



华南理工大学

South China University of Technology

计算方法实验报告

学 院： 计算机科学与工程学院

班 级： 09 计科 2 班

姓 名： 冯少泳

学 号： 200930581024

指导老师： 韩国强

提交日期： 2012 年 6 月

邮 箱： nemosail@gmail.com

目录

1.	三次样条插值.....	P3
2.	P97 第 8 题上机题, 要求 $e=10^{-6}$ 使用算法	P5
	1) 自动选取步长复化梯形	
	2) 自动步长复化抛物线	
	3) Romberg 求积 (需要打印过程的三角形)	
3.	解线性方程组直接法.....	P9
	1) 顺序高斯消去法	
	2) 列主元高斯	
	3) 全主元高斯消去	
4.	解线性方程组迭代法.....	P12
	1) 简单迭代 Jacobi	
	2) Seidel;	
	3) SOR;	

附录 (实现源码):	P15
------------------	-----

- A. 三次样条插值实现代码
- B. 自动选取步长复化梯形源码
- C. 自动选取步长复化抛物线
- D. Romberg
- E. 顺序高斯
- F. 列主元高斯
- G. 全主元高斯
- H. 简单 Jacobi 迭代
- I. Seidel 迭代
- J. SOR 迭代

实验目的:

通过实现书上的算法, 加深对数值计算的认识。

实验环境:

Fedora 14

GCC 4.5.1

实验内容:

三次样条插值

三次样条函数是一个分段代数多项式, 在每一个分段上它是一个不超过三次的代数多项式, 它在节点上连续, 其一阶导数和二阶导数在节点上也连续。三次样条提法是:

1) 在 $[X(i), X(i+1)]$ ($i = 0, 1, \dots, n-1$) 上位不超过三次的代数多项式;

2) $s(x_i) = y_i$ ($i = 0, 1, \dots, n$);

3) $s(x) \in C^2[a, b]$, 其中 $a = x_0$, $b = x_n$.

三次样条插值有两种边界条件:

A $a_0 = 0$; $b_0 = 2 * m_0$

$a_n = 1$; $b_n = 2 * m_n$

B $a_0 = 1$; $b_0 = 3 / h_0 * (y_1 - y_0)$

$a_n = 0$; $b_n = 3 / h(n-1) * (y_n - y(n-1))$

P52

9 给定函数表, 和边界条件 $S''(75) = 0$, $S''(80) = 0$
(第二边界条件) 求 $f(78.3)$ 近似值:

X_n Y_n

```
[alpha@Cameron numberCompute]$ cat data7
75 2.768
76 2.833
77 2.903
78 2.979
79 3.062
80 3.153
```

运行结果:

```
[alpha@Cameron numberCompute]$ ./fenDuan3YangTiao data7 78.3
Please select Boundary : 1 or 2
: 2
alpha[0] = 1.000000    beta[0] = 0.195000
alpha[1] = 0.500000    beta[1] = 0.202500
alpha[2] = 0.500000    beta[2] = 0.219000
alpha[3] = 0.500000    beta[3] = 0.238500
alpha[4] = 0.500000    beta[4] = 0.261000
alpha[5] = 0.000000    beta[5] = 0.273000
m[5] = 0.092732
m[4] = 0.087536
m[3] = 0.079124
m[2] = 0.072967
m[1] = 0.067010
m[0] = 0.063995
Result: 3.003045
[alpha@Cameron numberCompute]$
```

10.

给定函数 $y=f(x)$ 函数表, 边界条件 $s'(0.25) = 1$, $s'(0.53) = 0.6868$

求 $f(0.35)$ 的近似值 (第一边界条件)

X_n Y_n

```
[alpha@Cameron numberCompute]$ cat data10
0.25 0.5
0.3 0.5477
0.39 0.6245
0.45 0.6708
0.53 0.728
[alpha@Cameron numberCompute]$
```

