方程2的导数

}

void dichotomy(double x,double y,bool id){

//二分法求解

double middle=(x+y)/2,tmp=x;//初始middle为中间值，tmp用来保留上次middle值，从而达到xk-xk-1判断精度

int i=0;

printf("-----------------二分法-------------------\n");

printf("初始值为:左边界点%.10lf,右边界点%.10lf\n",x,y);//设立初始边界点

while(fabs(middle-tmp)>precise){//当(xk-xk-1)大于精度或|f(middle)|大于精度时

func1(middle,id)\*func1(x,id)>0?x=middle:y=middle;//判断f(middle)与f(x)乘积是否为正，从而确立下一次区间

tmp=middle; //保留这次xk结果

middle=(x+y)/2;

i++;

printf("迭代次数为:%d,左边界点%.10lf,右边界点%.10lf\n",i,x,y);

}

printf("最终结果为%.10lf\n",middle);

}

void NewtonMethod(double x,bool id){

int i=0;

printf("-----------------牛顿法-------------------\n");

printf("初始值为%.10lf\n",x);//设立初始近似值

while(fabs(func1(x,id)/func2(x,id))>precise){//当函数内值即为(xk-xk-1),当其未到达精度要求时循环

x-=func1(x,id)/func2(x,id);//x减去该值

i++;

printf("迭代次数为:%d,此时x为%.10lf\n",i,x);

}

printf("最终结果为%.10lf\n",x);

}

void NewtonDownMethod(double x,bool id){

int i=0;

double x1,u=1;

printf("---------------牛顿下山法------------------\n");

printf("初始值为%.10lf\n",x);//设立初始值

while(u\*fabs(func1(x,id)/func2(x,id))>precise){ //当(xk-xk-1)大于精度时

u=1;

x1=x-u\*func1(x,id)/func2(x,id);//xk减去该值转变为下一次的xk+1

if(fabs(func1(x1,id))>=fabs(func1(x,id)))//当|f(xk+1)|>=|f(xk)|的时候要将u减半，使得迭代满足下降条件

u\*=0.5;

i++;

printf("迭代次数为:%d,此时x为%.10lf,x1为%.10lf,u为%.10lf\n",i,x,x1,u);

x=x1;

}

printf("最终结果为%.10lf\n",x);

}

void SecantMethod(double x,double x1,bool id){//给定初始的x，x1

//这里的计算逻辑是x存储xk-1，x1存储xk，tmp存储xk+1从而达成循环

int i=0;

double tmp;//存储xk+1，从而完成循环

printf("-----------------弦割法-------------------\n");

while(fabs(func1(x,id)\*(x1-x)/(func1(x1,id)-func1(x1,id)))>precise){//当未达到精度时

tmp=x1-func1(x1,id)\*(x1-x)/(func1(x1,id)-func1(x,id));//计算此时的xk+1

x=x1;//将xk赋值给xk-1

x1=tmp;//xk+1赋值给xk

i++;

printf("迭代次数为:%d,此时x为%.10lf\n",i,x);

}

printf("最终结果为%.10lf\n",x);

}

int main() {

dichotomy(0,1,1);dichotomy(1,2,0);//用二分法求两个方程的解，这里对于方程1初始区间为0到1，方程2初始区间为1到2

NewtonMethod(0,1);NewtonMethod(1,0);//用牛顿法求两个方程的解，这里初始近似值方程1取为0，方程2取为2

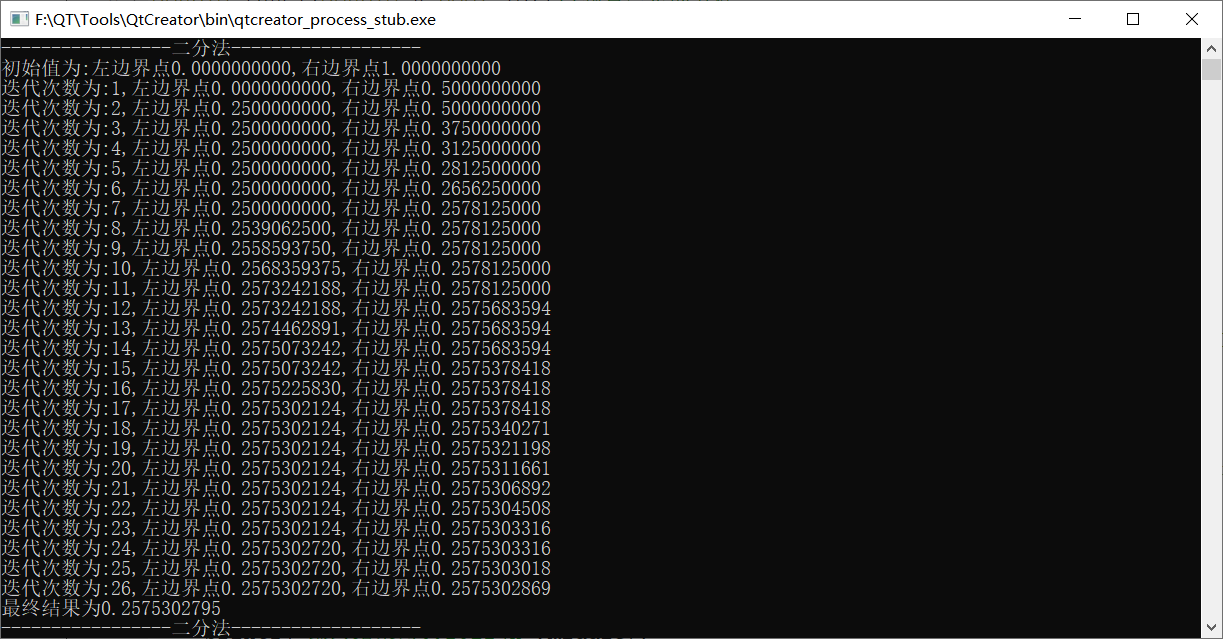
NewtonDownMethod(0,1);NewtonDownMethod(1,0);//用牛顿下山法求两个方程的解，方程1初始值取为0，方程2取为2

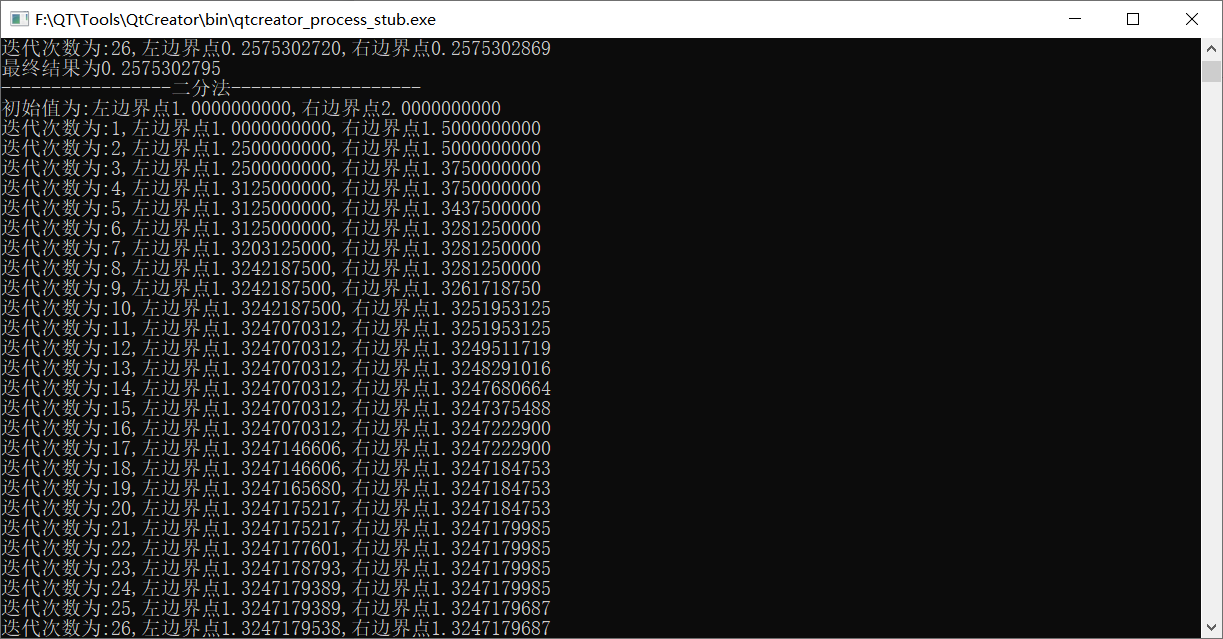
SecantMethod(0,1,1);SecantMethod(1,2,0);//用弦割法求两个方程的解，方程1的初始x，x1取为0和1，方程2取为1和2

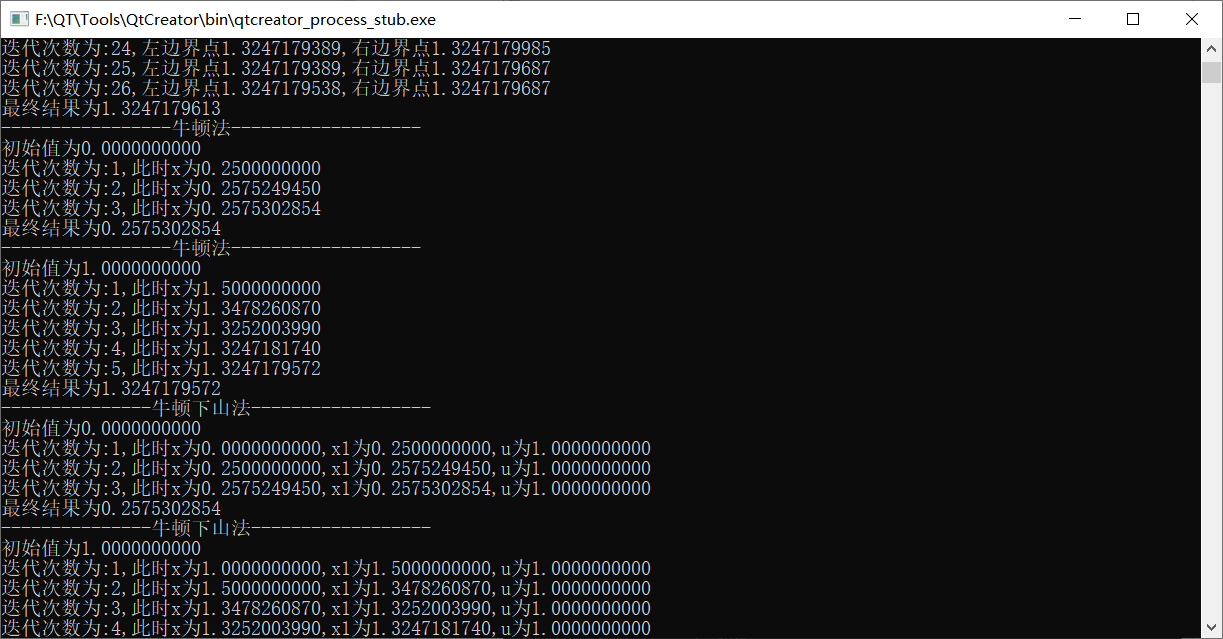
return 0;

