武漢理工大學
WUHAN UNIVERSITY OF TECHNOLOGY

理学院

高等数值分析实验报告

线性方程组的迭代解法

——迭代算法及对比

姓　　名：袁　子　豪

学　　号：104971210327

指导教师：朱 国 甫 教 授

2021 年 12 月 29 日

摘要

考虑线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 中，当非奇异矩阵 $\boldsymbol{A}$ 为高阶稀疏矩阵时，利用直接的解析法求解会接近奇异阵，这个时候利用迭代法求解线性方程组是更为合适的。迭代法是用某种极限过程去逐步逼近线性方程组精确解的方法，它是解高阶稀疏方程组的重要方法。在计算机内存和运算两方面，迭代法通常都可利用 A 中有大量零元素的特点。迭代法的基本思想是用逐次逼近的方法求解线性方程组。常用的迭代算法包括雅可比迭代法、高斯-塞德尔迭代法、超松弛迭代法和共轭梯度法等。本试验报告将从雅可比迭代法、高斯-塞德尔迭代法、超松弛迭代法入手，模拟计算如何用迭代算法去逼近求解线性方程组。

关键词: 线性方程组迭代解，雅可比迭代法，高斯-赛德尔迭代法，超松弛迭代法

目录

第 1 章　实验目标及准备工作

本实验的目标为在 matlab 环境中使用 matlab 自带的基础函数对三个迭代算法进行编程以实现对任意线性方程组的求解问题，并且使用 matlab 自带的 plot 函数对迭代中的误差大小进行绘图，具象各方法的收敛速度，对各方法进行性能分析比较。具体为：

1. 掌握用迭代法求解线性方程组的基本思想和步骤。

2. 掌握雅可比迭代法，高斯-赛德尔法和松弛法在求解过程中的优缺点。

1.1　线性方程组的迭代解法

迭代法是用某种极限过程去逐步逼近线性方程组精确解的方法，它是解高阶稀疏方程组的重要方法。迭代法的基本思想是用逐次逼近的方法求解线性方程组。

设有线性方程组

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.1}$$

其中，$\boldsymbol{A} = (a_{ij}) \in \mathbf{R}^{n\times n}$ 为非奇异矩阵。将其转化为等价的迭代形式的相容方程组

$$\boldsymbol{x} = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{f} \tag{1.2}$$

并由此构造迭代公式

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{G}\boldsymbol{x}^{(k)} + \boldsymbol{f} \tag{1.3}$$

式中 $\boldsymbol{G}$ 称为迭代矩阵，$\boldsymbol{f}$ 称为迭代向量。对任意的初始向量 $\boldsymbol{x}^0$，由式 (1.3) 可求得向量序列 $\{\boldsymbol{x}^{(k)}\}_0^{\infty}$，若 $\lim\limits_{k\to\infty}\boldsymbol{x}^{(k)} = \boldsymbol{x}^*$，则 $\boldsymbol{x}^*$ 就是方程 (1.1) 或 (1.2) 的解。此时迭代公式 (1.2) 是收敛的，否则称为发散的。构造的迭代公式 (1.3) 是否收敛，取决于迭代矩阵 $\boldsymbol{G}$ 的性质。引进误差向量

$$\boldsymbol{\varepsilon}^{(k)} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^* \tag{1.4}$$

由 (1.3) 式减去 (1.2) 式，得 $\varepsilon^{(k+1)} = \boldsymbol{G}\boldsymbol{\varepsilon}^{(k)}(k = 0, 1, 2, \ldots)$，递推得

$$\boldsymbol{\varepsilon}^{(k)} = \boldsymbol{G}\boldsymbol{\varepsilon}^{(k-1)} = \cdots = \boldsymbol{G}^k\boldsymbol{\varepsilon}^{(0)}$$

要考察 $\{\boldsymbol{x}^{(k)}\}$ 的收敛性，就要研究 $\boldsymbol{G}$ 在什么条件下有 $\lim\limits_{k\to\infty}\boldsymbol{\varepsilon}^{(k)}$，亦即要研究 $\boldsymbol{G}$ 满足什么条件时有 $\boldsymbol{G}^{(k)} \to \boldsymbol{0}(k \to \infty)$。将 $\boldsymbol{A}$ 分裂为

$$\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N} \tag{1.5}$$

其中，$\boldsymbol{M}$ 为可选择的非奇异矩阵。于是，求解问题转化为求解 $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{N}\boldsymbol{x} + \boldsymbol{b}$。即

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \Leftrightarrow \boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{N}\boldsymbol{x} + \boldsymbol{M}^{-1} + \boldsymbol{b} \tag{1.6}$$

也就是求解线性方程组 $\boldsymbol{x} = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{f}$，从而可构造一阶定常迭代法：

$$\begin{cases} \boldsymbol{x}^{(0)} \quad (\text{初始向量}) \\ \boldsymbol{x}^{(k+1)} = \boldsymbol{G}\boldsymbol{x}^{(k)} + \boldsymbol{f}, k = 0, 1, \ldots \end{cases} \tag{1.7}$$

其中，$\boldsymbol{G}=\boldsymbol{I}-\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{f}=\boldsymbol{M}^{-1}\boldsymbol{b}$。迭代法 (1.7) 式收敛的充分必要条件为：对任意给定的线性方程组 (1.1)，选取初始向量 $\boldsymbol{x}^{(0)}$，迭代法收敛的充要条件为：迭代矩阵 $\boldsymbol{G}$ 的谱半径 $\rho(\boldsymbol{G})<1$。其中 $\rho(\boldsymbol{G})=\max\limits_{1\leq i\leq n}|\lambda_i|$，$\lambda_1,\lambda_2,\ldots,\lambda_n$ 为迭代矩阵 $\boldsymbol{G}$ 的全部特征值。

1.2 实验环境

本实验运行在 matlab(R2021b) 的 macOS 版本下，界面如图1.1所示。

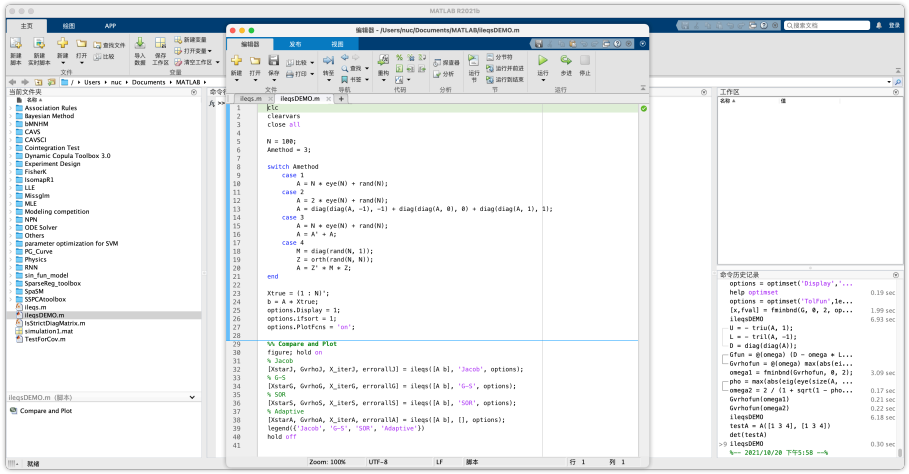

图 1.1: matlab 运行环境示意图

1.3 编译环境

本报告编译环境为 macOS 下的 Texpad，界面如图1.2所示。

图 1.2: Texpad 编译排版环境示意图

第 2 章　算法

设有线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，若将系数矩阵 $\boldsymbol{A}$ 分解为

$$\boldsymbol{A}=\boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U} \tag{2.1}$$