运行结果:

```
[alpha@Cameron numberCompute]$ ./fenDuan3YangTiao data10 0.35
Please select Boundary : 1 or 2
: 1
Please input m0 and mn : 1
0.6868
alpha[0] = 0.000000    beta[0] = 2.000000
alpha[1] = 0.357143    beta[1] = 2.754143
alpha[2] = 0.600000    beta[2] = 2.413000
alpha[3] = 0.428571    beta[3] = 2.242143
alpha[4] = 1.000000    beta[4] = 1.373600
m[4] = 0.686800
m[3] = 0.745217
m[2] = 0.800392
m[1] = 0.912716
m[0] = 1.000000
Result: 0.591607
[alpha@Cameron numberCompute]$
```

P97 第 8 题上机题, 要求 $\epsilon=10^{-6}$ 使用算法

8. 计算积分 $\int_1^9 \sqrt{x} dx$

1) 自动选取步长复化梯形

由于使用复化梯形求积分的近似值是一种比较有效的方法, 但是, 在使用复化梯形求积公式求积分之前必须选定 n , 如果 n 选取得太大, 虽然能是复化梯形求积公式与积分精确值的误差变得很小, 但会导致计算量的增加, 如果 n 取得太小, 复化梯形求积公式与积分精确值就会相差很大, 从而使得复化梯形求积公式的精度难以保证, 因此, 假定给定一个精度, 那如何选择 n 使得积分近似值满足精度要求? 为了避免盲目选取 n , 可以使用自动选取步长。

自动选取步长算法计算步骤为:

1) 先用梯形求积公式计算出积分的第一次近似值:

$$T_1 = \frac{b-a}{2} [f(a) + f(b)]$$

2) 把积分区间 $[a, b]$ 2 等分, 并用公式 (4.20) 求出积分的第二次近似值

$$T_2 = \frac{T_1}{2} + \frac{b-a}{2} f\left(a + \frac{b-a}{2}\right)$$

然后判别 $|T_2 - T_1| < 3\epsilon$ 是否成立, 若成立, 计算停止, T_2 作为积分近似值, 否则继续加密区间。

3) 把积分区间 $[a, b]$ 4 等分, 继续近似

$$T_4 = \frac{T_2}{2} + \frac{b-a}{4} \sum_{k=1}^2 f\left(a + (2k-1)\frac{b-a}{4}\right)$$

然后判别 $|T_4 - T_2| < 3\epsilon$ 是否成立, 若成立, 计算停止, T_4 作为积分近似值, 否则继续加密区间。

.....

这样一直做下去, 直到 $|T_{2n} - T_n| < 3\epsilon$ 成立

运行截图：

其中 a 为积分的下限

b 为积分的上限

e 为积分精度要求

函数 $f(x) = \sqrt{x}$ 已经在程序的 `double f(const double)` 中定义(可能需要使用 `-lm` 来编译)

可以看到积分出来（面积 square）为 17.33333

分成了 $n=1024$ 段

```
[alpha@Cameron numberCompute]$ ./autoLadder
Please input a: 1
Please input b: 9
Please input e: 0.000001
The square is 17.333333
n is 1024
[alpha@Cameron numberCompute]$
```

2) 自动步长复化抛物线

自动选取步长的复化 Simpson 道理跟自动选取步长的梯形求积算法道理类似，由于复化 Simpson 公式求积分算法需要制定 a, b, n, n 为分段数（n 需要为偶数），对于需要的精度 e，如果人为提前选定 n，保守点的 n 会满足精度，但是将有大量浪费计算，激进的 n 可能不符合精度要求，所以为了避免盲目第选取 n 来运行复化 Simpson，可以使用自动选取步长的算法。

自动步长复化抛物线（Simpson）算法：

① 输入 a, b, 定义 $f(x)$, 输入 e（精度要求）

② $n=2, h=(b-a)/2$;

③
$$S_2 = \frac{h}{3} [f(a) + f(b) + 4f(\frac{a+b}{2})]$$

$$T_1 = f(\frac{a+b}{2})$$

④ 区间加密

$n = 2 * n$;

$h = h / 2$;

$$T_2 = \sum_{i=0}^{n/2-1} f(a + (2*i+1)*h)$$

⑤ 计算 S_4

$$S_4 = \frac{h}{3} [f(a) + f(b) + 4T_2 + 2T_1]$$

⑥ 若 $|S_4 - S_2| \leq 15 * e$ 输出 n, S4, 停机

否则

S2 <--- S4

T1 <--- T1 + T2

转到 (4)

