高等数值算法与应用（八）

Advanced Numerical Algorithms & Applications

计算机科学与技术系 喻文健

内容概要

- 线性方程组的迭代解法（三）
  - 对称矩阵的**Arnoldi**过程—**Lanczos**过程
  - 从**Lanczos**过程到共轭梯度法
  - 第三类**Krylov**子空间迭代法基础—**Lanczos**双正交化过程
  - 构造预条件矩阵的技术

简单介绍

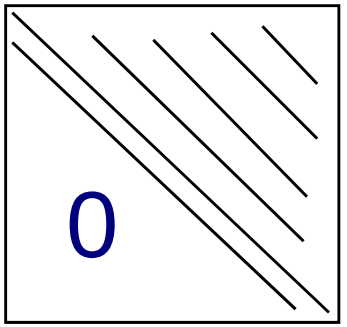
Krylov Methods - Symmetric Lanczos Procedure and CG

Wenjian Yu

Arnoldi过程：为Krylov子空间$\mathcal{K}_m(A, v_1)$构造一组单位正交基向量

Arnoldi过程的矩阵表示：

(H_m为上Hessenberg矩阵)



$$A V_m = V_{m+1} \bar{H}_m$$

$$= V_m H_m + w_m e_m^T \quad \Longrightarrow \quad V_m^T A V_m = H_m$$

考虑对实对称矩阵A应用Arnoldi过程，以及$\mathbb{L}=\mathbb{K}$的正交投影方法，其系数矩阵为H_m

(如果A对称正定，
此方法不会恶性中断)

$$H_m^T = V_m^T A^T V_m = H_m$$

对称矩阵！

+

H_m为对称三对角矩阵 $\Longleftarrow$ 上Hessenberg矩阵

Arnoldi过程 (针对实对称阵)

1) 选v_1使$\|v_1\|_2 = 1$

2) For j=1, …, m

$z := Av_j$

For ~~i=1, …,~~ j-1, j

$h_{i,j} := v_i^T z$

End

$w_j := z - h_{j-1,j}v_{j-1} - h_{j,j}v_j$

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

If $h_{j+1,j}$ ==0, stop

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

Endfor

$$H_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \beta_4 & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \cdot \\ & & & & \beta_m & \alpha_m \end{pmatrix}$$

第j-1步得到 $h_{j,j-1} = \beta_j$

$w_j := z - \beta_j v_{j-1}$ (j=1时设 $\beta_j v_{j-1} = 0$)

$\alpha_j := v_j^T w_j$

$w_j := w_j - \alpha_j v_j$

$\beta_{j+1} := \|w_j\|_2$

。 。 。

此过程称为Lanczos过程

The Lanczos algorithm

ALGORITHM : **Lanczos Procedure**

1. **Choose an initial vector** v_1 **of norm unity. Set** $\beta_1 \equiv 0, v_0 \equiv 0$
2. **For** $j = 1, 2, \ldots, m$ **Do:**
3. $w_j := Av_j - \beta_j v_{j-1}$
4. $\alpha_j := (w_j, v_j)$
5. $w_j := w_j - \alpha_j v_j$
6. $\beta_{j+1} := \|w_j\|_2$. **If** $\beta_{j+1} = 0$ **then Stop**
7. $v_{j+1} := w_j / \beta_{j+1}$
8. **EndDo**

当A为实对称矩阵，
为Krylov子空间$\mathcal{K}_m(A, v_1)$
构造一组单位正交基向量

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & \beta_m & \alpha_m \end{pmatrix}$$

计算量比**Arnoldi**过程小

ALGORITHM : **Lanczos Method for Linear Systems**

1. **Compute** $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, **and** $v_1 := r_0/\beta$
2. **For** $j = 1, 2, \ldots, m$ **Do:**
3. $w_j = Av_j - \beta_j v_{j-1}$ **(If** $j = 1$ **set** $\beta_1 v_0 \equiv 0$**)**
4. $\alpha_j = (w_j, v_j)$
5. $w_j := w_j - \alpha_j v_j$
6. $\beta_{j+1} = \|w_j\|_2$. **If** $\beta_{j+1} = 0$ **set** $m := j$ **and go to 9**
7. $v_{j+1} = w_j/\beta_{j+1}$
8. **EndDo**
9. **Set** $T_m = tridiag(\beta_i, \alpha_i, \beta_{i+1})$, **and** $V_m = [v_1, \ldots, v_m]$.
10. **Compute** $y_m = T_m^{-1}(\beta e_1)$ **and** $x_m = x_0 + V_m y_m$

针对实对称矩阵的正交投影方法

$\mathbb{L}=\mathbb{K}$

存储量仍然逐渐增大，怎么自动确定合适的m？

右端项 $\boldsymbol{V}_m^T \boldsymbol{r}_0 = \beta e_1$

利用系数矩阵T_m的特点，发展出直接从x_{m-1}→x_m的算法

Incremental form for solving $T_m y_m = \beta e_1^{(m)}$ (怎么从x_{m-1}→x_m ?)

