作者

2019年7月9日

第六讲　线性方程组的迭代解法

1 离散 Poisson 方程

2 定常迭代算法

3 收敛性分析

4 加速算法

5 交替方向与 HSS 算法

6 快速 Poisson 算法

方法综述

- 直接法　PLU 分解, LDLT 分解, Cholesky 分解法
- 迭代法
  -经典 (定常，不动点) 迭代法：Jacobi/Gauss-Seidel, SOR, AOR 等
- -Krylov 子空间迭代法: CG, MIRES, GMRES, BiCGStab 等
- 快速算法
  -基于快速变换, 如 FFT, DCT, DST 等
  -代数多重网格法 (Algebraic multigrid)
  -快速多极子算法 (Fast multipole)

有些方法可能只是对某类方程有效, 如快速算法. 在实际应用中, 这些方法可以结合使用, 如混合 (hybrid) 算法, 预处理算法 (preconditioning) 等

本讲主要介绍定常迭代算法

更多迭代方法可参见Templates for the Solution of Linear Systems:Building Blocks for Iterative Methods,SLAM,1994

1 离散 Poisson 方程

1.1 一维 Poisson 方程

1.2 二维 Poisson 方程

在本讲中, 我们以一个典型的线性方程组为例, 逐个介绍各种迭代方法, 并比较它们之间的性能. 这个方程组就是二维 Poisson 方程经过五点差分离散后得到的线性方程组.

1.1 一维 Poisson 方程

考虑如下带 Dirichlet 边界条件的一维 Poisson 方程

$$\begin{cases} \frac{\mathrm{d}^2 u(x)}{\mathrm{d}x^2} = f(x), 0 < x < 1, \\ u(0) = a, u(1) = b \end{cases} \tag{6.1}$$

其中 $f(x)$ 是给定的函数,$u(x)$ 是需要计算的未知函数.
差分离散
取步长 $h = \frac{1}{n+1}$, 为节点 $x_i = ih, i = 0, 1, 2, ..., n+1$ 我们采用中心差分离散，可得 $(i = 1, 2, ..., n)$

$$-\left.\frac{d^2u(x)}{dx^2}\right|_{x_i} = \frac{2u(x_i) - u(x_{i-1}) - u(x_{i+1})}{h^2} + O\left(h^2 \cdot \left\|\frac{d^4u}{dx^4}\right\|_\infty\right)$$

将其代入 (6.1), 舍去高阶项后可得 Poisson 方程在 x_i 点的近似离散方程

$$\boxed{-u_{i-1} + 2u_i - u_{i+1} = h^2 f_i,}$$

其中 $fi = f(xi), ui$ 为 $u(xi)$ 的近似.
令 $i = 1, 2, ..., n$, 则可 0 得 n 个线性方程, 写成矩阵形式

$$T_n u = f, \tag{6.2}$$

其中

$$T_n = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}, u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix}, f = \begin{bmatrix} f_1 + u_0 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n + u_{n+1} \end{bmatrix} \tag{6.3}$$

系数矩阵 T_n 的性质

引理　T_n 的特征值和对应的特征向量分别为

$$\lambda_k = 2 - 2cos\frac{k\pi}{n+1},$$

$$z_k = \sqrt{\frac{2}{n+1}} \cdot \left[\sin\frac{k\pi}{n+1}, \sin\frac{2k\pi}{n+1}, \ldots, \sin\frac{nk\pi}{n+1}\right]^T$$