其中

$$\boldsymbol{A}=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \qquad \boldsymbol{D}=\begin{pmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{pmatrix}$$

$$\boldsymbol{L}=\begin{pmatrix} 0 & & & \\ a_{21} & 0 & & \\ \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix} \qquad \boldsymbol{U}=\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ & 0 & \dots & a_{2n} \\ & & \ddots & \vdots \\ & & & 0 \end{pmatrix}$$

于是有

$$(\boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U})\boldsymbol{x}=\boldsymbol{b} \tag{2.2}$$

下面将根据式 (2.2) 推导出雅可比迭代法、高斯-塞德尔迭代法、超松弛迭代法的结构。

2.1　雅克比迭代算法原理

设有方程组

$$\sum_{i=1}^{n} a_{ij}x_j=b_j \quad (i=1,2,3,\cdots,n) \tag{2.3}$$

矩阵形式为 $\boldsymbol{Ax}=\boldsymbol{b}$，设系数矩阵 $\boldsymbol{A}$ 为非奇异矩阵，且 $a_{ii}\neq 0,(i=1,2,3,\cdots,n)$ 从式 (2.3) 中第 i 个方程中解出 $\boldsymbol{x}$, 得其等价形式

$$x_i=\frac{1}{a_{ii}}(b-\sum_{j=1,j\neq i}^{n} a_{ij}x_j) \tag{2.4}$$

取初始向量 $x^{(0)}=(x_1^{(0)},x_2^{(0)},\ldots,x_n^{(0)})$，对式 (2.4) 应用迭代法，可建立相应的迭代公式：

$$x_i^{(k+1)}=\frac{1}{a_{ii}}(-\sum_{j=1,j\neq i}^{n} a_{ij}x_j^{(k)}+b_i) \tag{2.5}$$

也可记为矩阵形式：

$$\boldsymbol{x}^{(k+1)}=\boldsymbol{G}_J\boldsymbol{x}^k+\boldsymbol{f}_J \tag{2.6}$$

因为

$$\boldsymbol{Dx}=(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}+\boldsymbol{b}$$

于是有

$$\begin{aligned}\boldsymbol{x} &= \boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x} + \boldsymbol{D}^{-1}\boldsymbol{b} \\ &= \boldsymbol{D}^{-1}(\boldsymbol{D}-\boldsymbol{A})\boldsymbol{x} + \boldsymbol{D}^{-1}\boldsymbol{b} \\ &= (\boldsymbol{I}-\boldsymbol{D}^{-1}\boldsymbol{A})\boldsymbol{x} + \boldsymbol{D}^{-1}\boldsymbol{b}\end{aligned} \tag{2.7}$$

将式 (2.7) 代入式 (2.6) 中，则 $\boldsymbol{G}_J$ 和 $\boldsymbol{f}_J$ 表示为

$$\begin{aligned}\boldsymbol{G}_J &= \boldsymbol{I}-\boldsymbol{D}^{-1}\boldsymbol{A} \\ \boldsymbol{f}_J &= \boldsymbol{D}^{-1}\boldsymbol{b}\end{aligned} \tag{2.8}$$

则迭代是否收敛的充分条件为：$\rho(\boldsymbol{G}_J) < 1$。其伪代码如下所示。

Algorithm 1 求解线性方程组的雅可比 (Jacob) 迭代法

Input: 增广矩阵 $(\boldsymbol{A}\ \boldsymbol{b})$, 范数 p，初始值 $\boldsymbol{x}^{(0)}$，迭代残差的容忍误差 $\|\boldsymbol{\varepsilon}^*\|$，迭代解的容忍误差 $\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^*\|$，最大迭代次数 K；

Output: 最终迭代解 $\boldsymbol{x}^*$，迭代矩阵的谱半径 $\rho(\boldsymbol{G}_J)$，每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$，每次迭代误差范数 $\|\boldsymbol{\varepsilon}\|_p$；

begin

对矩阵 $\boldsymbol{A}$ 进行如式 (2.1) 的分解得到 $\boldsymbol{D}$，$\boldsymbol{L}$ 和 $\boldsymbol{U}$；

根据式 (2.8) 计算矩阵 $\boldsymbol{G}_J$，$\boldsymbol{f}_J$；设置 $k=0$；$\boldsymbol{x}_{iter}=[\quad]$；$\|\boldsymbol{\varepsilon}\|_p=[\quad]$；

if $\rho(\boldsymbol{G}_J) >= 1$ **then**

    error(算法不收敛)；break;

while $k < K$ **do**

    $k=k+1$；根据式 (2.6) 计算 $\boldsymbol{x}^{(k)}$，$\|\boldsymbol{\varepsilon}^k\|_p = \|\boldsymbol{A}\boldsymbol{x}^{(k)}-\boldsymbol{b}\|_p$；$\boldsymbol{x}_{iter}=[\boldsymbol{x}_{iter}\quad \boldsymbol{x}_{(k)}]$；$\|\boldsymbol{\varepsilon}\|_p=[\|\boldsymbol{\varepsilon}\|_p\quad \|\boldsymbol{\varepsilon}^k\|_p]$；$\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^k\|_p = \|\boldsymbol{x}_k-\boldsymbol{x}_{k-1}\|_p$；

    if $(\|\boldsymbol{\varepsilon}^k\|_p < \|\boldsymbol{\varepsilon}^*\|)$ || $(\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^k\|_p < \|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^*\|)$ **then**

        break;

return $\boldsymbol{x}^*=\boldsymbol{A}\boldsymbol{x}^{(k)}$，$\rho(\boldsymbol{G}_J)$，$\boldsymbol{x}_{iter}$，$\|\boldsymbol{\varepsilon}\|_p$；

end

2.2 高斯-赛德尔迭代法

高斯-赛德尔 (Gauss-Seidel) 迭代法, 其迭代公式为

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + b_i\right) \quad (\mathrm{i}=1,2,\cdots,\mathrm{n}) \tag{2.9}$$

也可以写成矩阵形式

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{B}_{G-S}\boldsymbol{x}^{(k)} + \boldsymbol{f}_{G-S} \tag{2.10}$$

这时将式 (2.2) 变为

$$\boldsymbol{D}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{x} + \boldsymbol{U}\boldsymbol{x} + \boldsymbol{b} \tag{2.11}$$

将最新分量代替为旧分量, 得

$$\boldsymbol{D}\boldsymbol{x}^{(k+1)} = \boldsymbol{L}\boldsymbol{x}^{(k+1)} + \boldsymbol{U}\boldsymbol{x}^{(k)} + \boldsymbol{b} \tag{2.12}$$

即

$$(\boldsymbol{D} - \boldsymbol{L})\boldsymbol{x}^{(k+1)} = \boldsymbol{U}\boldsymbol{x}^{(k)} + \boldsymbol{b} \tag{2.13}$$

于是有

$$\boldsymbol{x}^{(k+1)} = (\boldsymbol{D} - \boldsymbol{L})^{-1}\boldsymbol{U}\boldsymbol{x}^{(k)} + (\boldsymbol{D} - \boldsymbol{L})^{-1}\boldsymbol{b} \tag{2.14}$$

则 $\boldsymbol{G}_{G-S}$ 和 $\boldsymbol{f}_{G-S}$ 表示为

$$\begin{aligned} \boldsymbol{G}_{G-S} &= (\boldsymbol{D} - \boldsymbol{L})^{-1}\boldsymbol{U} \\ \boldsymbol{f}_{G-S} &= (\boldsymbol{D} - \boldsymbol{L})^{-1}\boldsymbol{b} \end{aligned} \tag{2.15}$$