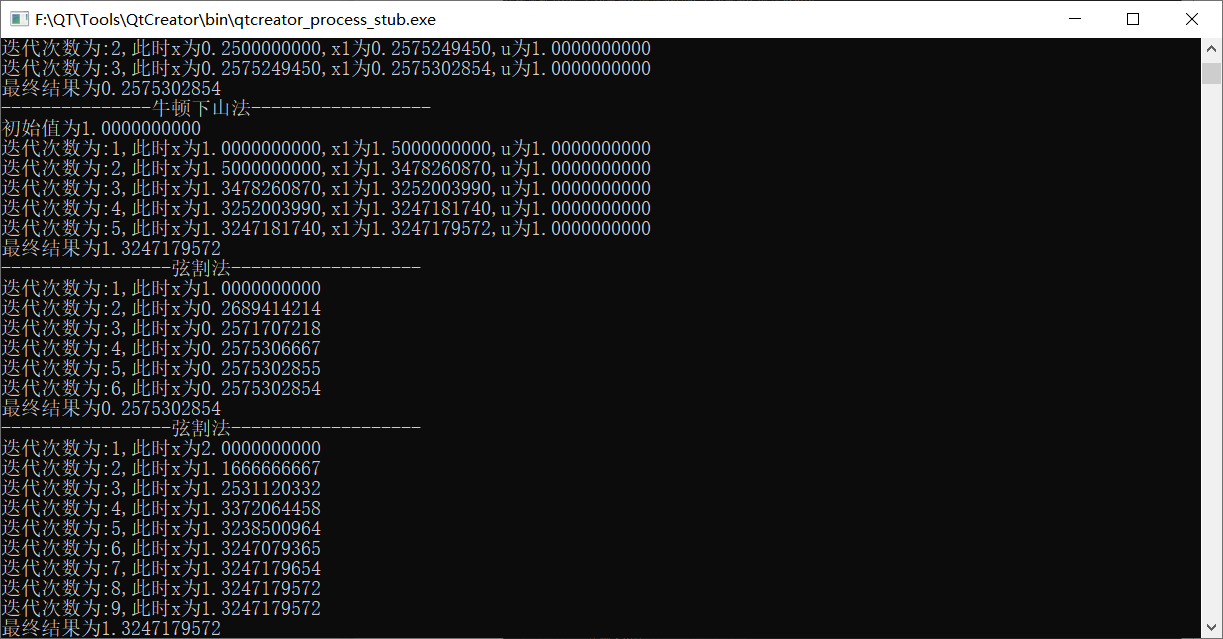
}

## 运行输出结果：









**优劣比较：**

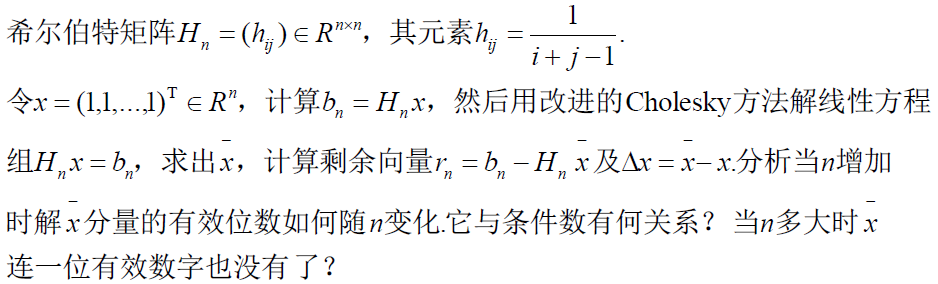
**从测试结果来看，牛顿下山法和牛顿法迭代次数要少于弦截法和二分法，牛顿迭代法虽然收敛高，但是需要计算导数，计算的复杂度要高，而且迭代法可能出现不收敛的情况，使得无法求解。弦截法慢于牛顿迭代法，可用差商取代牛顿迭代法中的导数从而降低了计算带来的复杂度。**

## 调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

开始由于未统一循环变量的类型，导致循环进入死循环。之后通过调试发现了这一问题，将int型tmp改变为double后解决了这个问题。一开始只输出前8位，但是按照精度要求的区间小于exp，位数要提高，之后将输出提高到前十位。之后在设计实现四种算法实现函数求解实根时，由于确立的判断条件不一致导致四种算法结果最后一位的精度不统一，通过确立为<exp来统一精度要求，同时由于二分法使用时，按照题目要求<exp要比|f(middle)|<exp慢，最终导致其精度并未像之后的四种算法那样一致。这也给我了一个教训，没有将区间统一，算法快慢的比较缺乏科学性，为了使得性能的比较具有一致性，需要统一将区间定为一样的标准，这也体现在解决比较方面的问题时需要统一标准。

# 实验二、线性方程组的直接解法

## 一、实验内容：



## 算法基本思想及复杂度分析：

**改进Cholesky算法：**

**求解步骤：**用Cholesky方法需要进行开方，这有可能损失精度和增加运算量，为了避免开方，改进的算法采用将对称矩阵*A*分解为*LDLT*的形式，其中L为下三角矩阵，D为对角矩阵，将原方程转化为*LDLTX=b*的形式，设D-1*Y=LTX*, *LY=b*, 即可将原线性方程组转换为两步求解，更易于计算机实现。

求解线性方程组IMG_256的解步骤如下

（1）对矩阵IMG_257进行分解得到IMG_258

（2）求解IMG_259，得到IMG_260

（3）求解IMG_261，得到IMG_262

**复杂度分析：**

**时间复杂度：**因为分解系数矩阵时，有存在，可以看到上式中有ijk三个循环变量，因此时间复杂度为**O(n^3)**。

**空间复杂度：**需要二维数组存放分解后的c和l元素，因此复杂度为**O(n^2)**。

## 源程序及注释：

#include<bits/stdc++.h>

using namespace std;

const int maxn=55;//最大值为55

double H[maxn][maxn];//希尔伯特矩阵

double x[maxn],b[maxn],\_x[maxn],y[maxn];//\_x为 HX=b 的计算结果

double u[maxn][maxn],d[maxn],l[maxn][maxn];//分解后的矩阵

double r[maxn],detx[maxn];//剩余向量 和 detx

void init(){

//构造初始Hn矩阵和x向量

for(int i=1;i<maxn;++i){

for(int j=1;j<maxn;++j){

H[i][j]=1.0/(i+j-1);

}

x[i]=1;

}

}

void H\_x(int n){

//计算b的值

memset(b,0,sizeof(b));//初始b值为零

for(int i=1;i<=n;++i){//每个b[i]为 Hn矩阵i行j元素均乘以x向量j行1列元素之和

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

b[i]+=H[i][j]\*x[j];

}

}

}

void LDLt(int n){

//LDLt分解法

//分解H，uij=lij\*dj，先求出uij,即可分解求出lij以及dj

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

for(int j=1;j<i;++j){//一开始不进入循环跳过求d[1]的值

//求uij，uij=lij\*dj

//累减误差较大，改成先累加再减

double sum=0;

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

sum+=u[i][k]\*l[j][k];

}

u[i][j]=H[i][j]-sum;

//求lij

l[i][j]=u[i][j]/d[j];

}

//求di=h[i][j]-ul之前的求和

double sum=0;

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

sum+=u[i][k]\*l[i][k];

}

d[i]=H[i][i]-sum;

}

}

//计算y和\_x

void Cal\_x(int n){

//求y，由公式l\*y=b反推y表达式，画出矩阵理解,由于当前l行乘以y列等于b[i],则y[i]等于b[i]减去之前的求和

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

double sum=0;

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

sum+=l[i][k]\*y[k];

}

y[i]=b[i]-sum;

}

//求\_x,y/d=(l^t)\*(\_x)，反推\_x表达式，画出矩阵进行理解，等于左边的矩阵减去之前的求和

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

double sum=0;

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

sum+=l[k][i]\*\_x[k];

}

\_x[i]=y[i]/d[i]-sum;

}

}

//改进Cholesky算法

void Improved\_Cholesky(int n){

LDLt(n);

Cal\_x(n);

}

//计算r和detx

void Cal\_r\_detx(int n){

//计算r和detx

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

double sum=0;

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

sum+=H[i][j]\*\_x[j];

}

r[i]=b[i]-sum;

detx[i]=\_x[i]-x[i];

}

}

//H输出

void print2(int n){

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

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

printf("%.9f ",H[i][j]);

}

cout<<endl;

}

cout<<endl;

}

//任意一个一维数组输出

void print1(int n,double\* a){

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

printf(" %12.9f\n",a[i]);

}

cout<<endl;

}

void way(){

init();//初始化

int n=0;

while(++n){ //直到满足条件后退出循环

if(n==1){ //开始时一样

cout<<" n\t有效数字数\n";

cout<<" 1\t 相等\n";

continue;

}

H\_x(n);//计算b

Improved\_Cholesky(n);//调用优化Cholesky算法计算\_x;

Cal\_r\_detx(n);//计算r和detx;

double det=0;

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

det=max(fabs(detx[i]),det);//求出最大的detx

}

//det<=1/2\*1e-n -> n位有效数字

double eps=0.5;

int s=0;

while(++s){ //求有效数字位数算法

eps/=10;

if(det>eps) break;

}

printf(" %d\t %d\n",n,s-1);//需要减去一位

if(s==1) break;//一位有效数字都没有的时候退出

}

}

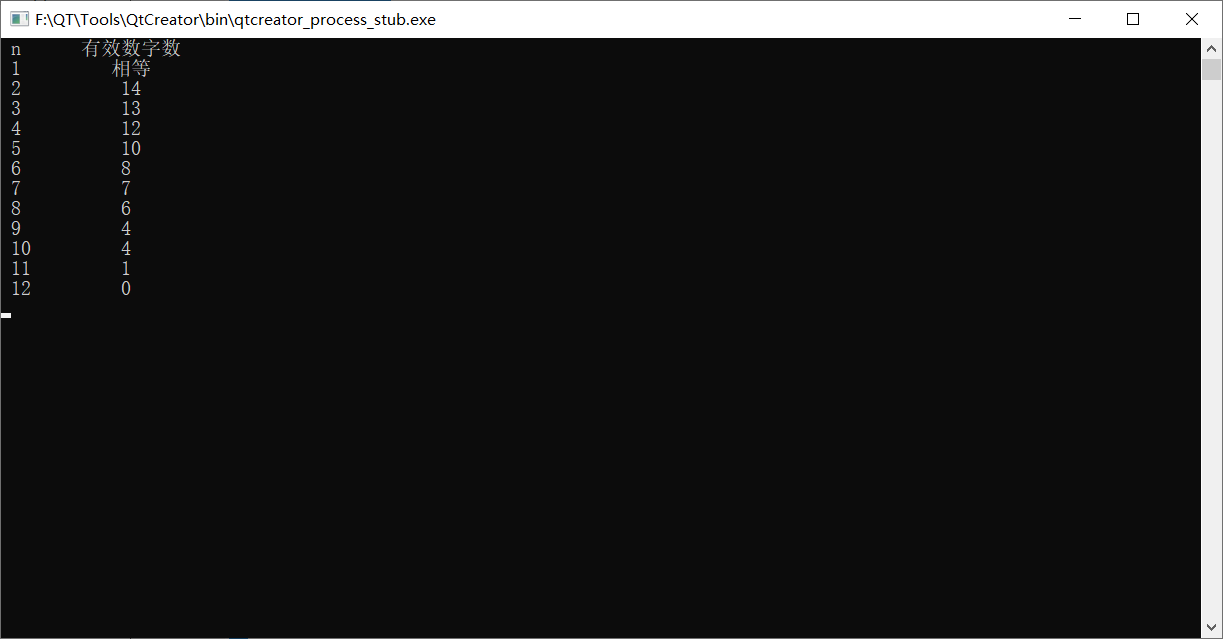
int main(){

way();//调用方法解决问题

return 0;

