实验三 *Jacobi* 迭代法
与 *Gauss-Seidel* 迭代法
实验报告

课　　程	计算方法与 MATLAB
院（系）	计算机科学与技术学院
专　　业	人工智能
级　　别	2021 级
学　　号	2115102001
姓　　名	安炳旭
指导老师	陈叶旺

2023 年 12 月 8 日

实验三　*Jacobi* 迭代法与 *Gauss-Seidel* 迭代法

1　算法思想

1.1 *Jacobi* 迭代法简介

对于方程$A\mathbf{x}=\mathbf{b}$而言, 其中$A\in\mathbb{R}^{m\times n}$,$\mathbf{x}\in\mathbb{R}^{n}$,$\mathbf{b}\in\mathbb{R}^{m}$,同时若矩阵$A$非奇异，则可以写成以下形式:

$$x=Gx+f \tag{1}$$

因此，对于其矩阵形式

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}=\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

可以改写成

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}=\begin{bmatrix} 0 & -\dfrac{a_{12}}{a_{11}} & \dots & -\dfrac{a_{1n}}{a_{11}} \\ -\dfrac{a_{21}}{a_{22}} & 0 & \dots & -\dfrac{a_{2n}}{a_{22}} \\ \vdots & \vdots & & \vdots \\ -\dfrac{a_{n1}}{a_{nn}} & \dots & \dots & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}+\begin{bmatrix} \dfrac{b_1}{a_1} \\ \dfrac{b_2}{a_2} \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \tag{2}$$

矩阵形式可以用(3)式表示:

$$\boldsymbol{x}=G_{\boldsymbol{J}}\boldsymbol{x}+\boldsymbol{f} \tag{3}$$

其迭代序列与分量形式分别如(4)式和(5)式表示:

$$\boldsymbol{x}^{(k+1)}=G_J\boldsymbol{x}^{(k)}+\boldsymbol{f}\quad k=0,1,2,\dots \tag{4}$$

$$x_i^{(k+1)}=(b_i-\sum_{j=1,j\neq i}^{n}a_{ij}x_j^{(k)})/a_{ii}\quad i=1,2,\dots,n \tag{5}$$

Jacobi 迭代法基本思想是求解第 i 个方程，以得到第 i 个未知量。对于其线性方程组形式或许更加直观一些，如(6)式所示:

$$\begin{cases} x_1^{(k+1)} = \dfrac{-1}{a_{11}}(a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)} - b_1) \\ x_2^{(k+1)} = \dfrac{-1}{a_{22}}(a_{21}x_1^{(k)} + a_{23}x_3^{(k)} + \cdots + a_{2n}x_n^{(k)} - b_2) \\ \vdots \\ x_n^{(k+1)} = \dfrac{-1}{a_{nn}}(a_{n1}x_1^{(k)} + \cdots + a_{nn-1}x_{n-1}^{(k)} - b_n) \end{cases} \tag{6}$$

$$x_i^{(k+1)} = \frac{-1}{a_{ii}}\left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} + \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} - b_i\right)$$

还可以写出其矩阵形式：

令 A=D-L-U, 其中 D=$(a_{11}, a_{22}, ..., a_{nn})$

$$L = \begin{bmatrix} 0 & & & \\ -a_{21} & 0 & & \\ \cdots & \cdots & \ddots & \\ -a_{n1} & -a_{n2} & \cdots & 0 \end{bmatrix} \quad U = \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ & 0 & \cdots & \cdots \\ & & \ddots & -a_{n-1n} \\ & & & 0 \end{bmatrix}$$

还可以写出其矩阵形式如(7)式所示：

$$\boldsymbol{x}^{(k+1)} = D^{-1}(L+U)\boldsymbol{x}^{(k)} + D^{-1}b \tag{7}$$

于是有$G_J = D^{-1}(L+U), \boldsymbol{f}_J = D^{-1}$

1.2 *Gauss-Seidel* 迭代法简介

对于 *Gauss-Seidel* 迭代法而言，其核心思想是利用最新计算出的分量进行迭代，只需要对(5)式加以修正，得到(8)式

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) \quad (i=1,2,...,n)(k=0,1,...) \tag{8}$$