从T_{m-1}→T_m :

作**Lanczos**过程第**j=m**步得到 α_m ，即得到T_m

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \beta_m \\ & & & & \beta_m & \alpha_m \end{pmatrix}$$

怎么加快求解 $T_m y_m = \beta e_1^{(m)}$**?**

做法**1:** 通过**Givens**旋转将其化为上三角阵，不断检查残差向量是否足够小, ……, 导出算法类似**FOM**算法

做法**2:** 直接分解技术： T_m的**LU**分解的特点：

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \beta_m \\ & & & & \beta_m & \alpha_m \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \\ & & & & \lambda_m & 1 \end{pmatrix}}_{L_m} \underbrace{\begin{pmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \eta_3 & \cdot & \\ & & & \cdot & \cdot \\ & & & & \cdot & \beta_m \\ & & & & & \eta_m \end{pmatrix}}_{U_m}$$

Incremental form for solving $T_m y_m = \beta e_1^{(m)}$ (怎么从x_{m-1}→x_m ?)

求矩阵L_m和U_m的算法

$$\eta_1 = \alpha_1$$

For i= 1, …, m-1

$$\lambda_{i+1} = \beta_{i+1} / \eta_i$$

$$\eta_{i+1} = \alpha_{i+1} - \lambda_{i+1}\beta_{i+1}$$

Endfor

$$\begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \beta_4 & \\ & & \ddots & \ddots & \ddots \\ & & & \beta_m & \alpha_m \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda_m & 1 \end{pmatrix} \begin{pmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \eta_3 & \ddots & \\ & & & \ddots & \beta_m \\ & & & & \eta_m \end{pmatrix}$$

从L_{m-1}, U_{m-1}→L_m, U_m：需要 λ_m, η_m，多做一步迭代即可

$$y_m = T_m^{-1}\beta e_1^{(m)} = U_m^{-1}L_m^{-1}\beta e_1^{(m)}$$

$$x_m = x_0 + \underline{V_m U_m^{-1}} \underline{L_m^{-1}\beta e_1^{(m)}}$$

$$\xrightarrow{?} x_{m-1} + \zeta_m p_m$$

增量式

$$x_m = x_0 + (V_{m-1}U_{m-1}^{-1} \mid p_m)\begin{pmatrix} L_{m-1}^{-1}\beta e_1^{(m-1)} \\ \zeta_m \end{pmatrix} \quad ?$$

$$= x_0 + V_{m-1}U_{m-1}^{-1}L_{m-1}^{-1}\beta e_1^{(m-1)} + \zeta_m p_m = \underline{x_{m-1} + \zeta_m p_m}$$

设 $P_m = V_m U_m^{-1}$, $z_m = L_m^{-1}\beta e_1^{(m)}$

$$P_m = (V_{m-1}U_{m-1}^{-1} \mid p_m)$$

$$P_m U_m = V_m \quad \text{即} \quad (p_1 \cdots \mid p_m)\left(\begin{array}{ccccc|c} \eta_1 & \beta_2 & & & & \\ & \eta_2 & \beta_3 & & & \\ & & \eta_3 & \ddots & & \\ & & & \ddots & \beta_m & \\ \hline & & & & & \eta_m \end{array}\right) = (v_1 \cdots \mid v_m)$$

→ 根据分块矩阵乘法，

$$(p_1 \cdots p_{m-1})U_{m-1} = (v_1 \cdots v_{m-1}) = V_{m-1}$$

Incremental form for solving $T_m y_m = \beta e_1^{(m)}$ (怎么从x_{m-1}→x_m ?)

