第三讲　预处理方法

目录

Nothing will be more central to computational science in the next century than the art of transforming a problem that appears intractable into another whose solution can be approximated rapidly. For Krylov subspace matrix iterations, this is **preconditioning**.

— Trefethen & Bau, 1997.

Lloyd N. Trefethen

Professor of University of Oxford, Head of Oxford's Numerical Analysis Group

- First customer to buy MATLAB
- Royal Society (英国皇家学会), National Academy of Engineering (美国国家工程院), Academia Europaea (欧洲人文与自然科学院)
- Fellow of SIAM and AMS, President of SIAM
- IMA Gold Medal, LMS Naylor Prize, SIAM Pólya Prize
- SIAM John von Neumann Prize
- Invited speaker at ICIAM, ICM, and ECM congresses
- Author of SIAM's all-time bestseller
- Winner of teaching prizes at MIT, Cornell and Oxford
- Winner of the first Fox Prize in Numerical Analysis
- Inventor of Chebfun

L.N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, 1997.

为什么要预处理

工程中出现的大规模线性方程组往往是病态的, 对数值求解带来很大的困难:

- 使得迭代法 (比如 Krylov 子空间迭代法) 收敛变得非常缓慢
- 对数值解的精度产生很大的影响 (在有限精度计算情形下)

对于第一个问题, 当前的有效处理方法是 **预处理**, 预处理是当前公认的能有效改善迭代法 (特别是 Krylov 子空间迭代法) 收敛性质的有效手段.

什么是预处理

通俗地说, 预处理就是将**难以求解**的问题转化成**等价**的**容易求解**的新问题

- 对于线性方程组而言, 预处理就是对 (病态) 系数矩阵进行适当的线性变换, 转换为一个 (良态) 新矩阵, 从而达到改善迭代法收敛性的目的.
- 预处理方法的研究是当前科学计算领域中的重要研究课题和热门课题.

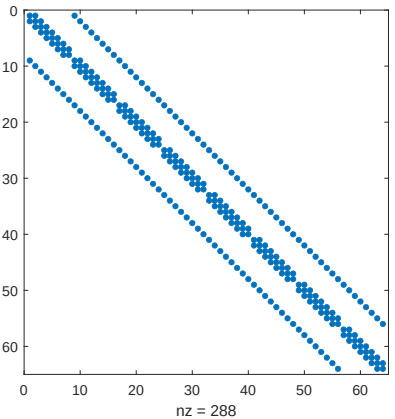
预处理举例: 二维 Poisson 方程

$$
\begin{aligned}
-\Delta u(x,y) &= -\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) = f(x,y),\\
u(x,y) &= 0, \quad (x,y)\in\partial\Omega, \quad \Omega \triangleq [0,1]\times[0,1].
\end{aligned}
$$

- 用五点差分离散, 得代数方程组

$$(I\otimes T + T\otimes I)u = h^2 f,$$

其中 $T = \mathrm{tridiag}(-1, 2, -1)$.



不同预处理效果

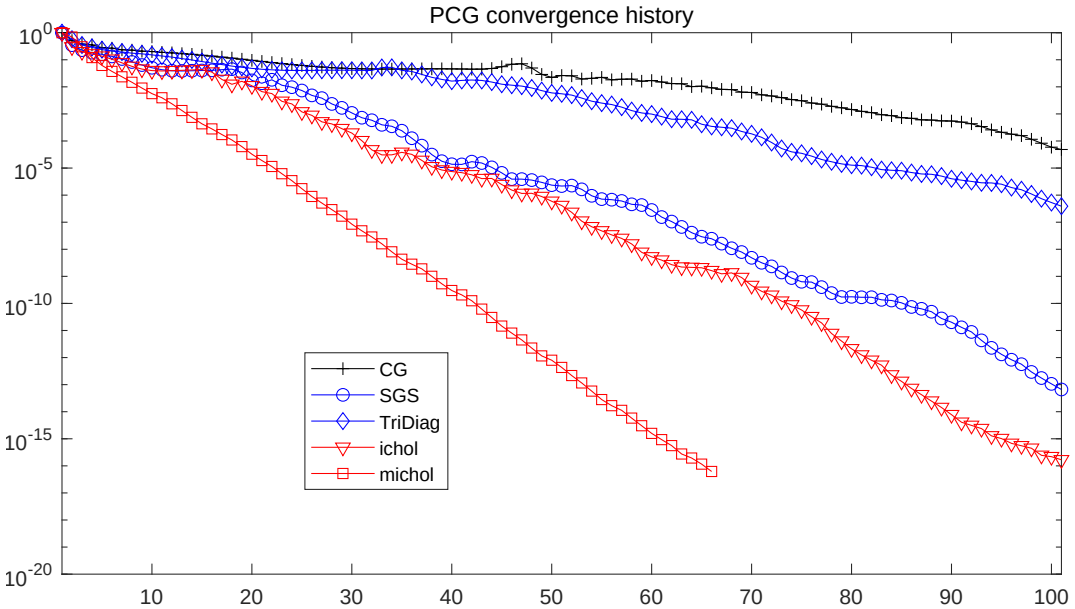
$$P_1 = (D - L)D^{-1}(D - L^{\mathsf{T}})$$

$$P_2 = \mathrm{TriDiag}(A)$$

$$P_3 = \mathrm{ichol}(A)$$

$$P_4 = \mathrm{michol}(A)$$

n	8^2	16^2	32^2	64^2
$\kappa(A)$	32.2	166	441	1712
$\kappa(P_1^{-1}A)$	4.89	15.5	56.3	216
$\kappa(P_2^{-1}A)$	16.6	58.7	221	856
$\kappa(P_3^{-1}A)$	3.69	11.1	39.8	152
$\kappa(P_4^{-1}A)$	3.00	6.65	16.5	43.5



Performance of different preconditioners for 2D Poisson with $n = 64^2$

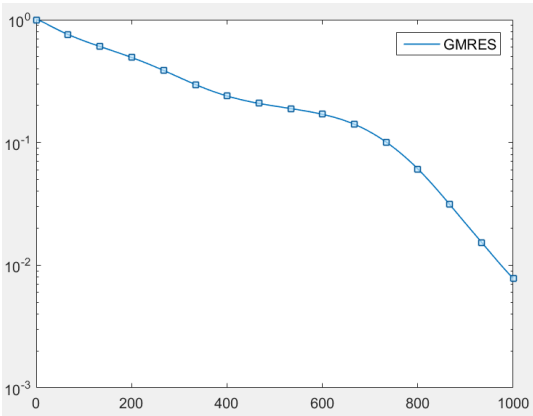
预处理举例: 分数阶扩散方程

$$\begin{cases} -\mathrm{D}\left[K(x)\left(\theta\,{}_0\mathrm{D}_x^{1-\beta} - (1-\theta)\,{}_x\mathrm{D}_1^{1-\beta}\right)u(x)\right] = f(x), \\ x \in [0,1], \quad K(x) = 1 + 8x, \quad \beta = 0.5, \quad \theta = 0.5. \end{cases}$$

Condition number

n	$\mathrm{cond}(A)$
2^8	5191
2^9	14813
2^{10}	42143
2^{11}	119653
2^{12}	339259
2^{13}	961081

