第四讲 线性方程组迭代解法

定常迭代法

目录

为什么迭代法

直接法 VS 迭代法

- 直接法运算量 $\mathcal{O}(n^3)$, 随着矩阵规模的增大, 运算量也随之 **快速增长**
- 对于大规模线性方程组, 特别是稀疏方程组, 当前的首选方法是 **迭代方法**

迭代法基本思想

$$Ax = b, \quad A \in \mathbb{R}^{n\times n} \text{ 非奇异}$$

给定一个初始值 $x^{(0)}$, 通过 **一定的迭代格式** 生成一个迭代序列 $\{x^{(k)}\}_{k=0}^{\infty}$, 使得

$$\lim_{k\to\infty} x^{(k)} = x_* \triangleq A^{-1}b$$

目前常用的两类迭代法

- **定常迭代法**: 如 Jacobi, Gauss-Seidel, SOR, ADI 等.
- **子空间迭代法**: 如 CG, MINRES, GMRES, BiCGStab 等.

4-1 定常迭代法

4.1 定常迭代法

4-1-1 迭代法基本概念

基本思想

直接求解 $Ax = b$ 比较困难, 我们可以

(1) 求解一个近似线性方程组 $Mx = b$, 其中 M 是 A 在某种意义下的近似.

(2) 根据精度需求对近似解进行更新.

记 $Mx = b$ 的解为 $x^{(1)}$, 与原方程的解 $x_* = A^{-1}b$ 之间的误差满足

$$A\left(x_* - x^{(1)}\right) = b - Ax^{(1)}.$$

如果 $x^{(1)}$ 已经满足精度要求, 则可以停止计算, 否则需要 修正.

记 $\Delta x \triangleq x_* - x^{(1)}$, 则 Δx 满足方程

$$A\Delta x = b - Ax^{(1)}.$$

由于直接求解比较困难, 因此我们还是求解近似方程

$$M\Delta x = b - Ax^{(1)}$$

得到一个近似的修正量 $\tilde{\Delta}x$. 于是修正后的近似解为

$$x^{(2)} = x^{(1)} + \tilde{\Delta}x = x^{(1)} + M^{-1}(b - Ax^{(1)}).$$

如果 $x^{(2)}$ 已经满足精度要求, 则停止计算, 否则 **继续按以上的方式进行修正**.

不断重复以上步骤, 于是, 我们就得到一个向量序列

$$x^{(1)}, x^{(2)}, \ldots, , x^{(k)}, \ldots.$$

它们都是真解 x_* 的近似值, 且满足下面的递推关系

$$x^{(k+1)} = x^{(k)} + M^{-1}(b - Ax^{(k)}), \quad k = 1, 2, \ldots.$$

这就构成了一个迭代方法.

由于每次迭代的格式是一样的, 因此称为 **定常迭代法**.

构造定常迭代法需要考虑的两个方面

(1) 以 M 为系数矩阵的线性方程组必须比原线性方程组 **更容易求解**

(2) M 应该是 A 的一个很好的近似: 迭代序列 $\{x_k\}$ **快速收敛**

如何选取 M? — 矩阵分裂

目前一类比较常用的定常迭代法是基于矩阵分裂的迭代法, 如经典的 Jacobi 方法, Gauss-Seidel 方法, SOR (Successive Over-Relaxation, 超松弛) 方法等.

定义 (矩阵分裂 Matrix Splitting) 设 $A \in \mathbb{R}^{n\times n}$ 非奇异, 我们称

$$A = M - N \tag{4.1}$$

为 A 的一个**矩阵分裂**, 其中 M 非奇异.

基于矩阵分裂的迭代法

给定一个矩阵分裂, 则原方程组 $Ax = b$ 就等价于 $Mx = Nx + b$.

于是我们就可以构造出以下的迭代格式

$$Mx^{(k+1)} = Nx^{(k)} + b, \quad k = 0, 1, \ldots, \tag{4.2}$$

或

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b \triangleq Gx^{(k)} + f, \quad k = 0, 1, \ldots, \tag{4.3}$$

其中 $G \triangleq M^{-1}N$ 称为 **迭代矩阵**.

这就是基于矩阵分裂的迭代方法, 选取不同的 M, 就得到不同的迭代方法.

三种经典定常迭代法

$$A = D - L - U$$

- Jacobi 迭代法
- Gauss-Seidel 迭代法
- SOR 迭代法

4-1-2 Jacobi 迭代法

Jacobi 迭代法: $M = D, N = L + U$

- 迭代格式: $x^{(k+1)} = D^{-1}(L+U)x^{(k)} + D^{-1}b$, $k = 0, 1, \ldots$
- 迭代矩阵: $G_{\mathrm{J}} = D^{-1}(L+U)$
- 分量形式: $$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, n.$$

Jacobi 迭代算法

算法 Jacobi 迭代算法

1: Given an initial guess $x^{(0)}$
2: **while** not converge **do**　% 停机准则
3: 　**for** $i = 1$ to n **do**
4: 　　$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$
5: 　**end for**
6: **end while**

几点说明

- Jacobi 迭代中 $x_i^{(k+1)}$ 的更新顺序与 i 无关, 因此非常 适合并行计算.
- 在编程实现该算法时, "停机准则" 一般是要求相对残量满足一定的精度, 即

$$\frac{\|b - Ax^{(k)}\|}{\|b - Ax^{(0)}\|} < \text{tol},$$

其中 tol 是一个事前给定的精度要求, 如 10^{-6} 或 10^{-8} 等.