其线性方程组形式如(9)式所示：

$$\begin{cases} x_1^{(k+1)} = \dfrac{-1}{a_{11}}(a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)} - b_1) \\ x_2^{(k+1)} = \dfrac{-1}{a_{22}}(a_{21}x_1^{(k+1)} + a_{23}x_3^{(k)} + \cdots + a_{2n}x_n^{(k)} - b_2) \\ \vdots \\ x_n^{(k+1)} = \dfrac{-1}{a_{nn}}(a_{n1}x_1^{(k+1)} + \cdots + a_{nn-1}x_{n-1}^{(k+1)} - b_n) \end{cases} \tag{9}$$

$$x_i^{(k+1)} = \frac{-1}{a_{ii}}\left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} - b_i\right)$$

其迭代矩阵形式如(10)所示：

$$\boldsymbol{x}^{(k+1)} = (D-L)^{-1}U\boldsymbol{x}^{(k)} + (D-L)^{-1}b \tag{10}$$

于是有$G_s = (D-L)^{-1}U, \boldsymbol{f}_s = (D-L)^{-1}b$

然而，*Jacobi* 迭代法与 *Gauss-Seidel* 迭代法并不往往保证解的收敛性，两个迭代法收敛的充要条件是

$$\rho(G) < 1 \tag{11}$$

一般情况下，*Gauss-Seidel* 迭代法快于 *Jacobi* 迭代。

2 算法步骤

本次实验需要使用 MATLAB 来具体实现这两个算法，根据第一部分的介绍，很容易能够得到这两个算法的伪代码，首先来分析 *Jacobi* 迭代法的伪代码，详情如表 1 所示：

表 1 *Jacobi* 迭代法的伪代码

```
Input:
   n, (a_ij), (b_i), (x_i), M, ε
1:  for k = 1 to M do
2:    for i = 1 to n do
3:      u_i ← (b_i − Σ_{j≠i}^{n} a_ij x_j)/a_ii;
4:    end for
5:    if ||u − x|| < ε then
6:      break;
7:    else
8:      for i = 1 to n do
9:        x_i ← u_i;
10:     end for
11:   end if
12: end for
Output:
   (x_i)
```

可以看到，在伪代码的第 3 行，*Jacobi* 迭代法求解线性方程组时对于每次迭代，只会更新一次 x 值，刚刚进行迭代的 x 值没有马上进入下一次迭代，这会导致迭代速度变慢。因此 *Gauss-Seidel* 迭代法便是基于这个明显的缺点加以改进，表 2 是 *Gauss-Seidel* 迭代法的伪代码：

表 2 *Jacobi* 迭代法的伪代码

Input:
n, (a_{ij}), (b_i), (x_i), M, ε
1: **for** $k = 1$ to M **do**
2:   **for** $i = 1$ to n **do**
3:     $u_i \leftarrow x_i$;
4:   **end for**
5:   **for** $i = 1$ to n **do**
6:     $x_i \leftarrow (b_i - \sum_{j \neq i}^{n} a_{ij} x_j)/a_{ii}$;
7:   **end for**
8:   **if** $||\mathbf{u} - \mathbf{x}|| < \varepsilon$ **then**
9:     **break;**
10:   **end if**
11: **end for**
Output:
(x_i)

同样是在第 3 行，可以看到在每次迭代结束的 x，会立即进入下一次的迭代过程，这样就大大减少了迭代所需的时间，因此 *Gauss-Seidel* 迭代法具有更高的收敛速度。

3　代码实现

本次实验代码及其功能如下表所示，具体功能请参考注释：

main.m

功能：main 函数，调用 Jacobi 迭代和 Gauss-Seidel 迭代，并进行输出

```
clc;clear;
A=input("请输入系数方阵 A:");
b=input("请输入常数向量 b:");

x_start=input("请输入迭代初始向量 x:");
n_limit=100;
tolerance=10^(-5);
[solution3,n1]=Jacobi(A,b,x_start,n_limit,tolerance);
fprintf('使用 Jacobi 迭代的方程组的解为: %s，迭代次数为: %d\n',
mat2str(solution3), n1);

[solution4,n2]=gauss_seidel(A,b,x_start,n_limit,tolerance);
fprintf('使用 Gauss-Seidel 迭代的方程组的解为: %s，迭代次数为: %d\n',
mat2str(solution4), n2);
```

