数值分析实验报告

课题一　迭代格式的比较

一、问题提出

设方程　$f(x)=x^3-3x-1=0$ 有三个实根 $x_1^*=1.8793$, $x_2^*=-0.34727$,$x_3^*=-1.53209$

现采用下面三种不同计算格式，求 $f(x)=0$ 的根 x_1^* 或 x_2^*

1、 $x=\dfrac{3x+1}{x^2}$

2、 $x=\dfrac{x^3-1}{3}$

3、 $x=\sqrt[3]{3x+1}$

二、要求

1、编制一个程序进行运算，最后打印出每种迭代格式的敛散情况；

2、用事后误差估计 $\left|x_{k+1}-x_k\right|<\varepsilon$ 来控制迭代次数，并且打印出迭代的次数；

3、初始值的选取对迭代收敛有何影响；

4、分析迭代收敛和发散的原因。

三、目的和意义

1、通过实验进一步了解方程求根的算法；

2、认识选择计算格式的重要性；

3、掌握迭代算法和精度控制；

4、明确迭代收敛性与初值选取的关系。

程序代码：

```
#include<iostream>
#include<cmath>
#include<cstdlib>
using namespace std;
double f(double i)          //外调函数f(x),每次更新新的函数
{//以第一种迭代方式为例子
    double k,m,sum;
    k=3*i+1;
    m=pow(i,2.0);
    sum=k/m;
    return sum;
}
int main()
{
    double x,x0;
    int N;//最大迭代次数
    int k;
    cout<<"输入初解:";
    cin>>x0;
    cout<<"输入最大迭代次数:";
    cin>>N;
     for(k=1;k<=N;k++)
```

```
{
    x=f(x0);
    if(fabs(x-x0)<0.0000001)
    {
        cout<<"迭代次数:"<<k<<endl;
        cout<<"输出得到的解:"<<x<<endl;
        system("pause");
        return 0;
    }
    else x0=x;
}
cout<<"已达到最大迭代次数:"<<N<<endl;
cout<<"输出得到的解:"<<x<<endl;
    system("pause");
    return 0;
}
```

实验结果：

```
输入初解:-1.5
输入最大迭代次数:50
迭代次数:42
输出得到的解:-1.53209
请按任意键继续. . .
```

```
输入初解:-0.3
输入最大迭代次数:50
已达到最大迭代次数:50
输出得到的解:-1.#IND
请按任意键继续. . .
```

```
输入初解:-0.3
输入最大迭代次数:50
迭代次数:8
输出得到的解:-0.347296
请按任意键继续. . .
```

```
输入初解:1.5
输入最大迭代次数:50
迭代次数:10
输出得到的解:-0.347296
请按任意键继续. . .
```

```
输入初解:1.5
输入最大迭代次数:50
迭代次数:13
输出得到的解:1.87939
请按任意键继续. . .
```

```
输入初解:2.0
输入最大迭代次数:50
迭代次数:12
输出得到的解:1.87939
请按任意键继续. . .
```

四、程序运行结果讨论和分析：

对于第一种迭代格式，收敛区间[-8.2　-0.4]，在该收敛区间内迭代收敛于-1.53209，只能求得方程的一个根；

对于第二种迭代格式，收敛区间[-1.5　1.8]，在该收敛区间内迭代收敛于-0.34730，同样只能求得方程的一个根；

对于第三种迭代格式，收敛区间[-0.3　+∞)，在该收敛区间内迭代收敛于 1.87937，只能求得方程的一个根；

由以上结果很容易发现，初值的选取对迭代敛散性有很大影响。以第一种迭代格式为例，当初值大于等于-0.3 时，迭代格式发散；当初值小于等于-8.3 时，迭代格式也发散；只有初值在-0.3 和-8.3 之间时，迭代格式才收敛于—1.53209。其他迭代格式也有这样的性质，即收敛于某个数值区间，超出这个区间迭代格式就是发散的，这就是所谓迭代格式的收敛性。

对于不同迭代格式在不同区间具有不同的敛散性的原因，我认为可以从一下两方面理解：1、迭代法是一种逐次逼近法，其基本思想是将隐式方程归结为一组显式的计算公式，就是说，迭代过程实质上是个逐步显式化的过程。2、我们可以用几何图像来更好地理解迭代过程。由图可知，在某些区间选取的初始值随着迭代次数的增加会越来越逼近精确值，即收敛于精确值，而在另外一些区间选取的初始值随着迭代次数的增加却离精确值越来越远，即不会收敛于一个确定值。

课题二　线性方程组的直接算法

一、问题提出

给出下列几个不同类型的线性方程组，请用适当算法计算其解。

1、设线性方程组

$$\begin{bmatrix} 4 & 2 & -3 & -1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 8 & 6 & -5 & -3 & 6 & 5 & 0 & 1 & 0 & 0 \\ 4 & 2 & -2 & -1 & 3 & 2 & -1 & 0 & 3 & 1 \\ 0 & -2 & 1 & 5 & -1 & 3 & -1 & 1 & 9 & 4 \\ -4 & 2 & 6 & -1 & 6 & 7 & -3 & 3 & 2 & 3 \\ 8 & 6 & -8 & 5 & 7 & 17 & 2 & 6 & -3 & 5 \\ 0 & 2 & -1 & 3 & -4 & 2 & 5 & 3 & 0 & 1 \\ 16 & 10 & -11 & -9 & 17 & 34 & 2 & -1 & 2 & 2 \\ 4 & 6 & 2 & -7 & 13 & 9 & 2 & 0 & 12 & 4 \\ 0 & 0 & -1 & 8 & -3 & -24 & -8 & 6 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \\ 3 \\ 2 \\ 3 \\ 46 \\ 13 \\ 38 \\ 19 \\ -21 \end{bmatrix}$$

$$x^* = (1, \ -1, \ 0, \ 1, \ 2, \ 0, \ 3, \ 1, \ -1, \ 2)^T$$

2、设对称正定阵系数阵线方程组

$$\begin{bmatrix} 4 & 2 & -4 & 0 & 2 & 4 & 0 & 0 \\ 2 & 2 & -1 & -2 & 1 & 3 & 2 & 0 \\ -4 & -1 & 14 & 1 & -8 & -3 & 5 & 6 \\ 0 & -2 & 1 & 6 & -1 & -4 & -3 & 3 \\ 2 & 1 & -8 & -1 & 22 & 4 & -10 & -3 \\ 4 & 3 & -3 & -4 & 4 & 11 & 1 & -4 \\ 0 & 2 & 5 & -3 & -10 & 1 & 14 & 2 \\ 0 & 0 & 6 & 3 & -3 & -4 & 2 & 19 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} 0 \\ -6 \\ 20 \\ 23 \\ 9 \\ -22 \\ -15 \\ 45 \end{bmatrix}$$

$$x^* = (1, \ -1, \ 0, \ 2, \ 1, \ -1, \ 0, \ 2)^T$$

3、三对角形线性方程组

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 7 \\ 5 \\ -13 \\ 2 \\ 6 \\ -12 \\ 14 \\ -4 \\ 5 \\ -5 \end{bmatrix}$$