即 $Tn = ZAZ_T$, 其中 $A = diag(\lambda_1, \lambda_2, ..., \lambda_n), Z = [z_1, z_2, ..., z_n]$.
证明. 直接带入验证即可.
引理　更一般的，设 $T = tridiag(a, b, c) \in \mathbb{R}^{n\times n}$，则 T 的特征值为

$$\lambda_k = b - 2\sqrt{ac}cos\frac{k\pi}{n+1}, k = 1, 2, ..., n$$

对应的特征向量为 z_k, 其第 j 个分量为

$$z_k(j) = (\frac{a}{c})^{\frac{j}{2}} sin\frac{jk\pi}{n+1}$$

特别地, 若 $a = c = 1$, 则对应的单位特征向量为

$$z_k = \sqrt{\frac{2}{n+1}} \cdot \left[\sin\frac{k\pi}{n+1}, \sin\frac{2k\pi}{n+1}, \ldots, \sin\frac{nk\pi}{n+1}\right]^T$$

由前面的结论可知,T_n 是对称正定的, 其最大特征值为

$$2\left(1 - \cos\frac{n\pi}{n+1}\right) = 4\sin^2\frac{n\pi}{2(n+1)} \approx 4,$$

最小特征值为

$$2\left(1 - \cos\frac{\pi}{n+1}\right) = 4\sin^2\frac{\pi}{2(n+1)} \approx \left(\frac{\pi}{n+1}\right)^2$$

因此, 当 n 很大时,T_n 的谱条件数约为

$$\kappa_2(T_n) \approx \frac{4(n+1)^2}{\pi^2}$$

矩阵 T_n 可以分解为 $T_n = DD^T$, 其中

$$D = \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{n\times(n+1)}$$

矩阵 D 也通常称为差分矩阵. 需要注意的是,D 不是方阵, 因此不能用这个分解来求解线性方程组 $T_n x = b$.

1.2 二维 Poisson 方程

现在考虑二维 Poisson 方程

$$\begin{cases} -\Delta u(x,y) = -\frac{\partial^2 u(x,y)}{\partial x^2} - \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y), & (x,y) \in \Omega \\ u(x,y) = u_0(x,y), & (x,y) \in \partial\Omega \end{cases} \tag{6.4}$$

其中 $\Omega = [0,1] \times [0,1]$ 为求解区域，$\partial\Omega$ 表示 Ω 的边界.

五点差分离散

为了简单起见，我们在 $x-$ 方向和 $y-$ 方向取相同的步长 $h = \frac{1}{n+1}$，节点设为 $x_i = ih, y_j = jh, i, j = 0, 1, 2, \ldots, n$. 在 $x-$ 方向和 $y-$ 方向同时采用中心差分离散可得

$$\left.\frac{\partial^2 u(x,y)}{\partial x^2}\right|_{(x_i,y_j)} \approx \frac{2u(x_i,y_j) - u(x_{i-1},y_j) - u(x_{i+1},y_j)}{h^2}$$

$$\left.\frac{\partial^2 u(x,y)}{\partial y^2}\right|_{(x_i,y_j)} \approx \frac{2u(x_i,y_j) - u(x_i,y_{j-1}) - u(x_i,y_{j+1})}{h^2}$$

代入 (6.4)，即得二维 Poisson 方程在 (x_i, y_j) 点的近似离散方程

$$\boxed{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = h^2 f_{i,j}}$$

其中 $f_{ij} = f(x_i, y_j), u_{i,j}$ 为 $u(x_i, y_j)$ 的近似。写成矩阵形式即为

$$T_N u = h^2 f \tag{6.5}$$

其中

$$T_N \triangleq I \otimes T_n + T_n \otimes I, \quad N = n^2,$$

$$u = [u_{1,1}, \ldots, u_{n,1}, u_{1,2}, \ldots, u_{n,2}, \ldots, u_{1,n}, \ldots, u_{n,n}],$$

在后面介绍的算法时，我们都以二维离散 Poisson 方程 (6.5) 为例.