- Jacobi 迭代格式也可以写为

$$x^{(k+1)} = x^{(k)} + D^{-1}(b - Ax^{(k)}) = x^{(k)} + D^{-1}r_k$$

其中 $r_k \triangleq b - Ax^{(k)}$. 这表明, $x^{(k+1)}$ 是通过对 $x^{(k)}$ 做一个修正得到的.

4-1-3 Gauss-Seidel (G-S) 迭代法

Gauss-Seidel 迭代法: $\boldsymbol{M = D - L, N = U}$

- 迭代格式: $x^{(k+1)} = (D-L)^{-1}Ux^{(k)} + (D-L)^{-1}b$, $\quad k = 0, 1, \ldots$
- 迭代矩阵: $G_{\mathrm{GS}} = (D-L)^{-1}U$
- 分量形式: $Dx^{(k+1)} = Lx^{(k+1)} + Ux^{(k)} + b$

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

Gauss-Seidel 迭代算法

算法 Gauss-Seidel 迭代算法

1: Given an initial guess $x^{(0)}$
2: **while** not converge **do**
3: **for** $i = 1$ to n **do**
4: $$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$
5: **end for**
6: **end while**

Gauss-Seidel 迭代法的优点

Gauss-Seidel 迭代法充分利用了已经获得的最新数据, 有望获得更快的收敛速度.

4-1-4 SOR 迭代法

SOR 迭代法: $M = \frac{1}{\omega}D - L, N = \frac{1}{\omega}(1-\omega)D + U$

- 基本思想: 将 G-S 迭代法的 $x^{(k+1)}$ 与 $x^{(k)}$ 加权平均, 以获得 更好 的近似解.
- 迭代格式: $x^{(k+1)} = (1-\omega)x^{(k)} + \omega D^{-1}\left(Lx^{(k+1)} + Ux^{(k)} + b\right)$

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

其中 ω 称为松弛参数.

- 迭代矩阵: $G_{\mathrm{SOR}} = (D-\omega L)^{-1}\big((1-\omega)D + \omega U\big)$

- 分量形式:

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

注记

- 当 $\omega = 1$ 时, SOR 即为 G-S 方法,
  当 $\omega < 1$ 时, 称为**低松弛**方法, 当 $\omega > 1$ 时, 称为**超松弛**方法.
  在大多数情况下, 当 $\omega > 1$ 时会取得比较好的收敛效果.

- SOR 方法曾经在很长一段时间内是科学计算中求解线性方程组的首选方法.

SOR 迭代算法

算法 求解线性方程组的 SOR 迭代方法

1: Given an initial guess $x^{(0)}$ and parameter ω
2: **while** not converge **do**
3: **for** $i = 1$ to n **do**
4: $$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$
5: **end for**
6: **end while**

几点说明

- SOR 的优点是引入了松弛参数 ω, 通过选取适当的 ω 可以大大提高收敛速度.
- 确定 SOR 的 **最优** 松弛因子通常是非常困难的!

例 分别用 Jacobi, G-S 和 SOR($\omega = 1.1$) 迭代方法求解线性方程组 $Ax = b$, 其中

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}.$$

初始向量设为 $x^{(0)} = [0, 0, 0]^{\mathsf{T}}$, 迭代过程中保留小数点后四位.

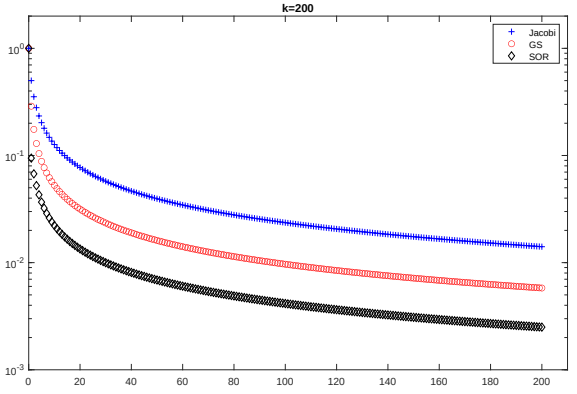
(Iter_Jacobi_GS_SOR_01.m)

例 编程实践: 用 Jacobi, G-S 和 SOR($\omega = 1.5$) 求解线性方程组 $Ax = b$, 其中

$$A = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

初始向量设为 $x^{(0)} = [0, 0, 0]^{\mathsf{T}}$.

(Iter_Jacobi_GS_SOR_02.m)



谢 谢
THANK YOU