Jacobi.m

功能：Jacobi 迭代，同时最后可视化误差与迭代次数的图像

```
function [X_reality, n_reality, norms] = Jacobi(A, b, X_start, n_limit, tolerance)
    % A 为迭代的系数矩阵
    % b 为方程组右边的常数项（列向量）
    % X_start 为迭代的初始向量
    % n_limit 为最大允许迭代的次数
    % tolerance 为精度上限值
    % X_reality 为最后结果
    % n_reality 为最后迭代次数
    % norms 为每次迭代后的范数记录

    [n, n] = size(A); % A 的行数和列数均为 n
    D = diag(diag(A)); % D 的对角线元素根 A 的对角线元素相同，其余为 0
    B = inv(D) * (D - A); % B 为雅克比迭代矩阵，也就是化简后的便于迭代的等价方程组的系数矩阵
    f = inv(D) * b; % f 为化简后的便于迭代的等价方程组的常数项向量
    n_reality = 0;
    norms = [];

    while 1
        if(n_reality > n_limit)
            disp('迭代次数超界');
            break;
        end
        X_reality = B * X_start + f; % 雅可比迭代公式
        n_reality = n_reality + 1;

        disp(['第', num2str(n_reality), '次迭代,解为: ', mat2str(X_reality)]);
        norm_value = norm(X_reality - X_start);
        norms = [norms, norm_value]; % 记录范数
        if(norm_value <= tolerance) % 如果满足条件||X(k+1) - X(k)||的 2 范数小于等于 tolerance
            break; % 则退出函数
        else
            X_start = X_reality; % 循环迭代
        end
    end

    % 绘制范数随迭代次数的图像
    figure;
    plot(1:n_reality, norms, '-o');
    xlabel('迭代次数');
    ylabel('误差');
```

```
    title('Jacobi 迭代法误差随迭代次数变化');
end
```

Gauss_seidel.m
功能：Gauss-Seidel 迭代，同时最后可视化误差与迭代次数的图像

```
function [X_reality, n_reality, norms] = gauss_seidel(A, b, X_start,
n_limit, tolerance)
    % A: 系数矩阵，b: 常数向量，X_start: 初始解向量
    % n_limit: 最大迭代次数，tolerance: 精度上限值
    % X_reality: 最后结果，n_reality: 最后迭代次数，norms: 每次迭代后的范数记
录

    n = length(b); % 未知数个数
    X_reality = X_start; % 初始化解向量
    n_reality = 0; % 初始化迭代次数
    norms = []; % 初始化范数记录

    while 1
        if(n_reality > n_limit)
            disp('迭代次数超界');
            break;
        end
        X_old = X_reality; % 保存旧的解向量
        for i = 1:n
            X_reality(i) = (b(i) - A(i,1:i-1)*X_reality(1:i-1) -
A(i,i+1:n)*X_reality(i+1:n)) / A(i,i);
        end
        disp(['第', num2str(n_reality), '次迭代,解为: ',
mat2str(X_reality)]);

        n_reality = n_reality + 1; % 迭代次数加 1
        norm_value = norm(X_reality - X_old);
        norms = [norms, norm_value]; % 记录范数
        if(norm_value <= tolerance) % 如果满足条件||X(k+1) - X(k)||的 2 范数
小于等于 tolerance
            break; % 则退出函数
        end
    end

    % 绘制范数随迭代次数的图像
    figure;
    plot(1:n_reality, norms, '-o');
```

```
    xlabel('迭代次数');
    ylabel('误差');
    title('Gauss-Seidel 迭代法随迭代次数变化');
end
```

4　实验结果

本次实验测试选取了课本例题 3.20 与例 3.7.4, 如下所示：

例3.7.4　$\begin{bmatrix} 4 & -1 & 2 \\ 2 & -5 & 1 \\ -2 & 1 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 7 \\ -1 \\ 6 \end{bmatrix}$

3.20　$\begin{bmatrix} 10 & -2 & -2 \\ -2 & 10 & -1 \\ -1 & -2 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ \frac{1}{2} \\ 1 \end{bmatrix}$

对于例 3.7.4，选取$\boldsymbol{x_0}=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$, $\varepsilon=10^{-5}$，分别使用 *Jacobi* 和 *Gauss-Seidel* 迭代法所得到的迭代过程与结果如表 3 所示：