$$x^* = (2, \ 1, \ -3, \ 0, \ 1, \ -2, \ 3, \ 0, \ 1, \ -1)^T$$

二、要求

1、 对上述三个方程组分别利用 Gauss 顺序消去法与 Gauss 列主元消去法；平方根法与

改进平方根法；追赶法求解（选择其一）；

2、 应用结构程序设计编出通用程序；

3、 比较计算结果，分析数值解误差的原因；

4、 尽可能利用相应模块输出系数矩阵的三角分解式。

三、目的和意义

1、通过该课题的实验，体会模块化结构程序设计方法的优点；

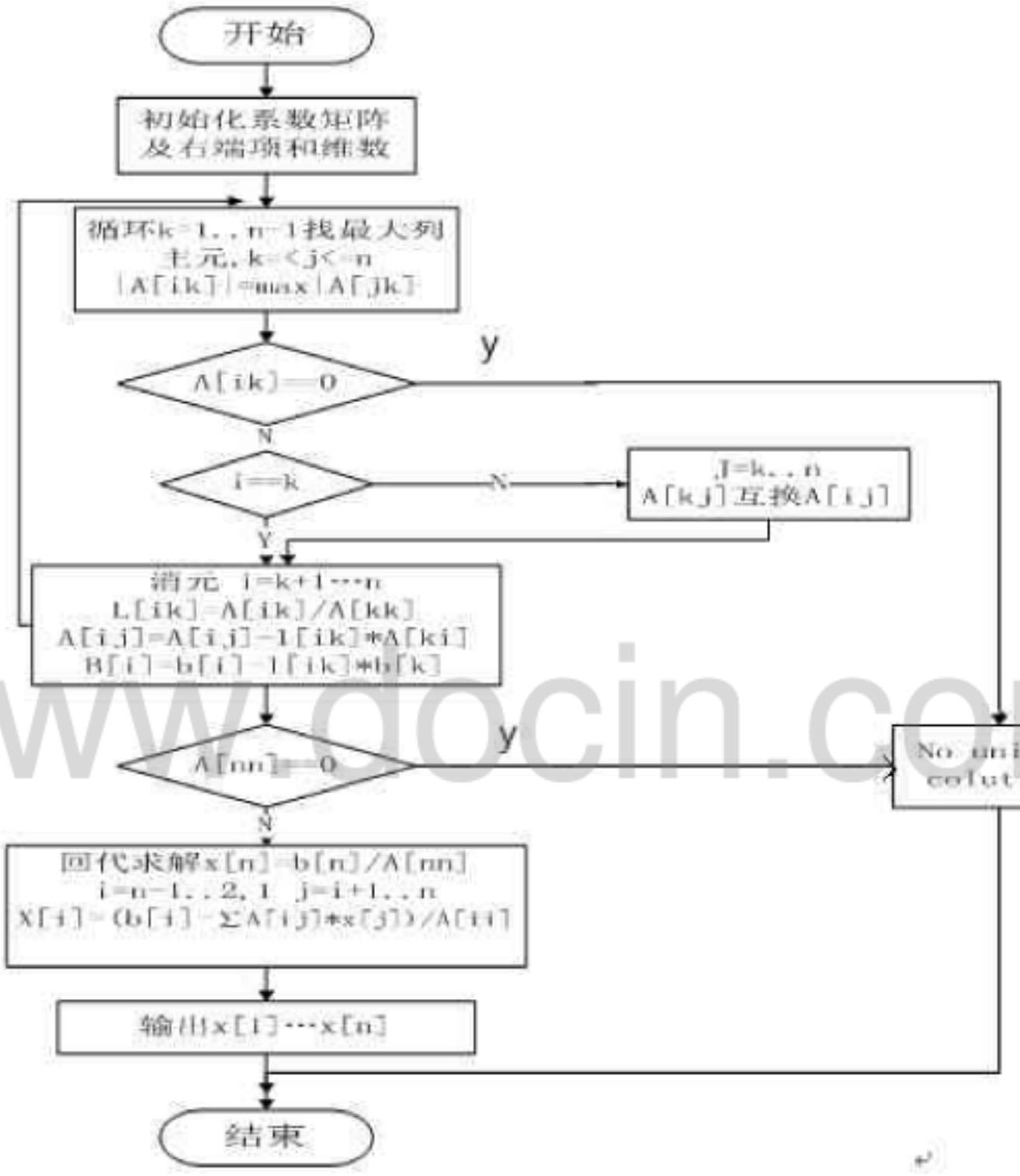
2、运用所学的计算方法，解决各类线性方程组的直接算法；

3、提高分析和解决问题的能力，做到学以致用；

4、通过三对角形线性方程组的解法，体会稀疏线性方程组解法的特点。

程序代码：

1. Gsuss 列主元消去法




```
#include<iostream>
#include<cmath>
#include<cstdlib>
using namespace std;
int main()
{
    int n,  i,j,k,m;
```

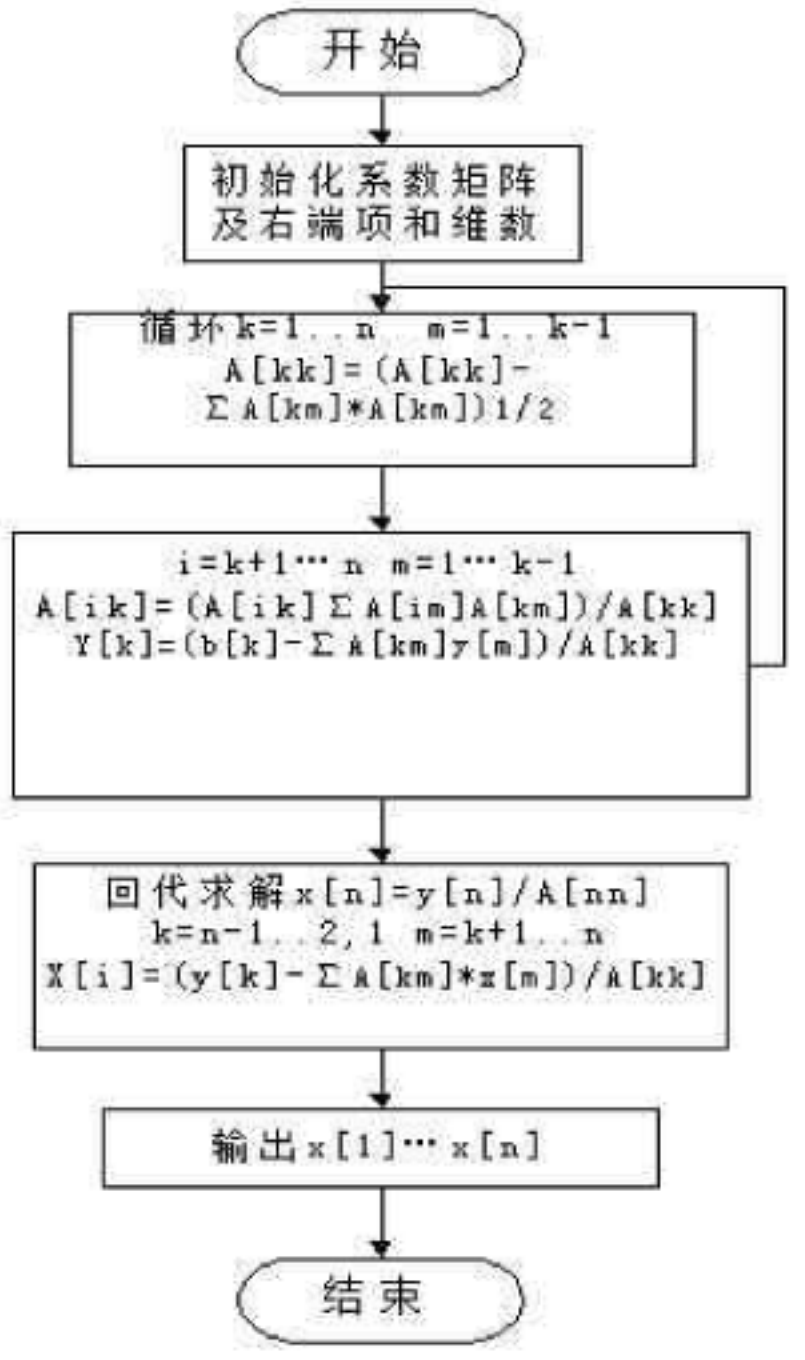
```
cout<<"输入维数:";
 cin>>n;
float **A=new double*[(n+1)];
for(i=1;i<=n;i++)
    A[i]=new double[n+1];
float *b=new double[n+1];
float *x=new double[n+1];
float l;
float temp1,temp2,temp3;
cout<<"输入系数矩阵A[][]:"<<endl;
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        cin>>A[i][j];
    cout<<"输入向量b[]:";
    for(i=1;i<=n;i++)
        cin>>b[i];
    cout<<endl;
     for(k=1;k<n;k++)
    {
        temp1=abs(A[k][k]);
        m=k;
        for(i=k;i<=n;i++)//找最大值的列主元
         {
            if(temp1<abs(A[i][k]))  {temp1=abs(A[i][k]);m=i;}//m是确定
的最列主元的行标
        }
        if(temp1==0)  cout<<"no unique solution!"<<endl; exit(0);
        if(m!=k)//换行
        {
            for(j=1;j<=n;j++)
            {
                temp2=A[k][j];
                A[k][j]=A[m][j];
                A[m][j]=temp2;
            }
            temp3=b[k];
            b[k]=b[m];
            b[m]=temp3;
    }
    for(i=k+1;i<=n;i++)//消元
    {
        l=A[i][k]/A[k][k];
        for(j=k+1;j<=n;j++)
        {
```

```
            A[i][j]=A[i][j]-l*A[k][j];
        }
        b[i]=b[i]-l*b[k];
    }
}
if(A[n][n]==0)
{
    cout<<"no unique solution!"<<endl; exit(0);
}
x[n]=b[n]/A[n][n];//回代求解
for(i=n-1;i>=1;i--)
{
    float sum=0;
     for(j=i+1;j<=10;j++)
           sum=sum+A[i][j]*x[j];
    x[i]=(b[i]-sum)/A[i][i];
}
cout<<"输出结果向量x[]:"<<endl;
for(i=1;i<=10;i++)  cout<<x[i]<<endl;;
system("pause");
return 0;
}
```

```
输入系数矩阵A[][]:
4  2  -3  -1  2  1  0  0  0  0
8  6  -5  -3  6  5  0  1  0  0
4  2  -2  -1  3  2  -1  0  3  1
0  -2  1  5  -1  3  -1  1  9  4
-4  2  6  -1  6  7  -3  3  2  3
8  6  -8  5  7  17  2  6  -3  5
0  2  -1  3  -4  2  5  3  0  1
16  10  -11  -9  17  34  2  -1  2  2
4  6  2  -7  13  9  2  0  12  4
0  0  -1  8  -3  -24  -8  6  3  -1
输入向量b[]:5 12 3 2 3 46 13 38 19 -21

输出结果向量x[]:
1.00001
-1.00001
7.10323e-006
0.999997
2
1.46832e-007
3
1.00001
-1
2
请按任意键继续. . .
```