系数矩阵 T 的性质

因为 $T_N = I \otimes T_n + T_n \otimes I$ 由 Kronecker 乘积的性质即得定理　设 $T_n = Z\Lambda Z^T$ 其中 $Z = [z_1, z_2, \ldots, z_n]$ 为正交阵，$\Lambda = \text{diag}(\lambda_1\lambda_2, \ldots, \lambda_n)$ 为对角阵，则 T 的特征值分解为

$$T_N = (Z \otimes Z)(I \otimes \Lambda + \Lambda \otimes I)(Z \otimes Z)^T$$

即 T 的特征值为 $\lambda_i + \lambda_j$，对应的特征向量为 $z_i \otimes z_j, i, j = 1, 2, \ldots, n$ 条件数

$$\kappa(T_N) = \frac{\lambda_{\max}(T_N)}{\lambda_{\min}(T_N)} = \frac{1 - \cos\frac{n\pi}{n+1}}{1 - \cos\frac{\pi}{n+1}} = \frac{\sin^2\frac{n\pi}{2(n+1)}}{\sin^2\frac{\pi}{2(n+1)}} \approx \frac{4(n+1)^2}{\pi^2}$$

故当 n 越来越大时 $\kappa(T_N) \to \infty$，即 T_N 越来越病态.

二维离散 Poisson 方程的常用算法

	方法	串行时间	存储空间
直接法	稠密 Cholesky 分解	$O(N^3)$	$O(N^2)$
	显式求逆	$O(N^2)$	$O(N^2)$
	带状 Cholesky 分解	$O(N^2)$	$O(N^(3/2))$
	稀疏 Cholesky 分解	$O(N^(3/2)))$	$O(N\log N)$
经典迭代	Jacobi	$O(N^3)$	$O(N)$
	Gauss-Seidel	$O(N^3)$	$O(N)$
	SOR	$O(N^(3/2)))$	$O(N)$
	带 Chebyshev 加速的 SSOR	$O(N^(5/4)))$	$O(N)$
Krylov 子空间迭代	CG (共轭梯度法)	$O(N^{3/2})$	$O(N)$
	CG (带修正 IC 预处理)	$O(N^(5/4)))$	$O(N)$
快速算法	FFT(快速 Fourier 变换)	$O(N\log N)$	$O(N)$
	块循环约化	$O(N\log N)$	$O(N)$
	Multigrid	$O(N)$	$O(N)$

2 定常迭代方法

当直接求解方程组 $Ax=b$ 较困难时, 我们可以求解一个近似方程组

$$Mx=b$$

设其解为 $x^{(1)}$. 易知它与真解之间的误差满足

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

如果 $x^{(1)}$ 已经满足精度要求, 则停止计算, 否则需要修正. 设修正量为 Δx. 显然 Δx 满足方程 $A\Delta x=b-Ax^{(1)}$ 但由于直接求解该方程比较困难, 因此我们还是求解近似

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

于是得到修正后的近似解

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

若 $x^{(2)}$ 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, 我们就得到一个序列

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

满足以下递推关系

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

由于每次迭代的格式是一样的, 因此称为 定常迭代.
通常, 构造一个好的定常迭代, 需要考虑以下两点:

- (1) 以 M 为系数矩阵的线性方程组必须要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, 或者迭代序列 x_k 要收敛

下面我们就介绍几个常见的基于矩阵分裂的定常迭代方法

- Jacobi 算法
- Gauss-Seidel 算法
- SOR(Successive Over-Relaxation) 算法
- SSOR(Symmetric SOR) 算法
- AOR(Accelerated over-relaxation) 算法

2.1 矩阵分裂迭代方法

迭代方法的基本思想: 给定一个迭代初始值 $x_{(0)}$, 通过一定的迭代格式生成一个迭代序列 $\left\{x^{(k)}\right\}_{k=0}^{\infty}$, 使得 $\lim_{k\to\infty} x^{(k)} = x_* \triangleq A^{-1}b$

定义(矩阵分裂 Matrix splitting)

设 $A \in \mathbb{R}^{n\times n}$ 非奇异, 称

$$A = M - N$$

为 A 的一个矩阵分裂, 其中 M 非奇异
原方程组等价于 $Mx = Nx + b$. 于是我们就可以构造出以下的迭代格式

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b \triangleq Gx^{(k)} + g \quad , \quad k = 0, 1, \ldots, \tag{6.7}$$