Convergence history for $N = 4096$

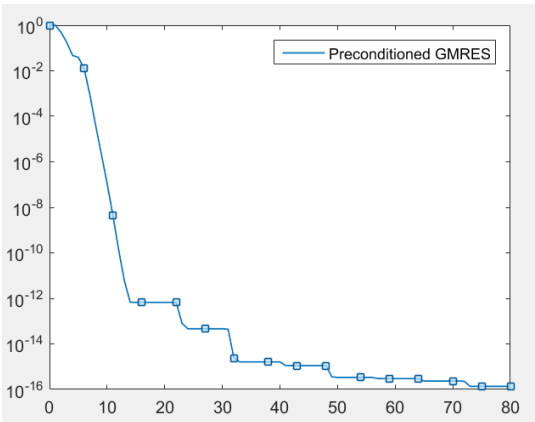


简单循环预处理子

Condition number

n	$\mathrm{cond}(A)$	$\mathrm{cond}(P^{-1}A)$
2^8	5191	76
2^9	14813	112
2^{10}	42143	162
2^{11}	119653	235
2^{12}	339259	337
2^{13}	961081	482

Convergence history after preconditioning



关于预处理方法的相关参考资料

- Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition, 2003.
- Chen, *Matrix Preconditioning Techniques and Applications*, 2005.
- Málek and Z. Strakoš, *Preconditioning and the Conjugate Gradient Method in The Context of Solving PDEs*, 2015.
- Bertaccini and Durastante, *Iterative Methods and Preconditioning for Large and Sparse Linear Systems With Applications*, 2018.
- **谷同祥等, 迭代方法和预处理技术 (下), 科学出版社**, 2015.
  - ▷ Benzi, Preconditioning techniques for large linear systems: A survey, JCP, 2002, 418–477.
  - ▷ Mardal and Winther, Preconditioning discretizations of systems of partial differential equations, NLAA, 2011, 1–40.
  - ▷ Wathen, Preconditioning, Acta Numerica, 2015, 329–376.
  - ▷ Pearson and Pestana, Preconditioners for Krylov subspace methods: An overview, GAMM, 2020.

3-1 预处理 Krylov 子空间方法

3.1 预处理 Krylov 子空间方法

线性方程组的预处理

考虑线性方程组

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

在方程组的两边同时左乘一个非奇异矩阵 $P \in \mathbb{R}^{n\times n}$ 的逆, 则可得

$$\boxed{P^{-1}Ax = P^{-1}b} \tag{3.1}$$

- 这个新方程组就是 **预处理后的方程组**
- P 就称为 **预处理子** (**preconditioner**)
- 这个过程就是对原线性方程组 **做预处理**
- 当 Krylov 子空间方法被用于求解 (3.1) 时, 就称为 **预处理 Krylov 子空间方法**

预处理子怎么选取

预处理子选取基本准则

一个好的预处理子 P 通常需满足下面两个要求:

(1) $P^{-1}A$ 具有更小的条件数和 (或) 更好的特征值分布;

(2) 线性方程组 $Pz = r$ 容易求解, 即预处理子 P 的使用成本低廉.

- 第一条是为了确保预处理后的线性方程组更容易求解, 即预处理子有效
- 第二条是因为在用 Krylov 子空间方法求解预处理后的方程组时, 每步迭代都需要额外求解一个以 P 为系数矩阵的线性方程组, 为了不增加太多额外的运算成本, 必须很容易求解.

预处理方式

- **左预处理** $P^{-1}Ax = P^{-1}b$
- **右预处理** $AP^{-1}u = b, \qquad x = P^{-1}u$
- **两边预处理**: 同时进行左预处理和右预处理

$$L^{-1}AR^{-1}u = L^{-1}b, \qquad x = R^{-1}u$$

对应的预处理子为 $P = LR$

三种预处理方式的选择

三种预处理方式的迭代矩阵

$$P^{-1}A, \quad AP^{-1}, \quad L^{-1}AR^{-1}$$

具有相同的特征值分布

- 若 A 和 P 都对称正定, 则三种方式的 PCG 效果基本一样
- 若 A 非对称 (非正规) 情形, 则预处理方式对 Krylov 子空间方法有不同影响

注记

在实际使用中, 该选取哪种预处理方式, 需要根据问题本身和所用的方法来确定. 比如 GMRES 方法, 建议右预处理 (右预处理极小化的是原始残量的范数).

3-1-1 预处理 CG 方法

设 $A \in \mathbb{R}^{n\times n}$ 对称正定, 要求预处理子 P 也对称正定.

考虑使用 **两边预处理**方式: 设 $P = LL^{\mathsf{T}}$

$$L^{-1}A(L^{\mathsf{T}})^{-1}u = L^{-1}b, \quad x = (L^{\mathsf{T}})^{-1}u$$

- 用 CG 方法求解, 迭代 k 步后的近似解记为 $u^{(k)}$ $\rightarrow x^{(k)} = (L^{\mathsf{T}})^{-1}u^{(k)}$
- 记 $\tilde{r}_k \triangleq L^{-1}b - L^{-1}A(L^{\mathsf{T}})^{-1}u^{(k)}$ $\rightarrow$ 预处理后的残量

CG 基本框架: $Ax = b$

1: **给定初值** $x^{(0)}$
2: **计算** $r_0 = b - Ax^{(0)}$
3: $p_1 = r_0$
4: **for** $k = 1, 2, \ldots$ **do**
5: $\quad \xi_k = \dfrac{(r_{k-1}, r_{k-1})}{(Ap_k, p_k)}$
6: $\quad x^{(k)} = x^{(k-1)} + \xi_k p_k$
7: $\quad r_k = r_{k-1} - \xi_k A p_k$
8: $\quad \mu_k = \dfrac{(r_k, r_k)}{(r_{k-1}, r_{k-1})}$
9: $\quad p_{k+1} = r_k + \mu_k p_k$
10: **end for**

PCG: $L^{-1}AL^{-\mathsf{T}}u = L^{-1}b$

1: **给定初值** $x^{(0)}$
2: **计算** $r_0 = b - Ax^{(0)}$
3: $\tilde{r}_0 = L^{-1}r_0, \quad \tilde{p}_1 = \tilde{r}_0$
4: **for** $k = 1, 2, \ldots$ **do**
5: $\quad \xi_k = \dfrac{(\tilde{r}_{k-1}, \tilde{r}_{k-1})}{(L^{-1}AL^{-\mathsf{T}}\tilde{p}_k, \tilde{p}_k)}$
6: $\quad u^{(k)} = u^{(k-1)} + \xi_k \tilde{p}_k$
7: $\quad \tilde{r}_k = \tilde{r}_{k-1} - \xi_k L^{-1}AL^{-\mathsf{T}}\tilde{p}_k$
8: $\quad \mu_k = \dfrac{(\tilde{r}_k, \tilde{r}_k)}{(\tilde{r}_{k-1}, \tilde{r}_{k-1})}$
9: $\quad \tilde{p}_{k+1} = \tilde{r}_k + \mu_k \tilde{p}_k$
10: **end for**

算法得到的是 $u^{(k)}$ 和 $\tilde{r}_k$, 因此需要考虑原方程的近似解和残量.