得到递推公式: $x_j = x_{j-1} + \zeta_j p_j$, 怎么求向量$p_j$和$z_j$的分量$\zeta_j$?

$$P_j U_j = V_j$$

$$(p_1 \cdots p_j) \begin{pmatrix} \eta_1 & \beta_2 & & & & \\ & \eta_2 & \beta_3 & & & \\ & & \eta_3 & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & \beta_j \\ & & & & & \eta_j \end{pmatrix} = (v_1 \cdots v_j)$$

$$v_j = \beta_j p_{j-1} + \eta_j p_j$$

$$p_j = \eta_j^{-1}(v_j - \beta_j p_{j-1})$$

$$z_j = (\zeta_1 \cdots \zeta_j)^T, \quad L_j z_j = \beta e_1^{(j)} \longrightarrow \zeta_j = -\lambda_j \zeta_{j-1}$$

注意：这些量的计算都可嵌入**Lanczos**过程的循环体中，计算顺序为：**(**下标**j-1**的量在上一步迭代中已得到**)**

v_j , β_j , α_j ,　λ_j , η_j ,　p_j , ζ_j

上一迭代步 → T_j矩阵　T_j的**LU**分解　得到新解x_j

$$x_j = x_{j-1} + \zeta_j p_j$$

发展出**D-Lanczos**算法**!**

ALGORITHM : **D-Lanczos**

1. **Compute** $r_0 = b - Ax_0$, $\zeta_1 := \beta := \|r_0\|_2$, **and** $v_1 := r_0/\beta$
2. **Set** $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3. **For** $m = 1, 2, \ldots$, **until convergence Do:**
4. **Compute** $w := Av_m - \beta_m v_{m-1}$ **and** $\alpha_m = (w, v_m)$
5. **If** $m > 1$ **then compute** $\lambda_m = \frac{\beta_m}{\eta_{m-1}}$ **and** $\zeta_m = -\lambda_m \zeta_{m-1}$
6. $\eta_m = \alpha_m - \lambda_m \beta_m$
7. $p_m = \eta_m^{-1}(v_m - \beta_m p_{m-1})$
8. $x_m = x_{m-1} + \zeta_m p_m$
9. **If** x_m **has converged then Stop**
10. $w := w - \alpha_m v_m$
11. $\beta_{m+1} = \|w\|_2$, $v_{m+1} = w/\beta_{m+1}$
12. **EndDo**

T矩阵的LU分解、x的计算都变成递推式，并嵌入Lanczos过程

计算量、存储量都不随迭代步增加而增大(只需存v_m和v_{m-1})

$\|x_m - x_{m-1}\|_2$ <eps ？

D-Lanczos算法针对实对称矩阵

对一般的实对称矩阵，可能发生“恶性中断”

- T_m矩阵奇异(某个η_m=0或w=0，算法执行发生中断)
- 对T_m矩阵的LU分解是不选主元的，计算误差大

考虑实对称正定矩阵，则上述两个问题都消失

D-Lanczos算法重要性质

1. $\{p_i\}$两两A正交， $p_i^T A p_j = 0, \text{ for } i \neq j$

$$P_m^T A P_m = U_m^{-T} V_m^T A V_m U_m^{-1}$$

$$P_m = V_m U_m^{-1} \qquad = U_m^{-T} T_m U_m^{-1}$$

$$T_m = L_m U_m \qquad = U_m^{-T} L_m$$

两个下三角矩阵乘积仍为下三角

$P_m^T A P_m$

对称矩阵+下三角矩阵：对角阵 $\begin{pmatrix} p_1^T \\ \vdots \\ p_m^T \end{pmatrix} A \begin{pmatrix} p_1 \cdots p_m \end{pmatrix}$ ⟹ $\left(P_m^T A P_m\right)_{ij} = p_i^T A p_j = 0, \text{ for } i \neq j$

D-Lanczos算法重要性质

2. 剩余向量 $r_m = b - Ax_m$ 与 v_{m+1} 成比例, $\{r_i\}$ 两两正交

$$
\begin{aligned}
b - Ax_m &= r_0 - AV_m y_m \qquad (AV_m = V_m H_m + w_m e_m^T) \\
&= \beta v_1 - (V_m H_m + w_m e_m^T) y_m \\
&= \beta v_1 - V_m (H_m y_m) - \eta_{m+1,m} v_{m+1} e_m^T y_m \\
&= \beta v_1 - V_m (\beta e_1) - \eta_{m+1,m} v_{m+1} e_m^T y_m \\
&= -(\eta_{m+1,m} e_m^T y_m) v_{m+1}
\end{aligned}
$$

数学上为**Arnoldi**投影方法的一个特殊情况

利用这两个性质得到针对对称正定矩阵的简洁算法

D-Lanczos算法 ——共轭梯度算法**(CG算法)**

。。。

7. $p_m = \xi_m r_{m-1} - \eta_m^{-1} \beta_m p_{m-1}$ → $p_{m+1} = \xi_{m+1} r_m - \eta_{m+1}^{-1} \beta_{m+1} p_m$