表 3　课本例 3.7.4 的迭代结果

```
请输入系数方阵 A:[4,-1,2;2,-5,1;-2,1,4]
请输入常数向量 b:[7;-1;6]
请输入迭代初始向量 x:[0;0;0]

Jacobi 迭代:

第 1 次迭代,解为：[1.75;0.2;1.5]
第 2 次迭代,解为：[1.05;1.2;2.325]
第 3 次迭代,解为：[0.8875;1.085;1.725]
第 4 次迭代,解为：[1.15875;0.9;1.6725]
第 5 次迭代,解为：[1.13875;0.998;1.854375]
第 6 次迭代,解为：[1.0723125;1.026375;1.819875]
第 7 次迭代,解为：[1.09665625;0.9929;1.7795625]
第 8 次迭代,解为：[1.10844375;0.994575;1.800103125]
第 9 次迭代,解为：[1.0985921875;1.003398125;1.805578125]
第 10 次迭代,解为：[1.09806046875;1.0005525;1.7984465625]
第 11 次迭代,解为：[1.10091484375;0.9989135;1.798892109375]
第 12 次迭代,解为：[1.1002823203125;1.000144359375;1.800729046875]
第 13 次迭代,解为：[1.09967156640625;1.0002587375;1.8001050703125]
```

```
第 14 次迭代,解为：[1.10001214921875;0.999889640625;1.79977109882813]
第 15 次迭代,解为：[1.10008686074219;0.999959079453125;1.80003366445312]
第 16 次迭代,解为：[1.09997293763672;1.0000414771875;1.80005366050781]
第 17 次迭代,解为：[1.09998353904297;0.99999990715625;1.79997609952148]
第 18 次迭代,解为：[1.10001192702832;0.999988635521484;1.79999179273242]
第 19 次迭代,解为：[1.10000126251416;1.00000312935781;1.80000880463379]
第 20 次迭代,解为：[1.09999638002256;1.00000226593242;1.79999984891763]
第 21 次迭代,解为：[1.10000064202429;0.999998521792549;1.79999762352817]
使用 Jacobi 迭代的方程组的解为:
[1.10000064202429;0.999998521792549;1.79999762352817]，迭代次数为：21

Gauss-Seidel 迭代

第 0 次迭代,解为：[1.75;0.9;2.15]
第 1 次迭代,解为：[0.9;0.99;1.7025]
第 2 次迭代,解为：[1.14625;0.999;1.823375]
第 3 次迭代,解为：[1.0880625;0.9999;1.79405625]
第 4 次迭代,解为：[1.102946875;0.99999;1.8014759375]
第 5 次迭代,解为：[1.09925953125;0.999999;1.799630015625]
第 6 次迭代,解为：[1.1001847421875;0.9999999;1.80009239609375]
第 7 次迭代,解为：[1.09995377695312;0.99999999;1.79997689097656]
第 8 次迭代,解为：[1.10001155201172;0.999999999;1.80000577625586]
第 9 次迭代,解为：[1.09999711162207;0.9999999999;1.79999855583604]
第 10 次迭代,解为：[1.10000072205698;0.99999999999;1.80000036103099]
使用 Gauss-Seidel 迭代的方程组的解为:
[1.10000072205698;0.99999999999;1.80000036103099]，迭代次数为：11
```

作出$\boldsymbol{x}_0=\begin{bmatrix}0\\0\\0\end{bmatrix}$的情况时，*Jacobi* 迭代和 *Gauss-Seidel* 迭代误差随迭代次数变化的曲线如图 1 所示：

