天津理工大学 实验报告

学院（系）名称：计算机科学与工程学院

姓名	王帆	**学号**	20152180	**专业**	计算机科学与技术
班级	计算机 1 班	**实验项目**	实验五　线性方程组的解法		
课程名称		数值计算方法		**课程代码**	0665026
实验时间		2017 年 6 月 6 日 第 3 -4 节连中午		**实验地点**	7-219
批改意见				**成绩**	

教师签字：

一、 实验目的

根据实验要求，使用迭代法和高斯消元法解决具体问题，设计相应的算法框图并编程实现，得到正确的运行结果并对结果进行有效分析。

二、 实验环境

- 硬件环境：IBM－PC 或兼容机
- 软件环境：Windows 操作系统，VC6.0
- 编程语言：C 或 C++

三、 实验内容

1.用雅可比迭代法、高斯—塞德尔迭代法和松弛法求下列线性方程组的解，要求准确到 $1/2\times10^{-3}$。

$$\begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}=\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

要求：

(1) 线性方程组的维数 n、系数矩阵 A 的元素、列向量 b 的元素从键盘输入；

(2) 误差 ε 和松弛因子 ω 从键盘输入；

(3) 在同一个程序里将雅可比迭代法、高斯—塞德尔迭代法和松弛法编程实现；

(4) 绘制这三种算法求解线性方程组的算法框图；

(5) 改变 ω 的值，根据迭代计算结果比较 ω 取不同值的情况下迭代过程的收敛速度；
(6) 将这三种算法的每一步迭代计算结果打印输出；
(7) 根据计算结果，比较这三种算法的迭代收敛速度。

2.用高斯—约当消元法或全选主元高斯消元法求解下列线性方程组的解

$$\begin{cases} 12x_1 - 3x_2 + 3x_3 = 15 \\ -18x_1 + 3x_2 - x_3 = -15 \\ x_1 + x_2 + x_3 = 6 \end{cases}$$

要求：
(1) 线性方程组的维数 n、系数矩阵 A 的元素、列向量 b 的元素从键盘输入；
(2) 用算法框图描述相应算法的实现过程；
(3) 将每一次消元过程得到的增广矩阵打印输出；
(4) 最后输出方程组的解。

四、 实验要求

1.每一个实验内容要求自己独立完成，不允许抄袭别人，否则按不及格处理；
2.按照实验要求，根据自己的程序编写情况绘制相应的算法框图或描述算法步骤；
3.按照实验内容和相应的要求书写实验报告；
4.在实验过程部分，要求根据实验内容和要求书写每一个实验相应的算法步骤或框图、运行过程和运行结果的截图、运行结果分析、以及程序源代码。每一个实验要求书写下述内容：
(1) 算法步骤描述或算法框图
(2) 程序源代码
(3) 运行结果（要求截图）
(4) 运行结果分析
5. 在规定的时间内上交实验报告。