定义 p'_{m+1} 使得 $p'_{m+1} = r_m + \beta'_{m+1} p'_m$

8. $x_m = x_{m-1} + \zeta_m p_m$ → $x_m = x_{m-1} + \alpha'_m p'_m$ $\{p'_i\}$ 仍然两两**A**正交**!**

。。。

通过p'更新x, 再计算剩余，然后得到下一个p'。。。

怎么算系数 α'_j, β'_j?

$$x_{j+1} = x_j + \alpha'_j p'_j \Longrightarrow r_{j+1} = r_j - \alpha'_j A p'_j \Longrightarrow (r_j,\ r_j - \alpha'_j A p'_j) = 0$$

$\{r_j\}$两两正交

$$\Downarrow$$

$$\alpha'_j = \frac{(r_j,\ r_j)}{(Ap'_j,\ p'_j)}$$

$$p'_{j+1} = r_{j+1} + \beta'_j p'_j \Longrightarrow \beta'_j = -\frac{(r_{j+1},\ Ap'_j)}{(r_j,\ Ap'_j)}$$

$\{p'_j\}$两两A正交!

$$Ap'_j = -\frac{1}{\alpha'_j}(r_{j+1} - r_j)$$

$$\alpha'_j (Ap'_j,\ r_j) = (r_j,\ r_j)$$

$$\beta'_j = \frac{(r_{j+1},\ r_{j+1})}{(r_j,\ r_j)}$$

去掉变量的'记号，得到**CG**算法

1. Start: $r_0 := b - Ax_0,\ p_0 := r_0.$

2. Iterate: Until convergence do,

(a) $\alpha_j := (r_j, r_j)/(Ap_j, p_j)$

(b) $x_{j+1} := x_j + \alpha_j p_j$

(c) $r_{j+1} := r_j - \alpha_j A p_j$

(d) $\beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$

(e) $p_{j+1} := r_{j+1} + \beta_j p_j$

- 算法形式简洁!
- 收敛判据可使用$\|r_j\|$
- 对于对称正定阵，不会恶性中断

Lanczos Bi-Orthogonalization and Other Krylov Methods

第3类Krylov子空间迭代法的基本思想

Lanczos Bi-Orthogonalization: 扩展到非对称矩阵

ALGORITHM : ***The Lanczos Bi-Orthogonalization Procedure***

1. ***Choose two vectors*** v_1, w_1 ***such that*** $(v_1, w_1) = 1.$
2. ***Set*** $\beta_1 = \delta_1 \equiv 0,\ w_0 = v_0 \equiv 0$
3. ***For*** $j = 1, 2, \ldots, m$ ***Do:***
4. $\alpha_j = (Av_j, w_j)$
5. $\hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6. $\hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7. $\delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. ***If*** $\delta_{j+1} = 0$ ***Stop***
8. $\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
9. $w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
10. $v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
11. ***EndDo***

构造两个子空间的基向量

V: $\mathcal{K}(A, v_1)$
W: $\mathcal{K}(A^T, w_1)$

(注意不是正交基)

由于初始条件：

$(\hat{v}_{j+1}, w_j) = 0 \quad (\hat{w}_{j+1}, v_j) = 0$
$(v_{j+1}, w_j) = 0 \quad (w_{j+1}, v_j) = 0$

用归纳法可证明：$(v_j, w_i) = \delta_{ij} = \begin{cases} = 1, & i = j \\ = 0, & i \neq j \end{cases}$

一对单位化双正交序列

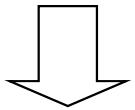
矩阵形式：$A V_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^T$

$A^T W_m = W_m T_m^T + \beta_{m+1} w_{m+1} e_m^T$

Methods Based on Lanczos Bi-Orthogonalization

基于Lanczos双正交过程的投影方法

取 $\mathbb{K}_m = \mathcal{K}(A, v_1)$, $\mathbb{L}_m = \mathcal{K}(A^T, w_1)$



投影方法求解的矩阵

$$W_m^T A V_m$$

Lanczos双正交过程的矩阵形式

$$A V_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^T$$

$$A^T W_m = W_m T_m^T + \beta_{m+1} w_{m+1} e_m^T$$

一对双正交序列：

$$(v_j, w_i) = \delta_{ij} = \begin{cases} = 1, & i = j \\ = 0, & i \neq j \end{cases}$$

$W_m^T V_m = I_m$

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \cdot & \cdot & \cdot & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix}$$