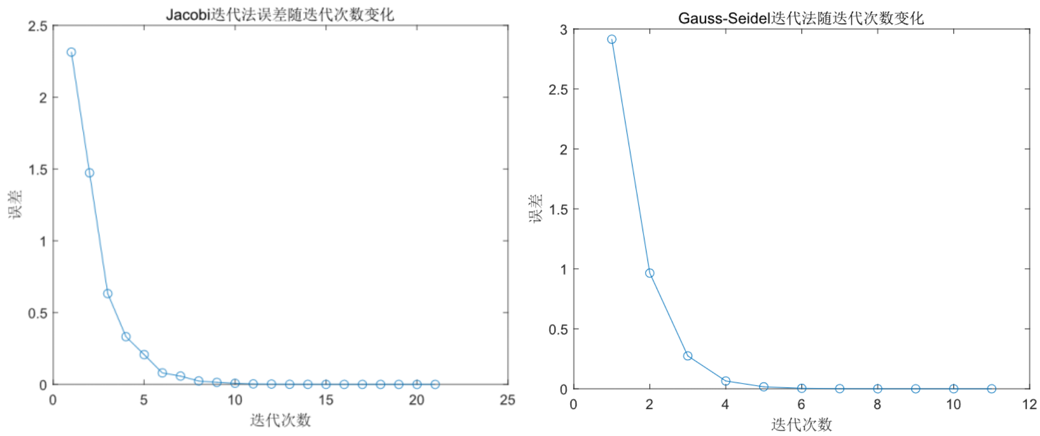

图 1 例 3.7.4 使用 *Jacobi* 迭代和 *Gauss-Seidel* 迭代的 误差-迭代次数 对比图

接下来对于例 3.20，选取$\boldsymbol{x_0}=\begin{bmatrix}0\\0\\0\end{bmatrix}$, $\varepsilon=10^{-5}$，分别使用 *Jacobi* 和 *Gauss-Seidel* 迭代法所得到的迭代过程与结果如表 4 所示：

表 4 课本题 3.20 的迭代结果

```
请输入系数方阵 A:[10,-2,-2;-2,10,-1;-1,-2,3]
请输入常数向量 b:[1;1/2;1]
请输入迭代初始向量 x:[0;0;0]

Jacobi 迭代法

第 1 次迭代,解为：[0.1;0.05;0.333333333333333]
第 2 次迭代,解为：[0.176666666666667;0.103333333333333;0.4]
第 3 次迭代,解为：[0.200666666666667;0.125333333333333;0.461111111111111]
第 4 次迭代,解为：[0.217288888888889;0.136244444444444;0.483777777777778]
第 5 次迭代,解为：[0.224004444444444;0.141835555555556;0.496592592592593]
第 6 次迭代,解为：[0.22768562962963;0.144460148148148;0.502558518518519]
第 7 次迭代,解为：[0.229403733333333;0.145792977777778;0.505535308641975]
第 8 次迭代,解为：[0.230265657283951;0.146434277530864;0.506996562962963]
第 9 次迭代,解为：[0.230686168098765;0.146752787753086;0.507711404115226]
第 10 次迭代,解为：[0.230892838373663;0.146908374031276;0.508063914534979]
第 11 次迭代,解为：[0.230994457713251;0.14698495912823;0.508236528812071]
第 12 次迭代,解为：[0.23104429758806;0.147022544423857;0.508321458656571]
第 13 次迭代,解为：[0.231068800616086;0.147041005383269;0.508363128811925]
第 14 次迭代,解为：[0.231080826839039;0.14705007300441;0.508383603794208]
第 15 次迭代,解为：[0.231086735359724;0.147054525747229;0.508393657615953]
第 16 次迭代,解为：[0.231089636672636;0.14705671283354;0.50839859561806]
使用 Jacobi 迭代的方程组的解为:
[0.231089636672636;0.14705671283354;0.50839859561806]，迭代次数为: 16

Gauss-Seidel 迭代法

第 0 次迭代,解为：[0.1;0.07;0.413333333333333]
第 1 次迭代,解为：[0.196666666666667;0.130666666666667;0.486]
第 2 次迭代,解为：[0.223333333333333;0.143266666666667;0.503288888888889]
第 3 次迭代,解为：[0.229311111111111;0.146191111111111;0.507231111111111]
第 4 次迭代,解为：[0.230684444444444;0.14686;0.508134814814815]
第 5 次迭代,解为：[0.230998962962963;0.147013274074074;0.508341837037037]
第 6 次迭代,解为：[0.231071022222222;0.147048388148148;0.50838926617284]
第 7 次迭代,解为：[0.231087530864198;0.147056432790123;0.508400132148148]
第 8 次迭代,解为：[0.231091312987654;0.147058275812346;0.508402621537449]
```

```
使用 Gauss-Seidel 迭代的方程组的解为:
[0.231091312987654;0.147058275812346;0.508402621537449]，迭代次数为: 9
```

