数值线性代数上机报告

周 旸

武汉大学数学与统计学院

日期：2025 年 5 月 29 日

摘　　要

本文聚焦于利用差分法求解两点边值问题，通过理论分析与上机实验，深入探讨不同参数及算法在求解该问题时的表现。证明了相关线性方程组系数矩阵的可逆性，对比了直接法、迭代法等多种算法，分析了计算速度、求解精度、迭代法收敛速度及模型误差等方面的特性。研究发现，不同算法在效率和精度上存在显著差异，模型误差对结果影响较大。此外，还讨论了两点边值问题的极限行为及更细致剖分的效果。文章最后总结了研究成果，并指出了有待解决的问题，为后续研究提供了方向。

关键词： 两点边值问题；差分法；线性方程组求解算法；数值线性代数

目录

1 引入

1.1 两点边值问题

考虑两点边值问题

$$\begin{cases} \varepsilon \dfrac{\mathrm{d}^2 y}{\mathrm{d}x^2} + \dfrac{\mathrm{d}y}{\mathrm{d}x} = f(x), \\ y(0) = 0,\ y(1) = 1. \end{cases} \tag{1}$$

其中 $\varepsilon > 0$。事实上，当 $f(x) \equiv a,\ 0 < a < 1$ 时，有解析解

$$y = \frac{1-a}{1-\mathrm{e}^{-1/\varepsilon}}(1-\mathrm{e}^{-x/\varepsilon}) + ax. \tag{2}$$

为了方便研究利用**差分法**求解这类问题的计算精度、效率和时间，本文只研究

$$\begin{cases} \varepsilon \dfrac{\mathrm{d}^2 y}{\mathrm{d}x^2} + \dfrac{\mathrm{d}y}{\mathrm{d}x} = a,\ 0 < a < 1, \\ y(0) = 0,\ y(1) = 1. \end{cases} \tag{3}$$

1.2 差分法

记问题 (3) 的解为 $y(x)$。把 $[0,1]$ 区间 n 等分，并令 $h = 1/n$，

$$x_i = ih,\ y_i = y_i, \quad i = 0, 1, \cdots, n-1, n,$$

利用 Taylor 公式，

$$\begin{aligned} y_{i+1} &= y_i + y'(x_i)h + \frac{y''(x_i)}{2}h^2 + \frac{y'''(x_i)}{6}h^3 + O(h^4), \\ y_{i-1} &= y_i - y'(x_i)h + \frac{y''(x_i)}{2}h^2 - \frac{y'''(x_i)}{6}h^3 + O(h^4). \end{aligned} \tag{4}$$

于是有

$$y''(x_i) = \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + O(h^2) \tag{5}$$

和

$$y'(x_i) = \frac{y_{i+1} - y_i}{h} + O(h). \tag{6}$$

当 h 充分小，即剖分充分细时，忽略 $O(h)$ 项，得到差分方程

$$\varepsilon \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + \frac{y_{i+1} - y_{i-1}}{h} = a, \tag{7}$$

或

$$(\varepsilon + h)y_{i+1} - 2\varepsilon y_i + (\varepsilon - h)y_{i-1} = ah^2. \tag{8}$$

写成线性方程组，得到

$$\begin{bmatrix} -2\varepsilon-h & \varepsilon+h & & & \\ \varepsilon & -2\varepsilon-h & \varepsilon+h & & \\ & \varepsilon & -2\varepsilon-h & \ddots & \\ & & \ddots & \ddots & \varepsilon+h \\ & & & \varepsilon & -2\varepsilon-h \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} ah^2 \\ ah^2 \\ \vdots \\ ah^2 \\ ah^2-\varepsilon-h \end{bmatrix}. \tag{9}$$

这个线性方程组有唯一解的充分必要条件是系数矩阵可逆。

若能证明线性方程组 (9) 的系数矩阵是**可逆矩阵**，那么两点边值问题 (3) 的求解就转化成线性方程组 (9) 的求解。

1.3 系数矩阵的奇异性分析

这一小节通过引入弱严格对角占优和不可约矩阵，证明线性方程组 (9) 的系数矩阵是**可逆矩阵**。

定义 1.1. （对角占优矩阵）设 $A=[a_{ij}]\in\mathbb{R}^{n\times n}$，

$$|a_{kk}|\geq\sum_{\substack{j=1\\ j\neq k}}^{n}|a_{kj}|,\quad k=1,2,\cdots,n, \tag{10}$$

且 (10) 中至少对一个 i 有严格不等号成立，则称 A 为**弱严格对角占优**的。

定义 1.2. （不可约矩阵）设 A 为 $n\geq 2$ 阶矩阵，$\mathcal{W}=\{1,2,\cdots,n\}$。若存在 $\mathcal{W}$ 两个非空的子集 $\mathcal{S}$ 和 $\mathcal{T}$ 满足

$$\mathcal{S}\cup\mathcal{T}=\mathcal{W},\quad \mathcal{S}\cap\mathcal{T}=\varnothing,$$

使得

$$a_{ij}=0,\quad i\in\mathcal{S},\quad j\in\mathcal{T},$$

则称 A 为**可约**的。若一个矩阵矩阵不可约且弱严格对角占优，则称该矩阵是**不可约对角占优**的。

定理 1.1. *线性方程组 (9) 的系数矩阵是不可约对角占优的。*

证明. 矩阵弱对角占优显然。要证明矩阵是不可约的，将矩阵看作有向图 G 的邻接矩阵。由矩阵的三对角线上的元素均不为 0，得 G 的顶点两两连通，即 G 是**强连通图**。于是，不存在将 G 上顶点的二划分，使得划分的两顶点集间不连通，即：矩阵是不可约的。 □

定理 1.2. *不可约对角占优矩阵是非奇异的。[1]*

由定理1.1和定理1.2，线性方程组 (9) 的系数矩阵是**可逆**的。

2 线性方程组求解算法的选择

2.1 直接法

常见求解线性方程组的直接法有 Gauss 消去法（包括不选主元的 Gauss 消去法、全主元 Gauss 消去法和列主元 Gauss 消去法）、Cholesky 分解（包括平方根法和改进平方根法）和追赶法。

对于 **Gauss 消去法**，若不换主元，则当且仅当**主元均不为** 0 时算法才能进行到底。由 (9)，线性方程组的系数矩阵的对角元均为 $-2\varepsilon - h \neq 0$，所以线性方程组可利用不换主元的 Gauss 消去法求解。

事实上，对于线性方程组 (9)，列主元 Gauss 消去法就是不选主元的 Gauss 消去法。

定理 2.1. *设 A 是对角占优矩阵（不一定严格），又设经过一步 Gauss 消去后，A 具有如下形状*

$$\begin{bmatrix} a_1 1 & a_1^\top \\ 0 & A_2 \end{bmatrix},$$

则矩阵 A_2 仍是对角占优矩阵 [1]。从而对于 A，利用不选主元的 Gauss 消去法和列主元 Gauss 消去法得到的结果是一致的。

因为系数矩阵是弱严格对角占优的，所以利用不选主元的 Gauss 消去法和列主元 Gauss 消去法求解线性方程组 (9) 的结果是一致的。

利用全主元 Gauss 消元法求解显然也是可行的，但计算成本较高。

对于 **Cholesky 分解**，无论是平方根法还是改进平方根法，均要求系数矩阵**对称**，而 (9) 中的系数矩阵不是对称的，因此无法利用 Cholesky 分解直接求解 (9)。虽然可以通过求解线性方程组 $A^\top Ax = A^\top b$ 的方式求解，但 $A^\top A$ 的条件数比 A 大，导致收敛性较差。

由于系数矩阵是可逆的三**对角矩阵**，所以可以利用**追赶法**求解线性方程组 (9)。

2.2 单步线性定常迭代方法

常见的单步线性定常迭代方法有 Jacobi 迭代法、Gauss-Seidel 迭代法和 SOR 迭代法。

定理 2.2. *对于线性方程组 $Ax = b$ 和不可约对角占优的系数矩阵 A，Jacobi 迭代和 Gauss-Seidel 迭代收敛。[1]*

结合定理1.1，线性方程组 (9) 利用 Jacobi 迭代和 Gauss-Seidal 迭代收敛。

定理 2.3. *对于线性方程组 $Ax = b$ 和不可约对角占优的系数矩阵 A，当 $0 < \omega < 1$ 时，SOR 迭代收敛。[1]*

由此可知，当 $0 < \omega < 1$ 时，线性方程组 (9) 对 SOR 迭代收敛。$1 \leq \omega < 2$ 时是否收敛?

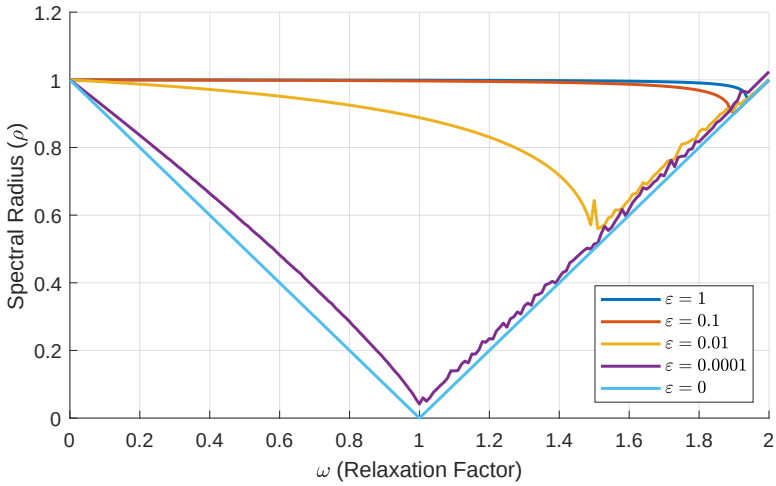


图 1: SOR 迭代矩阵的谱半径与松弛因子的关系

取 $a = 0.5$, $n = 100$ 生成的系数矩阵，得到 SOR 迭代矩阵的谱半径关于松弛因子的曲线如图1所示。可以看出，其实系数矩阵对 $0 < \omega < 2$ 的 SOR 迭代均收敛。且当 $0 < \omega < 1$ 时，SOR 迭代矩阵的谱半径均大于 Gauss-Seidel ($\omega = 1$) 的，起不到加速效果。