五、 实验过程

1.用雅可比迭代法、高斯—塞德尔迭代法和松弛法求下列线性方程组的解，要求准确到 1/2×10-3。

代码实现：

```
//三种迭代法求解线性方程组
#include <iostream>
#define MAX_NUM 100
#define N 100
using namespace std;
//结构体定义
typedef struct EQU{
    int A[MAX_NUM][MAX_NUM];
    int b[MAX_NUM];
}EQU;
//全局变量
```

```
EQU *T;
int n;
double e;
//原型声明
EQU *Init(int &n,double &e);
void show(int i);
void Jacobi(EQU *T,int n,double e);
void Gauss(EQU *T,int n,double e);
void Relaxation(EQU *T,int n,double e);
//实现
int main() {
    T = Init(n,e);
    Jacobi(T,n,e);
    Gauss(T,n,e);
    Relaxation(T,n,e);
    return 0;
}

EQU *Init(int &n,double &e) {
    EQU *T = new EQU();
    cout<<"维数 n = ";
    cin>>n;
    cout<<"精确度 e = ";
    cin>>e;
    cout<<"系数矩阵 A 的元素："<<endl;
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++) cin>>T->A[i][j];
    cout<<"列向量 b 的元素："<<endl;
    for(int i=0;i<n;i++) cin>>T->b[i];
    return T;
}

void show(int i){
    switch(i){
        case 0:{
            cout<<"次数 \t\tX1 \t\tX2 \t\tX3 \t\tX4  "<<endl;
            break;
        }
        case 1:{
            cout<<"雅可比迭代法："<<endl;
            break;
        }
        case 2:{
            cout<<"高斯-赛德尔迭代法："<<endl;
            break;
```

```
        }
        case 3:{
            cout<<"松弛法：<<endl;
            cout<<"松弛因子 w = ";
            break;
        }
    }
    return;
}

void Jacobi(EQU *T,int n,double e) {
    double x[MAX_NUM],x_t[MAX_NUM];
    double tol = 0.0,t = 0.0;
    bool f = true;

    for(int i=0; i<n;i++) x[i]=0.0;
    show(1); show(0);
    for(int k=0;f&&k<N;k++) {
        for(int i=0;i<n;i++) {
            for(int j=0;j<n;j++) {
                if(i!=j) tol+=T->A[i][j]*x[j];
            }
            x_t[i]=(T->b[i]-tol)/T->A[i][i];
            if(t<(x_t[i]-x[i])) t=x_t[i]-x[i];
            tol=0.0;
        }
        if(t<e) f = false;
        cout<<" "<<k;
        for(int i=0;i<n;i++) {
            printf("\t     %f",x[i]);
            x[i] = x_t[i];
        }
        cout<<"    t = "<<t<<endl;
        t = 0.0;
    }
}

void Gauss(EQU *T,int n,double e) {
    double x[MAX_NUM],x_t[MAX_NUM];
    double tol = 0.0,t = 0.0;
    bool f=true;

    for(int i=0;i<n;i++) {
        x[i] = 0.0;
    }
```

```
        show(2);
        show(0);
        for(int k=0;f&&k<N;k++){
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    if(i!=j) tol += T->A[i][j]*x[j];
                }
                x_t[i]=(T->b[i]-tol)/T->A[i][i];

                if(t<(x_t[i]-x[i])) t=x_t[i]-x[i];

                x[i]=x_t[i];
                tol=0.0;
            }
            if(t<e) f=false;
            cout<<" "<<k;
            for(int i=0;i<n;i++) cout<<"\t      "<<x[i]);
            cout<<"     t = "<<t<<endl;
            t = 0.0;
        }
    }

    void Relaxation(EQU *T,int n,double e) {
        double x[MAX_NUM],x_t[MAX_NUM],w;
        double tol = 0.0,t = 0.0;
        bool f;
        while(1) {
            show(3);
            cin>>w;
            if(w == 0) return;
            f = true;
            for(int i=0;i<n;i++) x[i]=0.0;
            show(0);
            for(int k=0;f&&k<N;k++) {
                for(int i=0;i<n;i++) {
                    for(int j=0;j<n;j++){
                        if(i!=j) tol+=T->A[i][j]*x[j];
                    }
                    tol=T->b[i]-tol;
                    tol=w/T->A[i][i]*tol;
                    x_t[i]=(1-w)*x[i]+tol;
                    if(t<(x_t[i]-x[i])) t=x_t[i]-x[i];
                    x[i]=x_t[i];
                    tol=0.0;
                }
```

```
            if(t<e) f=false;
            cout<<" "<<k;
            for(int i=0;i<n;i++) cout<<"\t      "<<x[i];
            cout<<"     t = "<<t<<endl;
            t = 0.0;
        }
    }

}
```

