实验 2 解线性方程组的迭代法

201311211914
赵帅帅
2016–3–29

一、实验问题

给定下列几个不同类型的线性方程组，请分别采用 **Jacobi**迭代法，**Gauss-Seidel**迭代法和 **SOR**迭代法求解。

1. 线性方程组

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

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

2. 对称正定方程组

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

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

3. 三对角线方程组

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

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

二、实验要求

1. 应用迭代法求解线性方程组，并与直接法做比较。
2. 分别对不同精度要求，如 $\varepsilon = 10^{-3}, 10^{-4}, 10^{-5}$，利用所需迭代次数体会该迭代法收敛快慢。
3. 对方程做 (2)，(3) 使用 **SOR**方法时，选取松弛因子 $\omega = 0.8, 0.9, 1, 1.1, 1.2$ 等，试观察对算法收敛的影响，并找出你所选用松弛因子的最佳值。
4. 编制出各种迭代法的程序并给出计算结果。

三、实验目的与意义

1. 通过上机计算了解迭代法求解线性方程组的特点；掌握求解线性方程组的各类迭代算法。。
2. 体会上机计算时，终止准则 $\left\|x^{(k+1)} - x^k\right\|_\infty < \varepsilon$，对控制迭代解精度的有效性。
3. 体会初始值 $x^{(0)}$ 和松弛因子 ω 的选取，对迭代收敛速度的影响。

四、算法

1. **Jacobi 迭代法**

   用 **Jacobi**迭代法求解方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，一维数组 $\boldsymbol{x}^{(0)}$ 和 $\boldsymbol{x}$ 分别存放迭代向量 $\boldsymbol{x}^{(k)}$ 和 $\boldsymbol{x}^{(k+1)}$。

   输入 矩阵 $\boldsymbol{A} = (a_{ij})$，右端项 $\boldsymbol{b} = (b_1, b_2, \cdots, b_n)^T$，维数 n，初始向量 $\boldsymbol{x}^{(0)}$，精度要求 ε，最大迭代次数 N

   输出 迭代解 $x_1, x_2, \cdots, x_n$ 和迭代次数 k

   (1) 对于 $k = 1, 2, \cdots, N$，可循环执行步 2 到步 5

(2) 对于 $i=1,2,\cdots,n$，计算

$$x_i \Leftarrow x_i^{(0)} + \frac{b_i - \sum\limits_{j=1}^{n} a_{ij}x_j^{(0)}}{a_{ii}}$$[1]

(3) 置 $R = \max\limits_{1\leqslant i\leqslant n}\left|x_i - x_i^{(0)}\right|$

(4) 如果 $R \leqslant \varepsilon$

输出 $x_1, x_2, \cdots, x_n, k$

(5) $x_i^{(0)} \Leftarrow x_i, i=1,2,\cdots,n$

(6) 已达到最大迭代次数，输出 $x_1, x_2, \cdots, x_n, k$

2. **Gauss-Seidel 迭代法**

用 **Gauss-Seidel**迭代法求解方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，一维数组 $\boldsymbol{x}^{(0)}$ 和 $\boldsymbol{x}$ 分别存放迭代向量 $\boldsymbol{x}^{(k)}$ 和 $\boldsymbol{x}^{(k+1)}$。

输入 矩阵 $\boldsymbol{A}=(a_{ij})$，右端项 $\boldsymbol{b}=(b_1,b_2,\cdots,b_n)^T$，维数 n，初始向量 $\boldsymbol{x}^{(0)}$，精度要求 ε，最大迭代次数 k

输出 迭代解 $x_1, x_2, \cdots, x_n$ 和迭代次数 k

(1) 对于 $k=1,2,\cdots,N$，可循环执行步 2 到步 5

(2) 对于 $i=1,2,\cdots,n$，计算

$$x_i \Leftarrow \frac{b_i - \sum\limits_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum\limits_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}}$$