为三对角矩阵！

类似于处理对称矩阵的**D-Lanczos**算法，将T_m进行**LU**分解，通过一系列推导得到从x_j到x_{j+1}的递推式

——**BCG**算法

ALGORITHM : **BiConjugate Gradient (BCG)** 也称BiCG算法

1. Compute $r_0 := b - Ax_0$. Choose r_0^* such that $(r_0, r_0^*) \neq 0$.
2. Set, $p_0 := r_0$, $p_0^* := r_0^*$
3. For $j = 0, 1, \ldots$, until convergence Do:,
4. $\alpha_j := (r_j, r_j^*)/(Ap_j, p_j^*)$
5. $x_{j+1} := x_j + \alpha_j p_j$
6. $r_{j+1} := r_j - \alpha_j A p_j$
7. $r_{j+1}^* := r_j^* - \alpha_j A^T p_j^*$
8. $\beta_j := (r_{j+1}, r_{j+1}^*)/(r_j, r_j^*)$
9. $p_{j+1} := r_{j+1} + \beta_j p_j$
10. $p_{j+1}^* := r_{j+1}^* + \beta_j p_j^*$
11. EndDo

推导过程类似于CG算法

$$T_m = L_m U_m$$

$$x_m = x_0 + V_m T_m^{-1} \beta e_1^{(m)}$$

$$= x_0 + V_m U_m^{-1} L_m^{-1} \beta e_1^{(m)}$$

$$= x_0 + P_m L_m^{-1} \beta e_1^{(m)}$$

$$P_m = V_m U_m^{-1}$$

- p_j 和 p_j^* 两组向量
- 存储量、计算量不随迭代过程增大
- 每步迭代计算量为CG法两倍
- 可能恶性中断, 收敛性不规律

Quasi-Minimal Residual Algorithm

- The Lanczos algorithm gives the relations $AV_m = V_{m+1}\bar{T}_m$ with $\bar{T}_m$ = $(m+1) \times m$ tridiagonal matrix $\bar{T}_m = \begin{pmatrix} T_m \\ \delta_{m+1} e_m^T \end{pmatrix}$.
- Let $v_1 \equiv r_0 / \beta$ and $x = x_0 + V_m y$. Residual norm $\|b - Ax\|_2$ is

$$\|r_0 - AV_m y\|_2 = \|\beta v_1 - V_{m+1}\bar{T}_m y\|_2 = \|V_{m+1}\left(\beta e_1 - \bar{T}_m y\right)\|_2$$

- Column-vectors of V_{m+1} are not orthonormal ($\neq$ GMRES).
- But: reasonable idea to minimize the function $J(y) \equiv \|\beta e_1 - \bar{T}_m y\|_2$
- Quasi-Minimal Residual Algorithm (Freund, 1990).

一定程度上能避免**break down**
收敛性质更有规律，整个算法更稳定

ALGORITHM : ***QMR***

1. **Compute** $r_0 = b - Ax_0$ **and** $\gamma_0 := \|r_0\|_2$, $w_1 := v_1 := r_0/\gamma_1$
2. **For** $m = 1, 2, \ldots,$ **until convergence Do:**
3. **Compute** α_m, δ_{m+1} **and** v_{m+1}, w_{m+1} **as in Lanczos Algor. [alg. ??]**
4. **Update the QR factorization of** $\bar{T}_m$**, i.e.,**
5. **Apply** Ω_i, $i = m-2, m-1$ **to the** m**-th column of** $\bar{T}_m$
6. **Compute the rotation coefficients** c_m, s_m
7. **Apply rotation** Ω_m, **to** $\bar{T}_m$ **and** $\bar{g}_m$**, i.e., compute:**
8. $\gamma_{m+1} := -s_m\gamma_m$; $\gamma_m := c_m\gamma_m$; **and** $\alpha_m := c_m\alpha_m + s_m\delta_{m+1}$
9. $p_m = \left(v_m - \Sigma_{i=m-2}^{m-1} t_{im}p_i\right)/t_{mm}$
10. $x_m = x_{m-1} + \gamma_m p_m$
11. **If** $|\gamma_{m+1}|$ **is small enough Stop**
12. **EndDo**

其他重要方法:
BiCG-stab, 1992

How to Construct Preconditioners

Wenjian Yu

Preconditioning – Basic principles

Basic idea is to use the Krylov subspace method on a modified system such as

$$M^{-1}Ax = M^{-1}b.$$

三种做法与额外计算量

1. 显式构造矩阵M: 每个迭代步的额外计算是求解以M为系数矩阵的线性方程组一次(注意并不实际形成矩阵$M^{-1}A$)

   希望能快速地求解以M为系数矩阵的线性方程组

2. 显式得到M的因子：M=LU，则每个迭代步增加解L, U分别为系数矩阵的方程各一次

3. 显式构造矩阵M^{-1}:每个迭代步的额外计算是做一次矩阵M与向量的乘法

   希望矩阵M^{-1}结构简单，易于进行矩阵向量乘

预条件的效果： $M^{-1}A$相比A应能导致更快的收敛，因此...