原方程的解 $x^{(k)} = L^{-\mathsf{T}} u^{(k)}$ 和残量 $r_k = L\tilde{r}_k$

对迭代公式进行适当的改写, 引入一个辅助变量 $p_k \triangleq L^{-\mathsf{T}}\tilde{p}_k$

$$\Longrightarrow \quad p_{k+1} = L^{-\mathsf{T}} p_{k+1} = L^{-\mathsf{T}}\tilde{r}_k + \mu_k L^{-\mathsf{T}}\tilde{p}_k = L^{-\mathsf{T}}\tilde{r}_k + \mu_k p_k$$

$$\begin{aligned}
r_k &= L\tilde{r}_k = L(\tilde{r}_{k-1} - \xi_k L^{-1} A L^{-\mathsf{T}}\tilde{p}_k) = r_{k-1} - \xi_k A p_k, \\
p_{k+1} &= L^{-\mathsf{T}}\tilde{r}_k + \mu_k p_k = L^{-\mathsf{T}} L^{-1} r_k + \mu_k p_k = P^{-1} r_k + \mu_k p_k, \\
x^{(k)} &= L^{-\mathsf{T}} u^{(k)} = L^{-\mathsf{T}}(u^{(k-1)} + \xi_k \tilde{p}_k) = x^{(k-1)} + \xi_k p_k,
\end{aligned}$$

$$\xi_k = \frac{(\tilde{r}_{k-1}, \tilde{r}_{k-1})}{(L^{-1}AL^{-\mathsf{T}}\tilde{p}_k, \tilde{p}_k)} = \frac{(L^{-1}r_{k-1}, L^{-1}r_{k-1})}{(L^{-1}AL^{-\mathsf{T}}\tilde{p}_k, \tilde{p}_k)} = \frac{(r_{k-1}, L^{-\mathsf{T}}L^{-1}r_{k-1})}{(AL^{-\mathsf{T}}\tilde{p}_k, (L^{-\mathsf{T}}\tilde{p}_k)} = \frac{(r_{k-1}, P^{-1}r_{k-1})}{(Ap_k, p_k)}$$

$$\mu_k = \frac{(\tilde{r}_k, \tilde{r}_k)}{(\tilde{r}_{k-1}, \tilde{r}_{k-1})} = \frac{(L^{-1}r_k, L^{-1}r_k)}{(L^{-1}r_{k-1}, L^{-1}r_{k-1})} = \frac{(r_k, P^{-1}r_k)}{(r_{k-1}, P^{-1}r_{k-1})}. \qquad (\text{记 } z_k \triangleq P^{-1}r_k)$$

算法 预处理 CG (PCG) 算法

1: 给定初值 $x^{(0)}$, (相对) 精度要求 $\varepsilon > 0$ 和最大迭代步数 IterMax
2: 计算 $r_0 = b - Ax^{(0)}$ 和 $\beta = \|r_0\|_2$
3: $z_0 = P^{-1} r_0$, $p_1 = z_0$
4: 计算 $\rho = r_0^{\mathsf{T}} z_0$
5: **for** $k = 1$ to IterMax **do**
6: $\quad q_k = Ap_k, \quad \xi_k = \rho/(p_k^{\mathsf{T}} q_k)$
7: $\quad x^{(k)} = x^{(k-1)} + \xi_k p_k$
8: $\quad r_k = r_{k-1} - \xi_k q_k$
9: $\quad$ relres $= \|r_k\|_2/\beta$
10: $\quad$ **If** relres$< \varepsilon$ **then** break **end if** % 收敛性检测
11: $\quad \rho_0 = \rho$
12: $\quad z_k = P^{-1} r_k, \quad \rho = r_k^{\mathsf{T}} z_k$
13: $\quad \mu_k = \rho/\rho_0$
14: $\quad p_{k+1} = z_k + \mu_k p_k$
15: **end for**

几点说明

- 矩阵 L 并没有出现在算法中, 这意味着并不需要对 P 做 Cholesky 分解
- 在 PCG 算法中, 每步迭代都需要计算 $z_k = P^{-1}r_k$, 一般情况下, 都是通过求解方程组 $Pz_k = r_k$ 来实现.

左预处理 CG 方法

PCG 也可以从左预处理方式导出:

$$P^{-1}Ax = P^{-1}b$$

定义 P-内积 (P 对称正定):

$$(x, y)_P \triangleq (Px, y) = (x, Py)$$

则 $P^{-1}A$ 关于 P-内积对称正定:

$$\begin{cases} (P^{-1}Ax, y)_P = (Ax, y) = (x, Ay) = \big(x, P(P^{-1}Ay)\big) = (x, P^{-1}Ay)_P \\ (P^{-1}Ax, x)_P = (Ax, x) > 0, \quad \forall\ x \neq 0 \end{cases}$$

记 $z_k \triangleq P^{-1}r_k$, 将 CG 中的内积改为 P-内积 ⟶ PCG

CG 基本框架: $Ax = b$

1: 给定初值 $x^{(0)}$
2: 计算 $r_0 = b - Ax^{(0)}$
3: $p_1 = r_0$
4: **for** $k = 1, 2, \ldots$ **do**
5: $\quad \xi_k = \dfrac{(r_{k-1}, r_{k-1})}{(Ap_k, p_k)}$
6: $\quad x^{(k)} = x^{(k-1)} + \xi_k p_k$
7: $\quad r_k = r_{k-1} - \xi_k A p_k$
8: $\quad \mu_k = \dfrac{(r_k, r_k)}{(r_{k-1}, r_{k-1})}$
9: $\quad p_{k+1} = r_k + \mu_k p_k$
10: **end for**

PCG: $P^{-1}Ax = b$, P-内积

1: 给定初值 $x^{(0)}$
2: 计算 $r_0 = b - P^{-1}Ax^{(0)}$
3: $p_1 = r_0$
4: **for** $k = 1, 2, \ldots$ **do**
5: $\quad \xi_k = \dfrac{(r_{k-1}, r_{k-1})_P}{(P^{-1}Ap_k, p_k)_P} = \dfrac{(r_{k-1}, z_{k-1})}{(Ap_k, p_k)}$
6: $\quad x^{(k)} = x^{(k-1)} + \xi_k p_k$
7: $\quad r_k = r_{k-1} - \xi_k P^{-1} A p_k$
8: $\quad \mu_k = \dfrac{(r_k, r_k)_P}{(r_{k-1}, r_{k-1})_P} = \dfrac{(r_k, z_k)}{(r_{k-1}, z_{k-1})}$
9: $\quad p_{k+1} = r_k + \mu_k p_k$
10: **end for**

- 不难看出, 两边预处理 CG 算法和左预处理 CG 算法是**完全一样**的.
- 类似地, 也可以从右预处理方式来推导 PCG 方法, 留作练习.

3-1-2 预处理 GMRES 方法

对于非对称 Krylov 子空间方法, 也有三种预处理方式.
但与对称正定情形不同的是, 这三种方式并不等价, 而且有时效果会相差很大.

左预处理

$$P^{-1}Ax = P^{-1}b$$

- $r_k = b - Ax^{(k)}$ $\quad \rightarrow$ 原始残量
- $\tilde{r}_k = P^{-1}r_k$ $\quad \rightarrow$ 预处理后的残量

将 GMRES 作用于预处理后的方程组即可得左预处理 GMRES 方法.