}

## 运行输出结果：



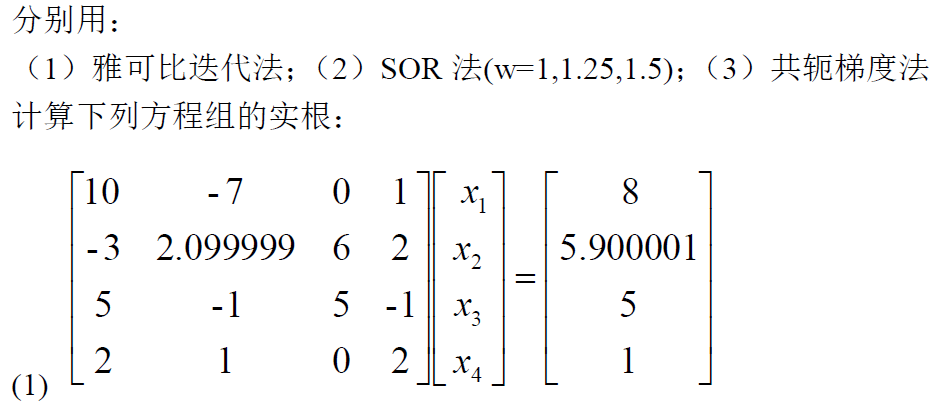
**由上图可以看出，当n=12时，\_x连一位有效数字都没有了。**

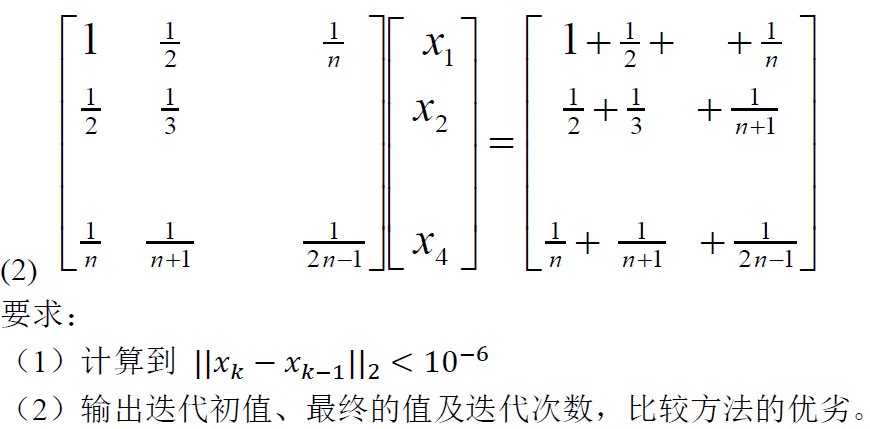
## 五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

设计代码过程中，由于牵扯到许多变量，以及变量之间的矩阵求解关系，刚开始设计完代码时，跑出来的结果一直与设想结果不符，之后通过将每种变量均取一特定意义名称，同时尽量简化变量间关系，减少中间变量的方式，直接将初始变量与末态变量间关系对应起来，最终解决了这个问题，还有就是求解有效位数的算法问题，一开始的精度设置不好，运行结果始终相差甚远，之后通过定义exp=0.5，循环除以10比较精度的方法解决了有效位数的问题，这也提醒了我算法不一定越复杂，精度和效果越好，能解决问题的代码就是好代码。

# 实验三、线性方程组的迭代解法

## 实验内容：





## 算法基本思想及复杂度分析：

**Jacobi 迭代法：**

**基本思想：**对于一个线性方程组***Ax* = *b***来说，将***A***分裂成***A* = *D* - *L – U***，其中D是对角矩阵，L、U是三角矩阵，从而将原线性方程组化为迭代公式形式求解。

**基本公式：**

**收敛性分析：**Jacobi 迭代收敛的充要条件 *ρ* (*J*) < 1，Jacobi 迭代收敛的充分条件 ||*J*|| <1，其中。

**复杂度分析：**

**时间复杂度：** 对于一次迭代来说，因为涉及到矩阵乘法，因此时间复杂度为**O(n^2)**，但是因为有多次迭代，应该还要乘上迭代的次数m，因此总的时间复杂度应该为**O(mn^2)**。

**空间复杂度：**涉及到系数矩阵存储，二维数组，所以空间复杂度为**O(n^2)**。

**SOR迭代法：**

**分析**：Jacobi迭代法虽然算法简单，易于实现，但是在进行下一次迭代时，需要保存上一次的迭代解，因此在Jacobi迭代法基础上衍生出了Gauss-Seidel迭代法，将每一次迭代中的代替作为迭代条件，不再需要保留上一步迭代解，还可能会得到更好的收敛效果。再对Gauss-Seidel迭代法进一步改进，引入松弛因子***ω*** ，通过选取合适的 ***ω*** ，可获得更快的收敛速度，但是如何确定 SOR 迭代中的最优松弛因子是一件很困难的事。

**基本思想：**与Jacobi迭代法类似，是Jacobi迭代法的双重改进。

**基本公式：**

**收敛性分析：**SOR 迭代收敛的充要条件 *ρ* (*Lω*) < 1，SOR 迭代收敛的充分条件 ||*Lω*|| < 1，其中。

**复杂度分析：**

**时间复杂度：** SOR法可以理解成两次改进的雅可比迭代法，因此时间复杂度应该依旧为**O(mn^2)**。

**空间复杂度：**涉及到系数矩阵存储，二维数组，所以空间复杂度为**O(n^2)**。

**共轭梯度法（CG法）：**

**分析：**一阶定常迭代法（SOR）的收敛速度很慢，而且其收敛条件对系数矩阵有一定的要求，如果我们采用“**变分原理**”，将迭代求解线性方程组转化为在**n维线性空间中搜索**最优解，理论上来说求解效率和收敛条件都会有一定的提升，于是便有了**最速下降法**。虽然最速下降法每一步都能找到局部最优解，但是整体效率低，因为局部最优未必为全局最优，因此在最速下降法的基础上通过改进得出了**共轭梯度法**（CG法）。

**基本思想：**当前点的最速下降方向为，同时记录下搜索的前一个方向是，过，在上述两个方向张成的平面上寻找函数的最小值，引入变量，故下一次的搜索方向为又由最速下降法的迭代公式：，从而得出搜索步长应为。

**复杂度分析：**

**时间复杂度：** 对于一次迭代来说，因为涉及到矩阵乘法，因此时间复杂度为**O(n^2)**，但是因为有多次迭代，应该还要乘上迭代的次数m，因此总的时间复杂度应该为**O(mn^2)**。虽然分析出来的复杂度和雅可比迭代法是相同的，但是很显然CG法收敛速度要快很多很多，因此可能差异体现在了m上，同时也再次告诉我们复杂度和收敛速度没有必然关系。

**空间复杂度：**涉及到系数矩阵存储，二维数组，所以空间复杂度为**O(n^2)**。

## 源程序及注释：

#include<bits/stdc++.h>

using namespace std;

const double eps=1e-6;//按照题目要求设置精度

const int maxn=105;

double \*\*A1,\*\*A2;//二级指针，便于函数方程输入以及传参

double x1[maxn],b1[maxn],x2[maxn],b2[maxn],x0[maxn]={0};//x1和x2分别用来存储各种方法对函数1和函数2输出的结果

double r[maxn],p[maxn],Ap[maxn];

void init(int n){

//对题目要求求解的两个方程进行输入

//方程 1输入

A1 = new double\*[5];

for(int i=1;i<5;++i){

A1[i] = new double[5];

}

A1[1][1]=10,A1[1][2]=-7,A1[1][3]=0,A1[1][4]=1;

A1[2][1]=-3,A1[2][2]=2.099999,A1[2][3]=6,A1[2][4]=2;

A1[3][1]=5,A1[3][2]=-1,A1[3][3]=5,A1[3][4]=-1;

A1[4][1]=2,A1[4][2]=1,A1[4][3]=0,A1[4][4]=2;

b1[1]=8,b1[2]=5.900001,b1[3]=5,b1[4]=1;//b[i]按照题目已经算好

//方程 2输入

A2 = new double\*[maxn];

for(int i=1;i<maxn;++i){

A2[i] = new double[maxn];

}

for(int i=1;i<maxn;++i){

for(int j=1;j<maxn;++j){

A2[i][j]=1.0/(i+j-1);

}

}

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

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

b2[i]+=1.0/(i+j-1);

}

}

}

void print(int n,double\* a){

//输出任意一个一维数组

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

printf(" %12.9f\n",a[i]);//占位为12，精度为9

}

cout<<endl;

}

int Jacobi(int n,double \*\*a,double \*x,double \*b,int cnt=1){//初始迭代次数设为1

//雅可比迭代求解

//迭代上限，防止不收敛

if(cnt==100) return cnt;//谱半径ρ≥1则不收敛，迭代次数cnt过多则不收敛

if(cnt==1){ //为x0设置初值

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

x0[i]=0;

}

}

//迭代公式

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

double sum=0;

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

sum+=a[i][j]\*x0[j];

}//排除第a[i][i]和x0[i]的乘积

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

sum+=a[i][j]\*x0[j];

}

x[i]=(b[i]-sum)/a[i][i];//由AX=B求得x0[i]的结果保存在x[i]中，根据外部传入的数组可能是x1[i]或者x2[i]

}

//输出

printf("第 %d 次迭代：\n",cnt);

print(n,x);//调用自定义的print函数输出数组x[n]

//计算精度

double det=0;

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

det+=(x[i]-x0[i])\*(x[i]-x0[i]);

}

if(sqrt(det)<eps) return cnt;//迭代精度达到要求，返回迭代次数

for(int i=1;i<=n;++i){//更新x0，为下次迭代做准备

x0[i]=x[i];

}

return Jacobi(n,a,x,b,cnt+1);//递归调用下一次迭代

}

int SOR(int n,int w,double \*\*a,double \*x,double \*b,int cnt=1){

//SOR迭代求解

//迭代上限，防止不收敛

if(cnt==100) return cnt;//谱半径ρ≥1则不收敛，迭代次数cnt过多则不收敛

if(cnt==1){ //为x、x0设置初值

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

x[i]=x0[i]=0;

}

}

//迭代公式

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

double sum=0;

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

sum+=a[i][j]\*x[j];

}

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

sum+=a[i][j]\*x[j];

}

x[i]=(1-w)\*x[i]+w\*(b[i]-sum)/a[i][i];//由基本公式加松弛因子以及前一次迭代结果xn-1推出

}

//输出

printf("第 %d 次迭代：\n",cnt);

print(n,x);//打印x数组内容

//计算停止条件

double det=0;//判断精度是否达到要求

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

det+=(x[i]-x0[i])\*(x[i]-x0[i]);//求平方，使得正负对精度影响效果相同

}

if(sqrt(det)<eps) return cnt;