Left, Right, and Split preconditioning

Left preconditioning

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

Right preconditioning

$$AM^{-1}u = b, \text{ with } x = M^{-1}u$$

矩阵A非对称

Split preconditioning . Assume M is factored: $M = M_L M_R$.

$$M_L^{-1}AM_R^{-1}u = M_L^{-1}b, \text{ with } x = M_R^{-1}u$$

矩阵A对称，要求*M*也对称

Standard preconditioners

Preconditioner M^{-1}

- Simplest preconditioner: M = Diag(A)
- Next to simplest: SSOR. (Symmetric SOR)

一个迭代步含前向、后向两遍SOR过程

$$M = (D - \omega E)D^{-1}(D - \omega F)$$

- Still simple but often more ef£cient: ILU(0).
- ILU(p) – ILU with level of £ll p – more complex.
- Class of ILU preconditioners with threshold

构造过程比较费时间，但对于大规模问题、或多右端方程求解有用

慢 → 快：收敛速度

小 → 大：构造及每步计算代价

根据固定格式迭代构造预条件

$$\boldsymbol{Ax}=\boldsymbol{b} \quad \Leftrightarrow \quad \boldsymbol{x}=\boldsymbol{Bx}+\boldsymbol{f}$$

构造迭代法

$$x^{(k+1)}=Bx^{(k)}+f,\ k=0,1,\ldots$$

收敛充分必要条件 $\rho(B)<1$

谱半径越小(B近似为零矩阵)，收敛越快

Spliting方法 $Mx^{(k+1)}=Nx^{(k)}+b,\ k=0,1,\ldots$

$$B=M^{-1}N$$

$$M^{-1}N\approx 0 \ \Leftrightarrow \ I-M^{-1}N=M^{-1}A\approx I$$

取***M***做预条件矩阵较好

根据固定格式迭代构造预条件

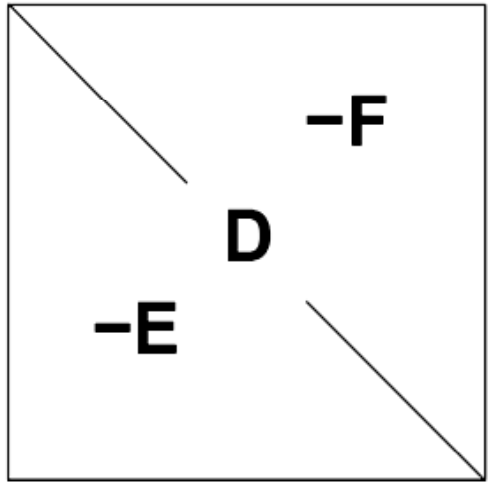
In splitting methods, $A = M - N$, **and** $Mx = Nx + b$

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b$$

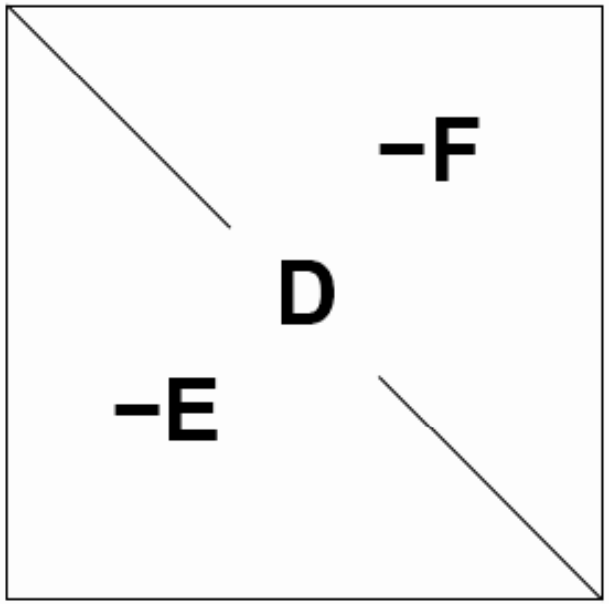
比如，**Jacobi**迭代法中M=**Diag(**A**)**，对应的预条件也叫**Jacobi**预条件