算法 左预处理 GMRES 方法

1: **给定初值** $x^{(0)}$ **和 (相对) 精度要求** $\varepsilon > 0$

2: **计算** $\tilde{r}_0 = P^{-1}(b - Ax^{(0)})$ **和** $\beta = \|\tilde{r}_0\|_2$

3: **令** $v_1 = \tilde{r}_0/\beta$, $\xi = \beta e_1$

4: **for** $j = 1, 2, \ldots$ **do**

5: $\quad \tilde{w}_j = Av_j$

6: $\quad w_j = P^{-1}\tilde{w}_j$ % Apply preconditioning

7: $\quad$ **for** $i = 1, 2, \ldots, j$ **do**

8: $\quad\quad h_{ij} = (w_j, v_i), \quad w_j = w_j - h_{ij}v_i$

9: $\quad$ **end for**

10: $\quad h_{j+1,j} = \|w_j\|_2$

11: $\quad$ **for** $i = 1, 2, \ldots, j-1$ **do**

12: $\quad\quad \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix}$

13: $\quad$ **end for**

14: **If** $h_{j+1,j} = 0$ **then** set $k = j$ and break **end if**

15: $v_{j+1} = w_j / h_{j+1,j}$

16: **if** $|h_{j,j}| > |h_{j+1,j}|$ **then**

17: $c_j = \frac{1}{\sqrt{1+\tau^2}}$, $s_j = c_j\tau$ where $\tau = \frac{h_{j+1,j}}{h_{j,j}}$

18: **else**

19: $s_j = \frac{1}{\sqrt{1+\tau^2}}$, $c_j = s_j\tau$ where $\tau = \frac{h_{j,j}}{h_{j+1,j}}$

20: **end if**

21: $h_{j,j} = c_j h_{j,j} + s_j h_{j+1,j}$, $h_{j+1,j} = 0$

22: $$\begin{bmatrix} \xi_j \\ \xi_{j+1} \end{bmatrix} = \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} \begin{bmatrix} \xi_j \\ 0 \end{bmatrix}$$

23: **If** $|\xi_{j+1}|/\beta < \varepsilon$ **then** set $k = j$ and break **end if**

24: **end for**

25: **计算** $y^{(k)} = R_k^{-1}\xi^{(k)}$, **其中** $R_k = H(1:k, 1:k)$, $\xi^{(k)} = \xi(1:k)$

26: **计算近似解** $x^{(k)} = x^{(0)} + V_k y^{(k)}$

注记

- 在左预处理 GMRES 方法中, 停机准则中的 $|\xi_{j+1}|$ 不是 r_k 的模, 而是 $\tilde{r}_k$ 的模.
- 如果要计算真实残量, 除了显式计算外, 没有其它更好的办法.
  所以, 如果停机准则是基于真实残量的, 需要额外去计算 r_k.

右预处理方式

$$AP^{-1}u = b, \quad x = P^{-1}u$$

- 设 $u^{(k)} = u^{(0)} + V_k y_k$ 是迭代 k 步后的近似解, 则原方程组 近似解和残量

$$x^{(k)} = P^{-1}u^{(k)} = P^{-1}(u^{(0)} + V_k y_k) = x^{(0)} + P^{-1}V_k y_k$$

$$r_k = b - Ax^{(k)} = r_0 - AP^{-1}V_k y_k = V_{k+1}(\beta e_1 - H_{k+1,k}y_k)$$

- 对应的最小二乘问题

$$\min_{y\in\mathbb{R}^k} \|\beta e_1 - H_{k+1,k}y\|_2 = \beta|q_1(k+1)| = \|r_k\|_2$$

右预处理后的残量是原方程的残量

算法 右预处理 GMRES 算法

1: **给定初值** $x^{(0)}$ **和 (相对) 精度要求** $\varepsilon > 0$

2: **计算** $r_0 = b - Ax^{(0)}$ **和** $\beta = \|r_0\|_2$

3: **令** $v_1 = \tilde{r}_0/\beta$, $\xi = \beta e_1$

4: **for** $j = 1, 2, \ldots$ **do**

5: $\quad \tilde{w}_j = P^{-1} v_j$ % Apply preconditioning

6: $\quad w_j = A\tilde{w}_j$

7: $\quad$ **for** $i = 1, 2, \ldots, j$ **do**

8: $\quad\quad h_{ij} = (w_j, v_i), \quad w_j = w_j - h_{ij} v_i$

9: $\quad$ **end for**

10: $\quad h_{j+1,j} = \|w_j\|_2$

11: $\quad$ **for** $i = 1, 2, \ldots, j-1$ **do**

12: $$\begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix}$$

13: **end for**

14: **If** $h_{j+1,j} = 0$ **then** set $k = j$ and break **end if**

15: $v_{j+1} = w_j / h_{j+1,j}$

16: **if** $|h_{j,j}| > |h_{j+1,j}|$ **then**

17: $c_j = \frac{1}{\sqrt{1+\tau^2}}$, $s_j = c_j\tau$ where $\tau = \frac{h_{j+1,j}}{h_{j,j}}$

18: **else**

19: $s_j = \frac{1}{\sqrt{1+\tau^2}}$, $c_j = s_j\tau$ where $\tau = \frac{h_{j,j}}{h_{j+1,j}}$

20: **end if**

21: $h_{j,j} = c_j h_{j,j} + s_j h_{j+1,j}$, $h_{j+1,j} = 0$

22: $$\begin{bmatrix} \xi_j \\ \xi_{j+1} \end{bmatrix} = \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} \begin{bmatrix} \xi_j \\ 0 \end{bmatrix}$$

23: **If** $|\xi_{j+1}|/\beta < \varepsilon$ **then** set $k = j$ and break **end if**

24: **end for**

25: **计算** $y^{(k)} = R_k^{-1}\xi^{(k)}$, **其中** $R_k = H(1:k, 1:k)$, $\xi^{(k)} = \xi(1:k)$

26: **计算近似解** $x^{(k)} = x^{(0)} + P^{-1}\, V_k y^{(k)}$

两边预处理方式

如果预处理子 P 是由乘积形式给出: $P = P_l P_r$, 则可构造两边预处理方程组

$$P_l^{-1} A P_r^{-1} u = P_l^{-1} b, \quad x = P_r^{-1} u$$

将 GMRES 用于求解上述线性方程组, 则可得到两边预处理 GMRES 方法.

注记

- 两边预处理后的残量为 $\tilde{r}_k = P_l^{-1}(b - Ax^{(k)})$
- 左预处理和两边预处理所得残量并不是原方程组的真实残量, 因此都可能会导致迭代法提前终止或延迟终止, 特别是当预处理子 P 的条件数较大时.