算法框图：

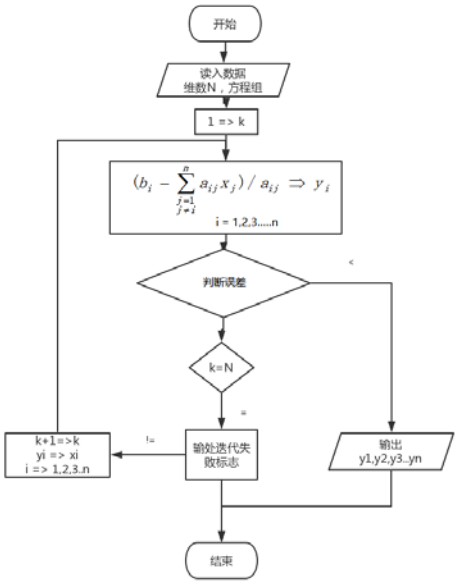

图 1-1 雅可比迭代法框图

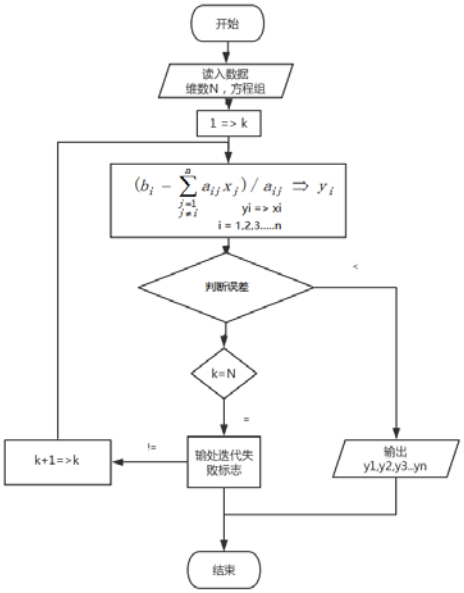

图 1-2 高斯-赛德尔迭代法框图

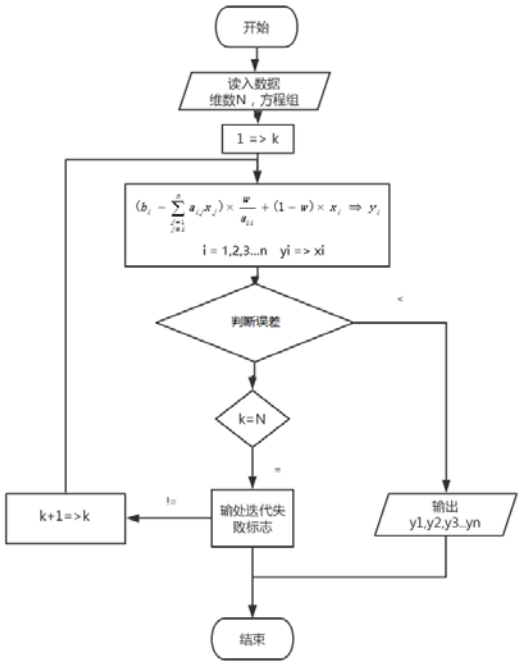

图 1-3 松弛法框图

运行结果：

雅可比迭代法:

次数	X1	X2	X3	X4	
0	0.000000	0.000000	0.000000	0.000000	temp = 0.25
1	0.000000	0.000000	0.250000	0.250000	temp = 0.0625
2	0.062500	0.062500	0.312500	0.312500	temp = 0.03125
3	0.093750	0.093750	0.343750	0.343750	temp = 0.015625
4	0.109375	0.109375	0.359375	0.359375	temp = 0.0078125
5	0.117188	0.117188	0.367188	0.367188	temp = 0.00390625
6	0.121094	0.121094	0.371094	0.371094	temp = 0.00195313
7	0.123047	0.123047	0.373047	0.373047	temp = 0.000976563
8	0.124023	0.124023	0.374023	0.374023	temp = 0.000488281

图 1-4 雅可比迭代法运行结果

高斯-赛德尔:

次数	X1	X2	X3	X4	
0	0.000000	0.000000	0.250000	0.312500	temp = 0.3125
1	0.062500	0.093750	0.343750	0.359375	temp = 0.09375
2	0.109375	0.117188	0.367188	0.371094	temp = 0.046875
3	0.121094	0.123047	0.373047	0.374023	temp = 0.0117188
4	0.124023	0.124512	0.374512	0.374756	temp = 0.00292969
5	0.124756	0.124878	0.374878	0.374939	temp = 0.000732422
6	0.124939	0.124969	0.374969	0.374985	temp = 0.000183105