则迭代是否收敛的充分条件为：$\rho(\boldsymbol{G}_{G-S}) < 1$。其伪代码如下所示。

Algorithm 2 求解线性方程组的高斯一赛德尔 (Gauss-Seidel) 迭代法

Input: 增广矩阵 $(\boldsymbol{A}\ \boldsymbol{b})$, 范数 p, 初始值 $\boldsymbol{x}^{(0)}$, 迭代残差的容忍误差 $\|\boldsymbol{\varepsilon}^*\|$, 迭代解的容忍误差 $\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^*\|$, 最大迭代次数 K;

Output: 最终迭代解 $\boldsymbol{x}^*$, 迭代矩阵的谱半径 $\rho(\boldsymbol{G}_{G-S})$, 每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$, 每次迭代误差范数 $\|\boldsymbol{\varepsilon}\|_p$;

begin

对矩阵 $\boldsymbol{A}$ 进行如式 (2.1) 的分解得到 $\boldsymbol{D}$, $\boldsymbol{L}$ 和 $\boldsymbol{U}$;

根据式 (2.15) 计算矩阵 $\boldsymbol{G}_{G-S}$, $\boldsymbol{f}_{G-S}$; 设置 $k = 0$; $\boldsymbol{x}_{iter} = [\,]$; $\|\boldsymbol{\varepsilon}\|_p = [\,]$;

if $\rho(\boldsymbol{G}_{G-S}) >= 1$ **then**

    error(算法不收敛); break;

while $k < K$ **do**

    $k = k + 1$; 根据式 (2.14) 计算 $\boldsymbol{x}^{(k)}$, $\|\boldsymbol{\varepsilon}^k\|_p = \|\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}\|_p$; $\boldsymbol{x}_{iter} = [\boldsymbol{x}_{iter}\ \boldsymbol{x}_{(k)}]$; $\|\boldsymbol{\varepsilon}\|_p = [\|\boldsymbol{\varepsilon}\|_p\ \|\boldsymbol{\varepsilon}^k\|_p]$; $\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^k\|_p = \|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|_p$;

    if $(\|\boldsymbol{\varepsilon}^k\|_p < \|\boldsymbol{\varepsilon}^*\|)$ || $(\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^k\|_p < \|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^*\|)$ **then**

        break;

return $\boldsymbol{x}^* = \boldsymbol{A}\boldsymbol{x}^{(k)}$, $\rho(\boldsymbol{G}_{G-S})$, $\boldsymbol{x}_{iter}$, $\|\boldsymbol{\varepsilon}\|_p$;

end

定理 1. 设 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, 如果：

(1) $\boldsymbol{A}$ 为严格对角占优矩阵，则解的雅可比迭代法，高斯-塞德尔迭代法均收敛。

(2) $\boldsymbol{A}$ 为弱对角占优矩阵，且 $\boldsymbol{A}$ 为不可约矩阵，则解的雅可比迭代法，高斯-塞德尔迭代法均收敛。

定理 2. 设矩阵 $\boldsymbol{A}$ 对称，且对角元 $a_{ii} > 0 (i = 1, 2, \cdots, n)$，则

(1) 解线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 的雅可比迭代法收敛的充分必要条件是 $\boldsymbol{A}$ 及 $\boldsymbol{D} - \boldsymbol{A}$ 均为正定矩阵。

(2) 解线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 的高斯-塞德尔法收敛的充分条件是 $\boldsymbol{A}$ 正定。

2.3 超松弛迭代法

选取分裂矩阵 $\boldsymbol{M}$ 为带参数的下三角矩阵

$$\boldsymbol{M} = \frac{1}{\omega}(\boldsymbol{D} - \omega \boldsymbol{L}) \tag{2.16}$$

其中 $\omega > 0$ 为可选择的松弛因子.

于是, 由 (2.16) 式可构造一个迭代法, 其迭代矩阵为

$$\boldsymbol{G}_\omega \equiv \boldsymbol{I} - \omega(\boldsymbol{D} - \omega \boldsymbol{L})^{-1}\boldsymbol{A} = (\boldsymbol{D} - \omega \boldsymbol{L})^{-1}((1 - \omega)\boldsymbol{D} + \omega \boldsymbol{U}) \tag{2.17}$$

从而得到解 $\boldsymbol{Ax} = \boldsymbol{b}$ 的逐次超松弛迭代法 (successive over relaxation method, 简称 SOR 方法)。解 $\boldsymbol{Ax} = \boldsymbol{b}$ 的 SOR 方法为

$$\begin{cases} \boldsymbol{x}^{(0)}, \quad \text{初始向量}, \\ \boldsymbol{x}^{(k+1)} = \boldsymbol{G}_\omega \boldsymbol{x}^{(k)} + \boldsymbol{f}, \quad k = 0, 1, \cdots \end{cases} \tag{2.18}$$

其中

$$\begin{aligned} \boldsymbol{G}_\omega &= (\boldsymbol{D} - \omega \boldsymbol{L})^{-1}((1 - \omega)\boldsymbol{D} + \omega \boldsymbol{U}) \\ \boldsymbol{f}_\omega &= \omega(\boldsymbol{D} - \omega \boldsymbol{L})^{-1}\boldsymbol{b} \end{aligned} \tag{2.19}$$

下面给出解 $\boldsymbol{Ax} = \boldsymbol{b}$ 的 SOR 迭代法的分量计算公式。记 $\boldsymbol{x}^{(k)} = \left(x_1^{(k)}, \cdots, x_i^{(k)}, \cdots, x_n^{(k)}\right)^{\mathrm{T}}$, 由式 (2.18) 可得

$$(\boldsymbol{D} - \omega \boldsymbol{L})\boldsymbol{x}^{(k+1)} = ((1 - \omega)\boldsymbol{D} + \omega \boldsymbol{U})\boldsymbol{x}^{(k)} + \omega \boldsymbol{b} \tag{2.20}$$

或

$$\boldsymbol{D}\boldsymbol{x}^{(k+1)} = \boldsymbol{D}\boldsymbol{x}^{(k)} + \omega\left(\boldsymbol{b} + \boldsymbol{L}\boldsymbol{x}^{(k+1)} + \boldsymbol{U}\boldsymbol{x}^{(k)} - \boldsymbol{D}\boldsymbol{x}^{(k)}\right) \tag{2.21}$$

由此, 得到解的 SOR 方法的计算公式

$$\begin{cases} \boldsymbol{x}^{(0)} = \left(x_1^{(0)}, \cdots, x_n^{(0)}\right)^{\mathrm{T}}, \\ x_i^{(k+1)} = x_i^{(k)} + \omega\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)}\right)/a_{ii} \\ \quad i = 1, 2, \cdots, n, k = 0, 1, \cdots \\ \omega \text{ 为松弛因子} \end{cases} \tag{2.22}$$

(1) 显然, 当 $\omega = 1$ 时，SOR 方法即为高斯-塞德尔迭代法。

(2) SOR 方法每迭代一次主要运算量是计算一次矩阵与向量的乘法。

(3) 当 $\omega > 1$ 时, 称为超松弛法；当 $\omega < 1$ 时, 称为低松弛法。

(4) 实现算法时可用

$$\max_{1\le i\le n} |\Delta x_i| = \max_{1\le i\le n} \left| x_i^{(k+1)} - x_i^{(k)} \right| < \varepsilon \tag{2.23}$$

控制迭代终止，或用

$$\left\| \boldsymbol{\varepsilon}^{(k)} \right\|_\infty = \left\| \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)} \right\|_\infty < \varepsilon \tag{2.24}$$