左、右预处理 GMRES 的最优性质

记 $x_L^{(k)}$ 和 $x_R^{(k)}$ 分别是左预处理 GMRES 和右预处理 GMRES 迭代 k 步后的近似解

- 对于左预处理 GMRES 有

$$x_L^{(k)} = \underset{x\in x^{(0)}+\mathcal{K}_k(P^{-1}A,P^{-1}r_0)}{\operatorname{argmin}} \|P^{-1}(b-Ax)\|_2$$

- 对于右预处理 GMRES 有

$$\begin{aligned} u^{(k)} &= \underset{u\in u^{(0)}+\mathcal{K}_k(AP^{-1},r_0)}{\operatorname{argmin}} \|b-AP^{-1}u\|_2 \\ \Longleftrightarrow\ x_R^{(k)} &= \underset{x\in x^{(0)}+P^{-1}\mathcal{K}_k(AP^{-1},r_0)}{\operatorname{argmin}} \|b-Ax\|_2 \end{aligned}$$

$$
\begin{aligned}
P^{-1}\mathcal{K}_k(AP^{-1}, r_0) &= P^{-1}\,\mathrm{span}\{r_0, AP^{-1}r_0, (AP^{-1})^2 r_0, \ldots, (AP^{-1})^{k-1} r_0\} \\
&= \mathrm{span}\{P^{-1}r_0, P^{-1}AP^{-1}r_0, P^{-1}(AP^{-1})^2 r_0, \ldots, P^{-1}(AP^{-1})^{k-1} r_0\} \\
&= \mathrm{span}\{P^{-1}r_0, (P^{-1}A)P^{-1}r_0, (P^{-1}A)^2 P^{-1}r_0, \ldots, (P^{-1}A)^{k-1} P^{-1}r_0\} \\
&= \mathcal{K}_k(P^{-1}A, P^{-1}r_0)
\end{aligned}
$$

$x_L^{(k)}$ 和 $x_R^{(k)}$ 属于同一个仿射空间

定理 设 $x_L^{(k)}$ 和 $x_R^{(k)}$ 分别是左预处理 GMRES 和右预处理 GMRES 迭代 k 步后得到的近似解, 则 $x_L^{(k)}$ 是 $\|P^{-1}(b - Ax)\|_2$ 在 $x^{(0)} + \mathcal{K}_k(P^{-1}A, P^{-1}r_0)$ 的极小值点, 而 $x_R^{(k)}$ 则是 $\|b - Ax\|_2$ 在同一个仿射空间中的极小值点.

3-1-3 Flexible GMRES 方法

基本思想

在前面的讨论中, 我们假定预处理子 P 是固定不变的. 事实上, 我们可以在每步迭代中取不同的预处理子, 这就是 **Flexible GMRES** (**FGMRES**) 方法.

- 与右预处理 GMRES 区别: 将算法 3.3 中的第 5 步改为下面的语句

5': solve $z_j = P_j^{-1} v_j$

- 近似解表达式: $x^{(k)} = x^{(0)} + \sum_{j=1}^{k} (P_j^{-1} v_j)\tilde{y}_j = x^{(0)} + Z_k \tilde{y}$, $Z_k = [z_1, z_2, \ldots, z_k]$

算法 Flexible GMRES (FGMRES)

1: **给定初值** $x^{(0)}$ **和 (相对) 精度要求** $\varepsilon > 0$

2: **计算** $r_0 = b - Ax^{(0)}$ **和** $\beta = \|r_0\|_2$

3: **令** $\xi = \beta e_1$, $v_1 = r_0/\beta$

4: **for** $j = 1, 2, \ldots$ **do**

5: $z_j = P_j^{-1} v_j$ % Apply the preconditioner

6: $w_j = Az_j$

7: **for** $i = 1, 2, \ldots, j$ **do**

8: $h_{ij} = (w_j, v_i), \quad w_j = w_j - h_{ij} v_i$

9: **end for**

10: $h_{j+1,j} = \|w_j\|_2$

11: **for** $i = 1, 2, \ldots, j-1$ **do**

12: $$\begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix}$$

13: **end for**

14: **If** $h_{j+1,j} = 0$ **then** set $k = j$ and break **end if**

15: $v_{j+1} = w_j / h_{j+1,j}$

16: **if** $|h_{j,j}| > |h_{j+1,j}|$ **then**

17: set $c_j = \frac{1}{\sqrt{1+\tau^2}}$, $s_j = c_j\tau$ where $\tau = \frac{h_{j+1,j}}{h_{j,j}}$

18: **else**

19: set $s_j = \frac{1}{\sqrt{1+\tau^2}}$, $c_j = s_j\tau$ where $\tau = \frac{h_{j,j}}{h_{j+1,j}}$

20: **end if**

21: $h_{j,j} = c_j h_{j,j} + s_j h_{j+1,j}$, $h_{j+1,j} = 0$

22: $$\begin{bmatrix} \xi_j \\ \xi_{j+1} \end{bmatrix} = \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} \begin{bmatrix} \xi_j \\ 0 \end{bmatrix}$$

23: **if** $\|\xi_{j+1}\|_2 < \varepsilon$ **then**

24: set $k = j$ and break

25: **end if**

26: **end for**

27: 计算 $\tilde{y} = R_k^{-1}\xi_k$, 其中 $R_k = H(1:k, 1:k)$, $\xi_k = \xi(1:k)$

28: **计算近似解** $x^{(k)} = x^{(0)} + Z_k \tilde{y}$

注记

- 右预处理 GMRES 的区别仅仅在第 5 行和 28 行, 都与预处理子相关
- 向量 z_j 用来扩充 Krylov 子空间, 理论上可以自由选取, 也即预处理子 P_j 可以自由选取, 甚至不用预先给出, 只要保证 $z_j, j = 1, 2, \ldots$ 线性无关即可.
  - 这使得 FGMRES 具有很大的灵活性, 比如可通过任意的内迭代得到 z_j
  - 因此, 从这个意义上来讲, 迭代法也可以作为预处理手段.
  - 当然这也可能带来一些问题, 如果 z_j 选取不恰当, 算法可能会提前终止

FGMRES 的最优性

FGMRES 的重要等式 $$AZ_k = V_{k+1}H_{k+1,k} = V_kH_k + h_{k+1,k}v_{k+1}e_k^{\mathsf{T}}$$

设 $x = x^{(0)} + Z_k y$ 是 $x^{(0)} + \mathrm{span}\{Z_k\}$ 中的任意一个向量, 则

$$b - Ax = b - Ax^{(0)} + AZ_k y = r_0 - AZ_k y = \beta v_1 - V_{k+1}H_{k+1,k}y$$

对应的最小二乘问题

$$\min_{x \in x^{(0)} + \mathrm{span}\{Z_k\}} \|b - Ax\|_2 = \min_{y \in \mathbb{R}^k} \|\beta e_1 - H_{k+1,k}y\|_2$$

定理 设 $x^{(k)}$ 是由 FGMRES 算法得到的近似解, 则

$$x^{(k)} = \arg \min_{x \in x^{(0)} + \mathrm{span}\{Z_k\}} \|b - Ax\|_2.$$

提前终止

在 FGMRES 算法过程中, 如果 $h_{j+1,j} = 0$, 则算法会提前终止.

定理 设 $r_0 \neq 0$ 且当 $i < j$ 时有 $h_{i+1,i} \neq 0$. 如果 H_j 非奇异, 则 $x^{(j)}$ 是精确解的充要条件是 $h_{j+1,j} = 0$.

注记

在 FGMRES 算法中, 每次迭代的预处理子是在变化的, 有时甚至无法用矩阵显式表示, 此时子空间 $\mathrm{span}\{Z_m\}$ 不是一个标准的 Krylov 子空间. 因此关于 FGMRES 算法的收敛性比较难分析.