(3) 置 $R = \max\limits_{1\leqslant i\leqslant n}\left|x_i - x_i^{(0)}\right|$

(4) 如果 $R \leqslant \varepsilon$

输出 $x_1, x_2, \cdots, x_n, k$

(5) $x_i^{(0)} \Leftarrow x_i, i=1,2,\cdots,n$

(6) 已达到最大迭代次数，输出 $x_1, x_2, \cdots, x_n, k$

3. **SOR 迭代法**

用 **SOR**方法 ($\omega=1$ 时为 **GS**迭代法) 求解方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，计算公式为

$$\begin{aligned}
x_i^{k+1} &= x_i^k + \Delta x_i, i=1,2,\cdots,n \\
\Delta x_i &= \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)
\end{aligned}$$

[1]算法原始表达式 $x_i \Leftarrow \frac{b_i - \sum\limits_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum\limits_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}}$

用一维数组 $\boldsymbol{x}$ 存放迭代向量 $\boldsymbol{x}^k$，终止准则为

$$\max_{1\leqslant i\leqslant n}\left|x_i^{(k+1)}-x_i^{(k)}\right|=\max_{1\leqslant i\leqslant n}\left|\Delta x_i\right|\leqslant\varepsilon$$

ε 为设定的精度要求。为防止死循环，限制最大迭代次数为 N。

输入 $\boldsymbol{A}=(a_{ij}),\boldsymbol{b}=(b_1,b_2,\cdots,b_n)^T,\boldsymbol{x}^{(0)}=\left(x_1^{(0)},x_2^{(0)},\cdots,x_n^{(0)}\right),\varepsilon,N,\omega$

输出 迭代解 $x_1,x_2,\cdots,x_n$ 和迭代次数 k

(1) $x_i\Leftarrow x_i^{(0)},i=1,2,\cdots,n$

(2) 对于 $k=1,2,\cdots,N$，循环执行步 3 到步 8

(3) $R\Leftarrow 0$

(4) 对于 $i=1,2,\cdots,n$，循环执行步 5 到步 7

(5) $R_1\Leftarrow\Delta x_i=\frac{\omega}{a_{ii}}\left(b_i-\sum\limits_{j=1}^{n}a_{ij}x_j\right)$

(6) 如果 $|R_1|>|R|$，则 $R\Leftarrow R_1$

(7) $x_i\Leftarrow x_i+R_1$

(8) 如果 $|R|\leqslant\varepsilon$，输出：$x_1,x_2,\cdots,x_n,k$

(9) 已达到最大迭代次数，输出：$x_1,x_2,\cdots,x_n,k$
此时，可另选 $x^{(0)}$ 或 ω 重新迭代计算

五、实验代码

1. Jacobi 迭代法

```
bool Jacobi(double a[N][N], double b[N], double x[N], double e)
{
    double temp_x[N] = {0};
    for(int k = 1; k <= max_iteration; k ++)
    {
        double R = 0;
        for(int i = 0; i < N; i ++)
        {
            if(a[i][i] == 0)
            {
                cout<<"Divided by zero!"<<endl;
                return false;
            }
            // 迭代
            double sum = 0;
            for(int j = 0; j < N; j ++)
            {
                sum += a[i][j] * temp_x[j];
            }
            x[i] = temp_x[i] + (b[i] - sum) / a[i][i];
            if(fabs(x[i] - temp_x[i]) > R)
            {
```

```
                R = fabs(x[i] - temp_x[i]);
            }
            // 判断是否发散
            if(_isnan(x[i])!=0)
            {
                cout<<"算法不收敛! "<<endl;
                return false;
            }
        }
        //判断是否达到精度
        if(R <= e)
        {
            cout<<"迭代次数: "<<k<<endl;
            return true;
        }
        // 记录本次迭代结果
        for(int i = 0; i < N; i ++)
        {
            temp_x[i] = x[i];
        }
    }
    cout<<"迭代次数: "<<max_iteration<<endl;
    return true;
}
```