控制迭代终止。

SOR 迭代法是高斯-塞德尔迭代法的一种修正, 可由下述思想得到. 设已知 $\boldsymbol{x}^{(k)}$ 及已计算 $\boldsymbol{x}^{(k+1)}$ 的分量 $x_j^{(k+1)}(j = 1, 2, \cdots, i-1)$。

(1) 首先用高斯-塞德尔迭代法定义辅助量 $\widetilde{x}_i^{(k+1)}$,

$$\widetilde{x}_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii} \tag{2.25}$$

(2) 再由 $x_i^{(k)}$ 与 $\widetilde{x}_i^{(k+1)}$ 加权平均定义 $x_i^{(k+1)}$, 即

$$x_i^{(k+1)} = (1-\omega) x_i^{(k)} + \omega \widetilde{x}_i^{(k+1)} = x_i^{(k)} + \omega \left(\tilde{x}_i^{(k+1)} - x_i^{(k)} \right) \tag{2.26}$$

迭代是否收敛的充分条件为：$\rho(\boldsymbol{G}_\omega) < 1$。其伪代码如下所示。

Algorithm 3 求解线性方程组的逐次超松弛 (SOR) 迭代法

Input: 增广矩阵 $(\boldsymbol{A}\ \boldsymbol{b})$, 范数 p, 初始值 $\boldsymbol{x}^{(0)}$, 迭代残差的容忍误差 $\|\boldsymbol{\varepsilon}^*\|$, 迭代解的容忍误差 $\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^*\|$, 最大迭代次数 K, 松弛因子 ω;

Output: 最终迭代解 $\boldsymbol{x}^*$, 迭代矩阵的谱半径 $\rho(\boldsymbol{G}_{G-S})$, 每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$, 每次迭代误差范数 $\|\boldsymbol{\varepsilon}\|_p$;

begin

对矩阵 $\boldsymbol{A}$ 进行如式 (2.1) 的分解得到 $\boldsymbol{D}$, $\boldsymbol{L}$ 和 $\boldsymbol{U}$;

根据式 (2.19) 计算矩阵 $\boldsymbol{G}_\omega$, $\boldsymbol{f}_\omega$; 设置 $k = 0$; $\boldsymbol{x}_{iter} = [\,]$; $\|\boldsymbol{\varepsilon}\|_p = [\,]$;

if $\rho(\boldsymbol{G}_\omega) >= 1$ **then**

    error(算法不收敛); break;

while $k < K$ **do**

    $k = k + 1$; 根据式 (2.18) 计算 $\boldsymbol{x}^{(k)}$, $\|\boldsymbol{\varepsilon}^k\|_p = \|\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}\|_p$; $\boldsymbol{x}_{iter} = [\boldsymbol{x}_{iter}\ \boldsymbol{x}_{(k)}]$; $\|\boldsymbol{\varepsilon}\|_p = [\|\boldsymbol{\varepsilon}\|_p\ \|\boldsymbol{\varepsilon}^k\|_p]$; $\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^k\|_p = \|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|_p$;

    if $(\|\boldsymbol{\varepsilon}^k\|_p < \|\boldsymbol{\varepsilon}^*\|)$ || $(\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^k\|_p < \|\boldsymbol{\varepsilon}_{\boldsymbol{x}}^*\|)$ **then**

        break;

return $\boldsymbol{x}^* = \boldsymbol{A}\boldsymbol{x}^{(k)}$, $\rho(\boldsymbol{G}_\omega)$, $\boldsymbol{x}_{iter}$, $\|\boldsymbol{\varepsilon}\|_p$;

end

第 3 章　实验过程（附代码）

与一般的仿真处理全流程类似，本实验的过程主要包括数据的生成、迭代方法的选择、计算结果分析和数据的可视化展示等几个主要步骤。实验的基本思路如图3.1所示。

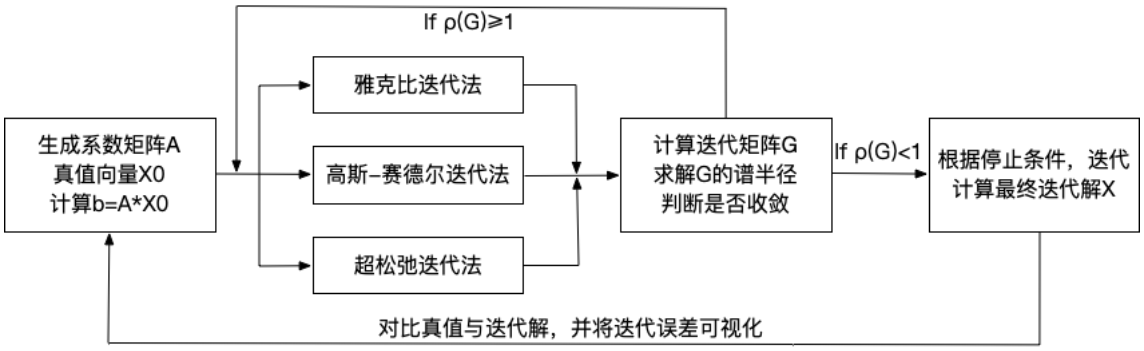

图 3.1: 本实验基本思路

根据基本思路图，本实验对计算所有流程进行了编程，且均已上传到 GitHub[1]上。

3.1　线性方程组迭代法的迭代矩阵代码编写

为了对不同的线性方程组求解，本实验采用将迭代方法也作为输入变量进行编程。

3.1.1　雅可比迭代矩阵代码 JacobLE.m 的编写

通过对系数矩阵的分裂可以计算出雅可比迭代矩阵，将代码保存为 “JacobLE.m”。

```
function [G, f] = JacobLE(A, b)
% Compute for iterative matrix and residual vector of Jacobian method
% Input:
%   A - Coefficient matrix
%   b - Constant column vector
% Output:
%   G - Iterative matrix
%   f - Residual vector

% Author  : ZH.Yuan
% Update  : 2021/10/19 (First Version: 2021/10/19)
% Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)

G = eye(size(A, 1)) - diag(1./ diag(A)) * A;
f = diag(1./ diag(A)) * b;
end
```

[1]https://github.com/yuanzihao945/ILEQS

3.1.2 高斯-赛尔德迭代矩阵代码 GSLE.m 的编写

通过对系数矩阵的分裂可以计算出高斯-赛尔德迭代矩阵，将代码保存为“GSLE.m”。

```
function [G, f] = GSLE(A, b)
% Compute for iterative matrix and residual vector of Gauss-Seidel method
% Input:
%   A - Coefficient matrix
%   b - Constant column vector
% Output:
%   G - Iterative matrix
%   f - Residual vector

% Author  : ZH.Yuan
% Update  : 2021/10/19 (First Version: 2021/10/19)
% Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)

U = - triu(A, 1);
L = - tril(A, -1);
D = diag(diag(A));
G = (D - L)^-1 * U;
f = (D - L)^-1 * b;

end
```

3.1.3 逐次超松弛迭代矩阵 SORLE.m 的编写

通过对系数矩阵的分裂可以计算出逐次超松弛迭代矩阵，将代码保存为“SORLE.m”。

```
function [G, f] = SORLE(A, b, omega)
% Compute for iterative matrix and residual vector of SOR method
% Input:
%   A - Coefficient matrix
%   b - Constant column vector
%   omega - Relaxation factor
% Output:
%   G - Iterative matrix
%   f - Residual vector

% Author  : ZH.Yuan
% Update  : 2021/10/19 (First Version: 2021/10/19)
% Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)

U = - triu(A, 1);
L = - tril(A, -1);
D = diag(diag(A));
Gfun = @(omega) (D - omega * L)^-1 * ((1 - omega) * D + omega * U);
Gvrhofun = @(omega) max(abs(eig(Gfun(omega))));