3-1-4 预处理 BiCGSTAB 方法

$$AM^{-1}u = b \quad \text{with} \quad u = Mx$$

BiCGSTAB: $Ax = b$

$p_1 = r_0$

$q_k = r_{k-1} - \alpha_k A p_k, \quad \alpha_k = \dfrac{(r_{k-1}, \tilde{r}_0)}{(Ap_k, \tilde{r}_0)}$

$r_k = q_k - \omega_k A q_k, \quad \omega_k = \dfrac{(q_k, Aq_k)}{(Aq_k, Aq_k)}$

$x^{(k)} = x^{(k-1)} + \alpha_k p_k + \omega_k q_k$

$p_{k+1} = r_k + \beta_k(p_k - \omega_k A p_k),$

$\beta_k = \dfrac{\alpha_k}{\omega_k} \cdot \dfrac{(r_k, \tilde{r}_0)}{(r_{k-1}, \tilde{r}_0)}$

右预处理 BiCGSTAB: $AM^{-1}u = b$

$p_1 = r_0$

$q_k = r_{k-1} - \alpha_k A \tilde{p}_k, \quad \alpha_k = \dfrac{(r_{k-1}, \tilde{r}_0)}{(A\tilde{p}_k, \tilde{r}_0)}$

$r_k = q_k - \omega_k A \tilde{q}_k, \quad \omega_k = \frac{(q_k, A\tilde{q}_k)}{(A\tilde{q}_k, A\tilde{q}_k)}$

$x^{(k)} = x^{(k-1)} + \alpha_k \tilde{p}_k + \omega_k \tilde{q}_k$

$p_{k+1} = r_k + \beta_k(p_k - \omega_k A \tilde{p}_k),$

$\beta_k = \dfrac{\alpha_k}{\omega_k} \cdot \dfrac{(r_k, \tilde{r}_0)}{(r_{k-1}, \tilde{r}_0)}$

其中 $\tilde{p}_k = M^{-1}p_k$, $\tilde{q}_k = M^{-1}q_k$ （算法讲义）

如何构造预处理子?

Finding a good preconditioner to solve a given sparse linear system is often viewed as a **combination of art and science**. Theoretical results are rare and some methods work surprisingly well, often despite expectations.

— Saad, 2003.

3-2 | 基于矩阵分裂的预处理子

预处理子的分类 (按构造方式)

(a) **代数预处理子** (Algebraic Preconditioner), **即仅仅根据所给方程组的系数矩阵来构造预处理子, 这类预处理子具有一定的通用性, 便于做成黑盒子.**

(b) **专属预处理子** (Problem-Specific Preconditioner), **即根据问题的机理或物理背景所构造的专属预处理子, 这类预处理子往往具有很好的数值表现.**

这里只介绍代数预处理子.

矩阵分裂预处理子

$$A = M - N$$

构造迭代格式:

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b = x^{(k)} + M^{-1}(b - Ax^{(k)}), \quad k = 0, 1, 2, \ldots.$$

- 等价方程组

$$M^{-1}(b - Ax) = 0 \quad \text{或} \quad M^{-1}Ax = M^{-1}b.$$

- 矩阵 M 可作为预处理子.

矩阵分裂预处理子

- Jacobi 预处理子 (或对角预处理子): $P = D$;
- Gauss-Seidel 预处理子 (或三角预处理子): $P = D - L$ 或 $P = D - U$;
- SOR 预处理子: $P = D - \omega L$ 或 $P = D - \omega U$;
- SSOR 预处理子: $P = (D - \omega L)D^{-1}(D - \omega U)$;
- SGS 预处理子: $P = (D - L)D^{-1}(D - U)$.
- HSS 预处理子: $P = (\alpha I + H)(\alpha I + S)$
- ADI 预处理子: $P = (\alpha I + A_1)(\alpha I + A_2)$

✍ 对于任意非零常数 α, 预处理子 αP 与 P 的效果是完全一样的, 因此我们在构造和使用预处理子时, 可以忽略预处理子前面的常系数.

✍ 相应地, 我们可以构造分块预处理子.

好的预处理子

要从理论上给出解释是比较困难的, 最直接的方法是通过数值实验进行测试.

通常我们认为一个好的预处理子 P 应该:

1. 使用成本低
2. 是 A 在某种意义下的一个很好的近似, 即 $P-A$ 或者 $I-P^{-1}A$ 很小, 或者说 P 包含原线性方程组尽可能多的信息.

✍ 注: 从 CG 方法的收敛性质可知, 如果能够有效降低系数矩阵的条件数, 或者使得系数矩阵的特征值分布更加聚集, 则 CG 方法的收敛速度会得到很大的改善.

✍ 一个比较有意思的事情是, 对病态矩阵 A 做预处理, 要使得其条件数大大降低, 预处理子 P 也通常是病态的.

例 以二维 Poisson 方程为例, 比较各种矩阵分裂预处理子对 CG 算法的加速效果.

PCG_Poisson2D.m

n	32^2	64^2	128^2	256^2
CG	58	114	224	438
PCG(Jacobi)	58	114	224	438
PCG(SGS)	31	53	89	156
PCG(TriDiag)	51	98	172	311

3-3 | 不完全分解预处理子

3.3 不完全分解预处理子

大规模稀疏矩阵的不完全分解

- 对于 大规模稀疏矩阵, 不完全分解 (incomplete factorization) 是一类比较通用预处理方法, 在实际应用中通常具有比较好的数值效果.
- 早在 1960 年左右, Buleev 和 Varga 就提出了矩阵不完全分解的思想, 但重要的突破是 1977 年由 Meijerink 和 van der Vorst 所提出的不完全 Cholesky 分解, 他们给出了基于不完全 Cholesky 分解的预处理共轭梯度方法 (incomplete Cholesky-conjugate gradient, ICCG).
- 不完全分解是当前针对大规模稀疏线性方程组的主流预处理技术之一.

注记

不完全矩阵分解预处理方法可以看作是融合直接法与迭代法的混合算法.

3-3-1 不完全 LU 分解

$$A = LU$$

定理 (LU 分解的存在性和唯一性) 矩阵 $A \in \mathbb{R}^{n\times n}$ 存在 LU 分解 (即存在单位下三角矩阵 L 和非奇异上三角矩阵 U 使得 $A = LU$) 的充要条件是 A 的所有顺序主子矩阵都非奇异. 进一步, 若 A 存在 LU 分解, 则分解是唯一的.

例 设 $A \in \mathbb{R}^{n\times n}$ 非奇异且列对角占优, 则 A 存在 LU 分解.