2.3 Krylov 方法

常见的 Krylov 方法有最速下降法和共轭梯度法。

这两种方法均要求矩阵**对称正定**。因此只能通过 $A^\top Ax = A^\top b$ 来间接求解线性方程组 (9)。但是 $A^\top A$ 性质更差，可能对算法的收敛效果产生比较显著的影响。

然而，由于共轭梯度法有非常好的收敛性（忽略舍入误差，理论上 n 次迭代必收敛），也不失为一种可行的算法。下面的实验中可以看到，事实上共轭梯度法表现良好。

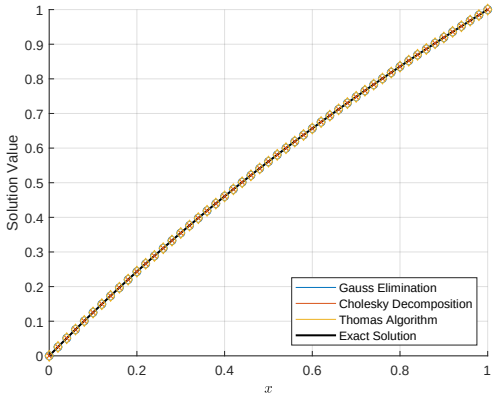
3 上机实现

3.1 直接法

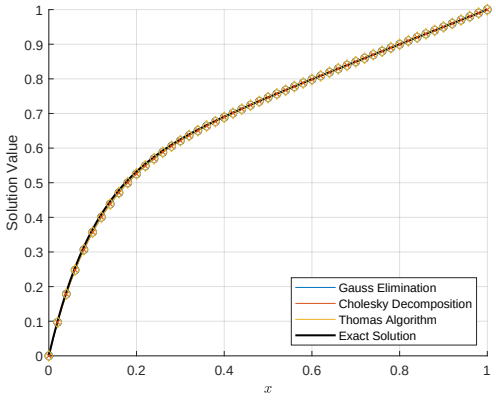
对线性方程组 (9)，取 $a = 0.5$, $\varepsilon = 1, 0.1, 0.01, 0.0001$, $n = 100$，分别利用不选主元的 Gauss 消元法、平方根法（求解 $A^\top Ax = A^\top b$）和追赶法，在相同的硬件条件下，得到的结果如表1所示。其中，MSE、Max Error 分别表示数值解与解析解之间的平均平方误差和最大误差。

表 1: 直接法结果

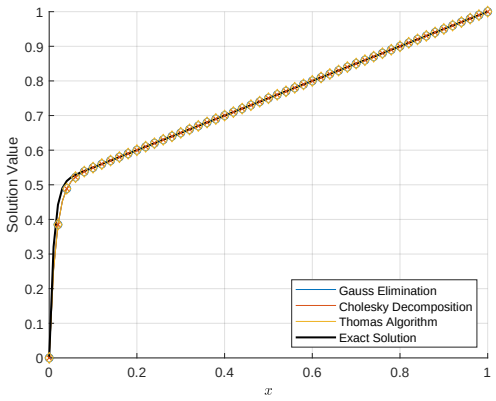
算法		**Gauss 消元法**	**平方根法**	**追赶法**
$\varepsilon = 1$	$\|b - Ax\|_\infty$	2.918×10^{-16}	1.189×10^{-15}	2.220×10^{-16}
	MSE	4.654×10^{-8}	4.654×10^{-8}	4.654×10^{-8}
	Max Error	3.001×10^{-4}	3.001×10^{-4}	3.001×10^{-4}
	时间	0.016 s	0.016 s	0.004 s
$\varepsilon = 0.1$	$\|b - Ax\|_\infty$	2.773×10^{-17}	1.119×10^{-17}	2.373×10^{-17}
	MSE	1.458×10^{-5}	1.458×10^{-5}	1.458×10^{-5}
	Max Error	8.824×10^{-3}	8.824×10^{-3}	8.824×10^{-3}
	时间	0.016 s	0.013 s	< 0.001 s
$\varepsilon = 0.01$	$\|b - Ax\|_\infty$	3.469×10^{-18}	2.334×10^{-19}	4.73×10^{-18}
	MSE	9.666×10^{-5}	9.666×10^{-5}	9.666×10^{-5}
	Max Error	6.606×10^{-2}	6.606×10^{-2}	6.606×10^{-2}
	时间	0.019 s	0.015 s	< 0.001 s
$\varepsilon = 0.0001$	$\|b - Ax\|_\infty$	2.229×10^{-18}	4.215×10^{-20}	2.202×10^{-18}
	MSE	2.427×10^{-7}	2.427×10^{-7}	2.427×10^{-7}
	Max Error	4.950×10^{-3}	4.950×10^{-3}	4.950×10^{-3}
	时间	0.019 s	0.018 s	< 0.001 s



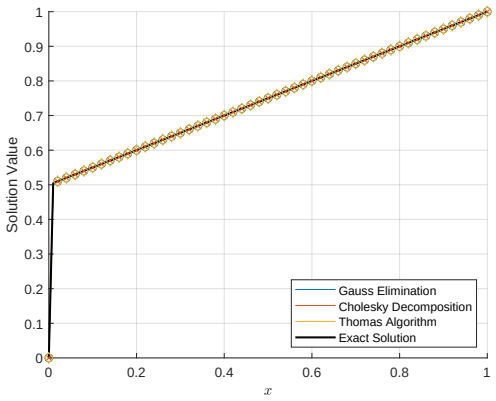
(a) $\varepsilon = 1$



(b) $\varepsilon = 0.1$



(c) $\varepsilon = 0.01$



(d) $\varepsilon = 0.0001$

图 2: 直接法结果图

从表1中可以看出：

- **求解线性方程组的数值精度**：三种直接法的精度都比较良好，三种算法差距不大；且 ε 越小，$\|b-Ax\|_\infty$ 越小，原因与系数矩阵的性质有关。
- **对比解析解的误差**：求解两点边值问题 (3) 误差的主要来源是**模型误差**；ε 与模型误差可能没有**线性关系**，数值解与解析解的误差随 ε 的减小先增后减。
- **计算时间**：三种算法在计算时间上有显著差异，Gauss 消去法最慢；平方根法次之，与 Gauss 消去法用时同量级；追赶法极快，对于 100 阶三对角矩阵用时几乎可以忽略。

3.2 迭代法

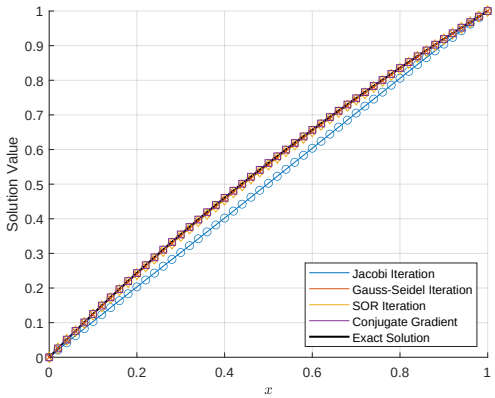
对线性方程组 (9)，取 $a=0.5$，$\varepsilon=1,0.1,0.01,0.0001$，$n=100$，同时取初值 $x=0$，**停机条件** $\|b-Ax\|_\infty<10^{-13}$ 和迭代次数 $k\le 5000$，分别利用 Jacobi 迭代、Gauss-Seidel 迭代、SOR 迭代（松弛因子的选取见表2注）和共轭梯度法（求解 $A^\top Ax=A^\top b$），在相同的硬件条件下，得到的结果如表2所示。

表 2: 迭代法结果

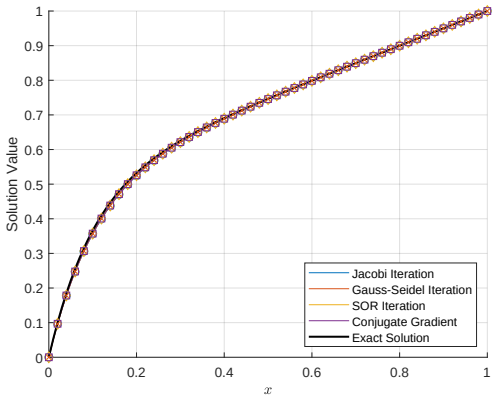
算法		**Jacobi**	**G-S**	**SOR ****	**CG**
$\varepsilon=1$	$\|b-Ax\|_\infty$	6.240×10^{-5}	4.917×10^{-6}	9.987×10^{-14}	5.003×10^{-14}
	MSE	1.736×10^{-3}	1.307×10^{-5}	4.654×10^{-8}	4.654×10^{-8}
	Max Error	5.948×10^{-2}	5.156×10^{-3}	3.001×10^{-4}	3.001×10^{-4}
	迭代次数	5000 *	5000 *	1243	442
	时间	0.017 s	1.648 s	0.421 s	0.005 s
$\varepsilon=0.1$	$\|b-Ax\|_\infty$	4.505×10^{-6}	1.392×10^{-9}	9.082×10^{-14}	2.170×10^{-14}
	MSE	1.992×10^{-5}	1.458×10^{-5}	1.458×10^{-5}	1.458×10^{-5}
	Max Error	9.927×10^{-3}	8.824×10^{-3}	8.824×10^{-3}	8.824×10^{-3}
	迭代次数	5000 *	5000 *	329	374
	时间	0.010 s	1.755 s	0.119 s	0.005 s
$\varepsilon=0.01$	$\|b-Ax\|_\infty$	9.409×10^{-14}	9.315×10^{-14}	2.220×10^{-15}	9.232×10^{-14}
	MSE	9.666×10^{-5}	9.666×10^{-5}	9.666×10^{-5}	9.666×10^{-5}
	Max Error	6.606×10^{-2}	6.606×10^{-2}	6.606×10^{-2}	6.606×10^{-2}
	迭代次数	876	487	101	186
	时间	0.002 s	0.169 s	0.037 s	0.005 s
$\varepsilon=0.0001$	$\|b-Ax\|_\infty$	1.776×10^{-14}	1.809×10^{-14}	1.809×10^{-14}	8.236×10^{-17}
	MSE	2.427×10^{-7}	2.427×10^{-7}	2.427×10^{-7}	2.427×10^{-7}
	Max Error	4.951×10^{-3}	4.951×10^{-3}	4.951×10^{-3}	4.951×10^{-3}
	迭代次数	132	116	116	99
	时间	0.001 s	0.042 s	0.034 s	0.003 s