运行结果:

其中 a 为积分的下限

b 为积分的上限

e 为积分精度要求

函数 $f(x) = \sqrt{x}$ 已经在程序的 `double f(const double)` 中定义(可能需要使用 `-lm` 来编译)

可以看到 求出的积分为 17.33333

分段数 $n = 64$ 要比自动步长梯形的少得多

```
[alpha@Cameron numberCompute]$ ./autoSimpson
Please input a: 1
Please input b: 9
Please input e: 0.000001
Result: S = 17.333333
        n = 64
[alpha@Cameron numberCompute]$
```

3) Romberg 求积 (需要打印过程的三角形)

Romberg 求积法是应用 Richardson 外推算法的一个典型例子, Romberg 求积发的计算过程为:

$$T_n = T_0(h)$$

$$T_{2n} = T_0\left(\frac{h}{2}\right) \quad T_1(h)$$

$$T_{4n} = T_0\left(\frac{h}{2^2}\right) \quad T_1\left(\frac{h}{2}\right) \quad T_2(h)$$

$$T_{8n} = T_0\left(\frac{h}{2^3}\right) \quad T_1\left(\frac{h}{2^2}\right) \quad T_2\left(\frac{h}{2}\right) \quad T_3(h)$$

·
·
·

运行截图：

其中 a 为积分的下限

b 为积分的上限

e 为积分精度要求

被积函数 $f(x) = \sqrt{x}$ 已经在程序的 `double f(const double)` 中定义

图为过程使用的的下三角矩阵

可以看到 求出的积分为 17.33333

```
[alpha@Cameron numberCompute]$ ./romberg
Please input a: 1
Please input b: 9
Please input e: 0.000001
16.000000
16.944272      17.259029
17.227740      17.322230      17.326443
17.306001      17.332087      17.332744      17.332845
17.326420      17.333226      17.333302      17.333311      17.333313
17.331599      17.333326      17.333332      17.333333      17.333333      17.333333
17.332899      17.333333      17.333333      17.333333      17.333333      17.333333      17.333333
Result : 17.333333
[alpha@Cameron numberCompute]$
```


解线性方程组直接法

1) 顺序高斯消去法

高斯消去法是一个古老方法，基本思想为通过初等变换将方程组转化为一个等价的三角形方程组：

$$b_{11}x_1 + b_{12}x_2 + \dots + b_{1n}x_n = g_1$$

$$b_{22}x_2 + \dots + b_{2n}x_n = g_2$$

$$b_{nn}x_n = g_n$$

然后就可以逐个求出 x_n, x_{n-1}, \dots, x_1 , 这个过程成为回代。

例子：

系数矩阵 A:

12 -3 3

-18 3 -1

1 1 1

有端项 b:

15

-15

6

除数过小控制 e:

0.01

运行结果：

```
alpha@Cameron numberCompute]$ ./seq
X[0] = 1.000000
X[1] = 2.000000
X[2] = 3.000000
alpha@Cameron numberCompute]$
```

2) 列主元高斯

主元高斯消去法是为了控制舍入误差而提出来的一种算法，在顺序高斯消去的消元过程中，若出现 $a_{kk} = 0$ ，则消元无法进行，即使 $a_{kk} \neq 0$ ，但是其值很小，用它作为除数也会导致其他元素的巨大增长和舍入误差的扩散，引起解的失真。

列主元消去法除了每步需要按列选取主元并可能进行矩阵行交换外，其消元过程跟顺序高斯消去法的过程一样。

例子矩阵：

```
[alpha@Cameron numberCompute]$ cat matrix6
0.780 0.563
0.913 0.659

0.217
0.254

0.00001
[alpha@Cameron numberCompute]$
```

运行结果：

```
[alpha@Cameron numberCompute]$ cat matrix5
10 -7 0
-3 2.099 6
5 -1 5

7
3.901
6

0.001
[alpha@Cameron numberCompute]$
```

rix6
rix6
s matrix6

运行结果：

```
[alpha@Cameron numberCompute]$ ./colGauss matrix5
X[0] = 0.000000
X[1] = -1.000000
X[2] = 1.000000
[alpha@Cameron numberCompute]$
```