其中 $G = M^{-1}N$ 称为该迭代格式的迭代矩阵

2.2 Jacobi 迭代

将矩阵 A 分裂为

$$A = D - L - U$$

其中 D 为 A 的对角部分,$-L$ 和 $-U$ 分别为 A 的严格下三角和严格上的三角部分
在矩阵分裂 $A = M - N$ 中取 $M = D, N = L + U$,则可得到Jacobi 迭代算法:

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b \quad , \quad k = 0, 1, 2, \ldots \tag{6.8}$$

迭代矩阵为

$$G_1 = D^{-1}(L + U)$$

写成分量形式即为

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

由于 Jacobi 迭代中 $x_i^{(k+1)}$ 的更新顺序与 i 无关，即可以按照顺序 $i = 1, 2, ..., n$ 计算。因此 Jacobi 迭代非常适合并行计算。

算法 2.1求解线性方程组的 Jacobi 迭代方法

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

我们也可以将 Jacobi 迭代格式写成

$$x^{(k+1)} = x^{(k)} + D^{-1}\left(b - Ax^{(k)}\right) = x^{(k)} + D^{-1} r_k \quad , \quad k = 0, 1, 2, \ldots$$

其中 $r_k \triangleq b - Ax^{(k)}$ 是 k 次迭代后的残量。

二维离散 poisson 方程 Jacobi 迭代方法

算法 2.2求解二维离散 poisson 方程的 Jacobi 迭代方法

1：Choose an initial guess $v^{(0)}$
2: while not converge do
3: for $i = 1$ to N do
4: for $j = 1$ to N do
5: $x_i^{(k+1)} = \left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)}\right) / a_{ii}$
6: end for
7: end for
8:end while

2.3 Gauss-Seidel 迭代

取 $M = D - L, N = U$, 即可得 colorblueGauss-Seidel (G-S) 迭代算法：

$$x^{(k+1)} = (D - L)^{-1} U x^{(k)} + (D - L)^{-1} b \tag{6.9}$$

迭代矩阵为

$$G_{GS} = (D - L)^{-1} U$$

将 $G - S$ 迭代改写为

$$Dx^{(k+1)} = Lx^{(k+1)} + Ux^{(k)} + b$$

即可得分量形式

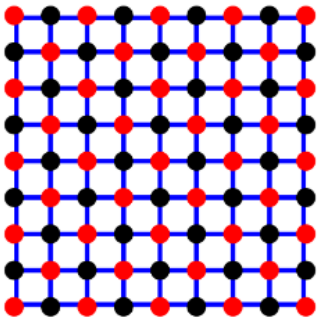
$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \quad i = 1, 2, \ldots, n$$

算法 2.1求解线性方程组的 Jacobi 迭代方法

1:Choose 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

$G-S$ 算法的主要优点是充分利用了已经获得的最新数据。
但由于 $G-S$ 算法中未知量的更新是按自然顺序进行的。因此不适合并行计算。

G-S 算法的并行计算: 红黑排序



下面我们介绍一种适合并行计算的更新顺序: 红黑排序, 即将二维网络点依次做红黑记号, 如右图
在计算过程中, 对未知量的值进行更新时, 我们可以先更新红色节点, 此时所使用的只是黑色节点的数据, 然后再更新黑色节点, 这时使用的是红色节点的数据. 于是我们得到红黑排序 G-S 迭代方法.