* 达到指定的迭代次数上限仍未达到精度要求；

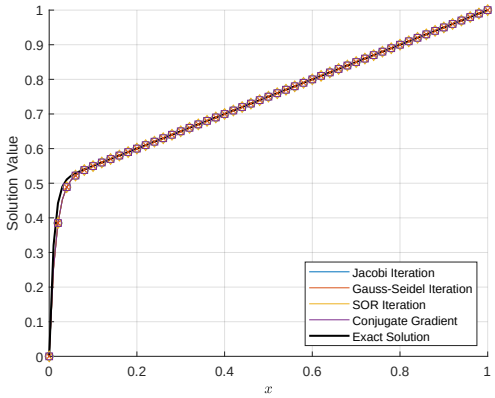
** SOR 迭代松弛因子的选取：$\varepsilon=1$，0.1 时，$\omega=1.9$；$\varepsilon=0.01$ 时，$\omega=1.5$；$\varepsilon=0.0001$ 时，$\omega=1$。



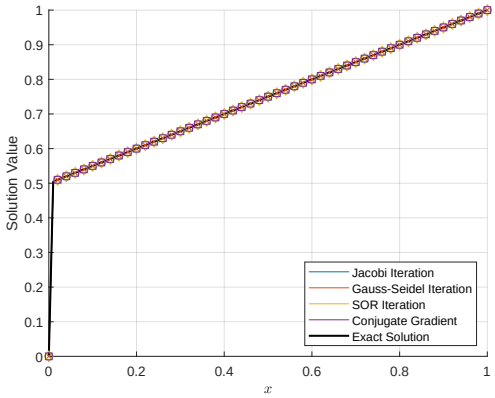
(a) $\varepsilon = 1$



(b) $\varepsilon = 0.1$



(c) $\varepsilon = 0.01$



(d) $\varepsilon = 0.0001$

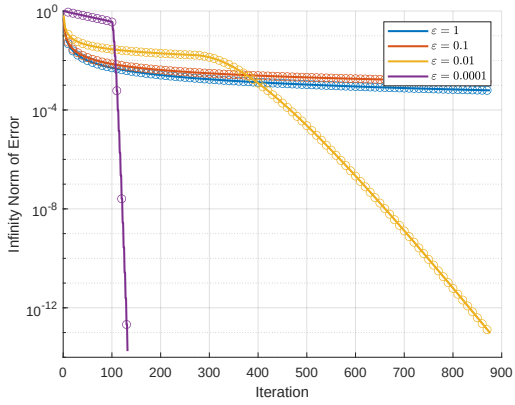
图 3: 迭代法结果图

从表2中可以看出：

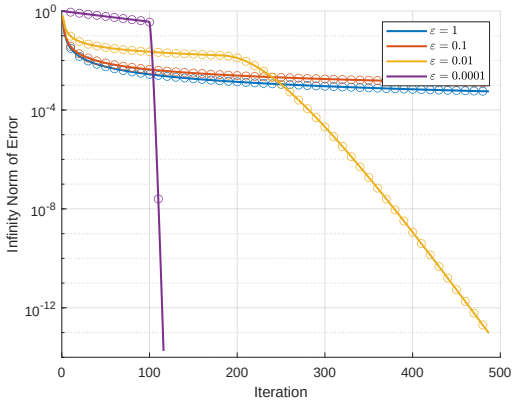
- **求解线性方程组的数值精度**：ε 越小，达到指定精度所需的迭代次数越大；SOR 迭代（松弛因子取得足够好）和共轭梯度法表现更好，尤其是共轭梯度法。
- **对比解析解的误差**：求解两点边值问题 (3) 误差的主要来源是**模型误差**；ε 与模型误差可能没有**线性关系**，数值解与解析解的误差随 ε 的减小先增后减。
- **计算时间**：四种算法在计算时间上有显著差异，Jacobi 迭代和共轭梯度法的速度较快，SOR 迭代次之，Gauss-Seidel 迭代明显更慢。

下面将对这些现象的原因做出解释。

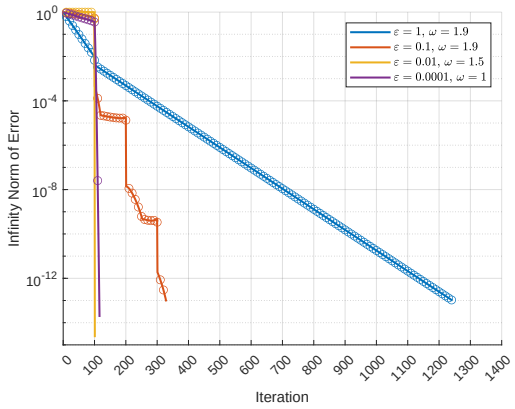
各个算法的迭代曲线如图4所示，其中两个圆圈之间差 10 次迭代。



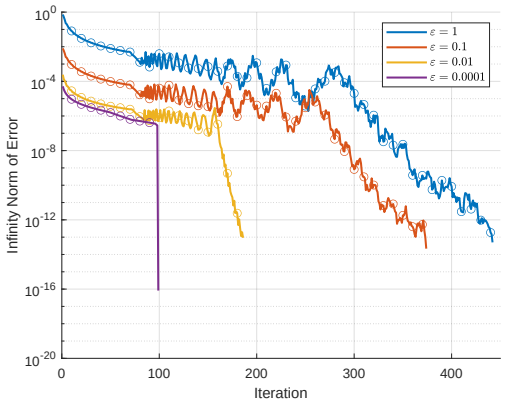
(a) Jacobi



(b) Gauss-Seidel



(c) SOR



(d) CG

图 4: 迭代法的迭代曲线

3.3 最速下降法的实现

利用最速下降法求解线性方程组 (9) 不是好选择。这里使用

$$\begin{bmatrix} -2\varepsilon-1 & \varepsilon & & & \\ \varepsilon & -2\varepsilon-1 & \varepsilon & & \\ & \varepsilon & -2\varepsilon-1 & \ddots & \\ & & \ddots & \ddots & \varepsilon \\ & & & \varepsilon & -2\varepsilon-1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} \tag{11}$$

来测试最速下降法。得到的结果如表3所示，收敛曲线如图5。

表 3: 最速下降法结果

ε	迭代次数	$\|b-Ax\|_\infty$
0.1	23	1.443×10^{-14}
0.01	9	6.661×10^{-14}
0.0001	5	1.110×10^{-16}

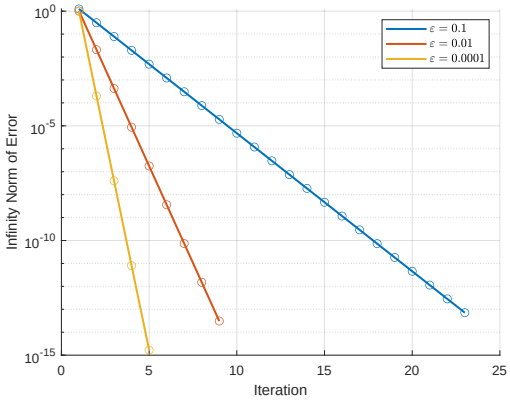


图 5: 最速下降法迭代曲线

4 对实验结果的分析

4.1 一些计算

在分析具体的数值现象之前，由系数矩阵的三对角特殊结构，系数矩阵的特征值、Jacobi 迭代矩阵的谱半径可以解析地计算出，便于后续的分析。

定理 4.1. （特殊三对角矩阵的特征值）矩阵

$$\begin{bmatrix} a & b & & & \\ c & a & b & & \\ & c & a & b & \\ & & \ddots & \ddots & b \\ & & & c & a \end{bmatrix} \tag{12}$$

的特征值为

$$\lambda_k = a + 2\sqrt{bc}\cos\frac{k\pi}{n+1},\ k = 1, 2, \cdots, n. \tag{13}$$

由定理4.1，系数矩阵的特征值均为负实数，最大特征值为

$$\lambda_{\max} = -2\varepsilon - h + 2\sqrt{(\varepsilon+h)\varepsilon}. \tag{14}$$

于是，Jacobi 迭代矩阵的谱半径可以直接给出解析表达式。由

$$M_{\text{Jacobi}} = D^{-1}(D-A) = I - D^{-1}A = I + \frac{1}{2\varepsilon+h}A, \tag{15}$$

其中 A 是系数矩阵，于是

$$\rho(M_{\text{Jacobi}}) = 1 + \frac{1}{2\varepsilon+h}\lambda_{\max} = 1 - \frac{2\varepsilon+h-2\sqrt{(\varepsilon+h)\varepsilon}}{2\varepsilon+h} = \frac{2\sqrt{(\varepsilon+h)\varepsilon}}{2\varepsilon+h}. \tag{16}$$

4.2 计算速度（时间）的分析

设系数矩阵的阶为 n，则直接法和迭代法的计算次数（记加、减、乘、除和开方的次数）如表4所示。对于迭代法，m 表示迭代次数。

表 4: 不同算法的计算次数和计算时间

算法		计算次数	迭代次数 *	时间 *
直接法	Gauss 消去法	$O(n^3)$	/	0.019 s
	平方根法	$O(n^3)$	/	0.015 s
	追赶法	$O(n)$	/	$<$ 0.001 s
迭代法	Jacobi 迭代	$O(mn^2)$	876	0.002 s
	G-S 迭代	$O(mn^2)$	487	0.169 s
	SOR 迭代	$O(mn^2)$	101	0.037 s
	共轭梯度法	$O(mn^2)$	186	0.005 s