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

Gauss-Seidel迭代法, $M=D-E$，对应的预条件简称**GS**预条件



The SOR/SSOR preconditioner



- SOR preconditioning

$$M_{SOR} = (D - \omega E)$$

- SSOR preconditioning

(若A对称，则它也对称)

$$M_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)$$

- $M_{SSOR} = LU$, $L =$ lower unit matrix, $U =$ upper triangular. One solve with $M_{SSOR} \approx$ same cost as a MAT-VEC. 矩阵向量乘

- Questions: Best ω? For preconditioning can take $\omega = 1$

$$M = (D - E)D^{-1}(D - F)$$

SGS preconditioner

ILU(0) and IC(0) preconditioners

不完全的LU分解或Cholesky分解

一般还包括对角线！

- **Notation:** $NZ(X) = \{(i,j) \mid X_{i,j} \neq 0\}$

L为单位下三角阵，U为上三角阵

- **Formal de£nition of ILU(0):**

$$A = LU + R$$
$$NZ(L) \cup NZ(U) = NZ(A)$$
$$(A-LU)_{ij} = r_{ij} = 0 \text{ for } (i,j) \in NZ(A)$$

对位于NZ(A)的矩阵元素，该分解精确

- **This does not de£ne $ILU(0)$ in a unique way.** 一般的ILU(0)不唯一

Constructive de£nition: Compute the LU factorization of A but drop any £ll-in in L and U outside of Struct(A).

做LU分解时丢弃NZ(A)以外的矩阵元素

- **ILU factorizations are often based on i, k, j version of GE.**

不完全Cholesky分解IC(0)

For $j = 1, 2, \ldots, n$

$$l_{jj} := (a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2)^{1/2};$$

For $i = j+1, \ldots, n$

If $a_{ij} = 0$ then $l_{ij} := 0;$

else

$$l_{ij} := (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}) / l_{jj};$$

End

End

End

$$A = LL^T + R$$

$$\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{bmatrix}$$

1. 逐列生成**L**矩阵

2. 可以证明$(A-LL^T)_{ij}=0$, **for** $(i,j) \in NZ(A)$

不完全LU分解ILU(0)

Gauss消元法(KIJ version)

1. For k= 1 to n-1
2. For i= k+1 to n
3. $a_{ik} := a_{ik} / a_{kk}$
4. For j= k+1 to n
5. $a_{ij} := a_{ij} - a_{kj} a_{ik}$ （乘子）
6. end
7. end
8. end

怎么变为**ILU(0)**算法？

第**3, 5**行计算前加判断 If $(i,j) \in NZ(A)$

1. 在**A**的非零元位置 $A = LU$

2. 对于压缩行的稀疏存储，效率不高

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

元素的计算顺序：

k=1, i=2: a_{21}; a_{22}, …, a_{2n};

i=3: a_{31}; a_{32}, …, a_{3n};

… …

i=n: a_{n1}; a_{n2}, …, a_{nn};

k=2, i=3: a_{32}; a_{33}, …, a_{3n};

… …

i=n: a_{n2}; a_{n3}, …, a_{nn};