```

```
if ~exist('omega', 'var') || isempty(omega)
    [Alower, Aupper] = bandwidth(A);
    if Alower + Aupper == 2
        pho = max(abs(eig(JacobLE(A, b))));
        omega = 2 / (1 + sqrt(1 - pho^2));
    else
        omega = fminbnd(Gvrhofun, 0, 2);
    end
end

G = Gfun(omega);
f = omega * (D - omega * L)^-1 * b;

end
```

3.1.4 自适应迭代矩阵 AdaptiveLE.m 的编写

通过对系数矩阵的分裂可以计算出逐次超松弛迭代矩阵，将代码保存为“AdaptiveLE.m”。

```
function [G, f, method] = AdaptiveLE(A, b, omega)
% Compute for iterative matrix and residual vector of adaptive choose method
% Input:
%   A - Coefficient matrix
%   b - Constant column vector
%   omega - Relaxation factor
% Output:
%   G - Iterative matrix
%   f - Residual vector
%   method - The fastest convergence method

% Author  : ZH.Yuan
% Update  : 2021/10/19 (First Version: 2021/10/19)
% Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)

if ~exist('omega', 'var')
    omega = [];
end

[G1, f1] = JacobLE(A, b);
[G2, f2] = GSLE(A, b);
[G3, f3] = SORLE(A, b, omega);

Gvrho1 = max(abs(eig(G1)));
Gvrho2 = max(abs(eig(G2)));
Gvrho3 = max(abs(eig(G3)));

[~, minGvrho] = min([Gvrho1, Gvrho2, Gvrho3]);
```

```
29
30 switch minGvrho
31     case 1
32         G = G1;
33         f = f1;
34         method = 'Jacob';
35     case 2
36         G = G2;
37         f = f2;
38         method = 'G-S';
39     case 3
40         G = G3;
41         f = f3;
42         method = 'SOR';
43 end
44
45 end
```

3.2 迭代求解模块 ileqs.m 的编写

根据第二节的计算公式和之前编写的迭代矩阵函数作为基础，将方法和增广矩阵作为输入，进行计算：

```
1  function [Xstar, Gvrho, X_iter, errorFun] = ileqs(Ab, method, options)
2  % ILEQS - Iterative optimization solution of linear equations
3  %
4  % Input:
5  %   Ab       - Augmented matrix of linear equations, N * (N + 1) double matrix
6  %   method   - Method of iteration: 'Jacob', 'G-S', 'SOR' or 'Adaptive'
7  %   options  - Options of algorithm(Struct data), include:
8  %              - options.Ifsort: input 0(default) or 1, if 1, swap rows and
9  %                                columns to ensure the diagonal elements maximum
10 %              - options.p: use p-norm
11 %              - options.X0: Intial value of algorithm
12 %              - options.TolFun: tolerance of equation
13 %              - options.TolX: tolerance of solution
14 %              - options.Maxiter: Maximum number of iterations
15 %              - options.Display: If display the final iterations number
16 %              - options.omega: omega is the relaxation factor of 'SOR' method
17 %              - options.PlotFcns: Draw the error value of each iteration
18 %                                  during the execution of the algorithm
19 %
20 % Output:
21 %   Xstar    - Iterative solutions of equations
22 %   Gvrho    - Spectral radius of Iterative matrix
23 %   X_iter   - Iterative solutions in every iteration
24 %   errorFun - Error of linear equations in each iteration
```

```
25 %
26 % Usage:
27 %    [] = ILEQS(Ab) uses the default settings soluting linear equations
28 %    [] = ILEQS(Ab, method) uses the input method to iterate linear equatioins
29 %    [] = ILEQS(Ab, [],options) uses the options' settings soluting linear equatioins
30 %    Xstar = ILEQS( ... ) returns the iterative solution of linear equations
31 %    [~, Gvrho] = ILEQS( ... ) returns the spectral radius of iterative matrix
32 %    [~, ~, X_iter] = ILEQS( ... ) returns the solutions in each iterations
33 %    [~, ~, ~, error] = ILEQS( ... ) returns the error of linear equations
34 %        in each iterations
35
36 % Author  : ZH.Yuan
37 % Update  : 2021/10/20 (First Version: 2021/10/19)
38 % Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)
39
40 [N, N1] = size(Ab);
41 if N1 - N ~= 1
42     error('Format of A is Wrong, please check it!')
43 end
44 A = Ab(:, 1 : end - 1);
45 b = Ab(:, end);
46 if det(A) == 0
47     error('The coefficient matrix is a singular matrix!')
48 end
49
50 % Set default value of value
51 if ~exist('method', 'var') || isempty(method)
52     method = 'Adaptive';
53 end
54
55 % Set default value of options
56 if ~exist('options', 'var') || isempty(options)
57     options.Ifsort = 1;
58     options.p = 2;
59     options.X0 = zeros(N, 1);
60     options.TolFun = 1e-08;
61     options.TolX = 1e-08;
62     options.Maxiter = max([200 N]);
63     options.Display = 0;
64     options.omega = [];
65     options.PlotFcns = 'off';
66 end
67
68 % Set default value of options.Ifsort
69 if ~isfield(options, 'Ifsort')
70     options.Ifsort = 1;
71 elseif isempty(options.Ifsort)
72     options.Ifsort = 1;
```

```
73  end
74
75  % Set default value of options.p
76  if ~isfield(options, 'p')
77      options.p = 2;
78  elseif isempty(options.p)
79      options.p = 2;
80  end
81
82  % Set default value of options.X0
83  if ~isfield(options, 'X0')
84      options.X0 = zeros(N, 1);
85  elseif isempty(options.X0)
86      options.X0 = zeros(N, 1);
87  end
88
89  % Set default value of options.TolFun
90  if ~isfield(options, 'TolFun')
91      options.TolFun = 1e-04;
92  elseif isempty(options.TolFun)
93      options.TolFun = 1e-04;
94  end
95
96  % Set default value of options.TolX
97  if ~isfield(options, 'TolX')
98      options.TolX = 1e-04;
99  elseif isempty(options.TolX)
100     options.TolX = 1e-04;
101 end
102
103 % Set default value of options.Maxiter
104 if ~isfield(options, 'Maxiter')
105     options.Maxiter = max([200 N]);
106 elseif isempty(options.Maxiter)
107     options.Maxiter = max([200 N]);
108 end
109
110 % Set default value of options.Display
111 if ~isfield(options, 'Display')
112     options.Display = 0;
113 elseif isempty(options.Display)
114     options.Display = 0;
115 end
116
117 % Set default value of options.omega
118 if ~isfield(options, 'omega')
119     options.omega = [];
120 elseif isempty(options.omega)
```

```
121     options.omega = [];
122 end
123 
124 % Set default value of options.PlotFcns
125 if ~isfield(options, 'PlotFcns')
126     options.PlotFcns = 'off';
127 elseif isempty(options.PlotFcns)
128     options.PlotFcns = 'off';
129 end
130 
131 % Sort the augmented matrix
132 if options.Ifsort
133     [A, b, S] = smatrix(A, b);
134 else
135     S = 1 : N;
136 end
137 
138 % Iterate linear equations solution by input method
139 switch method
140     case 'Jacob'
141         [G, f] = JacobLE(A, b);
142     case 'G-S'
143         [G, f] = GSLE(A, b);
144     case 'SOR'
145         [G, f] = SORLE(A, b, options.omega);
146 end
147 
148 if ~strcmp(method, 'Adaptive')
149     Gvrho = max(abs(eig(G)));
150     if Gvrho >= 1
151         warning([method ''''s L' num2str(options.p) '-norm is larger' ...
152             ' than 1, follow soultions use the adaptive method'])
153         method = 'Adaptive';
154     end
155 end
156 
157 if strcmp(method, 'Adaptive')
158     [G, f, method] = AdaptiveLE(A, b, options.omega);
159     Gvrho = max(abs(eig(G)));
160 end
161 
162 iter = 0;
163 X0 = options.X0;
164 
165 while iter < options.Maxiter
166     iter = iter + 1;
167     X_new = G * X0 + f;
168     errorX = norm(X_new - X0);
```

```
169     X_iter(:, iter) = X_new;
170     errorFun(iter) = norm(A * X_new - b, options.p);
171     if errorFun(iter) <= options.TolFun || errorX <= options.TolX
172         break
173     end
174     X0 = X_new;
175 end
176 
177 if options.Display
178     fprintf(['Algorithm-' mfilename ' stop at the %d-th iteration by ' ...
179         method ' method.\n'], iter);
180 end
181 
182 Xstar(S) = X_new;
183 X_iter(S, :) = X_iter;
184 
185 if strcmp(options.PlotFcns, 'on')
186     plot(errorFun, 'LineWidth', 2)
187     title('Error of equations in each iteration')
188     xlabel('Iteration number')
189     ylabel('Error of equations')
190 end
191 
192 end
193 
194 %% Algorithm for replacing the main element of the coefficient matrix
195 function [A, b, Scol] = smatrix(A, b)
196 % replacing diagonal element of the coefficient matrix
197 %
198 % Input:
199 %   A - Coefficient matrix
200 %   b - Constant column vector
201 %
202 % Output:
203 %   AS - Coefficient matrix after sort
204 %   bS - Constant column vector after sort
205 %   Scol - Sort of column
206 
207 % Author  : ZH.Yuan
208 % Update  : 2021/10/19 (First Version: 2021/10/19)
209 % Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)
210 
211 N = size(A, 1);
212 Scol = 1 : N;
213 k = 1;
214 ks = 1;
215 while k <= N
216     Aiter = abs(A(k : N, k : N));
```

```
    Adet = Aiter;
    [~, Sind] = maxk(abs(Aiter(:)), ks);
    [Maxrow, Maxcol] = ind2sub([N + 1 - k, N + 1 - k], Sind(end));
    Adet(Maxrow, :) = [];
    Adet(:, Maxcol) = [];
    if det(Adet) == 0
        ks = ks + 1;
    else
        ks = 1;
        A([k, Maxrow + k - 1], :) = A([Maxrow + k - 1, k], :);
        A(:, [k, Maxcol + k - 1]) = A(:, [Maxcol + k - 1, k]);
        b([k, Maxrow + k - 1], :) = b([Maxrow + k - 1, k], :);
        Scol(:, [k, Maxcol + k - 1]) = Scol(:, [Maxcol + k - 1, k]);
        k = k + 1;
    end
end

end
```