算法 2.4求解二维离散 poisson 方程的红黑排序 G-S 迭代方法

1:Choose an initial guess $v^{(0)}$

2: while not converge do

3: for (i,j) 为红色节点 do

4: $u_{i,j}^{(k+1)} = \frac{1}{4}\left(h^2 f_{i,j} + u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}\right)$

5: end for

6: for (i,j) 为黑色节点 do

7: $u_{i,j}^{(k+1)} = \frac{1}{4}\left(h^2 f_{i,j} + u_{i+1,j}^{(k+1)} + u_{i-1,j}^{(k+1)} + u_{i,j+1}^{(k+1)} + u_{i,j-1}^{(k+1)}\right)$

8: end for

9:end while

2.4 SOR 迭代

在 G-S 算法的基础上, 我们可以通过引入一个松弛参数 ω 来加快收敛速度. 这就是 SOR (Successive Overrelaxation) 算法, 即将 G-S 算法中的第 $k+1$ 步近似解与第 k 步近似解做一个加权平均:

$$x^{(k+1)} = (1-\omega)x^{(k)} + \omega\left(D^{-1}\left(Lx^{(k+1)} + Ux^{(k)}\right) + D^{-1}b\right) \tag{6.10}$$

整理后即为

$$x^{(k+1)} = (D-\omega L)^{-1}((1-\omega)D + \omega U)x^{(k)} + \omega(D-\omega L)^{-1}b \tag{6.11}$$

其中 ω 称为松弛参数。
当 $\omega = 1$ 时,SOR 即为 G-S 算法,当 $\omega <$ 时,称为低松弛 (under relaxation)算法,当 $\omega > 1$ 时,称为超松弛 (over relaxation)算法.
SOR 算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。在大多数情况下,当 $\omega > 1$ 时会取得比较好的收敛效果.
SOR 的迭代矩阵为

$$G_{\text{SOR}} = (D-\omega L)^{-1}((1-\omega)D + \omega U)$$

对应的矩阵分裂为

$$M = \frac{1}{\omega}D - L, \quad N = \frac{1-\omega}{\omega}D + U$$

由 (6.11) 可得 SOR 迭代的分量形式为

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

算法 2.5求解线性方程组的 SOR 迭代方法

1:Choose an initial guess $x^{(0)}$

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

SOR 算法最大的优点是引入了松弛参数 ω, 通过选取适当的 ω 可以大大提高算法的收敛速度.
但是 SOR 算法最大的难点就是如何选取最优的参数

算法 2.4求解二维离散 poisson 方程的红黑排序 G-S 迭代方法

1:Choose an initial guess $v^{(0)}$

2: while not converge do

3: for (i,j) 为红色节点 do

4: $u_{i,j}^{(k+1)} = (1-\omega)v_{i,j}^{(k)} + \omega\left(h^2 f_{i,j} + u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}\right)/4$

5: end for

6: for (i,j) 为黑色节点 do

7: $u_{i,j}^{(k+1)} = (1-\omega)v_{i,j}^{(k)} + \omega\left(h^2 f_{i,j} + u_{i+1,j}^{(k+1)} + u_{i-1,j}^{(k+1)} + u_{i,j+1}^{(k+1)} + u_{i,j-1}^{(k+1)}\right)/4$

8: end for

9:end while

2.5 SSOR 迭代方法

将 SOR 算法中的 L 和 U 相交换, 即可得迭代格式

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

将这个迭代格式与 SOR 相结合, 就可以得到下面的两步迭代方法

$$\begin{cases} x^{(k+\frac{1}{2})} = (D-\omega L)^{-1}[(1-\omega)D + \omega U]x^{(k)} + \omega(D-\omega L)^{-1}b \\ x^{(k+1)} = (D-\omega U)^{-1}[(1-\omega)D + \omega L]x^{(k+\frac{1}{2})} + \omega(D-\omega U)^{-1}b \end{cases}$$

这就是SSOR 迭代(对称超松弛) 算法, 相当于将 L 与 U 同等看待, 交替做两次 SOR 迭代.
消去中间迭代量 $x^{(k+\frac{1}{2})}$, 可得

$$x^{(k+1)} = G_{\text{SSOR}}x^{(k)} + g$$

其中迭代矩阵

$$G_{\text{SSOR}} = (D-\omega U)^{-1}[(1-\omega)D + \omega L](D-\omega L)^{-1}[(1-\omega)D + \omega U]$$