2. **Gauss-Seidel 迭代法**

```
bool Gauss_Seidel(double a[N][N], double b[N], double x[N], double e)
{
    double temp_x[N] = {0};
    for(int k = 1; k <= max_iteration; k ++)
    {
        double R = 0;
        for(int i = 0; i < N; i ++)
        {
            if(a[i][i] == 0)
            {
                cout<<"Divided by zero!"<<endl;
                return false;
            }
            //迭代计算解
            double sum = 0;
            for(int j = 0; j < N; j ++)
            {
                //不加自己
                if(j == i)
                {
                    continue;
                }
                sum += a[i][j] * x[j];
            }
            x[i] = (b[i] - sum) / a[i][i];
            if(fabs(x[i] - temp_x[i]) > R)
            {
                R = fabs(x[i] - temp_x[i]);
            }
            // 判断是否发散
```

```
            if(_isnan(x[i])!=0)
            {
                cout<<"算法不收敛! "<<endl;
                return false;
            }
        }
        // 判断是否达到精度
        if(R <= e)
        {
            cout<<"迭代次数: "<<k<<endl;
            return true;
        }
        //将上一次计算的结果放置到临时中，为下次迭代做准备x
        for(int i = 0; i < N; i ++)
        {
            temp_x[i] = x[i];
        }
    }
    cout<<"迭代次数: "<<max_iteration<<endl;
    return true;
}
```

3. **SOR 迭代法：**

```
bool SOR(double a[N][N], double b[N], double x[N], double e, double w)
{
    for(int k = 1; k <= max_iteration; k ++)
    {
        double R = 0;
        for(int i = 0; i < N; i ++)
        {
            // 迭代
            double sum = 0;
            for(int j = 0; j < N; j ++)
            {
                sum += a[i][j] * x[j];
            }
            double R1 = w * (b[i] - sum) / a[i][i];
            if(fabs(R1) > fabs(R))
            {
                R = R1;
            }
            x[i] += R1;
            // 判断是否发散
            if(_isnan(x[i])!=0)
            {
                cout<<"算法不收敛! "<<endl;
                return false;
            }
        }
        // 判断是否达到精度
        if(fabs(R) <= e)
        {
            cout<<"迭代次数: "<<k<<endl;
            return true;
        }
    }
```

```
    cout<<"迭代次数: "<<max_iteration<<endl;
    return true;
}
```

六、实验结果

操作系统： Windows 10 企业版

GCC 版本： 4.7.1

G++ 版本： 4.7.1

1. 直接法与迭代法比较

   (1) 直接法在主对角线元素不为 0 的情况下能保证解出方程组，但是有些算法要求方程组必须满足一些特殊条件；迭代法不一定能保证求解方程组。

   (2) 对于直接法，求解方程组的时间与矩阵的大小有关；迭代法的时间是不能确定的，即使对于同一方程组，不同的初值和精度会影响到方程组求解时间，对于 SOR 算法，收敛因子对于求解时间也有影响。

2. 精度对收敛的影响[2]

   (1) 方程组 1

算法	$e = 10^{-3}$	$e = 10^{-4}$	$e = 10^{-5}$
Jacobi 迭代法	不收敛	不收敛	不收敛
Gauss-Seidel 迭代法	不收敛	不收敛	不收敛
SOR 迭代法	不收敛	不收敛	不收敛

   (2) 方程组 2

算法	$e = 10^{-3}$	$e = 10^{-4}$	$e = 10^{-5}$
Jacobi 迭代法	不收敛	不收敛	不收敛
Gauss-Seidel 迭代法	133	570	1006
SOR 迭代法	197	554	991

   (3) 方程组 3

算法	$e = 10^{-3}$	$e = 10^{-4}$	$e = 10^{-5}$
Jacobi 迭代法	11	14	18
Gauss-Seidel 迭代法	7	9	11
SOR 迭代法	8	9	11

   由上表知，精度要求越高，算法收敛越慢。

3. SOR 算法中收敛因子对算法收敛的影响[3]

[2] SOR 算法收敛因子为 1.1

[3] 精度要求 10^{-5}

方程组	$\omega=0.8$	$\omega=0.9$	$\omega=1.0$	$\omega=1.1$	$\omega=1.2$
方程组 2	1083	1079	1006	911	809
方程组 3	15	13	11	11	13

由上表知，对于方程组 2，$\omega=1.2$ 时，算法收敛较快；对于方程组 3，$\omega=1.0,1.1$ 时，算法收敛较快。

七、实验感受

1. 因为算法都有，所以编程没有什么难处，但是得到实验数据需要有耐心。
2. 写完程序之后，感觉对于比较小的矩阵还是用直接法求解比较好。