LU 分解过程

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \xrightarrow{\text{第一轮}} \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix}$$

- $L_1 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ l_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}$, $l_{i1} = \dfrac{a_{i1}}{a_{11}}$ $\longrightarrow$ $L_1^{-1}A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix}$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix} \xrightarrow{\text{第二轮}} \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix}$$

- $L_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & l_{n2} & 0 & \cdots & 1 \end{bmatrix}$, $l_{i2} = \dfrac{a_{i2}^{(1)}}{a_{22}^{(1)}}$ $\longrightarrow$ $L_2^{-1}L_1^{-1}A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix}$
- **依此类推, 构造一系列 $L_3, L_4, \ldots, L_{n-1}$, 使得**

$$L_{n-1}^{-1} \cdots L_2^{-1}L_1^{-1}A = U \quad \Longleftrightarrow \quad A = L_1L_2 \cdots L_{n-1}U$$

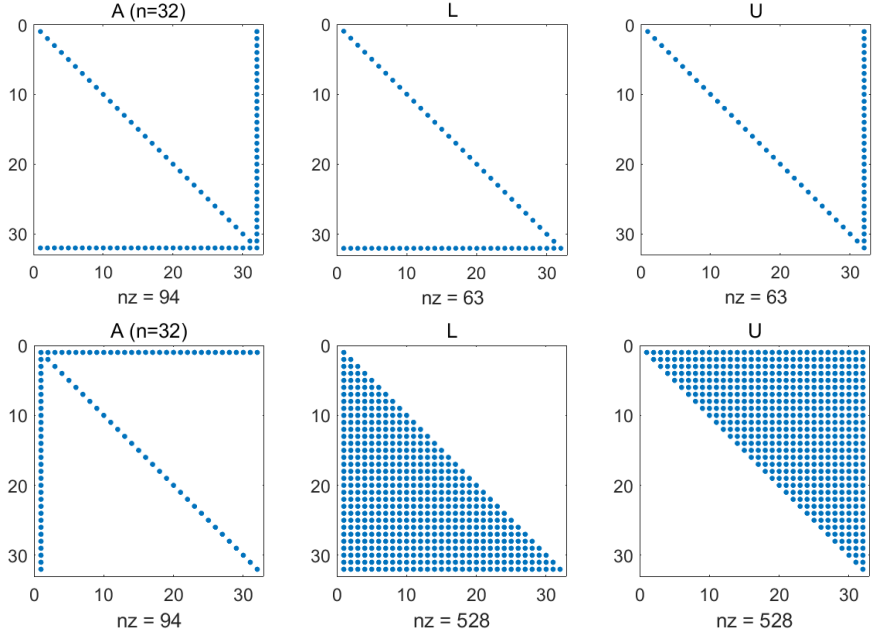
LU 分解算法

算法 LU 分解 (IKJ 型, 适合按行存储模式)

```
1: for i = 2 to n do
2:     for k = 1 to i − 1 do
3:         a_ik = a_ik / a_kk      % L 存放在 A 的严格下三角部分
4:         for j = k + 1 to n do
5:             a_ij = a_ij − a_ik a_kj      % U 存放在 A 的上三角部分
6:         end for
7:     end for
8: end for
```

- 对于稠密矩阵, 运算量为 $\frac{2}{3}n^3 + \mathcal{O}(n^2)$
- 如果 A 是一个稀疏矩阵, 运算量可能仍然是 $\mathcal{O}(n^3)$
- L 和 U 不一定是稀疏矩阵, 或者稀疏性远远低于 A

例 稀疏矩阵 LU 分解示例.



ILU 预处理子

为什么要不完全分解

- Krylov 子空间方法每次迭代的运算量大约为 $\mathcal{O}(\mathsf{nnz})$ (nnz 为非零元个数), 因此预处理子的构造成本和使用成本不能超过这个量级.
- 这就要求 L 和 U 满足一定的稀疏性, 因此只能进行 **不完全 LU 分解** (ILU)

$$A = LU - R,$$

其中 R 为误差矩阵.

- 不完全 LU 分解预处理子: $P_{\mathrm{ILU}} \triangleq LU$

怎么做 ILU: 稀疏模式、填充和舍弃准则

- **稀疏模式** (sparse pattern)

$$\mathcal{P}_A \triangleq \{(i,j) \ : \ a_{ij} = 0, \ i \neq j \}$$

- **填充**: 对 A 进行 LU 分解时, 如果 L 和 U 在 $\mathcal{P}_A$ 中的某个位置 (i,j) 上出现非零元, 我们就称之为填充 (fill-in)

- **不完全 LU 分解**: 怎么处理出现的填充? 全部舍弃或部分舍弃?

  → 预先定义 **舍弃准则** (dropping rule): 通常基于填充的 **值** 或 **位置**

方案一: 无填充 ILU

- 一种常用的简单舍弃准则就是将所有填充全部舍弃
  - ⟶ **无填充**的 ILU 预处理子, 记为 **ILU(0)**

算法 无填充 ILU 分解, ILU(0)

1: **for** $i = 2$ to n **do**
2: **for** $k = 1$ to $i-1$ and $(i,k) \notin \mathcal{P}_A$ **do**
3: $a_{ik} = a_{ik}/a_{kk}$
4: **for** $j = k+1$ to n and $(i,j) \notin \mathcal{P}_A$ **do**
5: $a_{ij} = a_{ij} - a_{ik}a_{kj}$
6: **end for**
7: **end for**
8: **end for**

ILU(0) 算法分析

$$A = LU - R \quad \Longrightarrow \quad a_{ij} = \sum_{k=1}^{i} l_{ik}u_{kj} - r_{ij} \quad \Longrightarrow \quad r_{ij} = \begin{cases} 0, & (i,j) \notin \mathcal{P}_A, \\ \sum\limits_{k=1}^{i} l_{ik}u_{kj}, & (i,j) \in \mathcal{P}_A. \end{cases}$$

注记

- 误差矩阵 R 决定了 ILU(0) 预处理子 $P_{\mathrm{ILU}} = LU$ 与 A 之间的近似度
- 若 A 对称正定, 则可得无填充不完全 Cholesky 分解 **IC(0)** $A = LL^{\mathsf{T}} - R$
- ILU(0) 和 IC(0) 的优点: 容易构造且使用成本低, 对于某些问题非常有效, 比如对角占优 M-矩阵. 特别地, 对于二阶椭圆方程五点差分离散后的线性方程组, 带 IC(0) 预处理的 CG 计算成本几乎与不带预处理的 CG 方法相当.

3-3-2 填充级别和 ILU(p)

ILU 的改进

- 对于较复杂的问题, ILU(0) 可能无法取得令人满意的效果, 此时需要改进.
- 改进方案: 适当保留部分填充, 提高 P_{ILU} 与 A 的近似程度

  $\rightarrow$ 引入概念 **填充级别** (level of fill).

填充级别

A 的每个元素 a_{ij} 都被赋予一个级别, 记为 lev_{ij}, 初始值设为

$$\mathsf{lev}_{ij} = \begin{cases} 0, & \text{if } i = j \text{ or } a_{ij} \neq 0, \\ \infty, & \text{otherwise.} \end{cases}$$

lev_{ij} 就是 a_{ij} 的填充级别, 其值越大, 意味着 $|a_{ij}|$ 越小

填充级别的更新 (模型: $a_{ij} \sim \varepsilon^{\mathsf{lev}_{ij}}$, $0 < \varepsilon < 1$)

$$a_{ij} = a_{ij} - a_{ik}a_{kj} \quad \Longrightarrow \quad \mathsf{lev}_{ij} = \min\{\mathsf{lev}_{ij}, \mathsf{lev}_{ik} + \mathsf{lev}_{kj}\}$$

☹ 若 $a_{ij} \neq 0$, 则 lev_{ij} 永远为 0;
若 $a_{ij} = 0$, 则 lev_{ij} 的初值为 ∞, 后面有可能变为 0 或保持为 ∞

修正 lev_{ij} 的更新公式

当前常用一种修改方法:

$$\mathsf{lev}_{ij} = \min\{\mathsf{lev}_{ij}, \mathsf{lev}_{ik} + \mathsf{lev}_{kj} + 1\ \}.$$

合理性解读: 如果 a_{ij} 的值从 0 变为非零, 则相应的 lev_{ij} 从 ∞ 变为一个有限值, 但这个有限值应该大于原来非零元 a_{ik} 和 a_{kj} 的填充级别 lev_{ik} 和 lev_{kj}.