* 取 $a = 0.5$, $\varepsilon = 0.01$, $n = 100$ 的实验结果。

对于直接法，从计算次数上可以理解追赶法的速度远快于 Gauss 消去法和平方根法；但对于 Gauss 消去法和平方根法的比较，事实上 Gauss 消去法的计算次数为 $2/3n^3 + O(n^2)$，平方根法为 $1/3n^3 + O(n^2)$，但实验结果来看二者时间差距不大，可能的原因是矩阵的规模太小，实际计算次数受尾项 $O(n^2)$ 影响较大（如前代法和回代法的过程是相同的）。

而对于迭代法，虽然计算次数均为 $O(mn^2)$，但实际单次迭代的时间差别很大：Jacobi 迭代和共轭梯度法的单次迭代速度要快得多。一个重要的原因是，Jacobi 迭代和共轭梯度法更依赖向量点积，而 Gauss-Seidal 迭代和 SOR 迭代需要逐个更新近似解的分量，对于现代计算机而言，向量点积的效率要快得多。

4.3 不同 ε 线性方程组求解的敏度分析

由表1和表2，无论是直接法还是迭代法，ε 越小，线性方程组求解的精度越高（对迭代法而言，迭代次数更少）。这说明了对于不同的 ε，系数矩阵的性质是有区别的。

定理 4.2. （线性方程组的敏度分析）设 $\|\cdot\|$ 是 $\mathbb{R}^{n\times n}$ 上的一个满足条件 $\|I\| = 1$ 的矩阵范数，并假定 $A \in \mathbb{R}^{n\times n}$ 非奇异，$b \in \mathbb{R}^n$ 非零；再假定 $\delta A \in \mathbb{R}^{n\times n}$ 满足 $\|A^{-1}\|\|\delta A\| < 1$。若 x 和 $x + \delta x$ 分别是线性方程组

$$Ax = b \quad \text{和} \quad (A + \delta A)(x + \delta x) = b + \delta b$$

的解，则

$$\frac{\|\delta x\|}{\|x\|} \leqslant \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right),$$

其中 $\kappa(A) = \|A^{-1}\|\|A\|$，称为矩阵 A 的**条件数**。

当 $\frac{\|\delta A\|}{\|A\|}$ 较小时，有

$$\frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \approx \kappa(A),$$

从而有

$$\frac{\|\delta x\|}{\|x\|} \lesssim \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right). \tag{17}$$

于是，可以利用矩阵范数作为条件数来评价矩阵良态与否。这里矩阵范数选取 2-范数和 ∞-范数，在 $a = 0.5$, $n = 100$ 下计算 $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$ 和 $\kappa_\infty(A) = \|A\|_\infty\|A^{-1}\|_\infty$ 的结果如表5。

表 5: 系数矩阵的条件数

ε	$\kappa_2 = \|A\|_2\|A^{-1}\|_2$	$\kappa_\infty = \|A\|_\infty\|A^{-1}\|_\infty$
1	3991	4933
0.1	2087	2766
0.01	371	555
0.0001	129	200

与表1和表2观察到的数值精度相符。

4.4 迭代法收敛速度的分析

定理 4.3. *设单步线性定常迭代矩阵为 M，则有*

$$R_\infty(M) = -\ln \rho(M). \tag{18}$$

*其中 $R_\infty(M)$ 是迭代法的***渐进收敛速度***。[1]*

定理 4.4. *用共轭梯度法求得的 x_k 有如下的误差估计：*

$$\|x_k - x_*\|_A \le 2\left(\frac{\sqrt{\kappa_2}-1}{\sqrt{\kappa_2}+1}\right)^k \|x_0 - x_*\|_A, \tag{19}$$

其中 $\kappa_2 = \|A\|_2\|A^{-1}\|_2$ 是矩阵 A 的条件数。[1]

由上述两个定理，可以通过单步线性定常迭代矩阵的谱半径和系数矩阵 2-范数下的条件数来估计迭代法的收敛速度。迭代矩阵的谱半径越小，单步线性定常迭代收敛越快；系数矩阵的条件数越小，共轭梯度法收敛越快。

表 6: 与迭代法收敛速度相关的数

ε	$\rho(M_{\text{Jacobi}})$	$\rho(M_{\text{G-S}})$	$\rho(M_{\text{SOR}})$	$\kappa_2(A)$
1	0.999	0.999	0.979	3991
0.1	0.998	0.997	0.900	2087
0.01	0.942	0.888	0.643	371
0.0001	0.195	0.041	0.041	129

与表1和表2观察到的迭代次数相符。

对于 Jacobi 迭代，可以定量地计算其迭代矩阵的谱半径并考虑迭代法收敛速度的极限行为。

$$\rho(M_{\text{Jacobi}}) = 1 + \frac{1}{2\varepsilon + h}\lambda_{\max} = 1 - \frac{2\varepsilon + h - 2\sqrt{(\varepsilon+h)\varepsilon}}{2\varepsilon + h} = \frac{2\sqrt{(\varepsilon+h)\varepsilon}}{2\varepsilon + h}. \tag{20}$$

可见，当 $\varepsilon \gg h$ 时，$\rho(M_{\text{Jacobi}}) \approx 1$；当 $\varepsilon \ll h$ 时，$\rho(M_{\text{Jacobi}}) \approx 2\sqrt{\varepsilon/h} \approx 0$。当 $\varepsilon = Ch$，C 为一正常数，则 $\rho(M_{\text{Jacobi}}) = 2\sqrt{C(C+1)}/(2C+1) \in (0,1)$。这从另一个角度证明了 Jacobi 迭代的收敛性，同时定量地解释了当 ε 取值较大时收敛性差的原因。

4.5 模型误差的分析

4.5.1 格点上的误差

从表中数据可以看出，解线性方程组的数值误差很小，但与解析解相比误差大了几个量级，说明在选取剖分数为 $n = 100$ 时，利用差分法求解两点边值问题的**误差主要来自模型误差**。因此，估计模型误差非常重要。

从1.2的分析可以看出，模型误差来自差分法对导数的近似。实际上，由

$$y''(x_i) = \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + O(h^2),$$
$$y'(x_i) = \frac{y_{i+1} - y_i}{h} - \frac{y''(x_i)}{2}h + O(h^2),$$

代入问题 (3)，得

$$\varepsilon\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + \frac{y_{i+1} - y_{i-1}}{h}\boxed{-\frac{y''(x_i)}{2}h + O(h^2)} = a, \tag{21}$$

$$(\varepsilon + h)y_{i+1} - 2\varepsilon y_i + (\varepsilon - h)y_{i-1} = ah^2 + \frac{y''(x_i)}{12}h^3 + O(h^4). \tag{22}$$

也即实际上问题等价的线性方程组是

$$\begin{bmatrix} -2\varepsilon - h & \varepsilon + h & & & \\ \varepsilon & -2\varepsilon - h & \varepsilon + h & & \\ & \varepsilon & -2\varepsilon - h & \ddots & \\ & & \ddots & \ddots & \varepsilon + h \\ & & & \varepsilon & -2\varepsilon - h \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} ah^2 + \frac{f''(x_1)}{12}h^3 + O(h^4) \\ ah^2 + \frac{f''(x_2)}{12}h^3 + O(h^4) \\ \vdots \\ ah^2 + \frac{f''(x_{n-2})}{12}h^3 + O(h^4) \\ ah^2 + \frac{f''(x_{n-1})}{12}h^3 + O(h^4) - \varepsilon \end{bmatrix}. \tag{23}$$

考虑 ∞-范数意义下的 $\|\delta b\|_\infty$。忽略 $O(h^4)$ 项，则 $\|\delta b\|_\infty \approx h^3 \cdot \max_{x\in\{x_1,\cdots,x_{n-1}\}} |f''(x)|$。又由

$$y''(x) = \frac{1-a}{1-\mathrm{e}^{-1/\varepsilon}}\left(1 - \frac{1}{\varepsilon^2}\mathrm{e}^{-x/\varepsilon}\right), \tag{24}$$

知当 ε 充分小时，

$$\|\delta b\|_\infty \approx h^3 \cdot \max_{x\in\{x_1,\cdots,x_{n-1}\}} |f''(x)| = h^3|f''(x_1)| \approx h^3\frac{1-a}{\varepsilon^2}\mathrm{e}^{-h/\varepsilon}.$$

再利用 (17)，当 $\|\delta A\|_\infty \ll \|A\|_\infty$ 时，估计解的误差为

$$\|\delta x\|_\infty \lesssim \kappa_\infty(A)\frac{(1-a)h}{a\varepsilon^2}\mathrm{e}^{-h/\varepsilon}. \tag{25}$$

这里利用了 $\|x\|_\infty \approx 1$ 和 $\|b\|_\infty \approx ah^2$。

考虑极限情形的模型误差。当 $\varepsilon \to \infty$ 时，A 收敛到矩阵 A_*，其逆矩阵为 A_*^{-1}，

$$A_* = \begin{bmatrix} -h & h & & & \\ & -h & h & & \\ & & -h & \ddots & \\ & & & \ddots & h \\ & & & & -h \end{bmatrix}, \quad A_*^{-1} = -\frac{1}{h} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ & 1 & 1 & \cdots & 1 \\ & & 1 & \cdots & 1 \\ & & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}, \tag{26}$$

所以 $\kappa_\infty(A) \to \kappa_\infty(A_*) = 2n$, $\varepsilon \to 0$。又由 $\varepsilon^{-2}\mathrm{e}^{-h/\varepsilon} \to 0$, $\varepsilon \to 0$，所以，当 $\varepsilon \to 0$ 时，模型误差趋于 0。

4.5.2 曲线拟合的误差

由于差分法计算的实际上是离散点上的近似解，描点绘制出来的曲线是 **Euler 折线**。

从图6和图3可以看出，随着 ε 的减小，

$$y'(x) = \frac{1-a}{1-\mathrm{e}^{-1/\varepsilon}}(1 + \frac{1}{\varepsilon}\mathrm{e}^{-x/\varepsilon}) + a, \tag{27}$$

解在 0 附近的斜率很大，且不是近似线性的；但格点在 $[0,1]$ 区间上是均匀的，数值解在 0 附近变化的刻画不够精细，导致整体误差更大。而在 1 附近，解近似线性，其实不需要过多的点去刻画。