//更新x0

for(int i=1;i<=n;++i){//传递进下一次结果

x0[i]=x[i];

}

return SOR(n,w,a,x,b,cnt+1);

}

void Cal\_r(int n,double \*\*a,double \*x,double \*b){

//计算r

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

double sum=0;

for(int j=1;j<=n;++j){//计算Axk

sum+=a[i][j]\*x[j];

}

r[i]=b[i]-sum;//初始时令p[i]与r[i]相等

p[i]=r[i];

}

}

int Conjugate\_gradient(int n,double \*\*a,double \*x,double \*b,int cnt=1){

//共轭梯度法求解，用于求解正定对称矩阵，对于第一个矩阵不收敛，第二个矩阵收敛

if(cnt==100) return cnt;//迭代上限，防止不收敛,cnt过多则不收敛

if(cnt==1){//初始时先初始化x0同时调用函数计算r和p

memset(x0,0,sizeof(x0));

Cal\_r(n,a,x0,b);//计算r和p

}

//求搜索步长ar的结果

memset(Ap,0,sizeof(Ap));//计算AP数组即A\*P

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

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

Ap[i]+=a[i][j]\*p[j];

}

}

double ar=0,temp=0;

double sum=0;

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

sum+=r[i]\*r[i];

temp+=r[i]\*Ap[i];

}

ar=sum/temp;

//更新x

for(int i=1;i<=n;++i){//用所得步长去更新x

x[i]=x0[i]+ar\*p[i];

}

printf("第 %d 次迭代：\n",cnt);

print(n,x);

//计算精度，获得停止条件

double det=0;

for(int i=1;i<=n;++i){//判断是不是精确解,如果r向量为0则上次结果即精确否则除数为零

det+=r[i];

}

if(det==0) return cnt-1;//说明上一次迭代就计算完毕，所以返回前一次次数

det=0;//判断精度是否达到要求的另一种方式

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

det+=(x[i]-x0[i])\*(x[i]-x0[i]);//求差值的平方，取消正负影响

}

if(sqrt(det)<eps) return cnt;

//参数更新，为下一次迭代做准备

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

x0[i]=x[i];

}

//更新r

for(int i=1;i<=n;++i){//计算r

r[i]=r[i]-ar\*Ap[i];

}

//更新p

double bet;//计算bet

double sum1=0;

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

sum1+=r[i]\*r[i];

}

bet=sum1/sum;

for(int i=1;i<=n;++i){//计算p

p[i]=r[i]+bet\*p[i];

}

return Conjugate\_gradient(n,a,x,b,cnt+1);//进行下一次迭代

}

void way(){

int n,cnt;

int choice\_1,choice\_2;

cout<<"请输入要求解的方程序号：（格式：1或2）"<<endl;

cin>>choice\_1;

if(choice\_1==2){

cout<<"请输入n:"<<endl;

cin>>n;

init(n);

}

else{

init(4);

}

cout<<"请输入要使用的求解方法：（雅可比迭代：1，SOR：2，共轭梯度：3）"<<endl;

cin>>choice\_2;

if(choice\_1==1&&choice\_2==1){

cnt=Jacobi(4,A1,x1,b1);

if(cnt!=100){

printf("迭代次数为：%d\n计算结果为：\n",cnt);

print(4,x1);

}

else {

printf("对于方程1系数矩阵，雅可比迭代方法不收敛。");

}

}

else if(choice\_1==1&&choice\_2==2){

double w;

cout<<"请输入松弛因子：（1，1.25，1.5）"<<endl;//松弛因子需要在[0,2]之间

cin>>w;

cnt=SOR(4,w,A1,x1,b1);

if(cnt!=100){

printf("迭代次数为：%d\n计算结果为：\n",cnt);

print(4,x1);

}

else {

printf("对于方程1系数矩阵，SOR迭代方法不收敛。");

}

}

else if(choice\_1==1&&choice\_2==3){

cnt=Conjugate\_gradient(4,A1,x1,b1);

if(cnt!=100){

printf("迭代次数为：%d\n计算结果为：\n",cnt);

print(4,x1);

}

else {

printf("对于方程1系数矩阵，共轭梯度法不收敛。");

}

}

else if(choice\_1==2&&choice\_2==1){

cnt=Jacobi(n,A2,x2,b2);

if(cnt!=100){

printf("迭代次数为：%d\n计算结果为：\n",cnt);

print(n,x2);

}

else {

printf("对于方程2系数矩阵，雅可比迭代方法不收敛。");

}

}

else if(choice\_1==2&&choice\_2==2){

double w;

cout<<"请输入松弛因子：（1，1.25，1.5）"<<endl;//松弛因子需要在[0,2]之间

cin>>w;

cnt=SOR(n,w,A2,x2,b2);

if(cnt!=100){

printf("迭代次数为：%d\n计算结果为：\n",cnt);

print(n,x2);

}

else {

printf("对于方程2系数矩阵，SOR迭代方法不收敛。");

}

}

else{

cnt=Conjugate\_gradient(n,A2,x2,b2);

if(cnt!=100){

printf("迭代次数为：%d\n计算结果为：\n",cnt);

print(n,x2);

}

else {

printf("对于方程2系数矩阵，共轭梯度法不收敛。");

}

}

for(int i=1;i<5;++i){ //将空间释放

delete A1[i];

delete A2[i];

}

delete[] A1;

delete[] A2;

}

int main()