对应的矩阵分裂为

$$\begin{aligned} M &= \frac{1}{\omega(2-\omega)}\left[D-\omega(L+U)+\omega^2 LD^{-1}U\right] \\ &= \frac{1}{\omega(2-\omega)}(D-\omega L)D^{-1}(D-\omega U) \\ N &= \frac{1}{\omega(2-\omega)}[(1-\omega)D+\omega L]D^{-1}[(1-\omega)D+\omega U] \end{aligned}$$

对于某些特殊问题, SOR 算法不收敛, 但仍然可能构造出收敛的 SSOR 算法.
一般来说, SOR 算法的渐进收敛速度对参数 ω 比较敏感, 但 SSOR 对参数 ω 不太敏感. (Poisson SOR omega.m,Poisson SSOR omega.m))

2.6 AOR 迭代

Hadjidimos 于 1978 年提出了 AOR (Accelerated over-relaxation, 快速松弛) 算法, 迭代矩阵为

$$G_{\mathrm{AOR}} = (D-\gamma L)^{-1}[(1-\omega)D+(\omega-\gamma)L+\omega U]$$

其中 γ 和 ω 为松弛参数. 对应的矩阵分解为

$$M = \frac{1}{\omega}(D-\gamma L), \quad N = \frac{1}{\omega}[(1-\omega)D+(\omega-\gamma)L+\omega U]$$

(1) 当 $\gamma=\omega$ 时, AOR 算法即为 SOR 算法;

(2) 当 $\gamma=\omega=1$ 时, AOR 算法即为 G-S 算法;

(3) 当 $\gamma=0,\omega=1$ 时, AOR 算法即为 Jacobi 算法

与 SSOR 类似, 我们也可以定义 SAOR 算法.

2.7 Richardson 算法

Richardson 算法是一类形式非常简单的算法, 其迭代格式为

$$x^{(k+1)} = x^{(k)} + \omega\left(b-Ax^{(k)}\right), \quad k=0,1,2,\ldots$$

对应的矩阵分裂和迭代矩阵分别为

$$M = \frac{1}{\omega}I, \quad N = \frac{1}{\omega}I - A, \quad G_{\mathrm{R}} = I-\omega A$$

如果在每次迭代时取不同的参数, 即

$$x^{(k+1)} = x^{(k)} + \omega_k\left(b-Ax^{(k)}\right), \quad k=0,1,2,\ldots$$

则称为 nonstationary Richardson 算法. 定理设 $A\in\mathbb{R}^{n\times n}$ 是对称正定矩阵,λ_1 和 λ_n 分别是 A 的最大和最小特征值, 则 Richardson 算法收敛当且仅当

$$0<\omega<\frac{1}{\lambda_1}$$

最优参数为

$$\omega_* = \arg\min_{\omega} \rho\left(G_{\mathrm{R}}\right) = \frac{2}{\lambda_1 + \lambda_n}$$