另一个常用的填充级别更新公式是

$$\mathsf{lev}_{ij} = \min\{\mathsf{lev}_{ij}, \max\{\mathsf{lev}_{ik}, \mathsf{lev}_{kj}\} + 1\ \}.$$

注: A 中非零元素的填充级别始终是 0 ($\rightarrow$ 引入填充级别是为了处理填充)

ILU(p) 预处理子

p 级 **ILU 预处理子** (ILU of level p), **记为 ILU(p)**:

舍弃所有 $\mathsf{lev}_{ij} > p$ 的填充

若 $p = 0$, 则 ILU(p) 就是 ILU(0).

算法 ILU(p) **分解**

Set the initial value of lev_{ij}
for $i = 2$ to n **do**
  for $k = 1$ to $i-1$ and $\mathsf{lev}_{ik} \le p$ **do**
    $a_{ik} = a_{ik}/a_{kk}$
    for $j = k+1$ to n **do**
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$
      $\mathsf{lev}_{ij} = \min\{\mathsf{lev}_{ij}, \mathsf{lev}_{ik} + \mathsf{lev}_{kj} + 1\}$
    end for
  end for
  for $j = 1$ to n and $\mathsf{lev}_{ij} > p$ **do**
    $a_{ij} = 0$
  end for
end for

几点说明

- 对于许多实际问题, ILU(1) 效果比 ILU(0) 有明显的提升, 运算量也在可接受范围内.
- 对于更高级别的 ILU, 效果会更好, 但由于运算量和存储量会大幅增加, 一般很少使用.
- ILU 也存在一些不足, 比如:

  (1) 增加的存储量和运算量无法事先估算;

  (2) 填充级别的更新是一笔不小的开支;

  (3) 对于不定矩阵, 根据填充级别制定的舍弃准则不一定有效, 因为此时 lev_{ij} 并不一定能正确描述填充的大小.

3-3-3 ILU 的存在性

引理 设 $A \in \mathbb{R}^{n\times n}$ 是 M-矩阵, A_1 是经过一次 Gauss 消去过程后得到的矩阵. 则 A_1 也是 M-矩阵.

(板书)

ILU(0) 的存在性

定理 设 $A \in \mathbb{R}^{n\times n}$ 是 M-矩阵, 则 ILU(0) 分解存在, 且

$$A = LU - R$$

构成 A 的一个正则分裂. (板书)

定理 设 $A \in \mathbb{R}^{n\times n}$ 是对称 M-矩阵, 则 IC(0) 分解存在, 且

$$A = LL^{\mathsf{T}} - R$$

构成 A 的一个正则分裂.

3-3-4 带 Threshold 的 ILU

当系数矩阵 A 既不是正定, 也不是对角占优时, 通过填充级别制定的舍弃准则不一定是好的方案: 有些填充的级别虽然比较大, 但它们的量 (绝对值) 并不小, 有的可能会很大.

双阈值 ILU 分解, 记为 ILUT(p, τ)

DS1 如果填充的绝对值小于其所在原始行的范数的 τ 倍, 则舍弃.

DS2 除对角线外, L 和 U 每行至多保留最大的前 p 个非零元素.

算法 ILUT(p, τ): ILU with dual threshold

for $i = 1$ to n **do**
  for $k = 1$ to n **do** % copy the i-th row of A to w
    $w_k = a_{ik}$
  end for
  $\tau_i = \tau \|w\|$ % relative drop tolerance
  for $k = 1$ to $i-1$ and $w_k \neq 0$ **do**
    $w_k = w_k / a_{kk}$
    if $|w_k| < \tau_i$ **then** % applying **DS1** on w_k
      $w_k = 0$
    else
      for $j = k+1, \ldots, n$ **do**
        $w_j = w_j - w_k u_{kj}$
      end for
    end if
  end for

16: **for** $k = 1$ to n and $|w_k| < \tau_i$ **do** % applying **DS1** on w
17: $w_k = 0$
18: **end for**
% applying **DS2** on w
19: Find the largest p entries of $w(1 : i-1)$ and drop others
20: Find the largest p entries of $w(i+1 : n)$ and drop others
21: **for** $k = 1$ to $i-1$ **do**
22: $l_{ik} = w_k$
23: **end for**
24: **for** $k = i$ to n **do**
25: $u_{ik} = w_k$
26: **end for**
27: set $w = 0$
28: **end for**

注记

- 关于非零元个数, 考虑到原始矩阵 A 本身每行的非零元素个数可能差别较大, 因此也可以改为**每行额外增加的填充个数不超过 p**.
- 通过调节 (p, τ) 的大小, ILUT(p, τ) 往往可以提供一个比较好的预处理子.
- ILUT(p, τ) 的一个困难是参数 (p, τ) 的选取.

3-3-5 修正的 ILU

修正的 ILU

为了进一步提升 ILU 的预处理效果, 可以对 ILU 进行修正. 一个简单的修正方法是将舍弃的填充全部加到到对角线上. 这种补偿方案可以使得 ILU 预处理子 $P_{\mathrm{ILU}} = LU$ 的行和与 A 的行和相等, 即

$$Ae = LUe, \quad \text{其中} \quad e = [1, 1, \ldots, 1]^{\mathsf{T}}.$$

修正后的 ILU 预处理子记为 **MILU** .

3-4 | 稀疏近似逆预处理子

什么是稀疏近似逆

给定稀疏矩阵 A, 寻找一个稀疏矩阵 M, 使得 $M \approx A^{-1}$.

注记

- 如果 A^{-1} 是稀疏矩阵, 则在很大程度上能找到一个较好的稀疏近似逆. 但需要注意的, 即使 A 非常稀疏, 但 A^{-1} 不一定稀疏.
- 虽然稀疏矩阵的逆不一定稀疏, 但往往可能存在很多绝对值相对很小的元素, 通过一定的准则将这些元素舍弃后, 就有可能得到一个较好的稀疏近似逆.
- 可以用两个或者多个稀疏矩阵的乘积作为 A^{-1} 的稀疏近似.

稀疏近似逆的计算

一类常用方法是基于矩阵 F-范数, 即

$$M = \underset{X\text{满足一定的稀疏模式}}{\operatorname{argmin}} \|AX - I\|_F^2 = \underset{X\text{满足一定的稀疏模式}}{\operatorname{argmin}} \sum_{i=1}^{n} \|Ax_i - e_i\|_2^2$$

3-5 | 其他预处理技术

基于问题特殊结构或特殊性质的预处理方法

- 多重网格法, 区域分解法
- 鞍点问题: Schur 补预处理子, 约束预处理子, 增广 Lagrange 预处理子, ...
- Toeplitz 结构线性方程组
- Hierarchical 结构线性方程组
- ... ...


谢 谢
THANK YOU