3.3 模拟求解

通过生成系数矩阵 $\boldsymbol{A}$ 和真值 $\boldsymbol{x}_0$，计算得到向量 $\boldsymbol{b}$，利用 3.2 节中的“ileqs.m”对生成的增广矩阵 $(\boldsymbol{A}\ \boldsymbol{b})$ 输入求解，整个过程保存为“ileqsMultyDEMO.m”运行。

```
clc
clearvars
close all

N = 100;
Amethod = 3;
switch Amethod
    case 1
        A = N * eye(N) + rand(N);
    case 2
        A = 2 * eye(N) + rand(N);
        A = diag(diag(A, -1), -1) + diag(diag(A, 0), 0) + diag(diag(A, 1), 1);
    case 3
        A = N * eye(N) + rand(N);
        A = A' + A;
    case 4
        Z = orth(rand(N, N));
        A = Z' * diag(rand(N, 1)) * Z;
end

Xtrue = (1 : N)';
b = A * Xtrue;
options = struct('Display', 1, 'ifsort', 1, 'PlotFcns', 'on');
```

```
24
25 figure; hold on
26 [XstarJ, GvrhoJ, X_iterJ, errorallJ] = ileqs([A b], 'Jacob', options); % Jacob
27 [XstarG, GvrhoG, X_iterG, errorallG] = ileqs([A b], 'G-S', options); % G-S
28 [XstarS, GvrhoS, X_iterS, errorallS] = ileqs([A b], 'SOR', options); % SOR
29 [XstarA, GvrhoA, X_iterA, errorallA] = ileqs([A b], [], options); % Adaptive
30 legend({'Jacob', 'G-S', 'SOR', 'Adaptive'})
31 hold off
```

通过上述过程进行 200 次模拟实验，在每次过程中，随机生成系数矩阵 $\boldsymbol{A}$，固定真值 $\boldsymbol{x}_0$，计算得到向量 $\boldsymbol{b}$，利用 3.2 节中的 “ileqs.m” 对生成的增广矩阵 $(\boldsymbol{A}\ \boldsymbol{b})$ 输入求解，整个过程保存为 “ileqsDEMO.m” 运行。

```
1  clc
2  clearvars
3  close all
4
5  times = 200;
6  Nall = [10 50 100 500];
7
8  NJ = zeros(1, times);
9  NG = zeros(1, times);
10 NS = zeros(1, times);
11 NA = zeros(1, times);
12
13 NJall = cell(1, numel(Nall));
14 NGall = cell(1, numel(Nall));
15 NSall = cell(1, numel(Nall));
16 NAall = cell(1, numel(Nall));
17
18 figure(1)
19
20 for iN = 1 : numel(Nall)
21     for iter = 1 : times
22         N = Nall(iN);
23         Amethod = 4;
24
25         switch Amethod
26             case 1
27                 A = N * eye(N) + rand(N);
28             case 2
29                 A = 2 * eye(N) + rand(N);
30                 A = diag(diag(A, -1), -1) + diag(diag(A, 0), 0) + diag(diag(A, 1), 1);
31             case 3
32                 A = N * eye(N) + rand(N);
33                 A = A' + A;
34             case 4
35                 A = N * 5 * sqrt(N) * diag(rand(1, N)) + rand(N);
36                 A = A' + A;
```

```
            case 5
                M = diag(rand(N, 1));
                Z = orth(rand(N, N));
                A = Z' * M * Z;
        end

        Xtrue = (1 : N)';
        b = A * Xtrue;
        options.Display = 0;
        options.ifsort = 1;
        options.PlotFcns = 'off';

        % Jacob
        [XstarJ, GvrhoJ, X_iterJ, errorallJ] = ileqs([A b], 'Jacob', options);
        % G-S
        [XstarG, GvrhoG, X_iterG, errorallG] = ileqs([A b], 'G-S', options);
        % SOR
        [XstarS, GvrhoS, X_iterS, errorallS] = ileqs([A b], 'SOR', options);
        % Adaptive
        [XstarA, GvrhoA, X_iterA, errorallA] = ileqs([A b], [], options);

        NJ(iter) = numel(errorallJ);
        NG(iter) = numel(errorallG);
        NS(iter) = numel(errorallS);
        NA(iter) = numel(errorallA);

    end

    figure(1)
    subplot(ceil(numel(Nall) / 2), 2, iN)
    fig = boxplot([NJ', NG', NS', NA'], ...
        {'Jacob', 'G-S', 'SOR', 'Adaptive'}, 'Widths',0.3);
    set(fig, 'Linewidth', 2);
    subtitle(['Random Simulation' num2str(N) ' Times'])
    ylabel('Stop Number of Iteration')

    NJall{iN} = NJ;
    NGall{iN} = NG;
    NSall{iN} = NS;
    NAall{iN} = NA;

end

figure(1)
sgtitle('Boxplot of Numbers of Iterations')
```