2. 平方根法



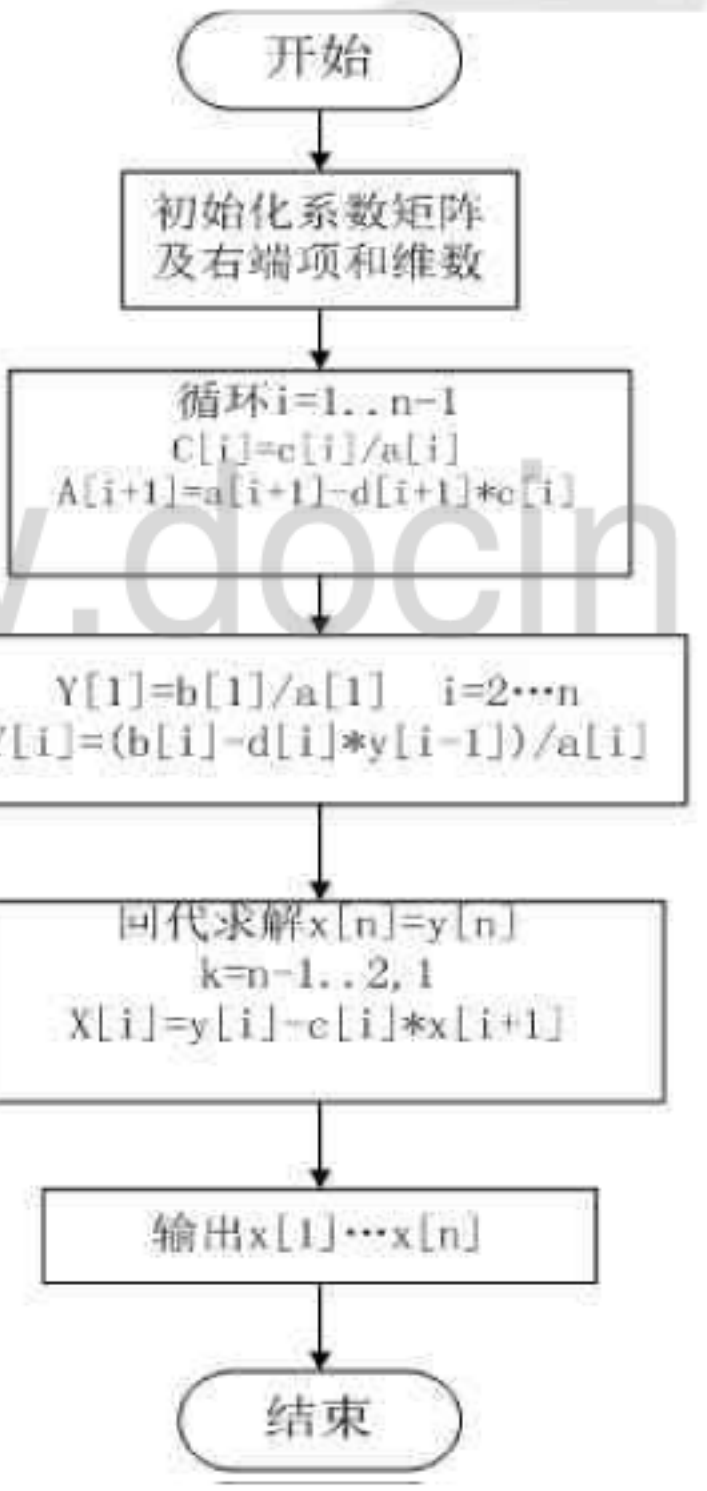

```
#include<iostream>
#include<cmath>
#include<cstdlib>
using namespace std;
int main()
{
    int n,i,j,k,m;
    cout<<"输入维数:";
    cin>>n;
    double **A=new double*[(n+1)];
    for(i=1;i<=n;i++)
        A[i]=new double[n+1];
    double *b=new double[n+1];
    double *x=new double[n+1];
    double *y=new double[n+1];
    cout<<"输入系数对称正定矩阵A[][]:"<<endl;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            cin>>A[i][j];
    cout<<"输入向量b[]:";
    for(i=1;i<=n;i++)
```

```
    cin>>b[i];
cout<<endl;
for(k=1;k<=n;k++)
{
    double sum=0;
    for(m=1;m<=k-1;m++)
    {
        sum=sum+pow(A[k][m],2.0);
    }
    sum=A[k][k]-sum;
    A[k][k]=sqrt(sum);
    for(i=k+1;i<=n;i++)
    {
        double temp1=0;
        for(m=1;m<=k-1;m++)
        {
            temp1=temp1+A[i][m]*A[k][m];
        }
        temp1=A[i][k]-temp1;
        A[i][k]=temp1/A[k][k];
    }
    double temp2=0;
    for(m=1;m<=k-1;m++)
    {
        temp2=temp2+A[k][m]*y[m];
    }
    y[k]=(b[k]-temp2)/A[k][k];
}
x[8]=y[8]/A[8][8];
for(k=n-1;k>=1;k--)
{
    double temp3=0;
    for(m=k+1;m<=n;m++)
    {
        temp3=temp3+A[m][k]*x[m];
    }
    x[k]=(y[k]-temp3)/A[k][k];
}
cout<<"输出结果向量x[]:"<<endl;
for(i=1;i<=n;i++)  cout<<x[i]<<endl;;
system("pause");
return 0;
}
```

```
输入维数:8
输入系数对称正定矩阵A[][]:
4 2 -4 0 2 4 0 0
2 2 -1 -2 1 3 2 0
-4 -1 14 1 -8 -3 5 6
0 -2 1 6 -1 -4 -3 3
2 1 -8 -1 22 4 -10 -3
4 3 -3 -4 4 11 1 -4
0 2 5 -3 -10 1 14 2
0 0 6 3 -3 -4 2 19
输入向量b[]:0 -6 20 23 9 -22 -15 45

输出结果向量x[]:
121.148
-140.113
29.7515
-60.1528
10.912
-26.7963
5.42593
-2.01852
请按任意键继续. . .
```