图 1-5 高斯-赛德尔迭代法运行结果

```
松弛法:
松弛因子：0.5
次数            X1              X2              X3              X4
 0          0.000000        0.000000        0.125000        0.140625        temp = 0.140625
 1          0.015625        0.019531        0.207031        0.223633        temp = 0.0830078
 2          0.036133        0.042236        0.260986        0.274719        temp = 0.0539551
 3          0.055969        0.062454        0.296829        0.307270        temp = 0.0358429
 4          0.072895        0.078748        0.320935        0.328595        temp = 0.024106
 5          0.086408        0.091249        0.337343        0.342872        temp = 0.0164078
 6          0.096778        0.100581        0.348628        0.352587        temp = 0.0112847
 7          0.104540        0.107431        0.356455        0.359279        temp = 0.00782701
 8          0.110256        0.112408        0.361919        0.363930        temp = 0.00571572
 9          0.114419        0.115997        0.365753        0.367184        temp = 0.00416295
 10         0.117428        0.118575        0.368453        0.369471        temp = 0.00300946
 11         0.119593        0.120421        0.370359        0.371083        temp = 0.00216445
 12         0.121144        0.121739        0.371708        0.372222        temp = 0.00155117
 13         0.122253        0.122679        0.372663        0.373029        temp = 0.00110892
 14         0.123044        0.123348        0.373341        0.373601        temp = 0.000791386
 15         0.123608        0.123825        0.373822        0.374006        temp = 0.000564094
 16         0.124010        0.124165        0.374163        0.374294        temp = 0.000401742

松弛法:
松弛因子：1.3
次数            X1              X2              X3              X4
 0          0.000000        0.000000        0.325000        0.430625        temp = 0.430625
 1          0.105625        0.174281        0.401781        0.383033        temp = 0.174281
 2          0.155533        0.122749        0.379499        0.373321        temp = 0.0499078
 3          0.116571        0.122390        0.370365        0.373149        temp = 0
```

图 1-6 松弛法运行结果（松弛因子分别为 0.5 与 1.3）

运行分析：

雅克比迭代将联立方程组的求解归结为重复计算一组彼此独立的线性表达式，使问题得到了简化。但更新 x 值是在计算完一次方程组后才更新的。

高斯-赛德尔公式的特点是一旦求出变元 xi 的某个新值 xi+1 后，将用新值代替旧值，一般来说高斯—赛德尔迭代法要雅克比迭代法好。但情况不总这样，也有高斯-赛德尔迭代法比雅克比迭代法收敛的慢，甚至有雅克比迭代法收敛但高斯-赛德尔迭代法发散的例子。该实验中，高斯-赛德尔迭代法稍比雅克比迭代法收敛的快一些。

使用迭代的困难在于那以估计计算量。有时迭代过程虽然收敛，但收敛速度缓慢，计算量大，因此，迭代过程的加速就有着重要的意义。松弛法是对高斯—赛德尔的一种能加速方法。将前一步的结果 xi 与高斯-赛德尔方法的迭代值适当加权平均，期望获得更好的近似值。其中，松弛因子 w 的取值对迭代的收敛速度影响极大，在本实验中，w 越大（0<w<2）,收敛速度越快，但精度越低。

2. 用高斯—约当消元法或全选主元高斯消元法求解下列线性方程组的解

代码实现：

```
//高斯-约当消元法求解线性方程组
#include <iostream>
#include <iomanip>
#define MAX_NUM 100
#define N 100
using namespace std;
//结构体定义
```

```
typedef struct EQU{
    double A[MAX_NUM][MAX_NUM];
    double b[MAX_NUM];
}EQU;
//原型声明
EQU *Init(int &n);
void GJ(EQU *T,int n);
//实现
int main(){
    EQU *T;
    int n;
    T = Init(n);
    GJ(T,n);
    return 0;
}
EQU *Init(int &n){
    cout<<setprecision(6);
    EQU *T=new EQU();
    printf("维数：");
    cin>>n;
    printf("系数矩阵 A 的元素：\n");
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++) cin>>T->A[i][j];

    printf("列向量元素：\n");
    for(int i=0;i<n;i++) cin>>T->b[i];
    cout<<endl<<endl;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++) cout<<T->A[i][j]<<"\t";
        cout<<T->b[i]<<endl;
    }
    cout<<endl;
    return T;
}

void GJ(EQU *T,int n){
    double t;
    for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
            if(k!=i){
                t=T->A[i][k]/T->A[k][k];
                for(int j=0;j<n;j++){
                    T->A[i][j]=T->A[i][j]-t*T->A[k][j];
                }
                T->b[i]=T->b[i]-t*T->b[k];
```

```
            }
        }
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                cout<<T->A[i][j]<<"\t";
            }
            cout<<T->b[i]<<endl;
        }
        cout<<endl;
    }
    for(int i=0;i<n;i++){
        T->b[i]=T->b[i]/T->A[i][i];
        T->A[i][i] = 1;
    }
    for(int i=0;i<n;i++) {
        for(int j=0;j<n;j++) {
            cout<<T->A[i][j]<<"\t";
        }
        cout<<T->b[i]<<endl;
    }
    cout<<endl;
    for(int i=1;i<=n;i++) cout<<"X"<<i<<" = "<<T->b[i-1]<<endl;
    return;
}
```