3) 全主元高斯消去

全主元高斯消去法与列主元高斯消去法类似，不同的是选取主元的范围不同，全主元高斯消去法在整个系数矩阵中找出绝对值最大的元素作为主元，以控制舍入误差的增长，将舍入误差控制在一个最小的范围，但是找出主元和交换行列次序需要花费大量机器时间。

矩阵例子，依次为 系数矩阵 A，右端项 b，除数控制 e：

```
X[2] = 1.000000
[alpha@Cameron numberCompute]$ cat matrix4
10 -7 0
-3 2 6
5 -1 5

7
4
6

0.001
[alpha@Cameron numberCompute]$ ./allGauss matrix4
X[0] = 1.400000
X[2] = 1.000000
X[1] = -1.000000
[alpha@Cameron numberCompute]$
```

运行结果：

```
0.001
[alpha@Cameron numberCompute]$ ./allGauss matrix4
X[0] = 1.400000
X[2] = 1.000000
X[1] = -1.000000
[alpha@Cameron numberCompute]$ ./sequenceGauss matrix4
X[0] = 0.000000
X[1] = -1.000000
X[2] = 1.000000
[alpha@Cameron numberCompute]$ ./colGauss matrix4
X[0] = 0.000000
X[1] = -1.000000
X[2] = 1.000000
[alpha@Cameron numberCompute]$
```

解线性方程组迭代法

解线性代数方程组 $AX=b$ 的一些直接法, A 是 $n \times n$ 的矩阵, X, b 都是 n 维向量, 二大多数计算过程均需要对系数矩阵进行分解, 因而一般不能保持 A 的稀疏性, 遇到偏微分方程数值求解, 会遇到大型稀疏矩阵, 直接法运算量大, 迭代法能充分利用系数矩阵的稀疏性, 当 n 较大, 能有效控制计算量。

1) 简单迭代 Jacobi

$$AX=b$$

$$A = B + D$$

$$B = D^{-1}(D - A) = I - D^{-1}A$$

$$g = D^{-1}b$$

方程可以表示为

$$X = BX + g$$

由此可以构造 Jacobi 迭代:

$$X^{(k+1)} = BX^{(k)} + g$$

当选定一个初始向量 X_0 后, 上面式子便产生一个向量序列:

$X_0, X_1, \dots, X_k, \dots$

以下为 Jacobi 运行情况:

系数矩阵 A

右端项 b

除数控制、精度 e

最大迭代次数 M

可以看到运行结果:

$$X[1]=1.099936$$

$$X[2]=1.199936$$

$$X[3]=1.299924$$

迭代次数 $K = 9$

```

[alpha@Cameron numberCompute]$ cat jacob1
10 -1 -2
-1 10 -2
-1 -1 5

7.2
8.3
4.2

0.001
100
[alpha@Cameron numberCompute]$ ./Jacobi jacob1
X[1] = 1.099936
X[2] = 1.199936
X[3] = 1.299924

K = 9
[alpha@Cameron numberCompute]$ 

```

2) Seidel;

Seidel 迭代特点在于计算第 i 个分量，用到前 $i-1$ 分量是最新算出的，这样有可能提高收敛速度。

实际运行效果，输入跟 Jacobi 的输入都是 jacob1 这个文件，可以看到迭代次数 $K=6$ ，比 Jacobi 快了 3 个迭代次数。

```

[alpha@Cameron numberCompute]$ ./Seidel jacob1
X[1] = 1.099986
X[2] = 1.199992
X[3] = 1.299996

K = 6
[alpha@Cameron numberCompute]$ 

```

3) SOR;

松弛法可以看做是 Seidel 迭代法的加速，但是这个过程跟 w 松弛因子有关，当 $w > 1$ 时候交超松弛； $w < 1$ 时候称为低松弛； $w = 1$ 就是 Seidel 迭代法：