于是，根据解曲线的行为，可以采取不等距剖分的方式，可以一定程度上减小曲线拟合的误差。

5 其他问题的分析

5.1 两点边值问题的极限行为

当 $\varepsilon \to 0$ 时，两点边值问题 (3) 退化为

$$\begin{cases} \dfrac{\mathrm{d}y}{\mathrm{d}x} = a, \ 0 < a < 1, \\ y(0) = 0, \ y(1) = 1. \end{cases} \tag{28}$$

当 $a \neq 1$ 时，方程 (28) 在 $[0,1]$ 区间上没有连续解，即：问题 (3) 对参数 ε 不连续依赖。

如果从非退化问题的解析解 (2) 出发，则有

$$\lim_{\varepsilon \to 0} \left[\frac{1-a}{1-\mathrm{e}^{-1/\varepsilon}} \left(1 - \mathrm{e}^{-x/\varepsilon}\right) + ax \right] = \begin{cases} 0, & x = 0, \\ ax + (1-a), & x \in (0,1]. \end{cases} \tag{29}$$

解析解对 $\varepsilon \to 0$ 不一致收敛。

考虑线性方程组 (9)。当 $\varepsilon \to 0$ 时，线性方程组退化为

$$\begin{bmatrix} -h & h & & & \\ & -h & h & & \\ & & -h & \ddots & \\ & & & \ddots & h \\ & & & & -h \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} ah^2 \\ ah^2 \\ \vdots \\ ah^2 \\ ah^2 - h \end{bmatrix}. \tag{30}$$

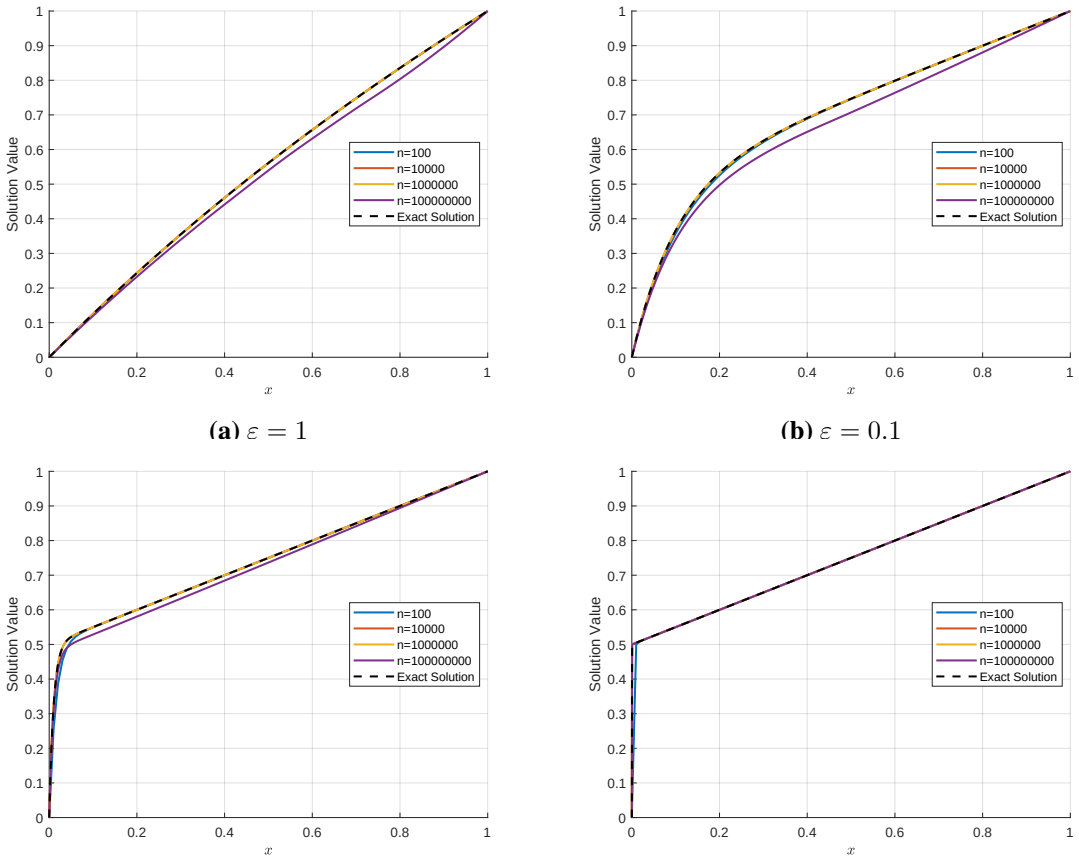
解得 $y_k = 1 - (n-k)ah,\ k = 1, 2, \cdots, n-1$，对应退化的解析解 (29)。

5.2 更细致的剖分

为了保证计算速度足够快和节约内存（对于追赶法，只需存储三个对角线向量即可，不需要存储任何矩阵），选择使用**追赶法**来测试不同剖分点数量 n 下差分法求解两点边值问题的效果。依然取 $a = 0.5$。

表 7: 不同 n 的实验结果

n		10^2	10^4	10^6	10^8
$\varepsilon = 1$	$\|b - Ax\|_\infty$	2.276×10^{-16}	4.137×10^{-16}	3.996×10^{-16}	3.941×10^{-16}
	MSE	4.654×10^{-8}	4.765×10^{-12}	1.238×10^{-10}	4.303×10^{-4}
	Max Error	3.001×10^{-4}	3.022×10^{-6}	1.535×10^{-5}	3.249×10^{-2}
	时间	1.555×10^{-3} s	1.587×10^{-3} s	1.371×10^{-1} s	2.171×10^{0} s
$\varepsilon = 0.1$	$\|b - Ax\|_\infty$	3.612×10^{-17}	4.427×10^{-17}	4.445×10^{-17}	4.715×10^{-17}
	MSE	1.458×10^{-5}	1.557×10^{-9}	1.241×10^{-13}	9.008×10^{-4}
	Max Error	8.824×10^{-3}	9.186×10^{-5}	8.573×10^{-7}	4.013×10^{-2}
	时间	4.019×10^{-4} s	5.921×10^{-4} s	1.908×10^{-2} s	4.249×10^{0} s
$\varepsilon = 0.01$	$\|b - Ax\|_\infty$	5.509×10^{-18}	4.307×10^{-18}	4.552×10^{-18}	5.511×10^{-18}
	MSE	9.666×10^{-5}	1.553×10^{-8}	4.364×10^{-12}	1.985×10^{-4}
	Max Error	6.606×10^{-2}	9.159×10^{-4}	1.060×10^{-5}	2.162×10^{-2}
	时间	3.686×10^{-4} s	1.015×10^{-3} s	1.790×10^{-2} s	5.228×10^{0} s
$\varepsilon = 0.0001$	$\|b - Ax\|_\infty$	3.165×10^{-18}	5.680×10^{-20}	3.866×10^{-20}	3.593×10^{-20}
	MSE	2.427×10^{-7}	9.762×10^{-7}	1.554×10^{-10}	3.258×10^{-9}
	Max Error	4.951×10^{-3}	6.606×10^{-2}	9.159×10^{-4}	9.123×10^{-5}
	时间	4.655×10^{-3} s	1.424×10^{-3} s	1.813×10^{-2} s	3.008×10^{0} s



(a) $\varepsilon = 1$ **(b)** $\varepsilon = 0.1$

(c) $\varepsilon = 0.01$ **(d)** $\varepsilon = 0.0001$

图 6: 不同 n 的实验结果图

从表7的结果看，一方面，追赶法在 n 较大时求解精度依然很高，稳定性很好；另一方面，对于 $\varepsilon = 0.1, 0.01, 0.001$, $n = 10^2, 10^4, 10^6$ 和固定的 ε，剖分点数量 n 越大，数值解与解析解之间的误差越小，即模型误差更小。

定性地看，由 (25)，当 $n \to \infty$ 时，理论上模型误差趋于 0，即趋势上，剖分点数量 n 越大，误差越小。这一结论符合直观。

当 $n = 10^8$ 以及 $\varepsilon = 1$ 时，出现反常现象。猜测可能因差分法的数值稳定性问题导致 n 较大时数值解较解析解产生更大的偏差。

6 总结与反思

本文简单介绍了差分法求解两点边值问题的思路，上机实验并对比了不同参数、不同算法求解问题的数值精度、效率和时间，尝试分析包括线性方程组的求解误差、模型误差、参数与剖分数量影响求解效果等数值现象。

从本文的研究对比不同的算法可以得到，对于大型稀疏线性方程组的求解，利用 Guass 消去法、根平方法这样通用的直接法，虽然精度很高，但没有很好地利用矩阵的稀疏性，计算成本较高、花费时间较长；对于迭代法，共轭梯度法的效果明显优于传统的单步线性定常迭代和最速下降法；追赶法是针对三对角矩阵设计的算法，在实验的算法之中表现最好，速度快、精

度高，且占用内存少。

本文尚且遗留诸多问题有待解决，如：

- 理论上，利用中心差分法估计一阶导数的误差更小，然而在实验中，利用中心差分估计一阶导数时出现线性方程组不收敛于解析解的反常现象（如图7所示），本文并未对此现象做出解释；
- 对于大型矩阵的计算，优化元素的存储方式能够缓解内存的压力。但本文并未讨论如何对于依赖矩阵的算法如何进行元素存储来释放内存压力；
- 仅定性地计算了算法的收敛速度，没有具体计算和讨论算法的收敛阶；
- 虽然本文对模型误差的上界做出了估计，但由于问题的解在 0 附近变化很大，因此将局部误差的估计推广到整体使得这一估计非常粗糙，只能定性地说明问题，无法定量地解析实际的误差大小；
- 对于 SOR 迭代松弛因子的选取，本文没有给出最佳松弛因子的估计或系统的选取方案；
- 对于改进模型来减少模型误差，本文虽提出了采取不等距剖分的模型，但没有对此进行详细的分析和实验；
- 没有讨论算法加速问题。

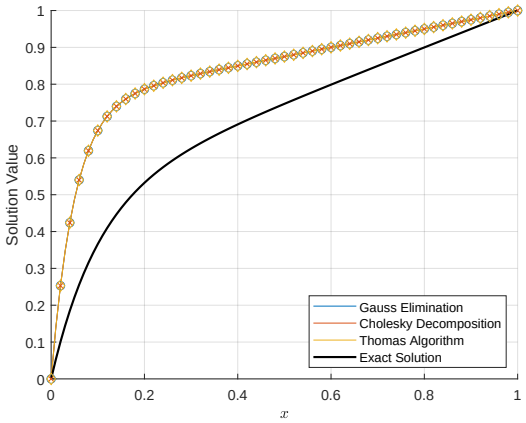


图 7: 利用中心差分估计一阶导数的反常现象

由于时间问题，包括但不限于上述问题还有待讨论和解决。希望老师批评指正！

参考文献

[1] 徐树方, 高立, and 张平文. 数值线性代数（第二版）北京大学出版社, 2013.