第 4 章　实验结果

4.1　一个简单的例子

设有线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，其中：

$$\boldsymbol{A} = \begin{pmatrix} 10 & 1 & \cdots & 1 & 1 \\ 1 & 10 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 10 & 1 \\ 1 & 1 & \cdots & 1 & 10 \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} 19 \\ 19 \\ \vdots \\ 19 \\ 19 \end{pmatrix}$$

很容易解析得到 $\boldsymbol{x}^* = (1, 1, \cdots, 1, 1)$。若是用第三节的函数运行可以得到：

```
A = ones(10) + 9 * diag(ones(10, 1));
x0 = ones(10, 1);
b = A * x0;
[Xstar, Gvrho, X_iter, errorFun] = ileqs([A b])
```

结果如4.1所示

```
>> [Xstar, Gvrho, X_iter, errorFun] = ileqs([A b])

Xstar =

    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000


Gvrho =

    0.1837


X_iter =

    1.7856    0.9408    0.9770    0.9989    1.0006    1.0001    1.0000    1.0000
    1.6178    0.9944    0.9819    0.9976    1.0002    1.0001    1.0000    1.0000
    1.4658    1.0295    0.9885    0.9971    0.9999    1.0001    1.0000    1.0000
    1.3280    1.0493    0.9954    0.9974    0.9997    1.0000    1.0000    1.0000
    1.2032    1.0562    1.0015    0.9981    0.9996    1.0000    1.0000    1.0000
    1.0901    1.0526    1.0061    0.9992    0.9996    1.0000    1.0000    1.0000
    0.9877    1.0403    1.0086    1.0002    0.9997    0.9999    1.0000    1.0000
    0.8949    1.0211    1.0086    1.0010    0.9999    1.0000    1.0000    1.0000
    0.8108    0.9962    1.0059    1.0013    1.0001    1.0000    1.0000    1.0000
    0.7346    0.9671    1.0005    1.0009    1.0001    1.0000    1.0000    1.0000


errorFun =

   14.9583    1.3781    0.3449    0.0657    0.0090    0.0023    0.0004    0.0001
```

图 4.1: 小例运行结果图示

其中，Xstar 表示方程解的输出解；Gvrho 表示迭代系数矩阵 $\boldsymbol{G}$ 的谱半径大小；X_iter 表示每次迭代的迭代解，每一列为一次解；errorFun 表示线性方程组的残差向量的 p-范数值。将迭代过程中的迭代解与迭代误差表示在表 4.1 中如下：

表 4.1: 自适应方法的迭代过程的解

迭代次数	迭代解 $\boldsymbol{x}$										残差范数
1	1.7856	1.6178	1.4658	1.3280	1.2032	1.0901	0.9877	0.8949	0.8108	0.7346	14.9583
2	0.9408	0.9944	1.0295	1.0493	1.0562	1.0526	1.0403	1.0211	0.9962	0.9671	1.3781
3	0.9770	0.9819	0.9885	0.9954	1.0015	1.0061	1.0086	1.0086	1.0059	1.0005	0.3449
4	0.9989	0.9976	0.9971	0.9974	0.9981	0.9992	1.0002	1.0010	1.0013	1.0009	0.0657
5	1.0006	1.0002	0.9999	0.9997	0.9996	0.9996	0.9997	0.9999	1.0001	1.0001	0.0090
6	1.0001	1.0001	1.0001	1.0000	1.0000	1.0000	0.9999	1.0000	1.0000	1.0000	0.0023
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0004
8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0001

4.2 各方法收敛速度的可视化展示

函数 “ileqs.m” 的运行样例函数 “ileqsDEMO.m” 运行结果展示如4.2所示。图中命令行窗口展示了各方法的迭代终止次数，工作区中为运行后的各方法结果：其中，GvrhoJ、GvrhoG、GvrhoS 和 GvrhoA 分别表示各迭代算法的迭代矩阵的谱半径；XstarJ、XstarG、XstarS 和 XstarA 分别表示各迭代算法的迭代解；errorallJ、errorallG、errorallS 和 errorallA 分别表示各迭代算法每次的迭代误差；X_iterJ、X_iterG、X_iterS 和 X_iterA 分别表示各迭代算法的所有迭代步骤过程中的向量值构成的矩阵。