算法框图：

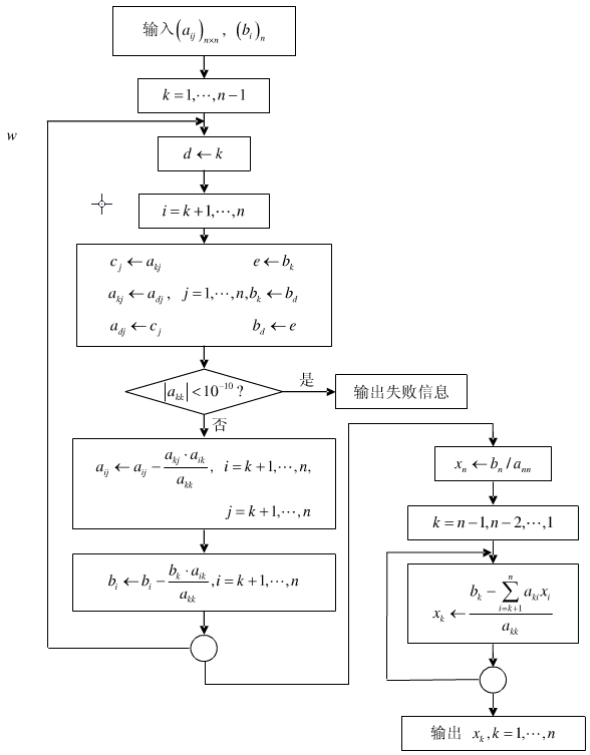

图 2-1 高斯-约当消元法算法框图

运行结果：

```
12.000000       -3.000000       3.000000        15.000000
-18.000000      3.000000        -1.000000       -15.000000
1.000000        1.000000        1.000000        6.000000

12.000000       -3.000000       3.000000        15.000000
0.000000        -1.500000       3.500000        7.500000
0.000000        1.250000        0.750000        4.750000

12.000000       0.000000        -4.000000       0.000000
0.000000        -1.500000       3.500000        7.500000
0.000000        0.000000        3.666667        11.000000

12.000000       0.000000        0.000000        12.000000
0.000000        -1.500000       0.000000        -3.000000
0.000000        0.000000        3.666667        11.000000

1.000000        0.000000        0.000000        1.000000
0.000000        1.000000        0.000000        2.000000
0.000000        0.000000        1.000000        3.000000

X1 = 1.000000
X2 = 2.000000
X3 = 3.000000
```

图 2-1 高斯-约当消元法执行过程

运行分析：

高斯-约当消去法执行一次归一化条件过程需要进行 n-k+1 次除法，而执行一次消去过程需要进行（n-1）(n-k+1)次乘法。因此，高斯-约当消元法进行的乘除法总计量与高斯消元法相比减少了很多。此外，高斯-约旦消去法不需要回代，算法结构稍许简单。因此，这是一种比较高效的方法。

六、 实验总结及心得体会

通过本次实验，我对求解线性方程组的雅可比、高斯-赛德尔、松弛法和列主消元法等方法有了更深刻的理解。其中，使用高斯-赛德尔和雅克比迭代都可以求出方程组的解，但是利用高斯-赛德尔迭代法所需的迭代次数比雅克比迭代少，能更早的达到精度要求此外，高斯-约旦消去法进行的乘除法总计量比高斯消去法少了很多计算工作量。高斯-约旦消去法不需要回代，算法结构稍许简单。通过分析，我了解了不同方法的适用情形，并对线性方程组的数值解法加深了认识与理解。