附录

A 大作业纸质部分

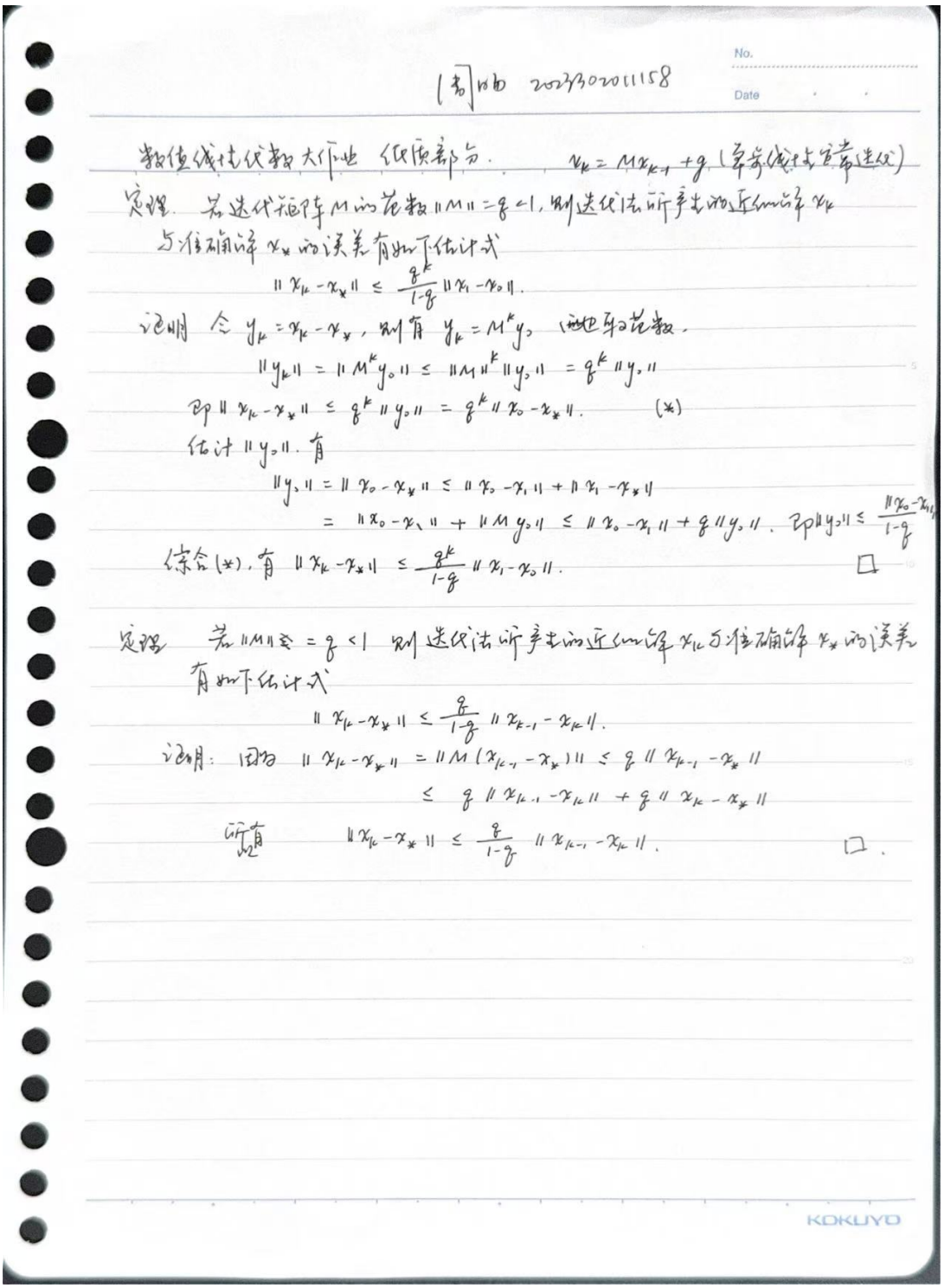


图 8: 纸质部分

B 算法实现的主要代码

B.1 Gauss 消去法

```
function [x, L, U, err, status] = lu_decomp1(A, b)
%LU_DECOMP1 简单 LU 分解解线性方程组 Ax = b
%   [x, L, U, err, status] = LU_DECOMP1(A, b) 使用简单 LU 分解求解线性方程组。
%
% 输入:
%   A - 系数矩阵 (n x n)
%   b - 右端项向量 (长度 n)
%
% 输出:
%   x - 解向量
%   L - 单位下三角矩阵
%   U - 上三角矩阵
%   err - 误差的二范数 ||Ax - b||
%   status - 状态码 (0: 成功, 1: 失败)

% 初始化输出
status = 0;
err = Inf;
x = [];
L = [];
U = [];

% 获取矩阵大小
[n, m] = size(A);
if n ~= m
    disp('Error: Matrix A must be square.');
    status = 1;
    return;
end

% LU 分解
L = eye(n); % 初始化 L 为单位矩阵
U = A;      % 初始化 U 为 A
for k = 1:n-1
```

```
    if abs(U(k, k)) < eps
        disp('Error: Zero pivot encountered.');
        status = 1;
        return;
    end
    for i = k+1:n
        L(i, k) = U(i, k) / U(k, k);
        U(i, k:n) = U(i, k:n) - L(i, k) * U(k, k:n);
    end
end

% 前向替换求解 Ly = b
y = zeros(n, 1);
for i = 1:n
    y(i) = b(i) - L(i, 1:i-1) * y(1:i-1);
end

% 后向替换求解 Ux = y
x = zeros(n, 1);
for i = n:-1:1
    if abs(U(i, i)) < eps
        disp('Error: Zero pivot encountered during back substitution.');
        status = 1;
        return;
    end
    x(i) = (y(i) - U(i, i+1:n) * x(i+1:n)) / U(i, i);
end

% 计算误差
err = norm(b - A * x, Inf);
end
```

B.2 平方根法

```
function [x, L, err, status] = cholesky_decomp1(A, b)
    %CHOLESKY_DECOMP1 Cholesky 分解解线性方程组 Ax = b
    %   [x, L, err, status] = CHOLESKY_DECOMP1(A, b) 使用 Cholesky 分解求解线性
        方程组。
```

```
4       %
5       % 输入:
6       %   A - 对称正定矩阵 (n x n)
7       %   b - 右端项向量 (长度 n)
8       %
9       % 输出:
10      %   x - 解向量
11      %   L - 下三角矩阵
12      %   err - 误差的二范数 ||Ax - b||
13      %   status - 状态码 (0: 成功, 1: 失败)
14
15      % 初始化输出
16      status = 0;
17      err = Inf;
18      x = [];
19      L = [];
20
21      % 获取矩阵大小
22      [n, m] = size(A);
23      if n ~= m
24          disp('Error: Matrix A must be square.');
25          status = 1;
26          return;
27      end
28
29      % 检查是否为对称正定矩阵
30      if ~isequal(A, A') || any(eig(A) <= 0)
31          disp('Error: Matrix A must be symmetric positive definite.');
32          status = 1;
33          return;
34      end
35
36      % Cholesky 分解
37      L = zeros(n, n);
38      for i = 1:n
39          for j = 1:i
40              if i == j
41                  L(i, j) = sqrt(A(i, i) - sum(L(i, 1:j-1).^2));
```

```
            else
                L(i, j) = (A(i, j) - sum(L(i, 1:j-1) .* L(j, 1:j-1))) / L(j, j);
            end
        end
    end
    U = L'; % U 为 L 的转置

    % 前向替换求解 Ly = b
    y = zeros(n, 1);
    for i = 1:n
        y(i) = (b(i) - L(i, 1:i-1) * y(1:i-1)) / L(i, i);
    end

    % 后向替换求解 Ux = y
    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (y(i) - U(i, i+1:n) * x(i+1:n)) / U(i, i);
    end

    % 计算误差
    err = norm(b - A * x, Inf);
    end
```

B.3 追赶法

```
function [x, err, status] = thomas(A, b)
    %THOMAS 追赶法解三对角系数矩阵的线性方程组 Ax = b
    %   [x, err, status] = THOMAS(A, b) 使用追赶法求解三对角矩阵的线性方程组。
    %
    % 输入:
    %   A - 三对角矩阵 (n x n)
    %   b - 右端项向量 (长度 n)
    %
    % 输出:
    %   x - 解向量
    %   err - 误差的二范数 ||Ax - b||
    %   status - 状态码 (0: 成功, 1: 失败)
```

```
13
14      % 初始化输出
15      status = 0;
16      err = Inf;
17      x = [];
18
19      % 获取矩阵大小
20      [n, m] = size(A);
21      if n ~= m
22          disp('Error: Matrix A must be square.');
23          status = 1;
24          return;
25      end
26
27      % 检查是否为三对角矩阵
28      if ~isequal(A, diag(diag(A, -1), -1) + diag(diag(A)) + diag(diag(A, 1), 1))
29          disp('Error: Matrix A must be tridiagonal.');
30          status = 1;
31          return;
32      end
33
34      % 提取三对角矩阵的下对角线、主对角线和上对角线
35      a = diag(A, -1); % 下对角线
36      b_diag = diag(A); % 主对角线
37      c = diag(A, 1); % 上对角线
38
39      % 检查主对角线是否有零元素
40      if any(abs(b_diag) < eps)
41          disp('Error: Zero element on the main diagonal.');
42          status = 1;
43          return;
44      end
45
46      % 获取右端项向量长度
47      if length(b) ~= n
48          disp('Error: Dimension of vector b does not match matrix A.');
49          status = 1;
50          return;
```

```
    end

    % 追赶法分解阶段
    c_prime = zeros(n-1, 1); % 修改后的上对角线
    d_prime = zeros(n, 1); % 修改后的右端项
    c_prime(1) = c(1) / b_diag(1);
    d_prime(1) = b(1) / b_diag(1);

    for i = 2:n-1
        denom = b_diag(i) - a(i-1) * c_prime(i-1);
        if abs(denom) < eps
            disp('Error: Zero pivot encountered during elimination.');
            status = 1;
            return;
        end
        c_prime(i) = c(i) / denom;
        d_prime(i) = (b(i) - a(i-1) * d_prime(i-1)) / denom;
    end
    d_prime(n) = (b(n) - a(n-1) * d_prime(n-1)) / (b_diag(n) - a(n-1) * c_prime
        (n-1));

    % 回代阶段
    x = zeros(n, 1);
    x(n) = d_prime(n);
    for i = n-1:-1:1
        x(i) = d_prime(i) - c_prime(i) * x(i+1);
    end

    % 计算误差的无穷范数
    err = norm(b - A * x, Inf);
    end
```