3. 追赶法




```
#include<iostream>
#include<cmath>
```

```
#include<cstdlib>
using namespace std;
int main()
{
    int n,i;
    cout<<"输入系数矩阵的维数:";
    cin>>n;
    double *a=new double[n+1];
    double *c=new double[n+1];
    double *d=new double[n+1];
    double *b=new double[n+1];
    double *x=new double[n+1];
    double *y=new double[n+1];
    cout<<"输入系数矩阵A[]数据: "<<endl;
    for(i=1;i<=n;i++) cin>>a[i];
    for(i=1;i<=n;i++) cin>>c[i];
    for(i=1;i<=n;i++) cin>>d[i];
    cout<<"输入b[] :"<<endl;
    for(i=1;i<=n;i++) cin>>b[i];
    for(i=1;i<=n-1;i++)
    {
        c[i]=c[i]/a[i];
        a[i+1]=a[i+1]-d[i+1]*c[i];
    }
    cout<<"输出解向量a[]:"<<endl;
    for(i=1;i<=n;i++)cout<<a[i]<<endl;
    cout<<"输出解向量c[]:"<<endl;
    for(i=1;i<=n;i++)cout<<c[i]<<endl;
    y[1]=b[1]/a[1];
    for(i=2;i<=n;i++)
    {
        y[i]=(b[i]-d[i]*y[i-1])/a[i];
    }
    cout<<"输出解向量y[]:"<<endl;
    for(i=1;i<=n;i++)cout<<y[i]<<endl;
    x[n]=y[n];
    for(i=n-1;i>=1;i--)
    {
        x[i]=y[i]-c[i]*x[i+1];
    }
    cout<<"输出解向量x[]:"<<endl;
    for(i=1;i<=n;i++)cout<<x[i]<<endl;
    system("pause");
    return 0;
```

}

```
输入系数矩阵的维数:10
输入系数矩阵A[]数据:
4 4 4 4 4 4 4 4 4 4
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
输入b[] :
7 5 -13 2 6 -12 14 -4 5 -5
输出解向量a[]:
4
3.75
3.73333
3.73214
3.73206
3.73205
3.73205
3.73205
3.73205
```

```
输出解向量c[]:
-0.25
-0.266667
-0.267857
-0.267943
-0.267949
-0.267949
-0.267949
-0.267949
-0.267949
-1
输出解向量y[]:
1.75
1.8
-3
-0.267943
1.5359
-2.80385
3
-0.267949
1.26795
-1
```

```
输出解向量x[]:
2
1
-3
1.11022e-016
1
-2
3
-5.55112e-017
1
-1
请按任意键继续. . .
```

四、　程序运行结果分析

在方法的选择上存在一定的误差，可以选一些更准确的方法求解；程序中对变量的类型设定，若设成 double 型，结果可以更精确；计算机在做运算时，会根据需要对中间结果进行舍入，这也会对最终结果有影响；

课题三　线性方程组的迭代法

一、问题提出

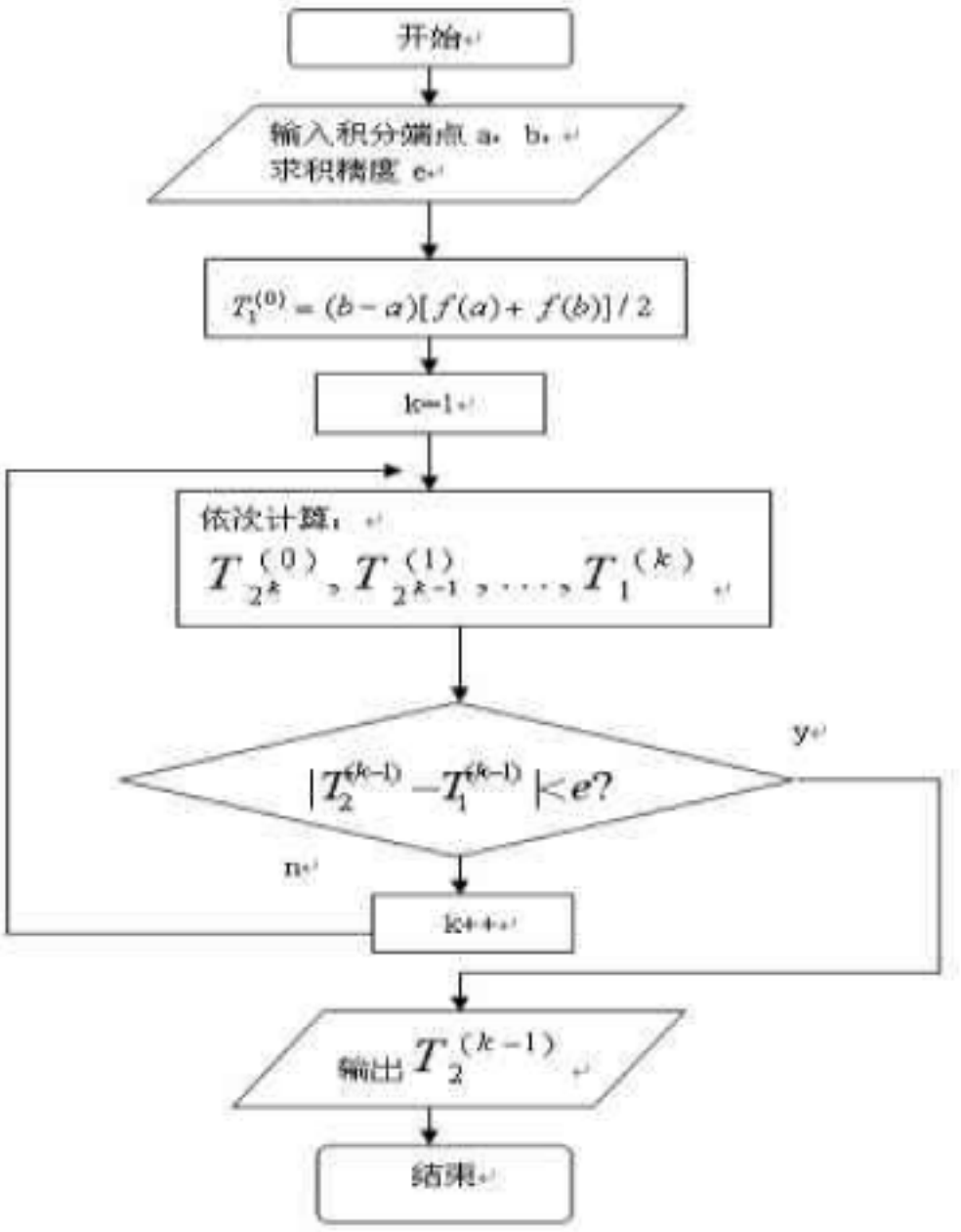
1、设线性方程组

$$\begin{bmatrix} 4 & 2 & -3 & -1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 8 & 6 & -5 & -3 & 6 & 5 & 0 & 1 & 0 & 0 \\ 4 & 2 & -2 & -1 & 3 & 2 & -1 & 0 & 3 & 1 \\ 0 & -2 & 1 & 5 & -1 & 3 & -1 & 1 & 9 & 4 \\ -4 & 2 & 6 & -1 & 6 & 7 & -3 & 3 & 2 & 3 \\ 8 & 6 & -8 & 5 & 7 & 17 & 2 & 6 & -3 & 5 \\ 0 & 2 & -1 & 3 & -4 & 2 & 5 & 3 & 0 & 1 \\ 16 & 10 & -11 & -9 & 17 & 34 & 2 & -1 & 2 & 2 \\ 4 & 6 & 2 & -7 & 13 & 9 & 2 & 0 & 12 & 4 \\ 0 & 0 & -1 & 8 & -3 & -24 & -8 & 6 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \\ 3 \\ 2 \\ 3 \\ 46 \\ 13 \\ 38 \\ 19 \\ -21 \end{bmatrix}$$