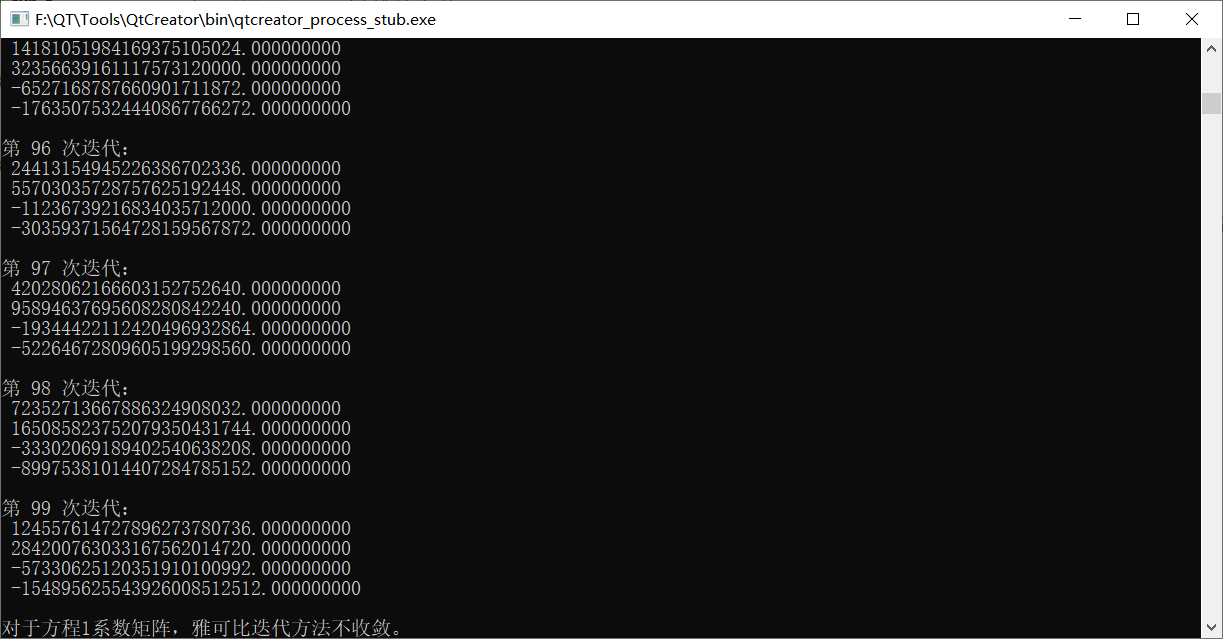
{

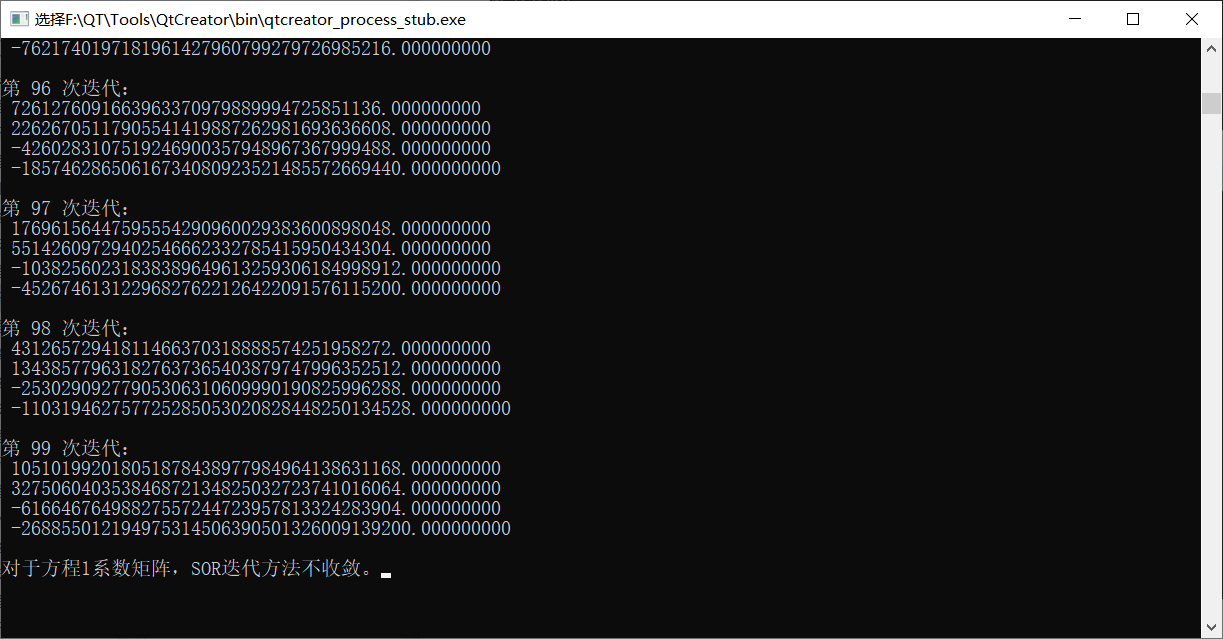
way();//调用函数解决问题

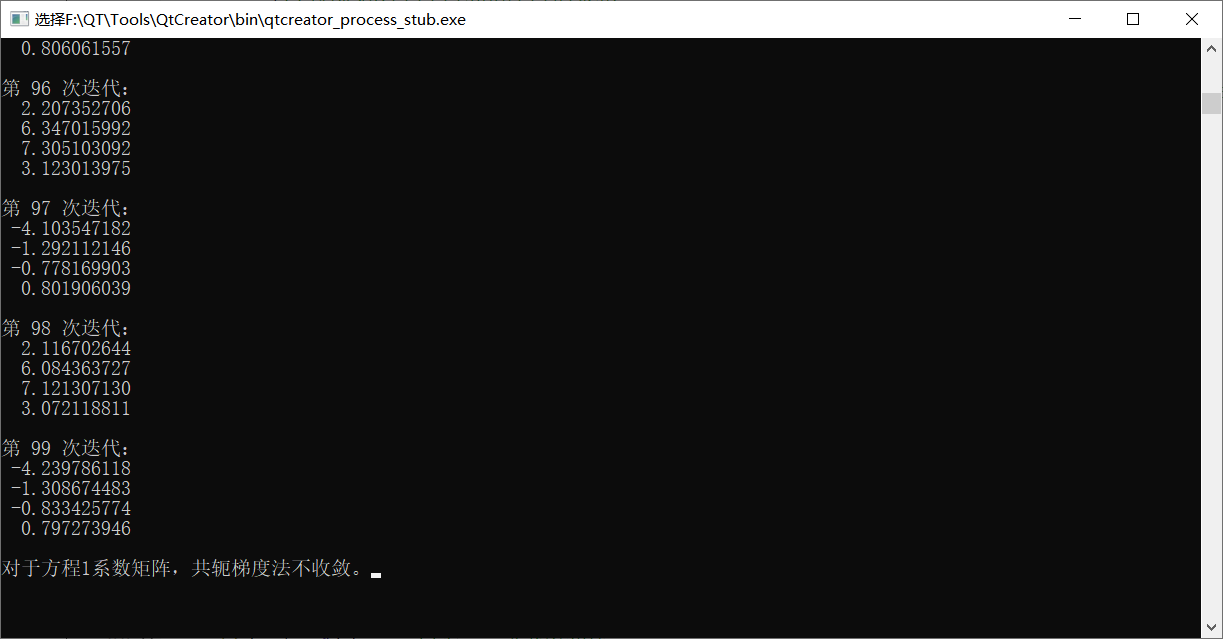
return 0;

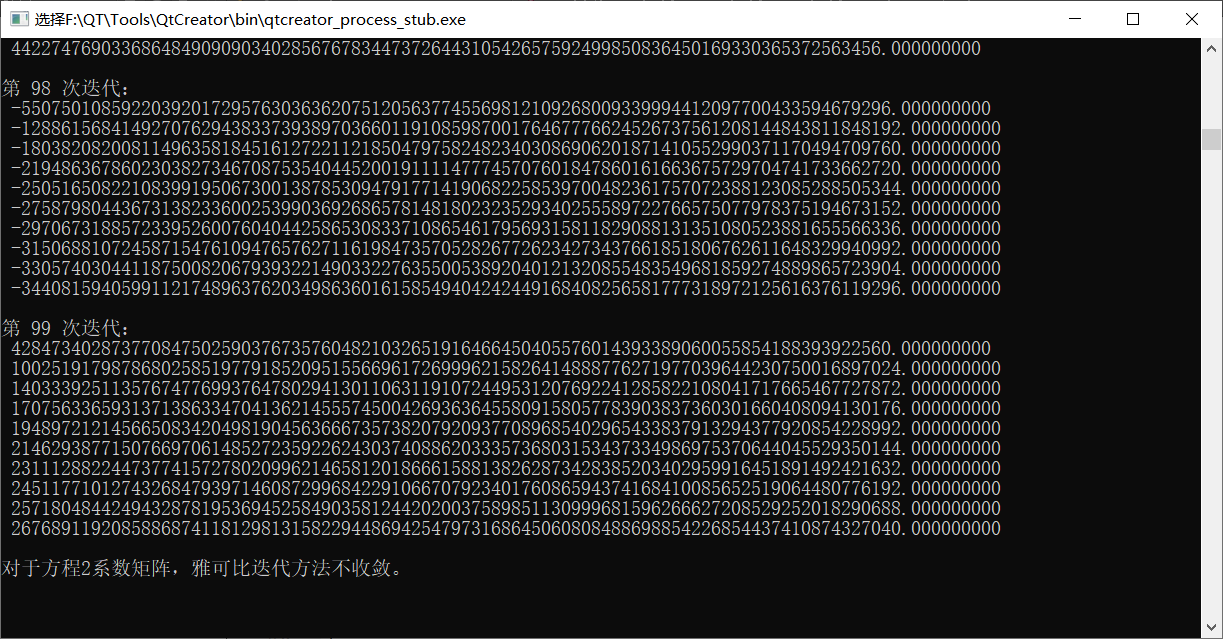
}

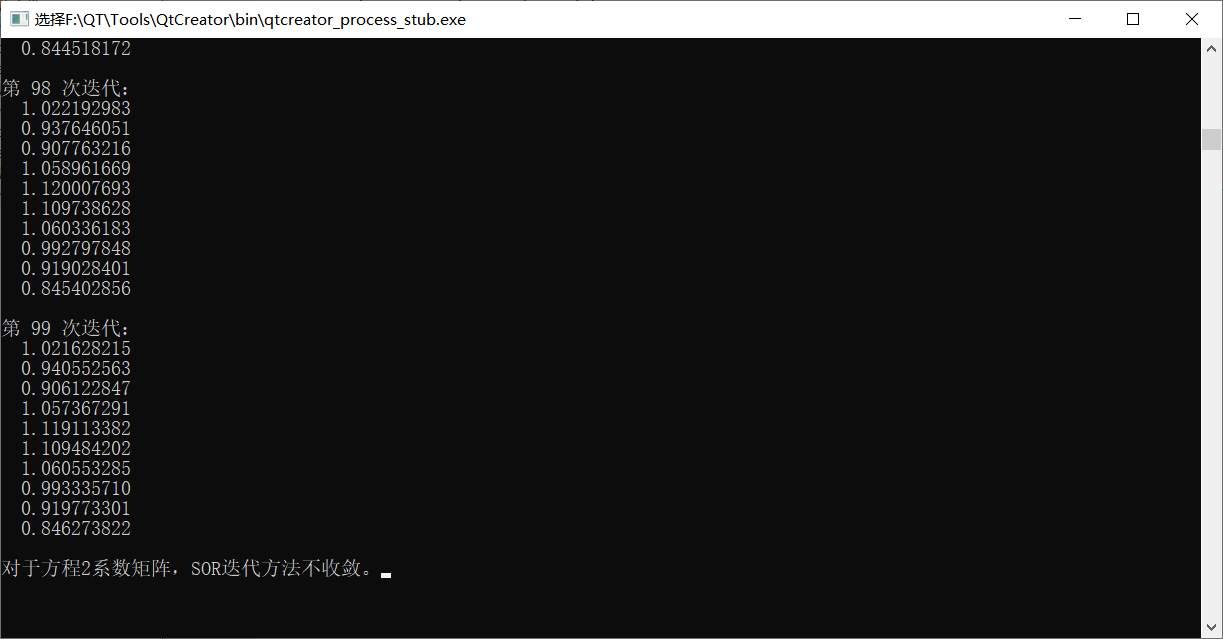
## 运行输出结果：

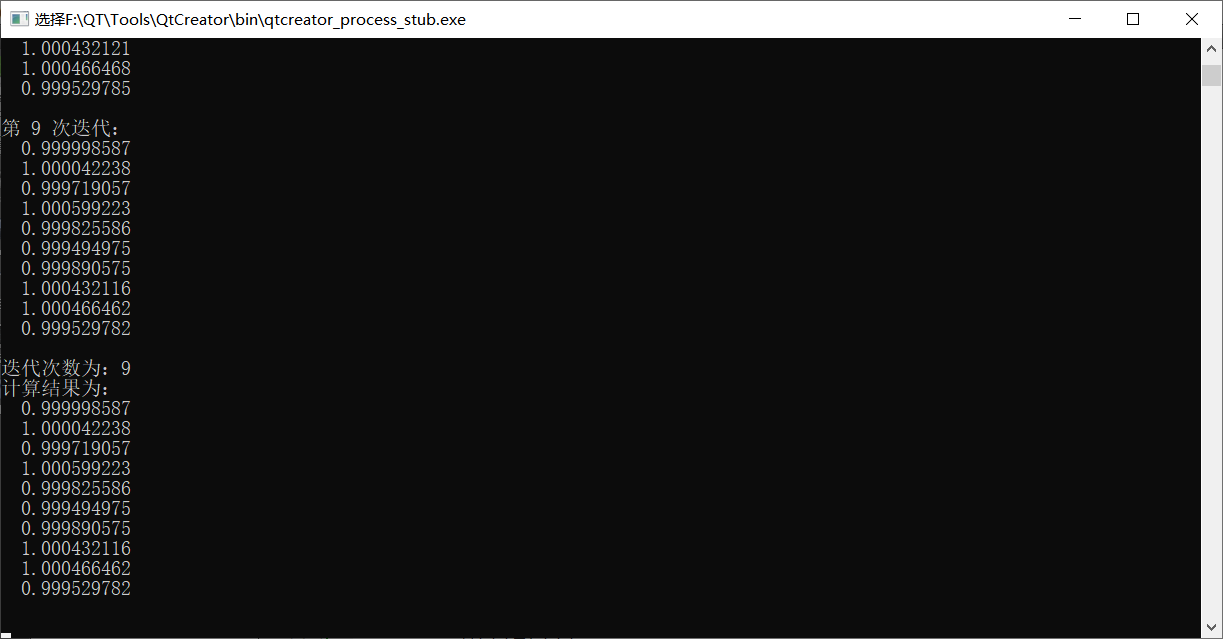










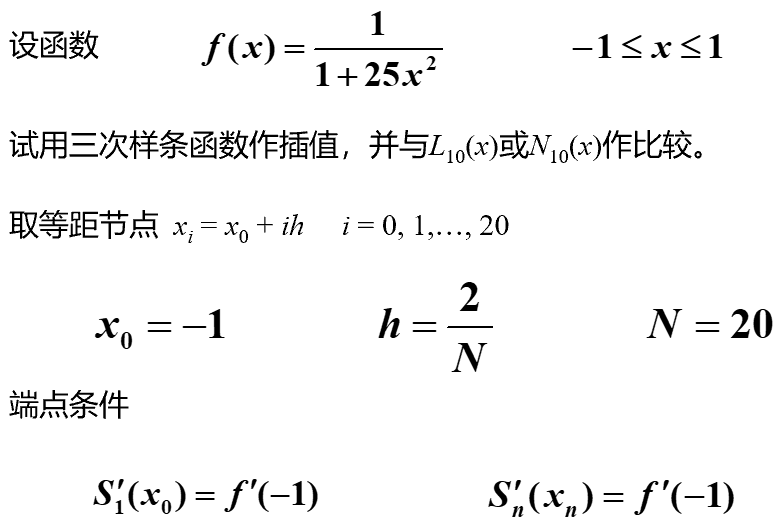


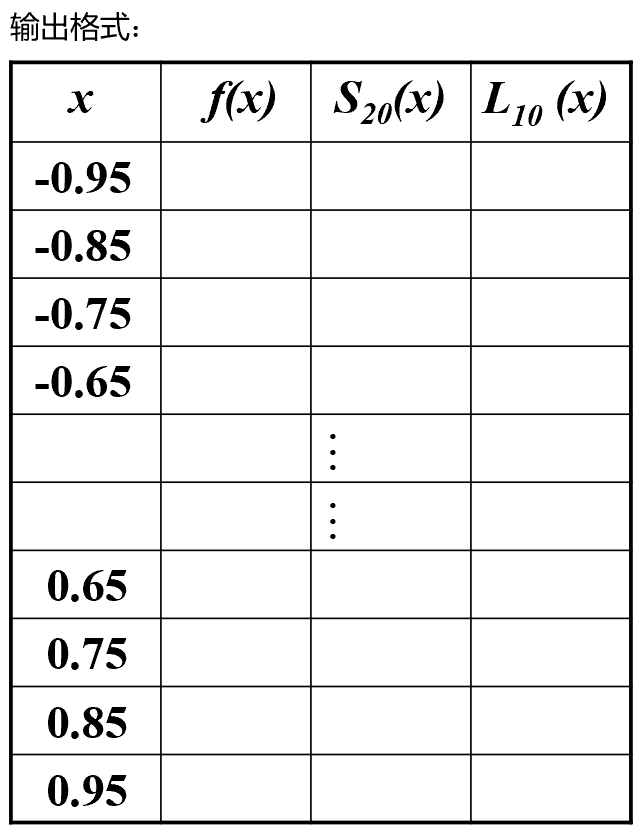
## 五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