B.4 Jacobi 迭代法

```
function [x, inf_norm_err, iter, status, error_list] = jacobi_iter(A, b, tol,
    kmax, plot_iter, x0)
%JACOBI_ITER Jacobi 迭代法解线性方程组 Ax = b
```

```
3  %  [x, inf_norm_err, iter, status, error_list] = JACOBI_ITER(A, b, tol, kmax,
       plot_iter, x0)
4  %  使用 Jacobi 迭代法解线性方程组，并根据误差无穷范数或最大迭代次数确定停机条件。
5  %
6  % 输入:
7  %  A         - 系数矩阵 (n x n)
8  %  b         - 右端项向量 (长度 n)
9  %  tol       - 收敛容限
10 %  kmax      - 最大迭代次数
11 %  plot_iter - 是否绘制误差迭代图 (布尔值, true/false)
12 %  x0        - 初始向量 (可选, 默认为零向量)
13 %
14 % 输出:
15 %  x         - 解向量
16 %  inf_norm_err - 最终误差的无穷范数
17 %  iter      - 实际迭代次数
18 %  status    - 状态码 (0: 成功, 1: 异常矩阵, 2: 超过最大迭代次数)
19 %
20 % 思路:
21 %  1. 确保 A 为方阵且对角元素非零。
22 %  2. 根据 Jacobi 迭代公式逐步逼近解向量。
23 %  3. 计算误差向量 (b - Ax) 的无穷范数。
24 %  4. 如果误差小于 tol 或迭代次数达到 kmax, 则停止迭代。
25 %  5. 根据需要绘制误差随迭代次数的变化曲线。
26
27 % 初始化输出
28 status = 0;
29 inf_norm_err = Inf;
30 iter = 0;
31 error_list = [];
32
33 % 检查输入矩阵是否是方阵
34 [n, m] = size(A);
35 if n ~= m
36     disp('Error: Matrix A must be square.');
37     status = 1;
38     x = [];
39     return;
```

```
end

% 检查 A 的对角元素是否为零
if any(abs(diag(A)) < eps)
    disp('Error: Matrix A has zero diagonal elements.');
    status = 1;
    x = [];
    return;
end

% 初始化
if nargin < 6 || isempty(x0)
    x = zeros(n, 1); % 默认初始解向量为零向量
else
    x = x0; % 使用用户提供的初始向量
end
x_old = x; % 保存上一轮的解向量
errors = zeros(kmax, 1); % 用于存储每次迭代的误差（无穷范数）

% Jacobi 迭代
D = diag(A); % 提取对角元素
R = A - diag(D); % 提取非对角部分
for k = 1:kmax
    % Jacobi 迭代公式
    x = (b - R * x_old) ./ D;

    % 计算误差的无穷范数
    inf_norm_err = norm(x - x_old, Inf);
    errors(k) = inf_norm_err;

    % 检查是否满足收敛条件
    if inf_norm_err < tol
        iter = k;
        status = 0; % 正常结束
        error_list = errors(1:k); % 截取有效的误差记录
        break;
    end

```

```
78      % 更新 x_old
79      x_old = x;
80  end
81
82  % 检查是否达到最大迭代次数
83  if k == kmax && inf_norm_err >= tol
84      disp('Warning: Jacobi iteration did not converge within the maximum number
            of iterations.');
85      iter = k;
86      status = 2; % 达到最大迭代次数
87      error_list = errors;
88  end
89
90  % 绘制误差变化曲线（对数尺度）
91  if plot_iter && ~isempty(error_list)
92      figure;
93      semilogy(1:iter, error_list, '-o');
94      xlabel('Iteration');
95      ylabel('Infinity Norm of Error');
96      title('Jacobi Iteration Convergence');
97      grid on;
98  end
99  end
```

B.5 Gauss-Seidel 迭代法

```
1   function [x, inf_norm_err, iter, status, error_list] = gauss_seidel_iter(A, b,
        tol, kmax, plot_iter, x0)
2   %GAUSS_SEIDEL_ITER Gauss-Seidel 迭代法解线性方程组 Ax = b
3   %   [x, inf_norm_err, iter, status, error_list] = GAUSS_SEIDEL_ITER(A, b, tol,
        kmax, plot_iter, x0)
4   %   使用 Gauss-Seidel 迭代法解线性方程组，并根据误差无穷范数或最大迭代次数确定停机
        条件。
5   %
6   % 输入:
7   %   A         - 系数矩阵 (n x n)
8   %   b         - 右端项向量 (长度 n)
9   %   tol       - 收敛容限
```

```
%   kmax     - 最大迭代次数
%   plot_iter - 是否绘制误差迭代图 (布尔值, true/false)
%   x0       - 初始向量 (可选, 默认为零向量)
%
% 输出:
%   x          - 解向量
%   inf_norm_err - 最终误差的无穷范数
%   iter       - 实际迭代次数
%   status     - 状态码 (0: 成功, 1: 异常矩阵, 2: 超过最大迭代次数)
%
    % 思路:
    %   1. 确保 A 为方阵且对角元素非零。
    %   2. 根据 Gauss-Seidel 迭代公式逐步逼近解向量。
    %   3. 计算误差向量 (b - Ax) 的无穷范数。
    %   4. 如果误差小于 tol 或迭代次数达到 kmax, 则停止迭代。
    %   5. 根据需要绘制误差随迭代次数的变化曲线。

% 初始化输出
status = 0;
inf_norm_err = Inf;
iter = 0;
error_list = [];

% 检查输入矩阵是否是方阵
[n, m] = size(A);
if n ~= m
    disp('Error: Matrix A must be square.');
    status = 1;
    x = [];
    return;
end

% 检查 A 的对角元素是否为零
if any(abs(diag(A)) < eps)
    disp('Error: Matrix A has zero diagonal elements.');
    status = 1;
    x = [];
    return;
```

```
48  end
49
50  % 初始化
51  if nargin < 6 || isempty(x0)
52      x = zeros(n, 1); % 默认初始解向量为零向量
53  else
54      x = x0; % 使用用户提供的初始向量
55  end
56  x_old = x; % 保存上一轮的解向量
57  errors = zeros(kmax, 1); % 用于存储每次迭代的误差（无穷范数）
58
59  % Gauss-Seidel 迭代
60  for k = 1:kmax
61      for i = 1:n
62          % 计算 Gauss-Seidel 迭代公式
63          sigma = A(i, 1:i-1) * x(1:i-1) + A(i, i+1:n) * x_old(i+1:n);
64          x(i) = (b(i) - sigma) / A(i, i);
65      end
66
67      % 计算误差的无穷范数
68      inf_norm_err = norm(x - x_old, Inf);
69      errors(k) = inf_norm_err;
70
71      % 检查是否满足收敛条件
72      if inf_norm_err < tol
73          iter = k;
74          status = 0; % 正常结束
75          error_list = errors(1:k); % 截取有效的误差记录
76          break;
77      end
78
79      % 更新 x_old
80      x_old = x;
81  end
82
83  % 检查是否达到最大迭代次数
84  if k == kmax && inf_norm_err >= tol
85      disp('Warning: Gauss-Seidel iteration did not converge within the maximum
```

```
        number␣of␣iterations.');
86      iter = k;
87      status = 2; % 达到最大迭代次数
88      error_list = errors;
89  end
90
91  % 绘制误差变化曲线（对数尺度）
92  if plot_iter && ~isempty(error_list)
93      figure;
94      semilogy(1:iter, error_list, '-o');
95      xlabel('Iteration');
96      ylabel('Infinity␣Norm␣of␣Error');
97      title('Gauss-Seidel␣Iteration␣Convergence');
98      grid on;
99  end
100 end
```