$x^* = (1, -1, 0, 1, 2, 0, 3, 1, -1, 2)^T$

J 迭代算法：

程序设计流程图：



源程序代码：

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
void main()
{
    float a[50][51],x1[50],x2[50],temp=0,fnum=0;
    int i,j,m,n,e,bk=0;
    printf("使用 Jacobi 迭代法求解方程组:\n");
    printf("输入方程组的元：\nn=");
    scanf("%d",&n);
    for(i=1;i<n+1;i++)
        x1[i]=0;
printf("输入方程组的系数矩阵:\n");
for(i=1;i<n+1;i++)
{
    j=1;
    while(j<n+1)
    {
        scanf("%f",&a[i][j]);
        j++;
    }
}
printf("输入方程组的常数项:\n");
for(i=1;i<n+1;i++)
{
    scanf("%f",&a[i][n+1]);
}
printf("\n");
printf("请输入迭代次数:\n");
```

```
scanf("%d",&m);
printf("请输入迭代精度:\n");
scanf("%d",&e);
while(m!=0)
{
      for(i=1;i<n+1;i++)
      {
            for(j=1;j<n+1;j++)
            {
                  if (j!=i)
                  temp=a[i][j]*x1[j]+temp;
            }
            x2[i]=(a[i][n+1]-temp)/a[i][i];
            temp=0;
      }
      for(i=1;i<n+1;i++)
      {
            fnum=float(fabs(x1[i]-x2[i]));
            if(fnum>temp) temp=fnum;
      }
      if(temp<=pow(10,-4)) bk=1;
      for(i=1;i<n+1;i++)
            x1[i]=x2[i];
      m--;
   }
   printf("原方程组的解为:\n");
   for(i=1;i<n+1;i++)
   {
      if((x1[i]-x2[i])<=e||(x2[i]-x1[i])<=e)
      {
            printf("x%d=%7.4f        ",i,x1[i]);
      }
   }
}
```

运行结果：

```
使用Jacobi迭代法求解方程组:
输入方程组的元:
n=10
输入方程组的系数矩阵:
4 2 -3 -1 2 1 0 0 0 0
8 6 -5 -3 6 5 0 1 0 0
4 2 -2 -1 3 2 -1 0 3 1
0 -2 1 5 -1 3 2 -1 0 3 1
-4 2 6 -1 6 7 -3 3 2 3
8 6 -8 5 7 17 2 6 -3 5
0 2 -1 3 -4 2 5 3 0 1
16 10 -11 -9 19 34 2 -1 2 2
4 6 2 -7 13 9 2 0 12 4
0 0 -1 8 -3 -24 -8 6 3 -1
输入方程组的常数项:
5 12 3 2 3 46 13 38 19 -21
请输入迭代次数:200
请输入迭代精度:0.0005
原方程组的解为:
x1= 1.0001
x2=-1.0001
x3= 0.0003
x4= 1.0002
x5= 2.0004
x6= 0.0003
x7= 3.0007
x8= 1.0001
x9=-1.0003
x10= 2.0005
x11= 0.0000
Press any key to continue
```

2、设对称正定阵系数阵线方程组

$$\begin{bmatrix} 4 & 2 & -4 & 0 & 2 & 4 & 0 & 0 \\ 2 & 2 & -1 & -2 & 1 & 3 & 2 & 0 \\ -4 & -1 & 14 & 1 & -8 & -3 & 5 & 6 \\ 0 & -2 & 1 & 6 & -1 & -4 & -3 & 3 \\ 2 & 1 & -8 & -1 & 22 & 4 & -10 & -3 \\ 4 & 3 & -3 & -4 & 4 & 11 & 1 & -4 \\ 0 & 2 & 5 & -3 & -10 & 1 & 14 & 2 \\ 0 & 0 & 6 & 3 & -3 & -4 & 2 & 19 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} 0 \\ -6 \\ 20 \\ 23 \\ 9 \\ -22 \\ -15 \\ 45 \end{bmatrix}$$

$$x^* = (1, -1, 0, 2, 1, -1, 0, 2)^T$$

GS 迭代算法：

```
#include<iostream.h>
#include<math.h>
#include<stdio.h>
const int m=11;
```

```
void main()
{
    int choice=1;

    while(choice==1)
    {

    double a[m][m],b[m],e,x[m],y[m],w,se,max;
    int n,i,j,N,k;
    cout<<"Gauss-Seidol 迭代法"<<endl;
    cout<<"请输入方程的个数："；
    cin>>n;
    for(i=1;i<=n;i++)
    {
    cout<<"请输入第"<<i<<"个方程的各项系数："；
    for(j=1;j<=n;j++)
        cin>>a[i][j];
    }
    cout<<"请输入各个方程等号右边的常数项:\n";
    for(i=1;i<=n;i++)
    {
        cin>>b[i];
    }
    cout<<"请输入最大迭代次数："；
    cin>>N;
    cout<<"请输入最大偏差："；
    cin>>e;
    for(i=1;i<=n;i++)
    {
        x[i]=0;
        y[i]=x[i];
    }
    k=0;
    while(k!=N)
    {
        k++;
    for(i=1;i<=n;i++)
    {
        w=0;
        for(j=1;j<=n;j++)
        {
            if(j!=i)
                w=w+a[i][j]*y[j];
        }
        y[i]=(b[i]-w)/double(a[i][i]);
    }

    max=fabs(x[1]-y[1]);
    for(i=1;i<=n;i++)
    {
        se=fabs(x[i]-y[i]);
```

```
    if(se>max)
        max=se;
}
if(max<e)
{
    cout<<endl;
    for(i=1;i<=n;i++)
        cout<<"x"<<i<<"="<<y[i]<<endl;
    break;
}
for(i=1;i<=n;i++)
{
    x[i]=y[i];
}
    }
    if(k==N)
        cout<<"迭代失败!! "<<endl;
    choice=0;

}
}
```

```
Gauss-Seidol迭代法
请输入方程的个数: 8
请输入第1个方程的各项系数: 4 2 -4 0 2 4 0 0
请输入第2个方程的各项系数: 2 2 -1 -2 1 3 2 0
请输入第3个方程的各项系数: -4 -1 14 1 -8 -3 5 6
请输入第4个方程的各项系数: 0 -2 1 6 -1 -4 -3 3
请输入第5个方程的各项系数: 2 1 -8 -1 22 4 10 -3
请输入第6个方程的各项系数: 4 3 -3 -4 4 11 1 -4
请输入第7个方程的各项系数: 0 2 5 -3 -10 1 14 2
请输入第8个方程的各项系数: 0 0 6 3 -3 -4 2 19
请输入各个方程等号右边的常数项:
0 -6 20 23 9 -22 -15 45
请输入最大迭代次数: 200
请输入最大偏差: 0.005
原方程组的解为:
x1=  1.001
 x2= -1.001
 x3=  0.003
 x4=  2.002
 x5=  1.004
 x6= -1.000
 x7=  0.001
 x8=  2.000
Press any key to continue
```

3、三对角形线性方程组

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 7 \\ 5 \\ -13 \\ 2 \\ 6 \\ -12 \\ 14 \\ -4 \\ 5 \\ -5 \end{bmatrix}$$

$$x^* = (2,\ 1,\ -3,\ 0,\ 1,\ -2,\ 3,\ 0,\ 1,\ -1)^T$$

SOR 方法：