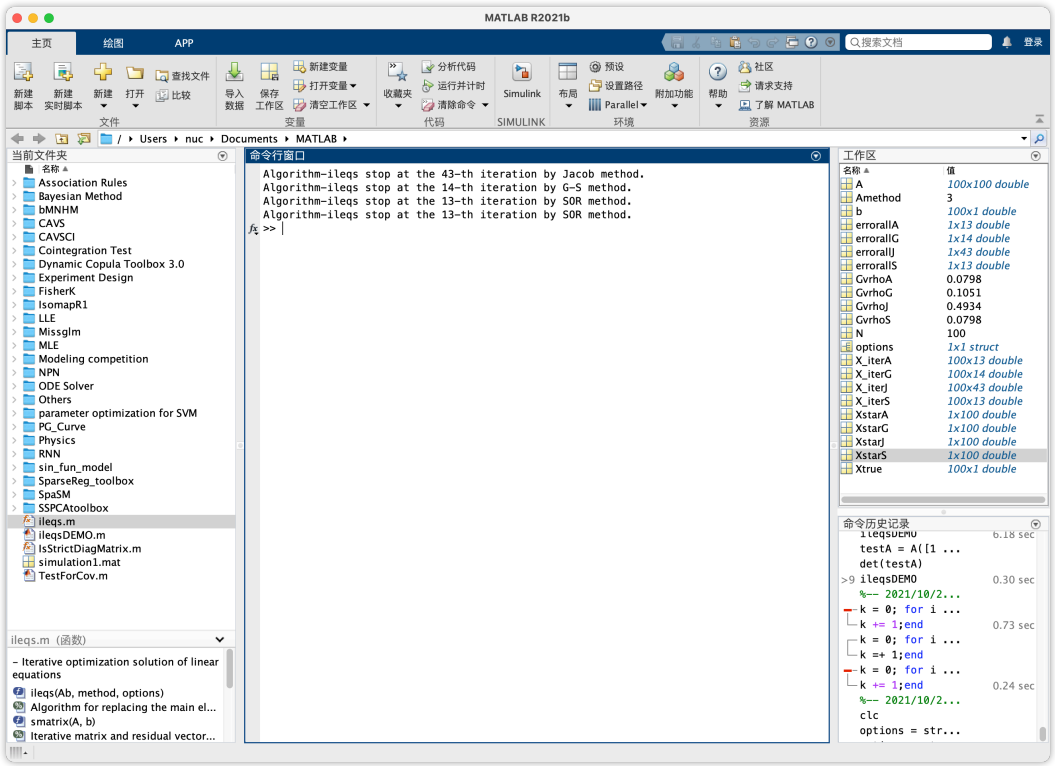

图 4.2: 运行结果图示

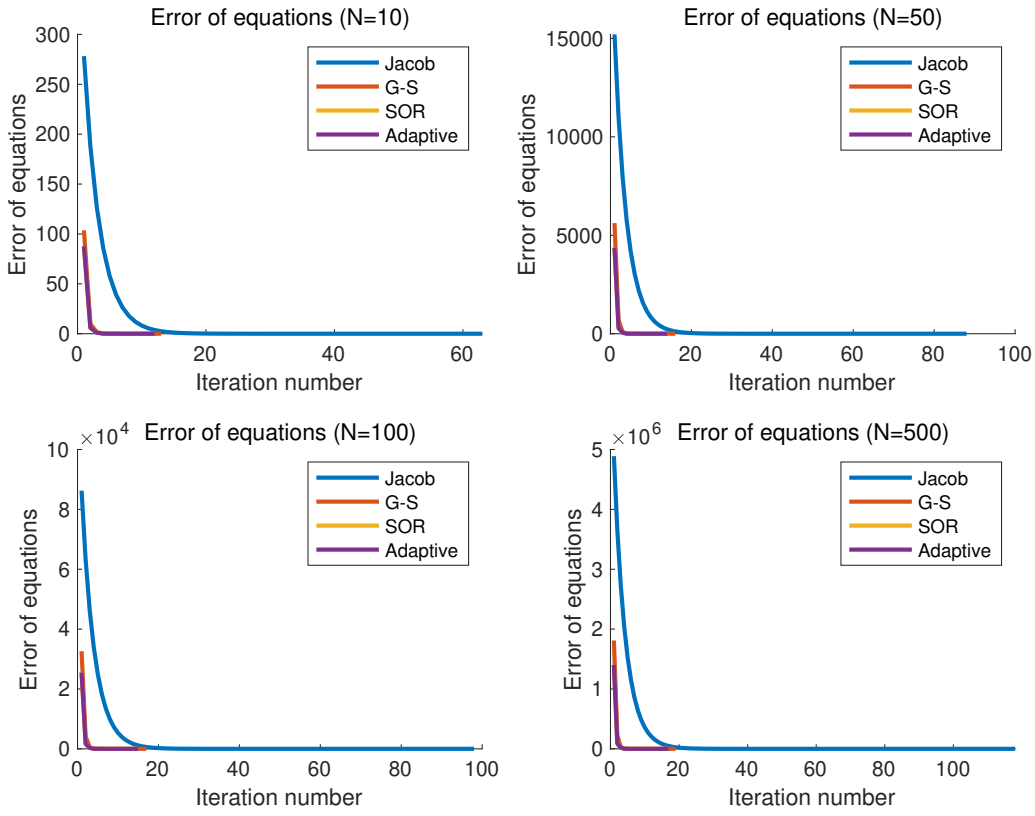

图 4.3: 收敛速度图示

在阶数 N 分别取 10, 50, 100, 500 时, 将 errorallJ、errorallG、errorallS 和 errorallA 分别对应的雅可比迭代法、高斯-赛德尔、SOR 和自适应选择的迭代算法每次的迭代误差随迭代次数的增加的趋势进行画图，所得结果如4.3所示。可以看出：在该次模拟中，无论阶数 N 取何值，雅可比迭代法的收敛速度都远小于高斯-赛德尔迭代法和逐次超松弛迭代法。最后各阶数下，各方法的误差都趋于 0，即各方法都收敛。

为了更避免模拟实验的随机性，本文又在阶数 N 分别取 10，50，100，500 时，对雅可比迭代法、高斯-赛德尔、SOR 和自适应选择的迭代算法进行了 200 次的模拟测试，测试结果取每个方法每次最终的迭代终止时的迭代次数。最终，将 200 次的迭代次数以箱线图4.4的形式进行展示。

当然,更多的参数可以进行设置与调整,详细的参数设置可以参照函数文件“ileqs.m”的 Input 注释进行阅读后调整，这里不再给出过多的说明。图 4.3 及图 4.4 的详细模拟过程见 “ileqsDEMO.m” 和 “ileqsMultyDEMO.m”。

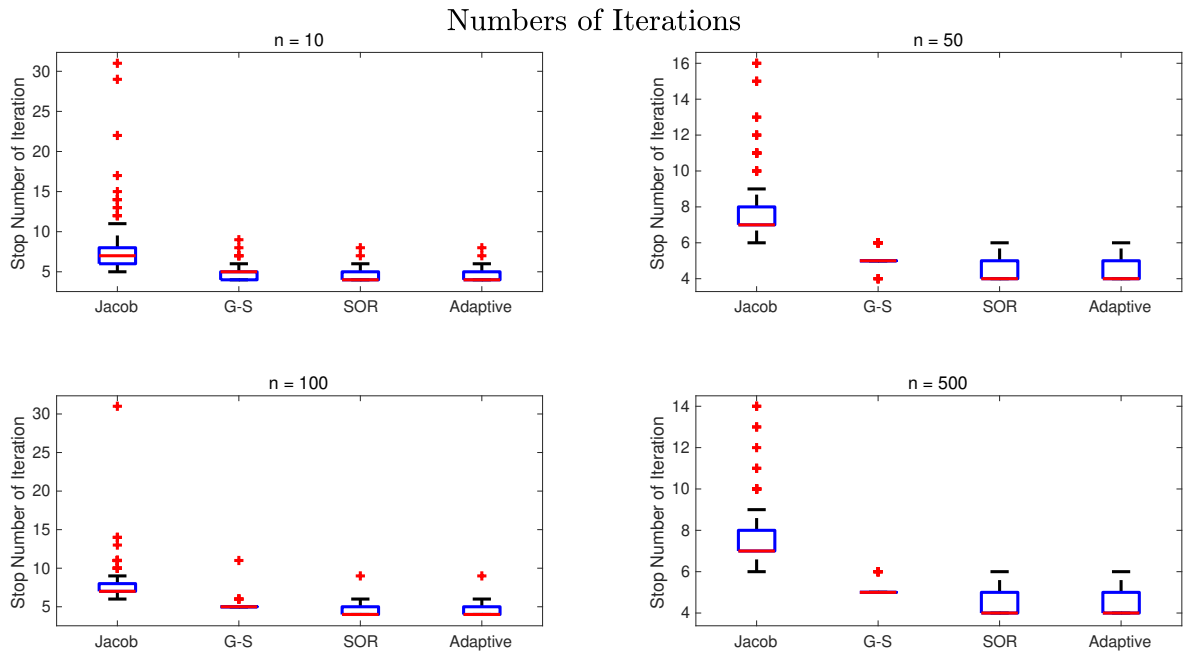

图 4.4: 200 次模拟实验的迭代次数的箱线图

4.3 总结展望

本文在 MATLAB_R2021b 环境下，编写了雅可比迭代法、高斯-赛德尔、SOR 和自适应选择的迭代算法的迭代矩阵的表达函数，利用迭代矩阵和表达式对线性方程组进行求解，已验证该迭代思想的正确性与稳定性。通过数值模拟分析，验证了其迭代收敛结果的正确性，并通过对误差的分析，确保了其收敛的稳定性。

本文还通过对矩阵的行列互换在确保矩阵非奇异的条件下，将主对角线元素更换为最大的 N 个数，这可以加速雅可比迭代法、高斯-赛德尔迭代法的收敛速度，在未来的研究中，可以对算法进行更深层地优化，以减少算法的计算时间。

致谢

感谢朱国甫老师对本实验报告的指导与鼓励！朱老师为我指点迷津，帮助我开拓研究思路，精心点拨、热忱鼓励。朱老师严谨细致、一丝不苟的作风，认真教学的态度，踏踏实实的精神让我深受感动。虽历时仅半年，却给我以终生受益无穷之道。对朱老师的感激之情是无法用言语表达的。