B.6 SOR 迭代法

```
1   function [x, inf_norm_err, iter, status, error_list] = sor_iter(A, b, tol,
        kmax, omega, plot_iter, x0)
2   %SOR_ITER SOR 迭代法解线性方程组 Ax = b
3   %   [x, inf_norm_err, iter, status, error_list] = SOR_ITER(A, b, tol, kmax,
        omega, plot_iter, x0)
4   %   使用 SOR 迭代法解线性方程组，并根据误差无穷范数或最大迭代次数确定停机条件。
5   %
6   % 输入:
7   %   A         - 系数矩阵 (n x n)
8   %   b         - 右端项向量 (长度 n)
9   %   tol       - 收敛容限
10  %   kmax      - 最大迭代次数
11  %   omega     - 松弛因子 (0 < omega < 2)
12  %   plot_iter - 是否绘制误差迭代图 (布尔值, true/false)
13  %   x0        - 初始向量 (可选, 默认为零向量)
14  %
15  % 输出:
16  %   x           - 解向量
17  %   inf_norm_err - 最终误差的无穷范数
```

```
%   iter      - 实际迭代次数
%   status    - 状态码 (0: 成功, 1: 异常矩阵, 2: 超过最大迭代次数)
%
    % 思路:
    %   1. 确保 A 为方阵且对角元素非零。
    %   2. 根据 SOR 迭代公式逐步逼近解向量。
    %   3. 计算误差向量 (b - Ax) 的无穷范数。
    %   4. 如果误差小于 tol 或迭代次数达到 kmax, 则停止迭代。
    %   5. 根据需要绘制误差随迭代次数的变化曲线。

% 初始化输出
status = 0;
inf_norm_err = Inf;
iter = 0;
error_list = [];

% 检查输入矩阵是否是方阵
[n, m] = size(A);
if n ~= m
    disp('Error: Matrix A must be square.');
    status = 1;
    x = [];
    return;
end

% 检查 A 的对角元素是否为零
if any(abs(diag(A)) < eps)
    disp('Error: Matrix A has zero diagonal elements.');
    status = 1;
    x = [];
    return;
end

% 检查松弛因子是否在有效范围内
if omega <= 0 || omega >= 2
    disp('Error: Relaxation factor omega must be in the range (0, 2).');
    status = 1;
    x = [];
```

```
    return;
end

% 初始化
if nargin < 7 || isempty(x0)
    x = zeros(n, 1); % 默认初始解向量为零向量
else
    x = x0; % 使用用户提供的初始向量
end
x_old = x; % 保存上一轮的解向量
errors = zeros(kmax, 1); % 用于存储每次迭代的误差（无穷范数）

% SOR 迭代
for k = 1:kmax
    for i = 1:n
        % 计算 SOR 迭代公式
        sigma = A(i, 1:i-1) * x(1:i-1) + A(i, i+1:n) * x_old(i+1:n);
        x(i) = (1 - omega) * x_old(i) + omega * (b(i) - sigma) / A(i, i);
    end

    % 计算误差的无穷范数
    inf_norm_err = norm(x - x_old, Inf);
    errors(k) = inf_norm_err;

    % 检查是否满足收敛条件
    if inf_norm_err < tol
        iter = k;
        status = 0; % 正常结束
        error_list = errors(1:k); % 截取有效的误差记录
        break;
    end

    % 更新 x_old
    x_old = x;
end

% 检查是否达到最大迭代次数
if k == kmax && inf_norm_err >= tol
```

```
    disp('Warning: SOR iteration did not converge within the maximum number of
        iterations.');
    iter = k;
    status = 2; % 达到最大迭代次数
    error_list = errors;
end

% 绘制误差变化曲线（对数尺度）
if plot_iter && ~isempty(error_list)
    figure;
    semilogy(1:iter, error_list, '-o');
    xlabel('Iteration');
    ylabel('Infinity Norm of Error');
    title('SOR Iteration Convergence');
    grid on;
end
end
```

B.7 最速下降法

```
function [x, inf_norm_err, iter, status, err_list] = steepest_descent(A, b,
    tol, kmax, plot_iter, x0)
    %STEEPEST_DESCENT 最速下降法解线性方程组 Ax = b
    %   [x, inf_norm_err, iter, status, err_list] = STEEPEST_DESCENT(A, b, tol,
         kmax, plot_iter, x0)
    %   使用最速下降法解线性方程组，并返回每一代的误差列表。
    %
    % 输入:
    %   A        - 系数矩阵 (n x n)，必须是对称正定矩阵
    %   b        - 右端项向量 (长度 n)
    %   tol      - 收敛容限
    %   kmax     - 最大迭代次数
    %   plot_iter - 是否绘制误差迭代图 (布尔值, true/false)
    %   x0       - 初始向量 (可选，默认为零向量)
    %
    % 输出:
    %   x          - 解向量
```

```
16      %   inf_norm_err - 最终误差的无穷范数
17      %   iter         - 实际迭代次数
18      %   status       - 状态码 (0: 成功, 1: 异常矩阵, 2: 超过最大迭代次数)
19      %   err_list     - 每一代的误差无穷范数列表
20      %
21      % 思路:
22      %   1. 确保 A 为对称正定矩阵。
23      %   2. 初始化解向量 x 和残差向量 r = b - Ax。
24      %   3. 计算步长 alpha = (r' * r) / (r' * A * r)。
25      %   4. 更新解向量 x 和残差向量 r。
26      %   5. 检查误差无穷范数是否小于 tol 或迭代次数是否达到 kmax。
27      %   6. 根据需要绘制误差随迭代次数的变化曲线。
28
29      % 初始化输出
30      status = 0;
31      inf_norm_err = Inf;
32      iter = 0;
33      err_list = [];
34
35      % 检查输入矩阵是否是对称正定矩阵
36      if ~isequal(A, A') || any(eig(A) <= 0)
37          disp('Error: Matrix A must be symmetric positive definite.');
38          status = 1;
39          x = [];
40          return;
41      end
42
43      % 初始化
44      n = size(A, 1);
45
46      % 检查是否提供初始向量 x0
47      if nargin < 6 || isempty(x0)
48          x = zeros(n, 1); % 默认初始向量为零向量
49      else
50          x = x0; % 使用用户提供的初始向量
51      end
52
53      r = b - A * x; % 初始残差向量
```

```
    errors = zeros(kmax, 1); % 用于存储每次迭代的误差（无穷范数）

    % 最速下降法迭代
    for k = 1:kmax
        % 计算步长 alpha
        alpha = (r' * r) / (r' * A * r);

        % 更新解向量 x 和残差向量 r
        x = x + alpha * r;
        r = b - A * x;

        % 计算误差的无穷范数
        inf_norm_err = norm(r, Inf);
        errors(k) = inf_norm_err;

        % 检查是否满足收敛条件
        if inf_norm_err < tol
            iter = k;
            status = 0; % 正常结束
            err_list = errors(1:k); % 截取有效的误差记录
            break;
        end
    end

    % 检查是否达到最大迭代次数
    if k == kmax && inf_norm_err >= tol
        disp('Warning: Steepest descent did not converge within the maximum
            number of iterations.');
        iter = k;
        status = 2; % 达到最大迭代次数
        err_list = errors;
    end

    % 绘制误差变化曲线（对数尺度）
    if plot_iter && ~isempty(err_list)
        figure;
        semilogy(1:iter, err_list, '-o');
        xlabel('Iteration');
```

```
        ylabel('Infinity Norm of Error');
        title('Steepest Descent Convergence');
        grid on;
    end
end
```

B.8 共轭梯度法

```
function [x, inf_norm_err, iter, status, err_list] = conjugate_gradient(A, b,
    tol, kmax, plot_iter, x0)
%CONJUGATE_GRADIENT 共轭梯度法解线性方程组 Ax = b
%   [x, inf_norm_err, iter, status, err_list] = CONJUGATE_GRADIENT(A, b, tol,
    kmax, plot_iter, x0)
%   使用共轭梯度法解线性方程组，并返回每一代的误差列表。
%
% 输入:
%   A         - 系数矩阵 (n x n)，必须是对称正定矩阵
%   b         - 右端项向量 (长度 n)
%   tol       - 收敛容限
%   kmax      - 最大迭代次数
%   plot_iter - 是否绘制误差迭代图 (布尔值，true/false)
%   x0        - 初始向量 (可选，默认为零向量)
%
% 输出:
%   x         - 解向量
%   inf_norm_err - 最终误差的无穷范数
%   iter      - 实际迭代次数
%   status    - 状态码 (0: 成功，1: 异常矩阵，2: 超过最大迭代次数)
%   err_list  - 每一代的误差无穷范数列表
%
% 思路:
%   1. 确保 A 为对称正定矩阵。
%   2. 初始化解向量 x 和残差向量 r = b - Ax。
%   3. 初始化搜索方向 p = r。
%   4. 迭代更新步长 alpha、解向量 x、残差向量 r 和搜索方向 p。
%   5. 检查误差无穷范数是否小于 tol 或迭代次数是否达到 kmax。
%   6. 根据需要绘制误差随迭代次数的变化曲线。
```

```
28
29  % 初始化输出
30  status = 0;
31  inf_norm_err = Inf;
32  iter = 0;
33  err_list = [];
34
35  % 检查输入矩阵是否是对称正定矩阵
36  if ~isequal(A, A') || any(eig(A) <= 0)
37      disp('Error: Matrix A must be symmetric positive definite.');
38      status = 1;
39      x = [];
40      return;
41  end
42
43  % 初始化
44  n = size(A, 1);
45
46  % 检查是否提供初始向量 x0
47  if nargin < 6 || isempty(x0)
48      x = zeros(n, 1); % 默认初始向量为零向量
49  else
50      x = x0; % 使用用户提供的初始向量
51  end
52
53  r = b - A * x; % 初始残差向量
54  p = r; % 初始搜索方向
55  errors = zeros(kmax, 1); % 用于存储每次迭代的误差（无穷范数）
56
57  % 共轭梯度法迭代
58  for k = 1:kmax
59      % 计算步长 alpha
60      alpha = (r' * r) / (p' * A * p);
61
62      % 更新解向量 x 和残差向量 r
63      x = x + alpha * p;
64      r_new = r - alpha * A * p;
65
```

```
    % 计算误差的无穷范数
    inf_norm_err = norm(r_new, Inf);
    errors(k) = inf_norm_err;

    % 检查是否满足收敛条件
    if inf_norm_err < tol
        iter = k;
        status = 0; % 正常结束
        err_list = errors(1:k); % 截取有效的误差记录
        break;
    end

    % 更新搜索方向 p
    beta = (r_new' * r_new) / (r' * r);
    p = r_new + beta * p;

    % 更新残差向量
    r = r_new;
end

% 检查是否达到最大迭代次数
if k == kmax && inf_norm_err >= tol
    disp('Warning: Conjugate gradient did not converge within the maximum
        number of iterations.');
    iter = k;
    status = 2; % 达到最大迭代次数
    err_list = errors;
end

% 绘制误差变化曲线（对数尺度）
if plot_iter && ~isempty(err_list)
    figure;
    hold on;
    semilogy(1:iter, err_list, '-o');
    xlabel('Iteration');
    ylabel('Infinity Norm of Error');
    title('Conjugate Gradient Convergence');
    grid on;
```

```
103         set(gca, 'YScale', 'log');
104         hold off;
105     end
106     end
```