```
# include <stdio.h>
# include <math.h>
#include<stdlib.h>

/**********定义全局变量**********/
float **a;     /*存放 A 矩阵*/
float *b;     /*存放 b 矩阵*/
float *x;     /*存放 x 矩阵*/
float p;     /*精确度*/
float w;     /*松弛因子*/
int n;      /*未知数个数*/
int c;      /*最大迭代次数*/
int k=1;     /*实际迭代次数*/

/**********SOR 迭代法**********/
void SOR(float xk[])
{
 int i,j;
 float t=0.0;
 float tt=0.0;
 float *xl;
 xl=(float *)malloc(sizeof(float)*(n+1));
 for(i=1;i<n+1;i++)
 {
  t=0.0;
  tt=0.0;
  for(j=1;j<i;j++)
   t=t+a[i][j]*xl[j];
  for(j=i;j<n+1;j++)
   tt=tt+a[i][j]*xk[j];
  xl[i]=xk[i]+w*(b[i]-t-tt)/a[i][i];
 }
 t=0.0;
 for(i=1;i<n+1;i++)
  {
```

```
  tt=fabs(xl[i]-xk[i]);
  tt=tt*tt;
  t+=tt;
 }
t=sqrt(t);
for(i=1;i<n+1;i++)
 xk[i]=xl[i];

if(k+1>c)
{
 if(t<=p)
  printf("\nReach the given precision!\n");
 else
  printf("\nover the maximal count!\n");
 printf("\nCount number is %d\n",k);
 }
 else
  if(t>p)
  {
   k++;
   SOR(xk);
  }
  else
  {
   printf("\nReach the given precision!\n");
   printf("\nCount number is %d\n",k);
  }
}

/**********程序*****开始**********/
void main()
{
 int i,j;
printf("SOR 方法\n");
printf("请输入方程个数:\n");
scanf("%d",&n);
a=(float **)malloc(sizeof(float)*(n+1));
for(i=0;i<n+1;i++)
 a[i]=(float*)malloc(sizeof(float)*(n+1));
printf("请输入三对角矩阵:\n");
for(i=1;i<n+1;i++)
 for(j=1;j<n+1;j++)
  scanf("%f",&a[i][j]);
for(i=1;i<n+1;i++)
 for(j=1;j<n;j++)
b=(float *)malloc(sizeof(float)*(n+1));
printf("请输入等号右边的值:\n");
for(i=1;i<n+1;i++)
 scanf("%f",&b[i]);
x=(float *)malloc(sizeof(float)*(n+1));
printf("请输入初始的 x:");
```

```
for(i=1;i<n+1;i++)
 scanf("%f",&x[i]);
printf("请输入精确度:");
scanf("%f",&p);
printf("请输入迭代次数:");
scanf("%d",&c);
printf("请输入 w(0<w<2):\n");
scanf("%f",&w);
SOR(x);
printf("方程的结果为:\n");
for(i=1;i<n+1;i++)
 printf("x[%d]=%f\n",i,x[i]);
}
```

```
SOR方法
请输入方程个数:
10
请输入三对角矩阵:
4 -1 0 0 0 0 0 0 0 0
-1 4 -1 0 0 0 0 0 0 0
0 -1 4 -1 0 0 0 0 0 0
0 0 -1 4 -1 0 0 0 0 0
0 0 0 -1 4 -1 0 0 0 0
0 0 0 0 -1 4 -1 0 0 0
0 0 0 0 0 -1 4 -1 0 0
0 0 0 0 0 0 -1 4 -1 0
0 0 0 0 0 0 0 -1 4 -1
0 0 0 0 0 0 0 0 -1 4
请输入等号右边的值:
7 5 -13 2 6 -12 14 -4 5 -5
请输入初始的x:2.1 1.1 -3 0.1 1.1 -2.1 3.1 0.1 1.1 -1.1
请输入精确度:0.005
请输入迭代次数:200
请输入w<0<w<2>:
1.2

Reach the given precision!