Sor1 输入文件内容依次为

系数矩阵 A

右端项 b

初始向量 Y0

除数、精度控制 e

最大迭代次数 M

松弛因子 w

```
[alpha@Cameron numberCompute]$ cat sor1
10 -1 -2
-1 10 -2
-1 -1 5

7.2
8.3
4.2

0
0
0

0.001
100
0.5
[alpha@Cameron numberCompute]$
```

```
[alpha@Cameron numberCompute]$ ./SOR sor1
X[1] = 1.099417
X[2] = 1.199474
X[3] = 1.299477

K = 17
[alpha@Cameron numberCompute]$
```

附录（源码）

A. 三次样条插值实现代码

```
double fenDuan3YangTiao( const double *x , const double *y , const size_t length,  
const double a0 , const double an , const double b0 , const double bn , const double  
queryX )
```

```
{  
  
    double *alpha;  
  
    double *beta;  
  
    alpha = malloc( sizeof( double ) * ( length ) );  
  
    beta = malloc( sizeof( double ) * ( length ) );  
  
  
    double *aArray;  
  
    double *bArray;  
  
    aArray = malloc( sizeof( double ) * ( length ) );  
  
    bArray = malloc( sizeof( double ) * ( length ) );  
  
  
    alpha[0] = a0;  
  
    alpha[length-1] = an;  
  
  
    beta[0] = b0;  
  
    beta[length-1] = bn;
```

```

printf("alpha[0] = %lf    beta[0] = %lf\n", alpha[0] , beta[0] );

size_t i;

for( i=1 ; i<length-1 ; i++ )

{

    alpha[i] = ( x[i] - x[i-1] ) / ( x[i+1] - x[i-1] );

    beta[i] = 3 * ( (1-alpha[i]) / (x[i]-x[i-1]) * (y[i] - y[i-1]) + alpha[i] / ( x[i+1]
- x[i] ) * (y[i+1] - y[i] ) );

    printf("alpha[%d] = %lf    beta[%d] = %lf\n", i , alpha[i] , i , beta[i] );

}

printf("alpha[%d] = %lf    beta[%d] = %lf\n", i , alpha[i] , i , beta[i] );


aArray[0] = -1 * alpha[0] / 2;

bArray[0] = beta[0] / 2;

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

{

    aArray[i] = -1 * alpha[i] / ( 2 + ( 1 - alpha[i] ) * aArray[i-1] ) ;

    bArray[i] = ( beta[i] - (1-alpha[i] ) * bArray[i-1] ) / ( 2 + ( 1 - alpha[i] )
* aArray[i-1] ) ;

}

```



```

double *m;

m = malloc( sizeof( double ) * ( length + 1 ) );

m[length] = 0;


int j;

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

{

m[j] = aArray[j] * m[j+1] + bArray[j];

printf("m[%d] = %lf\n" , j , m[j] );

}


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

{

if( queryX < x[i] )

    break;

}


double yy;

double t1 = ( queryX - x[i] ) / ( x[i-1] - x[i] );

double t2 = ( queryX - x[i-1] ) / ( x[i] - x[i-1] );

```

```

yy = (1+2*t2)*t1*t1*y[i-1];

yy +=( 1 + 2*t1)*t2*t2*y[i];

yy +=(queryX - x[i-1] ) * t1 * t1 * m[i-1];

yy +=(queryX - x[i] ) * t2 * t2 * m[i];


free( m );

free( aArray );

free( bArray );

free( alpha );

free( beta );


return yy;

}

```

B. 自动选取步长复化梯形源码

```

double autoLadder( const double a , const double b , const double e, int *p_n , double
( * f ) ( const double x) )
{
    double h;
    double T0,T1;
    double s;
    int k;

    h = ( b - a ) / 2;
    T1 = ( f(a) + f(b) ) * h;
    * p_n = 1;

    while( 1 )
    {
        T0 = T1;

```

```

s = 0;

for( k=1 ; k <= (* p_n) ; k++ )
{
    s = s + f( a + ( 2 * k - 1 ) * h / ( * p_n ) );
}

T1 = T0 / 2 + s * h / ( * p_n );

if( fabs( T1 - T0 ) < 3*e )
{
    return T1;
}
else
{
    * p_n = 2 * ( *p_n );
}
}

return 0.0; //keep the compiler happy
}