本实验涉及到的计算方法较多，同时也需要对方程的种类以及使用的方法进行判断，所以在发现结果储存在同一个数组中出现了交叉污染后，我选择了重复定义了多个数组用以储存各种方法计算出来的结果如x1[maxn],x2[maxn]，同时一开始在采用雅可比迭代法以及SOR迭代法时，发现循环一直不出现结果，一开始还以为代码设计的有所问题调试了许久，之后通过计算得出，两个函数的谱半径大于1，方法是不收敛的，而对于计算谱半径的算法较为复杂，于是我又考虑是否能够通过迭代次数的方法来判断是否收敛，于是我以100次为上限(更高也可以，一样不收敛)，进行运行，而对于第三种方法是用来解决正定对称矩阵的，对于方程组2是有解的，在知晓了大致结果的前提下进行设计，最终便较为轻松的解决了问题。

# 实验四、函数插值

## 实验内容：





## 算法基本思想及复杂度分析：

**拉格朗日插值：**

**基本公式：** 



***p*(*x*) = *y*0*l*0(*x*) + *y*1*l*1(*x*)+ · · · + *ynln*(*x*)** 

**复杂度分析：**

**时间复杂度：**因此有n个插值元素，每个插值元素都要计算系数，因此时间复杂度应该为**O(n^2)**。

**空间复杂度：**涉及到系数存储，一维数组，所以空间复杂度为**O(n)**。

**三次样条插值：**

**基本算法：**

*hi = xi – xi-1*



，，



**复杂度分析：**

**时间复杂度：**追赶法解三对角线性方程组，因此时间复杂度为**O(n)**。

**空间复杂度：**涉及到系数存储，一维数组，所以空间复杂度为**O(n)**。

## 源程序及注释：

#include<bits/stdc++.h>

using namespace std;

const int maxn=100;

const double x0=-1;

const double y0=50.0/676.0;

double diff[maxn];

double d[maxn];

double M[maxn];

double miu[maxn],lam[maxn],c[maxn];

double ar[maxn],bet[maxn],y[maxn];

//F(x)

double F(double x){

//初始函数

return 1/(1+25\*x\*x);

}

double Lagrange(double x,int n){

//拉格朗日插值法拟合函数

//按照题意要求以10次插值为例

double ans=0;

double h=2.0/n; //将区间(-1,1)十等分

for(int k=0;k<=n;++k){//对于每一个小块重复计算以下过程

double temp=F(x0+k\*h);//计算yk

for(int i=0;i<=n;++i){ //计算lk并实现pk=yk\*lk

if(i==k) continue;

temp\*=(x-(x0+i\*h));

temp/=((k-i)\*h);

}

ans+=temp;//算出pk之和即为结果

}

return ans;

}

double Newton(double x,int n){

//牛顿插值法

    double ans=F(x0);

    double h=2.0/n;

    //初始化差商表

    for(int i=0;i<=n;++i){

        diff[i]=F(x0+h\*i);

    }

    for(int i=1;i<=n;++i){

        double temp=1;

        for(int j=0;j<i;++j){

            temp\*=(x-(x0+h\*j));

        }

        for(int j=0;j<=n-i;++j){

            diff[j]=(diff[j+1]-diff[j])/(i\*h);

        }

        temp\*=diff[0];

        ans+=temp;

    }

    return ans;

}

void Catch(int n){

//追赶法解三对角线性方程组

ar[1]=c[1];

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

bet[i]=lam[i]/ar[i];

ar[i+1]=c[i+1]-miu[i+1]\*bet[i];

}

y[1]=d[1]/ar[1];

for(int i=2;i<=n;++i){

y[i]=(d[i]-miu[i]\*y[i-1])/ar[i];

}

M[n]=y[n];

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

M[i]=y[i]-bet[i]\*M[i+1];

}

void Cubic\_spline(int n){

//三次样条插值

double h=2.0/n;//这里的n为20，即划分为20个等份

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

lam[i]=miu[i]=0.5;//为每个等份的入[i]和u[i]按照公式设好初值

c[i]=2;//初始化均为2

}

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

d[i]=6\*((F(x0+(i+1)\*h)-F(x0+i\*h))/h-((F(x0+i\*h)-F(x0+(i-1)\*h))/h))/(2\*h);//计算每个等份内的d[i]

}

Catch(n-1);//调用追赶法解三对角线性方程组

M[0]=(6\*((F(x0+h)-F(x0))/h-y0)/h-M[1])/2;//设立初始m[0]，m0=f0‘‘

M[n]=M[0];//末尾等于初始

}

double S(double x,int n){

//S(x)三次样条插值结果

double h=2.0/n;//划分等份

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

if(x0+i\*h>=x){//返回离本次传入x最近且大于等于x的插值结果

double anss=0;

anss+=M[i-1]\*(x0+i\*h-x)\*(x0+i\*h-x)\*(x0+i\*h-x)/(6\*h);//按照公式模拟

anss+=M[i]\*(x-(x0+(i-1)\*h))\*(x-(x0+(i-1)\*h))\*(x-(x0+(i-1)\*h))/(6\*h);

anss+=(F(x0+(i-1)\*h)-(M[i-1]\*h\*h/6))\*(x0+i\*h-x)/h;

anss+=(F(x0+i\*h)-(M[i]\*h\*h/6))\*(x-(x0+(i-1)\*h))/h;

return anss;

}

}

return 0;

}

int main(){

double x=-0.95;//按照题目表格要求，从-0.95开始计算。

Cubic\_spline(20);//三次差值以20等份为例

printf(" x\t f(x)\t S(x)\t L(x)\n");

for(int i=1;i<=20;++i){//循环到x=0.95为止,依次打印表格

printf("%5.2f",x);

printf("%12.6f",F(x));

printf("%12.6f",S(x,20));

printf("%12.6f\n",Lagrange(x,10));

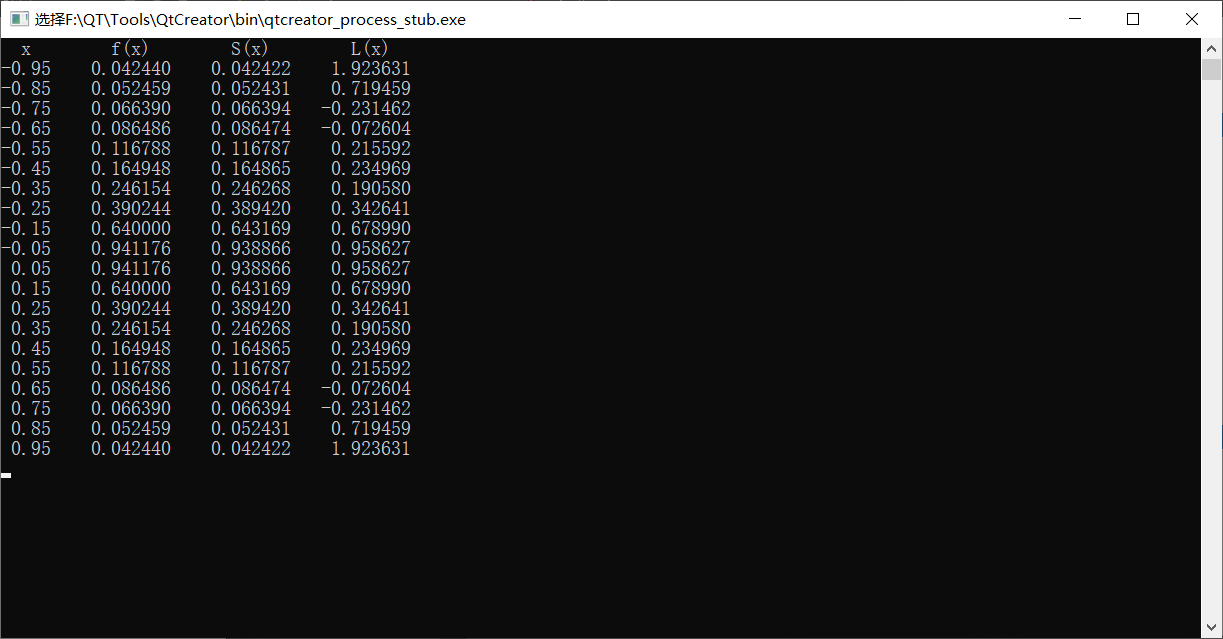
x+=0.1;

}

return 0;

}

## 运行输出结果：



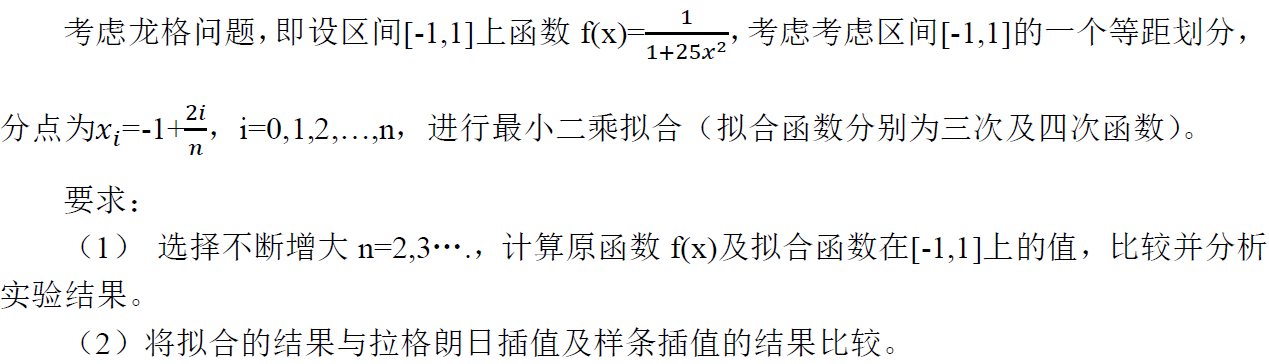
**拉格朗日法出现了龙格现象，在-0.95和0.95处与f(X)的结果相差甚远。**