Count number is 5
方程的结果为:
x[1]=1.999582
x[2]=1.000952
x[3]=-3.000353
x[4]=-0.000056
x[5]=0.999543
x[6]=-1.999329
x[7]=2.999813
x[8]=-0.000042
x[9]=1.000004
x[10]=-0.999997
Press any key to continue
```


四、程序运行结果讨论和分析：

①迭代法具有需要计算机的存贮单元较少,程序设计简单,原始系数矩阵在计算过程中始终不变等优点.
②迭代法在收敛性及收敛速度等方面存在问题.
[注:A必须满足一定的条件下才能运用以下三种迭代法之一.在Jacobi中不用产生的新数据信息,每次都要计算一次矩阵与向量的乘法,而在Gauss利用新产生的信息数据来计算矩阵与向量的乘法.在SOR中必须选择一个最佳的松弛因子,才能使收敛加速.]
经过计算可知Gauss-Seidel方法比Jacobi方法剩点计算量,也是Jacobi方法的改进.可是精确度底,计算量高,费时间,需要改进.SOR是进一步改进Gauss-Seidel而得到的比Jacobi,Gauss-Seidel方法收敛速度快,综合性强.改变松弛因子的取值范围来可以得到Jacobi,Gauss-Seidel方法.
③选择一个适当的松弛因子是关键.
结论：线性方程组 1 和 2 对于 Jacobi 迭代法，Gauss-Seidol 迭代法和 SOR 方法均不收敛，线性方程组 3 收敛。

课题四　数值积分

一、问题提出

选用复合梯形公式，复合 Simpson 公式，Romberg 算法，计算

（1）　$I = \int_0^{1/4} \sqrt{4 - \sin^2 x}\, dx$　　$(I \approx 1.5343916\quad)$

（2）　$I = \int_0^1 \frac{\sin x}{x} dx$　　$(f(0) = 1,\quad I \approx 0.9460831\quad)$

（3）　$I = \int_0^1 \frac{e^x}{4 + x^2} dx$

（4）

（5）　$I = \int_0^1 \frac{\ln(1 + x)}{1 + x^2} dx$

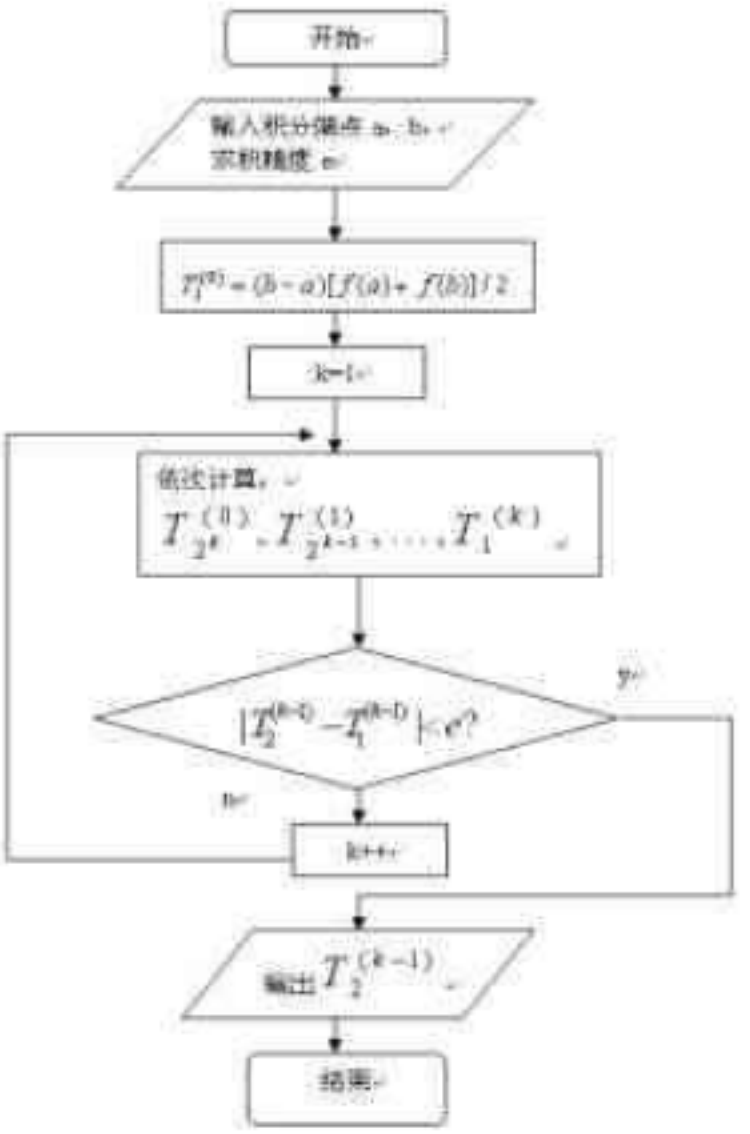
二、要求

1、 编制数值积分算法的程序；
2、 分别用两种算法计算同一个积分，并比较其结果；
3、 分别取不同步长　$h = (b - a)/n$，试比较计算结果（如 n = 10, 20 等）；
4、 给定精度要求 ε，试用变步长算法，确定最佳步长。

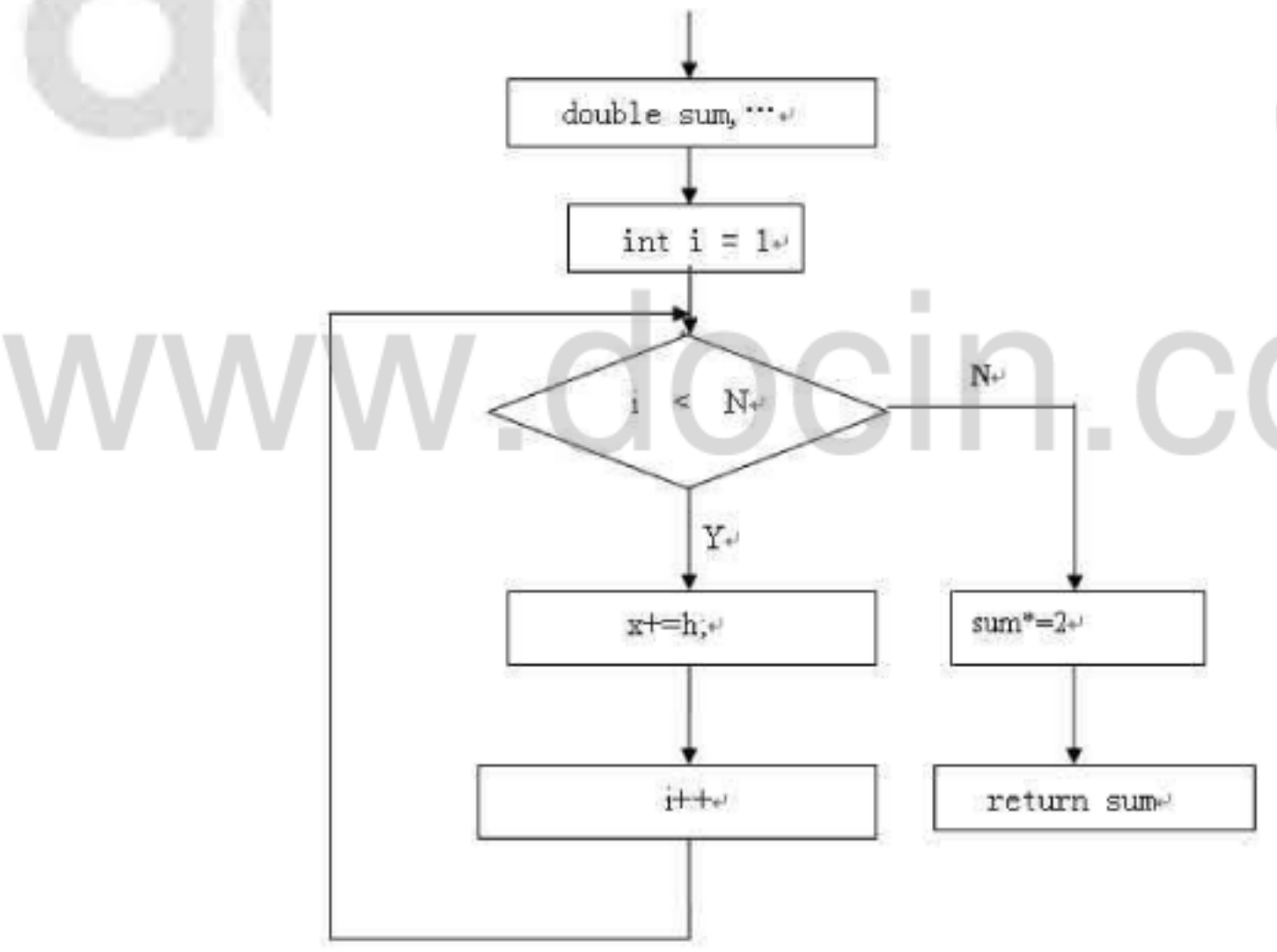
三、目的和意义

1、 深刻认识数值积分法的意义；
2、 明确数值积分精度与步长的关系；

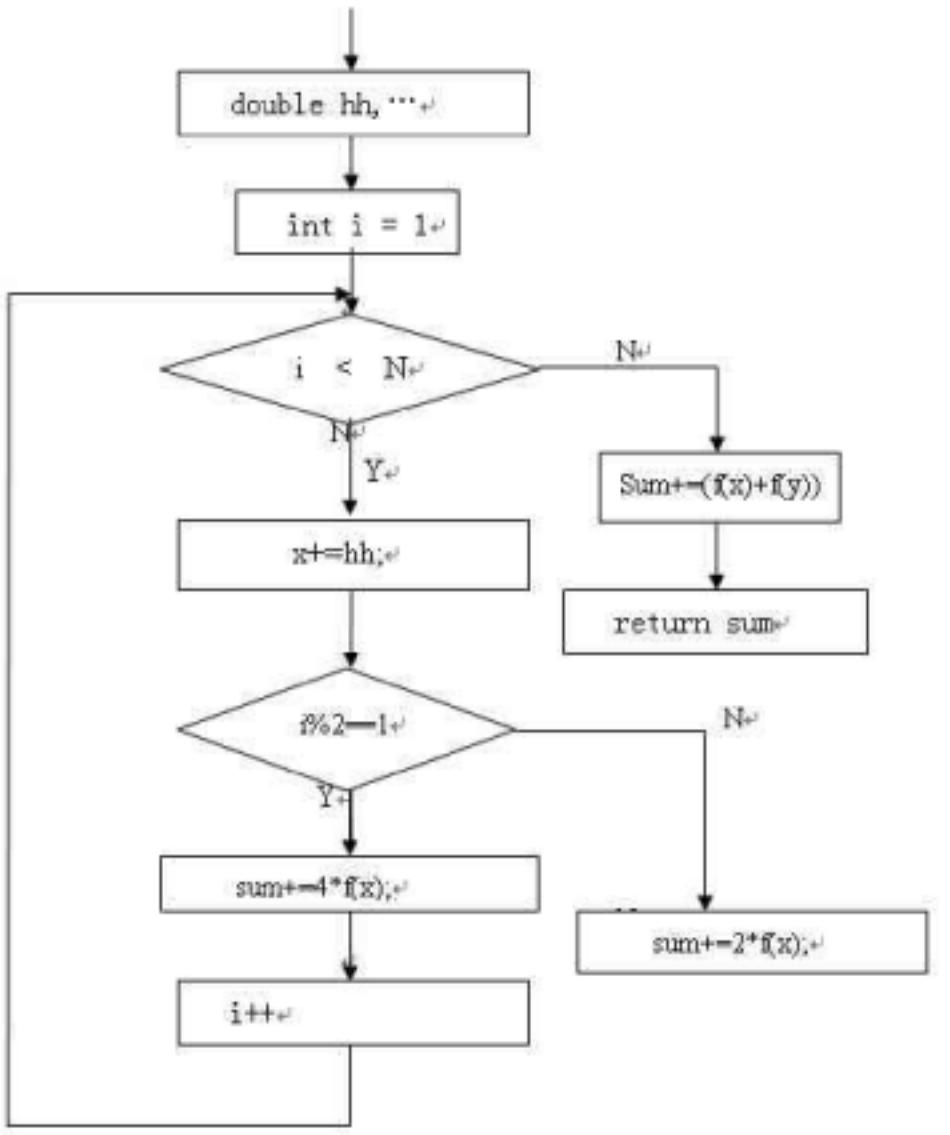
根据定积分的计算方法，可以考虑二重积分的计算问题。
流程图：
复化梯形：



复化simpson：



Romberg 求积法：



程序代码：

复化梯形：