```

C. 自动选取步长复化抛物线

```

double autoSimpson( const double a , const double b , const double e , size_t * p_n ,
double ( * f ) ( const double x ) )

```

```

{

    double h;

    double T1,T0,T2;

    double S1,S2,S4;

    int n;

    T0 = f(a) + f(b);

    T1 = f( ( a + b ) / 2 );

```

```
h = ( b - a ) / 2;
```

```
n = 2;
```

```
S2 = h / 3 * ( T0 + 4*T1 );
```

```
while( 1 )
```

```
{
```

```
int i;
```

```
    n= 2 * n;
```

```
    h = h / 2;
```

```
T2 = 0;
```

```
for( i=0 ; i<=(n/2-1) ; i++ )
```

```
{
```

```
    T2 += f( a + ( 2*i +1)*h );
```

```
}
```

```
S4 = h / 3 * ( T0 + 2 * T1 + 4*T2 );
```

```
if( fabs( S4 - S2 ) < 15*e )
```

```
    break;
```

```

else

{

    S2 = S4;

    T1 = T1 + T2;

}

}

* p_n = n;

return S4;

}

```

D. Romberg

```

double romberg( const double a , const double b , const double e, double ( * f )
(const double x ))
{
    double T[MATRIX_SIZE][MATRIX_SIZE];
    size_t k;
    size_t c2k1; //means 2^(k-1)
    double temp;

    T[0][0] = ( b - a ) / 2 * ( f(a) + f(b) );

    k=1;
    c2k1 = 1;

    while( 1 )
    {
        size_t i;
        temp=0;
        for( i=1; i<=c2k1 ; i++ )
        {
            temp += f( a + ( 2 * i - 1 ) * ( b - a ) / ( c2k1 * 2 ) );
        }
    }
}

```

```

T[0][k] = 0.5 * ( T[0][k-1] + ( b - a ) / c2k1 * temp );

size_t m;
int c4m=4;
for( m = 1 ; m <= k ; m++ )
{
    T[m][k-m] =( c4m * T[m-1][k-m+1] - T[m-1][k-m] )/(c4m - 1);
    c4m *=4;
}

if( fabs( T[k][0] - T[k-1][0] ) < e )
{
    //finish calcalating print the trangle
    int i,j;
    for( i=0 ; i <= k ; i++ )
    {
        for( j=i ; j>=0 ; j-- )
        {
            printf("%lf\t" , T[i-j][j] );
        }
        printf("\n");
    }

    return T[k][0];
}

k++;
c2k1 *=2; //why?
}

return 0.0; //to keep the compiler happy
}

```

E. 顺序高斯

```

void sequenceGauss( double m[][MATRIX_SIZE] , const int n , const double e)
{
    int k;
    int i;
    int j;

    double x[MATRIX_SIZE];

    for( k=0 ; k < n-1 ; k++ )

```

```

{
if( fabs( m[k][k] ) <= e )
{
    fprintf( stderr , "Sequence Gauss Can't solve it!");
    return;
}

for( i=k+1 ; i < n ; i++ )
{
    double t = m[i][k] / m[k][k];
    for( j=k+1 ; j<=n ; j++ )
    {
        m[i][j] = m[i][j] - t * m[k][j];
    }
}
}

if( fabs( m[n-1][n-1] ) <= e )
{
    fprintf( stderr , "Sequence Gausse Can't solve it!\n");
    return ;
}

x[n-1] = m[n-1][n] / m[n-1][n-1];
double tSum;
for( i=n-2 ; i>=0 ; i-- )
{
    tSum=0;
    for( j=i+1 ; j<n ; j++ )
    {
        tSum += m[i][j] * x[j];
    }

    x[i] = ( m[i][n] - tSum ) / m[i][i];
}

//now the x[...] store the answers print them out
for( i=0 ; i<n ; i++ )
{
    printf("X[%d] = %lf\n" , i , x[i]);
}