## 五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

这里首先我实现的是拉格朗日插值法，而其在-0.95和0.95拟合结果与原函数相差甚远，一开始试图找寻方式进行优化和精化，后来在网上发现是出现了龙格现象，这也是为什么拉格朗日插值法主要用于拟合靠近原点部分图像，算法主要复杂在三次样条插值部分，通过公式可以发现计算S(x)需要用到前置结点，那么就涉及到追赶法解方程组的问题了。代码方面主要就是按照公式模拟，进过前面几题后便较为轻松，主要注意需计算出y0开始迭代，同时采用c++11的版本，否则以分式的方式定义常量不被允许。

# 实验五、最小二乘拟合

## 实验内容：



## 算法基本思想及复杂度分析：

**多项式最小二乘拟合：**

**基本思想：**已知函数值表 ( *xi* , *yi* )，在函数空间 Φ 中求 *S*\*(*x*) ，使得



从而转化为线性返程组求解问题。

**基本公式：**



**复杂度分析：**

**时间复杂度：**构造矩阵的时间复杂度比较大，**O(nm^2)**，其中m为多项式次数，对于本题来说其实m可以忽略，降为**O(n)**，但又考虑到用到了CG法，因此总的复杂度为**O(mn^2)**。

**空间复杂度：**涉及到二维数组，所以空间复杂度为**O(n^2)**。

## 源程序及注释：

#include<bits/stdc++.h>

using namespace std;

const double eps=1e-6;//设置返回精度

const int maxn=155;

const double x00=-1;

double \*\*H;

double x[maxn],y[maxn];

double a[maxn],b[maxn];

double r[maxn],p[maxn],x0[maxn],Ap[maxn];

//F(x)

double F(double x){//定义函数

return 1/(1+25\*x\*x);

}

void init(int n,int k){

//构造矩阵并将其初始化

double h=2.0/n;//按照输入的n值将[-1,1]区间分块

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

x[i]=x00+i\*h;

y[i]=F(x[i]);

}

H = new double\*[150];//为二级指针H分配空间

for(int i=0;i<150;++i){//为每个指针指向分配空间

H[i] = new double[150];

}

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

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

double sum1=0,sum2=0;

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

sum2+=pow(x[l],i+j);

sum1+=pow(x[l],j)\*y[l];

}

H[i][j]=sum2;

b[j]=sum1;

}

}

}

void Conjugate\_gradient(int n,double \*\*a,double \*x,double \*b,int cnt=1){

//共轭梯度法

if(cnt==100) return ;//如果迭代次数过多则结束迭代并返回

if(cnt==1){//第一次迭代时进行初始化

memset(x0,0,sizeof(x0));

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

double sum=0;

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

sum+=a[i][j]\*x0[j];

}

r[i]=b[i]-sum;//计算r

p[i]=r[i];//计算p

}

}

//求ar

memset(Ap,0,sizeof(Ap));//计算AP数组

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

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

Ap[i]+=a[i][j]\*p[j];

}

}

double ar=0,temp=0;

double sum=0;

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

sum+=r[i]\*r[i];

temp+=r[i]\*Ap[i];

}

ar=sum/temp;//求出ar

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

x[i]=x0[i]+ar\*p[i];//利用ar进行更新x

}

double det=0;//判断循环中止条件

for(int i=0;i<=n;++i){//判断是不是精确解,否则除数为零

det+=r[i];

}

if(det==0) return ;//说明上一次迭代就计算完了，cnt-1

det=0;//判断精度是否达到要求

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

det+=(x[i]-x0[i])\*(x[i]-x0[i]);

}

if(sqrt(det)<eps){

return ;

}

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

x0[i]=x[i]; //更新x0为下一次循环做初始化

}

for(int i=0;i<=n;++i){//计算r

r[i]=r[i]-ar\*Ap[i]; //更新r

}

double bet;//计算bet

double sum1=0;

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

sum1+=r[i]\*r[i];

}

bet=sum1/sum;

for(int i=0;i<=n;++i){//计算p

p[i]=r[i]+bet\*p[i];//更新p值

}

Conjugate\_gradient(n,a,x,b,cnt+1);//调用下一次迭代

}

void LSF(int n,int k){

//多项式最小二乘拟合

init(n,k); //初始化

Conjugate\_gradient(k,H,a,b); //采用共轭梯度求解法来解方程

for(int i=1;i<150;++i){

delete H[i]; //释放指针H指向空间

}

delete[] H;

}

double LS(double x,int k){

//用于计算出LS(x)的结果

double sum=0;

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

sum+=a[i]\*pow(x,i);

}

return sum;

}

void way(){

int n,q;

double x=-0.95;

cout<<"请输入拟合函数的次数：";

cin>>q;//用于判断使用的是三次拟合函数还是四次拟合函数

cout<<"请输入等距划分数n：";

cin>>n;//输入等距划分点

LSF(n,q);

printf(" x\t f(x)\t LS(x)\n");

for(int i=1;i<=20;++i){

printf("%5.2f%12.6f%12.6f\n",x,F(x),LS(x,q));//打印输出表

x+=0.1;

}

}

int main()

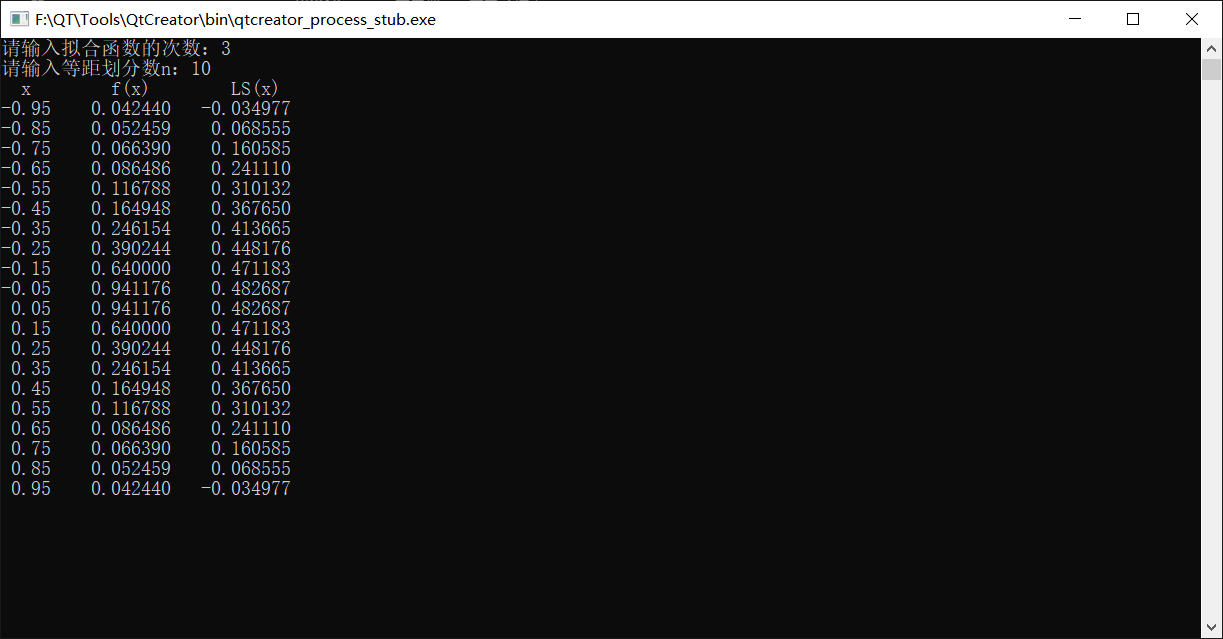
{

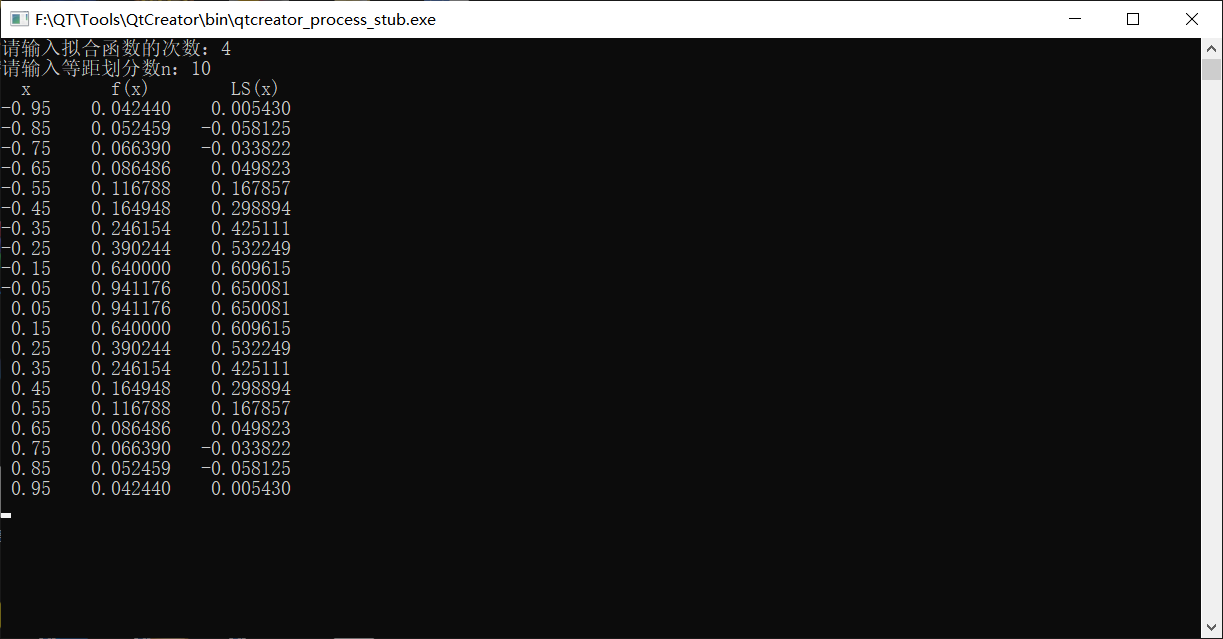
way();//调用way函数解决问题

return 0;

}

## 运行输出结果：





**由上述结果可以看出来，四次多项式拟合相较于三次多项式拟合要更加贴近原函数，能更加契合的反应原函数的大致变化情况，因此推断随着n的增加，也即次数的增加，拟合函数将逐步趋近于原函数。**

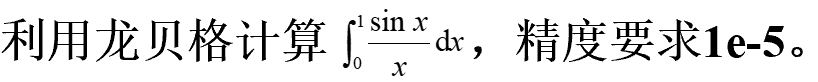
**作为比较而言，可以发现最小二乘拟合的拟合结果在细节方面以及拟合程度上是远远不如于拉格朗日插值在原点附近的拟合以及三次样条插值的总体拟合的，作为分析而言，我提出的看法是，由于三次样条插值保证了在等距节点处于原函数是相等的，而最小二乘拟合连这一点也无法保证，只是给了一个大体的变化趋势，因此认为其拟合的结果是不如于三次样条插值的。**