```
#include<iostream.h>
#include<math.h>
int N;
//声明被积分的函数 f(x)
double f(double x)
{     return(sqrt(4-sin(x)*sin(x)));
      //return(x==0? 1:sin(x)/x);
      //float e=2.718281828;
      //return(pow(e,x)/(4+x*x));
      //return(log(1+x)/(1+x*x));
}

inline double T(double x ,double y)
{
      double sum,h,a;
      sum=0;
      h=(y-x)/N;
```

```
        a=x;
        for(int i=1;i<N;i++)
        {//以 h 的大小为步长递增
              x+=h;
              sum+=f(x);
        }
        sum*=2;
        sum+=(f(x)+f(y));
        sum*=(h/2);
        return sum;
};
void main()
{
        cout<<"输入你想执行的步长：";
        cin>>N;
        double a ,b;
        cout<<"输入下界：";
        cin>>a;
        cout<<"输入上界：";
        cin>>b;
        //输出运行结果
        cout<<"经用梯形公式计算知原函数积分近似值为："<<T(a, b)<<endl;
}
```

复化 **simpson** 公式

```
#include<iostream.h>
#include<math.h>
int N;
double f(double x)
{     return(sqrt(4-sin(x)*sin(x)));
 //return(x==0? 1:sin(x)/x);
 //float e=2.718281828;
 //return(pow(e,x)/(4+x*x));
 //return(log(1+x)/(1+x*x));
}


inline double S(double x, double y)
{
 double sum, hh,a;
 sum=0;
 hh=(y-x)/N;
 a=x;
 for(int i=1;i<N;i++)
```

```
 {//以 hh 的大小为步长递增
        x+=hh;
 //对 i 的单双数进行不同处理
        if(i%2==1)
        sum+=4*f(x);
        else
        sum+=2*f(x);
 }
 sum+=(f(x)+f(y));
 sum*=hh/3;
 return sum;
}
void main()
{
 cout<<"输入你想执行的步长：";
 cin>>N;
 N*=2;
 //hh 表示公式中 h 的一半
 double a ,b;
 cout<<"输入下界：";
 cin>>a;
 cout<<"输入上界：";
 cin>>b;
 //输出运行结果
 cout<<"经用 Simpson 公式计算知原函数的积分近似值为："<<S(a, b)<<endl;
}
```


Romberg 求积法

1.

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <math.h>

double my_f(double x)
{   return(sqrt(4-sin(x)*sin(x)));
  //return(x==0? 1:sin(x)/x);
  //float e=2.718281828;
  //return(pow(e,x)/(4+x*x));
  //return(log(1+x)/(1+x*x));
}
```

```
double Romberg(double (*f)(double),double a,double b,double eps)
{ double T[64],h=b-a;
 int n=1;
 T[0]=h*(f(a)/4.0+f(b)/4.0+f(a+h/2.0)/2.0);//复化梯形公式
 T[1]=h*(f(a)/6.0+f(b)/6.0+f(a+h/2.0)/1.5);//Simpson 公式
 for(int i=2;fabs(T[i-2]-T[i-1])>eps;++i)
 {//计算递推梯形值,base
  h/=2.0;n<<=1;//分点数
  int j; double base=T[0]/h;
  double x=a+h/2.0;
  for(j=0;j<n;++j)
  {base+=f(x);
   x+=h;
  }
  base=base*h/2.0; //计算外推加速值,T[0]->T[i]
  double k1=4.0/3.0,k2=1.0/3.0;
  for(j=0;j<i;++j)
  {double hand=k1*base-k2*T[j];
   T[j]=base;
   base=hand;
   k2=k2/(4.0*k1-k2);
   k1=k2+1.0;
  }
  T[i]=base;
 }
 return T[i-1];
}
void main()
{   float a,b,e;
    cout<<"\n 请输入求积节点 a："<<endl; cin>>a;
    cout<<"\n 请输入求积节点 b："<<endl; cin>>b;
    cout<<"\n 请输入求积精度 e："<<endl; cin>>e;
    cout<<Romberg(my_f,a,b,e)<<endl;
}
```

运行结果：

1、

Simpson

n=10

```
Please input the dot a :0.0

Please input the dot b :0.25

Please input the number n :10

The result is :0.498711
Press any key to continue_
```

n=20

```
Please input the dot a :0.0

Please input the dot b :0.25

Please input the number n :20

The result is :0.498711
Press any key to continue_
```

Romberg

```
请输入求积节点 a:
0

请输入求积节点 b:
0.25

请输入求积精度 e:
0.00001
0.498711
Press any key to continue_
```


2、

Simpson

n=10

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :10

The result is :0.946083
Press any key to continue_
```

n=20

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :20

The result is :0.946083
Press any key to continue
```

Romberg

```
请输入求积节点 a:
0.0

请输入求积节点 b:
1.0

请输入求积精度 e:
0.00001
0.946083
Press any key to continue
```

3、

Simpson

n=10

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :10

The result is :0.390812
Press any key to continue
```

n=20

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :20

The result is :0.390812
Press any key to continue
```

Romberg

```
请输入求积节点 a:
0

请输入求积节点 b:
1

请输入求积精度 e:
0.00001
0.390811
Press any key to continue
```

5、

Simpson

n=10

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :10

The result is :0.272198
Press any key to continue
```

n=20

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :20

The result is :0.272198
Press any key to continue_
```

Romberg

```
请输入求积节点 a:
0.0

请输入求积节点 b:
1.0

请输入求积精度 e:
0.00001
0.222197
Press any key to continue
```

确定最佳步长

n=2

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :2

The result is :  0.946087
Press any key to continue_
```

n=4

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the nunber n :4

The result is :  0.946083
Press any key to continue
```

n=5

```
Please input the dot a :0.0

Please input the dot b :1.0

Please input the number n :5

The result is :0.946083
Press any key to continue_
```

实验心得：通过本次试验，深刻认识数值积分法的意义；明确数值积分精度与步长的关系；根据定积分的计算方法，可以考虑二重积分的计算问题。对不同步长不仅影响到运算时间运算量还影响着运算精度.,在实际操作中应根据不同的要求选取适当的步长。