```

```
}
```

F. 列主元高斯

```
void adjustMatrix( double m[][MATRIX_SIZE] , const int k , const int n)
{
    double absMax;
    int w,row;

    absMax = m[k][k];
    row = k;
    for( w=k+1; w<n ; w++ )
    {
        if( fabs( m[w][k] ) > absMax )
        {
            absMax = fabs( m[w][k] );
            row = w;
        }
    }

    if( row == k )    //the currunt row is the absMax no need to do
        return;

    double temp;
    for( w=k ; w<=n ; w++ )
    {
        temp = m[k][w];
        m[k][w] = m[row][w];
        m[row][w] = temp;
    }
}
```

```
void colGauss( double m[][MATRIX_SIZE] , const int n , const double e)
{
    int k;
    int i;
    int j;

    double x[MATRIX_SIZE];

    for( k=0 ; k < n-1 ; k++ )
    {
        adjustMatrix( m, k ,n);
    }
}
```



```

if( fabs( m[k][k] ) <= e )
{
    fprintf( stderr , "Sequence Gauss Can't solve it!");
    return;
}

for( i=k+1 ; i < n ; i++ )
{
    double t = m[i][k] / m[k][k];
    for( j=k+1 ; j<=n ; j++ )
    {
        m[i][j] = m[i][j] - t * m[k][j];
    }
}

if( fabs( m[n-1][n-1] ) <= e )
{
    fprintf( stderr , "Sequence Gausse Can't solve it!\n");
    return ;
}

x[n-1] = m[n-1][n] / m[n-1][n-1];
double tSum;
for( i=n-2 ; i>=0 ; i-- )
{
    tSum=0;
    for( j=i+1 ; j<n ; j++ )
    {
        tSum += m[i][j] * x[j];
    }

    x[i] = ( m[i][n] - tSum ) / m[i][i];
}

//now the x[...] store the answers print them out
for( i=0 ; i<n ; i++ )
{
    printf("X[%d] = %lf\n" , i , x[i]);
}

```

```
}
```

G. 全主元高斯

```
void adjustMatrix( double m[][MATRIX_SIZE], const int k, const int n, int * order )
```

```
{
```

```
    double absMax;
```

```
    int r, c, row, col;
```

```
    absMax = m[k][k];
```

```
    row = k;
```

```
    col = k;
```

```
    for( r=k; r<n ; r++ )
```

```
    {
```

```
        for( c=k; c<n ; c++ )
```

```
        {
```

```
            if( fabs( m[r][c] ) > absMax )
```

```
            {
```

```
                absMax = fabs( m[r][c] );
```

```
                row = r;
```

```
                col = c;
```

```
            }
```

```
        }
```

```
    }
```

```

//exchange the row

double temp;

int w;

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

{

temp = m[k][w];

m[k][w] = m[row][w];

m[row][w] = temp;

}

```

```

//exchange the column

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

{

temp = m[w][k];

m[w][k] = m[w][col];

m[w][col] = temp;

}

```

```

//exchange the order

int it;

it = order[k];

```

```

    order[k] = order[col];

    order[col] = it;

}

void colGauss( double m[][MATRIX_SIZE] , const int n , const double e)
{
    int k;

    int i;

    int j;

    double x[MATRIX_SIZE];

    int order[MATRIX_SIZE];

    //get the order ordered..

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

    {
        order[k] = k;
    }

    for( k=0 ; k < n-1 ; k++ )

    {

```

```

adjustMatrix( m, k ,n , order );

if( fabs( m[k][k] ) <= e )

{

    fprintf( stderr , "Sequence Gauss Can't solve it!");

    return;

}


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

{

    double t = m[i][k] / m[k][k];

    for( j=k+1 ; j<=n ; j++ )

    {

        m[i][j] = m[i][j] - t * m[k][j];

    }

}

}

if( fabs( m[n-1][n-1] ) <= e )

{

    fprintf( stderr , "Sequence Gausse Can't solve it!\n");

    return ;
}

```

```
}
```

```
x[n-1] = m[n-1][n] / m[n-1][n-1];
```

```
double tSum;
```

```
for( i=n-2 ; i>=0 ; i-- )
```

```
{
```

```
tSum=0;
```