即当 $\omega = \omega_*$ 时, 迭代矩阵的谱半径达到最小, 且有

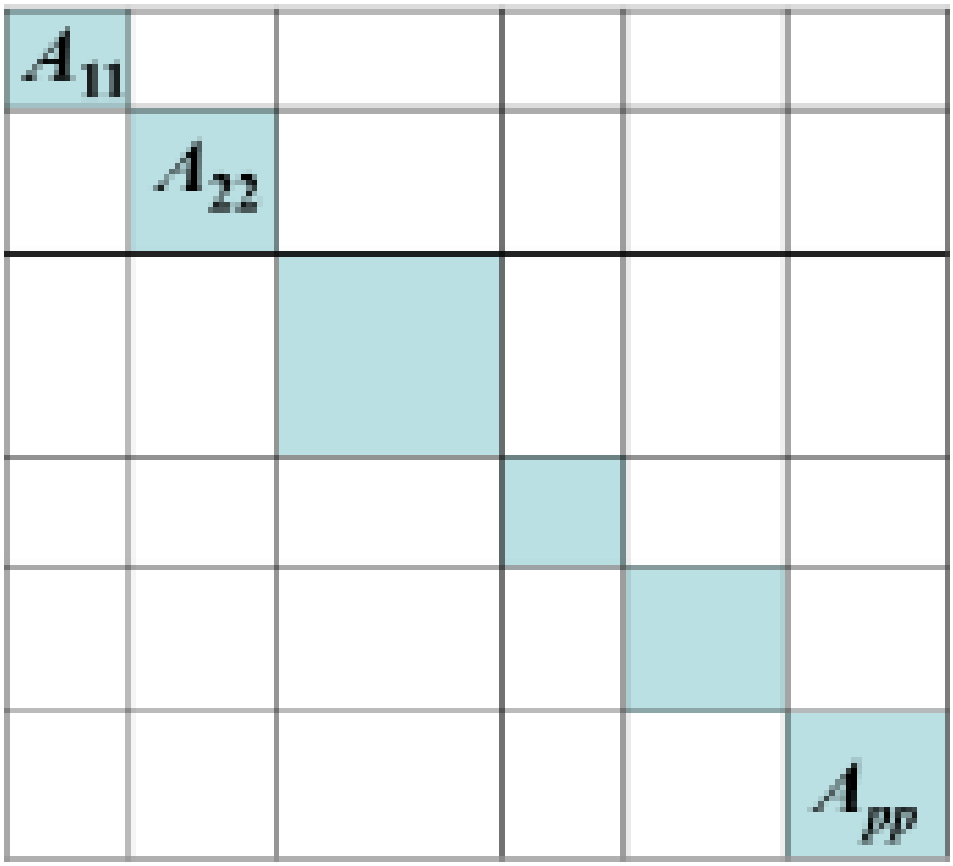
$$\rho\left(G_{\mathrm{R}}\right) = \begin{cases} 1 - \omega\lambda_n & \text{if } \omega \leq \omega \\ \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n} = \frac{\kappa(A)-1}{\kappa(A)+1} & \text{if } \omega = \omega \\ \omega\lambda_1 - 1 & \text{if } \omega \geq \omega \end{cases}$$

2.8 分块迭代方法

前面介绍的迭代方法可以推广到分块情形. 将 A 写成如下的分块形式

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{PP} \end{bmatrix}$$

设 $A = D - L - U$, 其中 $D, -L, -U$ 分别是 A 的快对角，块严格下三角矩阵和块严格上三角矩阵。则相应的分块 Jacobi, 分块 Gauss-Seidel 和分块 SOR 算法分别为



- 分块 Jacobi 迭代

$$A_{ii}\boldsymbol{x}_i^{(k+1)} = \boldsymbol{b}_i - \sum_{j=1,j\neq i}^{p} A_{ij}\boldsymbol{x}_j^{(k)}, \quad i = 1, 2, \ldots, p$$

- 分块 Gauss-seidel 迭代

$$A_{ii}\boldsymbol{x}_i^{(k+1)} = \boldsymbol{b}_i - \sum_{j=1}^{i-1} A_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j=i+1}^{p} A_{ij}\boldsymbol{x}_j^{(k)}, \quad i = 1, 2, \ldots, p$$

- 分块 SOR 迭代

$$\boldsymbol{x}_i^{(k+1)} = (1-\omega)\boldsymbol{x}_i^{(k)} + \omega A_{ii}^{-1}\left(\boldsymbol{b}_i - \sum_{j=1}^{i-1} A_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j=i+1}^{p} A_{ij}\boldsymbol{x}_j^{(k)}\right)$$
$$i = 1, 2, \ldots, p$$

3 收敛性分析

3.1 定常迭代方法的收敛性

[1]

参考文献

[1] TKH Tam and Cecil G Armstrong. 2d finite element mesh generation by medial axis subdivision. *Advances in engineering software and workstations*, 13(5):313–324, 1991.