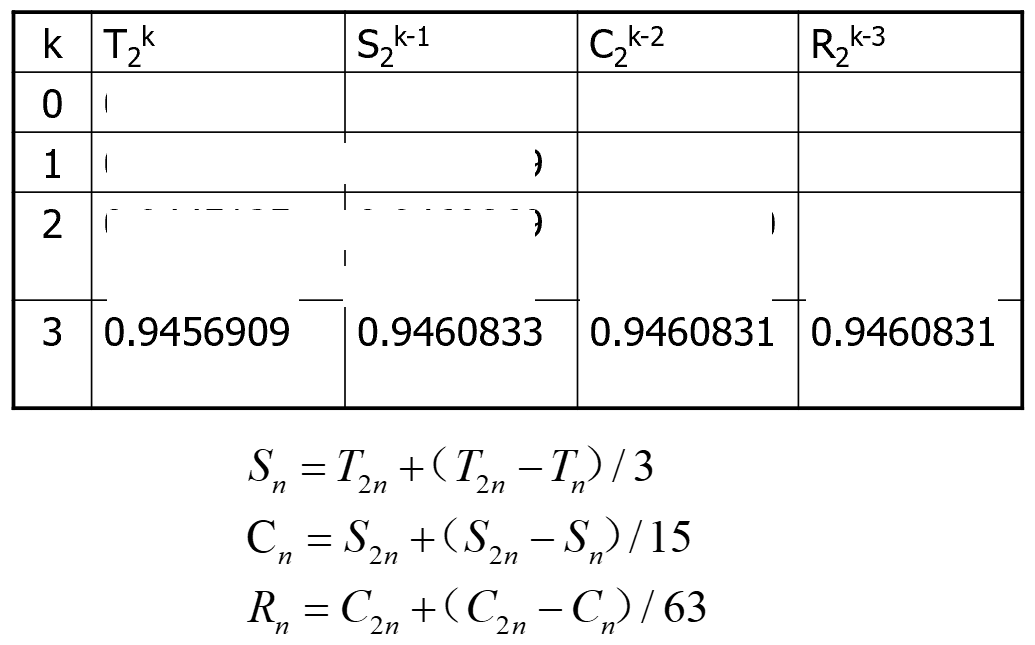
## 五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

共轭梯度法在之前的实验中已经使用过了，在这里便可以复用，稍加修改就可以十分契合本题的要求，主要在于构造初始矩阵和对函数次数以及区间段数的区分上，这里采用共轭梯度法也是因为通过实验三的比对发现共轭梯度法要优于其他的方法，主要体现在不依靠谱半径的情况上，而且性能也较为优异。

# 实验六、数值积分

## 实验内容：





## 算法基本思想及复杂度分析：

**龙贝格求积公式（逐次分半加速法）：**

**基本思想：**龙贝格求积公式是在梯形公式、抛物线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。

**基本公式：**













**复杂度分析：**

**时间复杂度：**根据算法基本思想，时间复杂度应该为O(2^n)，这是相当大的时间复杂度，因为指数函数是爆炸式增长的，后面会有具体的实践分析。

**空间复杂度：**使用了一维数组，所以空间复杂度为**O(n)**。

## 源程序及注释：

#include<bits/stdc++.h>

using namespace std;

const double eps=1e-5;//设置精度为1e-5

double f(double x){ //函数表达式，注意到当x=0时不能为分母，由洛必达得应返回1

if(x==0) return 1;

return sin(x)/x;

}

void Romberg(double a,double b){ //龙贝格法实现积分求解

double h,t1,s1,c1,r1,t2,s2,c2,r2;//梯形公式、抛物线公式、科斯特公式、龙贝格公式变量，其中下标为2的为循环遍量，在每次迭代中改变

t1=(b-a)/2\*(f(a)+f(b));//构造初始t1

h=(b-a);//构造初始h

int k=0;

printf("k T S C R\n");

while(1){//当满足条件时一直循环输出

double sum=0;

for(int i=0;a+i\*h+h/2<b;i++){//构造T2n变化方程中的后一部分即0至n-1求和部分

sum+=f(a+i\*h+h/2);

}

printf("%-4d",k);//输出当前循环次数k

if(k==0){//此时只有梯形公式有输出结果

t2=t1;k++;

printf("%-10.7lf\n",t2); //"-“号对应左对齐，占位10格，保留7位，精度为double

continue;

}

t2=t1/2+h/2\*sum;//按公式变化至T2

printf("%-10.7lf",t2);//输出T2

s2=t2+(t2-t1)/3;//按基本公式写出公式中意味着的S1，这里以S2代替S1做循环迭代

if(k==1){//此时抛物线公式出现结果

t1=t2;s1=s2;h/=2;k++;

printf("%-10.7lf\n",s2);

continue;

}

printf("%-10.7lf",s2);

c2=s2+(s2-s1)/15;//按基本公式写出公式中意味着的C1

if(k==2){//此时科特斯公式出现结果

t1=t2;s1=s2;c1=c2;h/=2;k++;

printf("%-10.7lf\n",c2);

continue;

}

printf("%-10.7lf",c2);

r2=c2+(c2-c1)/63;//按基本公式写出公式中意味着的r1

if(k==3){//此时龙贝格出现结果

t1=t2;s1=s2;c1=c2;r1=r2;h/=2;k++;

printf("%-10.7lf\n",r2);

continue;

}

printf("%-10.7lf\n",r2);

double temp=r1;

t1=t2;s1=s2;c1=c2;r1=r2;h/=2;//状态变化

k++;

if(fabs(t1-temp)<eps)

break;

}

cout<<"积分结果为："<<r2<<endl;

}

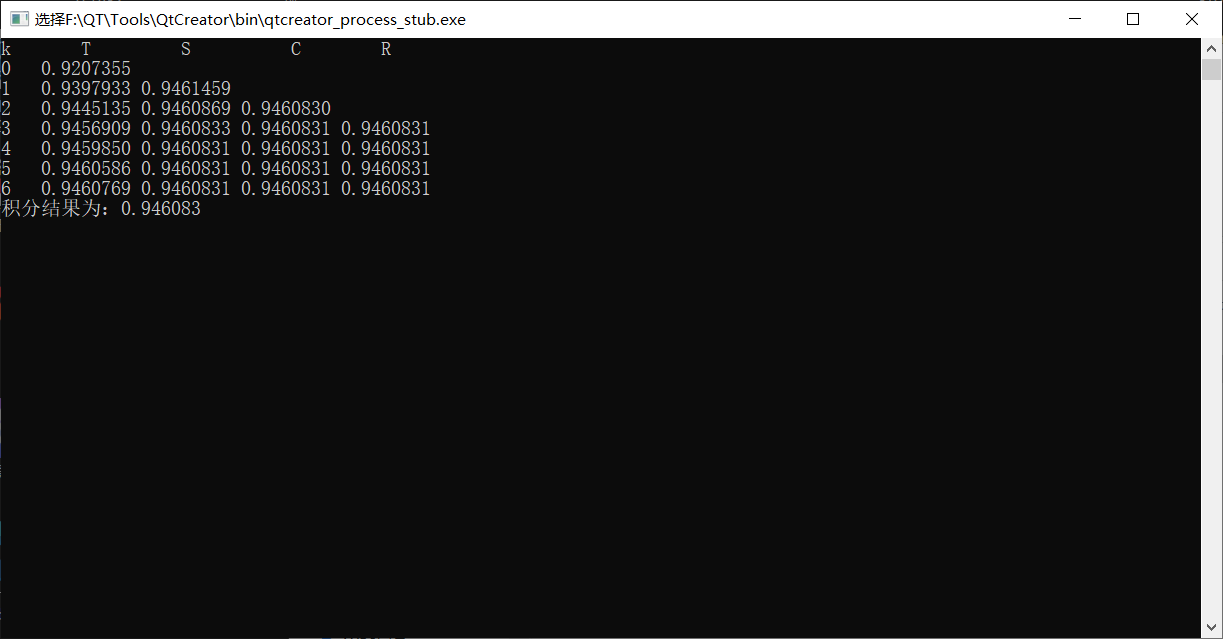
int main(){

Romberg(0,1);

return 0;

}

## 运行输出结果：



## 调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

本题在代码调试和运行时碰到的问题倒不是很多，但值得我注意的一点是，对于给定的函数而言，由于该函数在零点处分母为零无意义，因此需对函数求导，将该点导数值直接返回，从而实现对区间上的积分，代码表述即为if(x==0) return 1;一开始一直没想到这点，浪费了挺多时间。同时需要注意为了实现循环以及后一种方法调用前一种方法的结果的形式，需要定义两组变量，例如t1,t2,s1,s2这样，之后便是套用公式一步步求解，直到达到精度要求，输出即可。

**心 得 体 会**

转眼间，科学与工程计算的课程即将接近尾声，在这短短不到一个学期的时间里，我们从一开始对数值分析方面的一无所知，没有头绪的去完成老师布置的作业，再到渐渐的领会部分数值分析与计算的要点，通过线上视频后附带的ppt以及各大网站相关知识的学习，到最后在接近尾声的时间里圆满完成了科学与工程计算的实验报告。在这短短的一个流程中，我经历了从一开始的在家中看着习题的迷茫，再到逐渐转到线下和同学们进行面对面的交流和探讨，渐渐的，我开始理解如何使用代码语言去表述数值分析方面的知识，如何用一段代码去尽可能的复现公式想表达的含义。其中比较头疼的一点是初始化和迭代判断精度的问题，我一开始的想法是初始化尽可能的精简，精度要尽可能的严密和准确；然而事实恰恰相反，如果仅定义一两个数组用于所有方法的初始化，不仅会造成数据间的污染，还使得数组所表达的含义不够明确，使得代码晦涩难懂，同时精度的判断如果设计的复杂了，也会导致循环可能无法中止或是出现精度要求过高无法达到的问题。因此，我意识到，解决问题是一个动手实践的过程，自己设想的并不一定就是最正确和最恰当的，需要通过不断的实践去检验它，只有通过了数据考验的方法才是真正的好方法。

这门课的的确确的让我体会到了什么是数值分析：从一开始拿到报告，看着那实验内容的要求，不禁怀疑起短短几个月的时间是否可以圆满符合条件的完成，到之后通过一次次翻阅ppt和查找网站，通过在草稿纸上画图理清代码和公式间的关系，我逐渐的开始适应了这种学习模式，也可以较为有效迅速的完成每一个实验所提出的实验要求。我知道一个实验的背后往往是许多的汗水以及时间的付出，从一开始的毫无思路到查找到合适的公式方法再到代码实现，只有每个环节都亲力亲为才能够在出现问题的时候及时的找到错误，最终完成任务，因此培养查阅资料以及自学的能力对于我们来说就显得格外重要。同时，我觉得这门课带给我的收获不光是数组分析方面的知识，还有一个重大收获就是切切事事的增强了我的知识搜索能力和自学能力，从一次次完成实验的过程中，我体会到了数值分析实现的不易，也体会到了理解各种算法框架、实现思想的重要性，同时我也理解了如何去实现复用和修改代码，从而使得代码契合实验要求，这些都是使得实验能够完成的重要保证条件。

最后，我想感谢老师：虽然因为疫情的原因，我们与老师您的接触不多，但是这种以实践形式为考核方式的授课确实使得我收益良多。我知道，这仅仅是短暂的分别，之后的课程仍然会有王老师的身影出现，也希望之后的课程中可以跟着王老师领会高性能计算的魅力。

至此，表达我的感谢和对这次实验经历的珍重！