作出$\boldsymbol{x_0}=\begin{bmatrix}0\\0\\0\end{bmatrix}$的情况时，Jacobi 迭代和 Gauss-Seidel 迭代误差随迭代次数变化的曲线如图 2 所示：

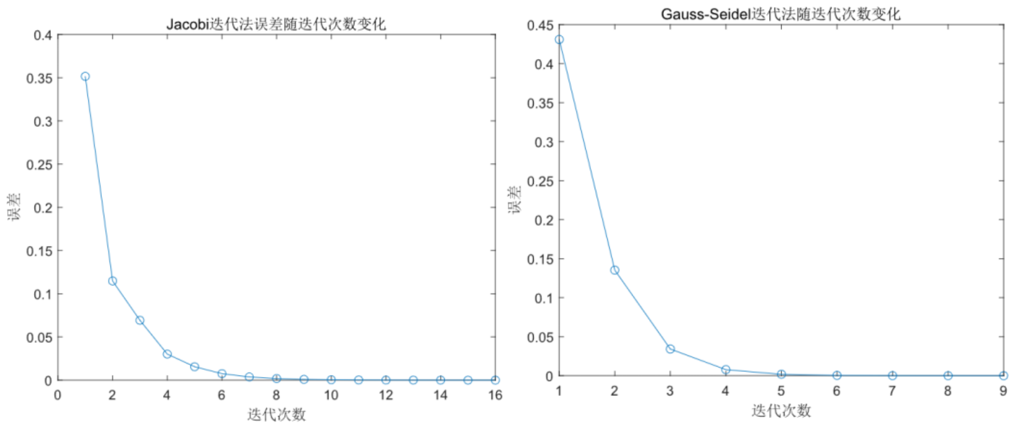

图 2 问题 3.20 使用 *Jacobi* 迭代和 *Gauss-Seidel* 迭代的 误差-迭代次数 对比图

通过实验结果的比较，可以看出对于例 3.7.4 来说，给定初始条件和阈值：

$$\boldsymbol{x_0}=\begin{bmatrix}0\\0\\0\end{bmatrix},\varepsilon=10^{-5}$$

根据图 2，*Jacobi* 迭代使用 21 步达到了所给精度，而 *Gauss-Seidel* 迭代则仅用了 11 步，收敛速度有着极大的提升。同样地，对于问题 3.20 来说，在同样条件下，根据图 3，*Jacobi* 迭代使用 16 步达到了所给精度，而 *Gauss-Seidel* 迭代则仅用了 9 步. 因此，使用 *Gauss-Seidel* 迭代法的收敛速度更快一些。

5　实验结论

为了进一步探索初始条件$\boldsymbol{x_0}$的不同对收敛速度的影响，本实验选取了不同的初始值$\boldsymbol{x_0}=\begin{bmatrix}10\\10\\10\end{bmatrix}$与$\boldsymbol{x_0}=\begin{bmatrix}100\\100\\100\end{bmatrix}$, 针对问题 3.20，得到实验结果如下表 5 所示：

表 5 选取不同初始值$\boldsymbol{x_0}$对收敛步数的影响

	Jacobi 迭代法		*Gauss-Seidel* 迭代法	
初始值$\boldsymbol{x_0}$	迭代	迭代解 x	迭代	迭代解 x

	步数		步数	
$\boldsymbol{x_0}=\begin{bmatrix}10\\10\\10\end{bmatrix}$	21	$\boldsymbol{x}_k=\begin{bmatrix}0.23110\\0.14706\\0.50841\end{bmatrix}$	11	$\boldsymbol{x}_k=\begin{bmatrix}0.23109\\0.14706\\0.50840\end{bmatrix}$
$\boldsymbol{x_0}=\begin{bmatrix}100\\100\\100\end{bmatrix}$	24	$\boldsymbol{x}_k=\begin{bmatrix}0.23110\\0.14706\\0.50840\end{bmatrix}$	13	$\boldsymbol{x}_k=\begin{bmatrix}0.23110\\0.14706\\0.50841\end{bmatrix}$

可以看到，选取的初始值距离解析解越远，需要的步数也就越多，但最终 *Gauss-Seidel* 迭代法还是收敛更快一些。图是使用 Jacobi 迭代和 Gauss-Seidel 迭代时调整不同初始值的误差-迭代次数对比图，左侧是选取$\boldsymbol{x_0}=\begin{bmatrix}10\\10\\10\end{bmatrix}$，右侧是选取$\boldsymbol{x_0}=\begin{bmatrix}100\\100\\100\end{bmatrix}$.