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

```
{
```

```
    tSum += m[i][j] * x[j];
```

```
}
```

```
x[i] = ( m[i][n] - tSum ) / m[i][i];
```

```
}
```

```
//now the x[...] store the answers print them out
```

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

```
{
```

```
printf("X[%d] = %lf\n" , order[i] , x[i]);
```

```
}
```

```
}
```

H. 简单 Jacobi 迭代

```
//initial Y
for( i = 0 ; i < n ; i++ )
{
    Y[i] = 0;
}
//now the matrix m , vector b , size n , e is usable

int k=1;
double T;
for( i=0 ; i<n ; i++ )
{
    if( fabs( m[i][i] ) < e )
    {
        fprintf( stderr , "Failed!" );
        exit( 1 );
    }

    T = m[i][i];

    for( j=0 ; j<n ; j++ )
    {
        m[i][j] = -1 * m[i][j] / T;
    }

    m[i][i] = 0;
    g[i] = b[i] / T;
}

while( 1 )
{
    for( i=0 ; i<n ; i++ )
    {
        double sum = 0;
        for( j=0 ; j<n ; j++ )
        {
            if( i == j )
                continue;
```

```

        else
        {
            sum += m[i][j] * Y[j];
        }
    }

    X[i] = g[i] + sum;
}

double s=0;
for( i=0 ; i<n ; i++ )
{
    s += fabs( X[i] - Y[i] );
}

if( s < e )
{
    for( i=0 ; i<n ; i++ )
    {
        printf("X[%d] = %lf\n", i+1 , X[i]);
    }

    printf("\n K = %d\n" , k );
    break;
}

if( k < M )
{
    k++;
    for( i = 0 ; i < n ; i++ )
    {
        Y[i] = X[i] ;
    }
}
else
{
    fprintf( stderr , "Failed!\n");
    break;
}

}

```

I.Seidel 迭代
 //initial Y


```

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

{

Y[i] = 0;

X[i] = Y[i];

}

//now the matrix m , vector b , size n , e is usable


int k=1;

double T;

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

{

if( fabs( m[i][i] ) < e )

{

    fprintf( stderr , "Failed!" );

    exit( 1 );

}

T = m[i][i];

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

{

    m[i][j] = -1 * m[i][j] / T;

```

```

}

m[i][i] = 0;

g[i] = b[i] / T;

}

while( 1 )

{

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

{

double sum = 0;

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

{

    if( i == j )

        continue;

    else

    {

        sum += m[i][j] * X[j];

    }

}

}

```

```

X[i] = g[i] + sum;

}


double s=0;

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

{

s += fabs( X[i] - Y[i] );

}


if( s < e )

{

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

{

printf("X[%d] = %lf\n", i+1 , X[i]);

}

printf("\n K = %d\n" , k );

break;

}


if( k < M )

{

```

```

k++;

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

{

    Y[i] = X[i] ;

}

}

else

{

    fprintf( stderr , "Failed!\n");

    break;

}

}

```

J. SOR 迭代

```

//initial X with Y

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

{

    X[i] = Y[i];

}

//now the matrix m , vector b , size n , e is usable

int k=1;

```

```

double T;

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

{

if( fabs( m[i][i] ) < e )

{

    fprintf( stderr , "Failed!" );

    exit( 1 );

}

T = m[i][i];

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

{

    m[i][j] = -1 * m[i][j] * w / T;

}

m[i][i] = 1-w;

g[i] = w * b[i] / T;

}

while( 1 )

{

```

```

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

{

double sum = 0;

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

{

    sum += m[i][j] * X[j];

}

```

```

X[i] = g[i] + sum;

}

```

```

double s=0;

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

{

s += fabs( X[i] - Y[i] );

}

```

```

if( s < e )

{

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

{

```

```
    printf("X[%d] = %lf\n", i+1 , X[i]);  
  
}
```

```
printf("\n K = %d\n" , k );  
  
break;  
  
}
```

```
if( k < M )  
  
{  
  
k++;  
  
for( i = 0 ; i < n ; i++ )  
  
{  
  
    Y[i] = X[i] ;  
  
}  
  
}
```

```
else  
  
{  
  
fprintf( stderr , "Failed!\n");  
  
break;  
  
}  
  
}
```