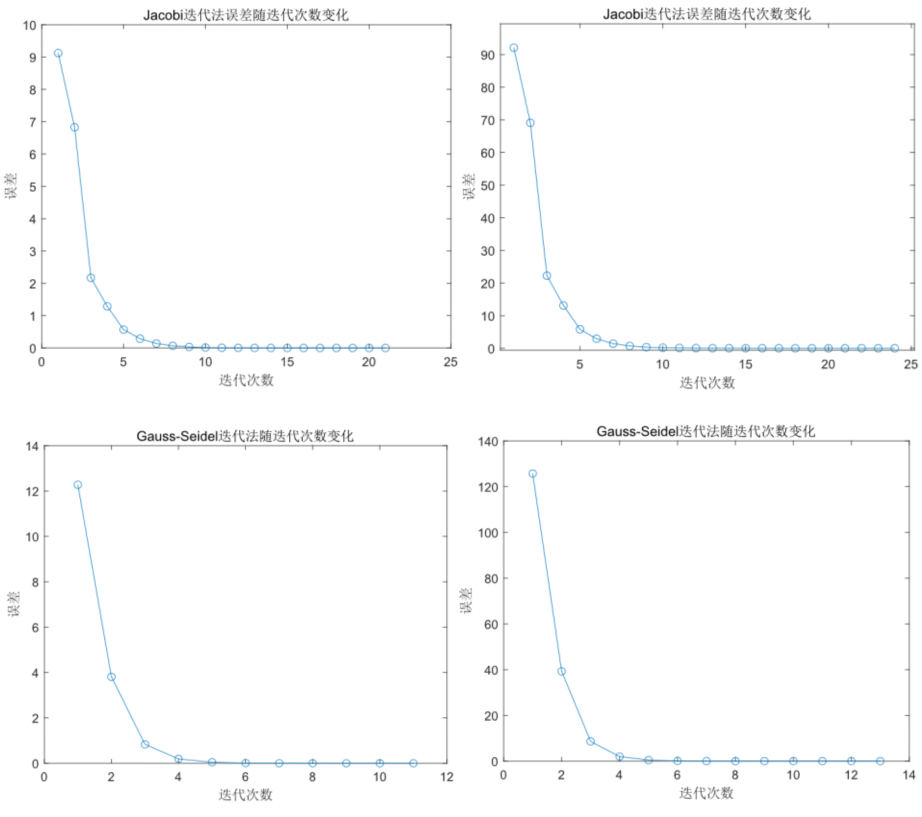

图 3 选取不同初始值$\boldsymbol{x_0}$的误差-迭代次数对比图

最后，给出 *Jacobi* 迭代法和 *Gauss-Seidel* 迭代法的比较，作为本实验的结论

部分：

- *Jacobi* 迭代法

  优点：

  ✧ 相对简单易实现，每个未知数的更新相互独立，可以并行化处理。

  缺点：

  ✧ 收敛速度较慢，特别是对于条件数较大的矩阵。

  ✧ 并不能保证一定是收敛的，要求谱半径$\rho(G)<1$,或者要求矩阵 $\boldsymbol{A}$ 是严格按行/列对角占优的。

- *Gauss-Seidel* 迭代法

  优点：

  ✧ 相对于 Jacobi 法，收敛速度较快，尤其是对于对称正定矩阵。

  缺点：

  ✧ 无法并行化处理，因为每个未知数的更新依赖于前面未知数的更新。

  ✧ 同样不能保证一定是收敛的，对于某些病态矩阵，可能发散而不收敛。

6　实验心得

在进行 Jacobi 迭代法与 Gauss-Seidel 迭代法的实验过程中，本次实验通过编写这两个算法的代码，体会到了这两种方法在解决线性方程组问题上的优缺点和适用场景，同时对迭代初始值进行调整实验，发现合适的迭代初始值迭代法的性能影响较大。

总的来说，通过这次实验，我对 Jacobi 迭代法与 Gauss-Seidel 迭代法有了更深入的理解，也认识到在实际应用中，方法的选择需要根据问题的具体特点来灵活运用。