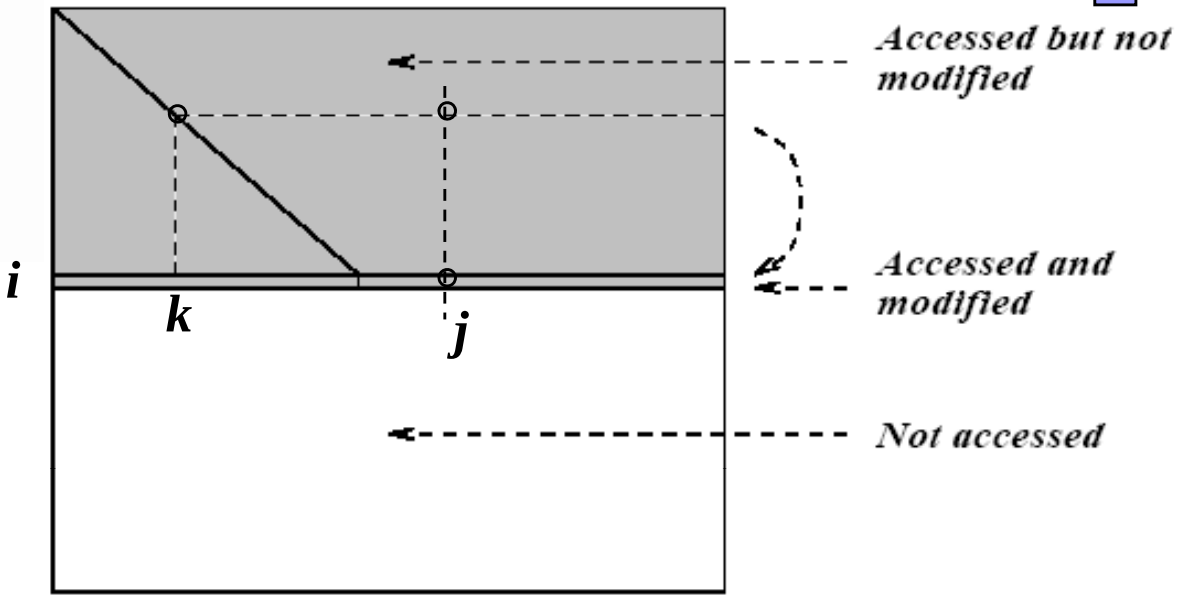
k=3, i=4: … …

What is the IKJ version of GE?

ALGORITHM : **Gaussian Elimination – IKJ Variant**

1. **For** $i = 2, \ldots, n$ **Do:**
2. **For** $k = 1, \ldots, i-1$ **Do:**
3. $a_{ik} := a_{ik}/a_{kk}$
4. **For** $j = k+1, \ldots, n$ **Do:**
5. $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
6. **EndDo**
7. **EndDo**
8. **EndDo**

常规GE：从前面取一行，多次使用来消后面的行；
IKJ variant: 考虑后面的某一行，多次使用前面不同的行来消自身。



对于行压缩的稀疏矩阵存储，效率较高，对当前行非零元排序，使用工作指针向量

ILU(0) – zero-fill ILU

ALGORITHM : ILU(0)

For $i = 2, \ldots, N$ Do:

  For $k = 1, \ldots, i-1$ and if $(i,k) \in NZ(A)$ Do:

    Compute $a_{ik} := a_{ik} / a_{kk}$

    For $j = k+1, \ldots$ and if $(i,j) \in NZ(A)$, Do:

    compute $a_{ij} := a_{ij} - a_{ik} a_{k,j}$.

  EndFor

EndFor

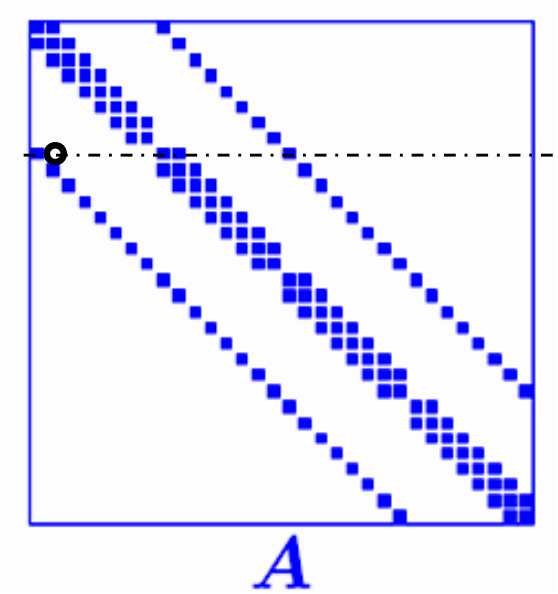
怎么证明$(A-LU)_{ij}=0$, for $(i,j) \in NZ(A)$?

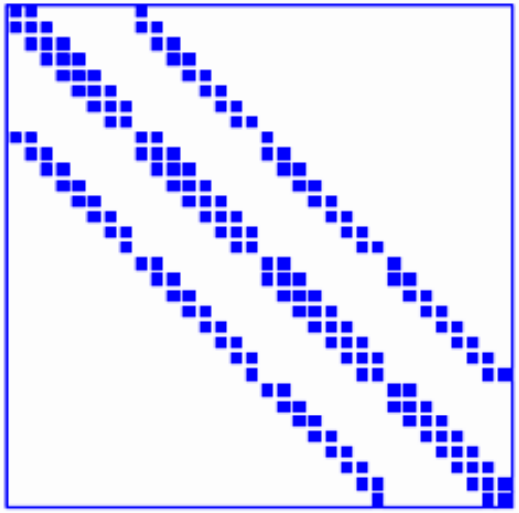
可以证明在数学上与KIJ version等价

Pattern of ILU(0) for 5-point matrix

5-point FD matrix for a 8×4 mesh

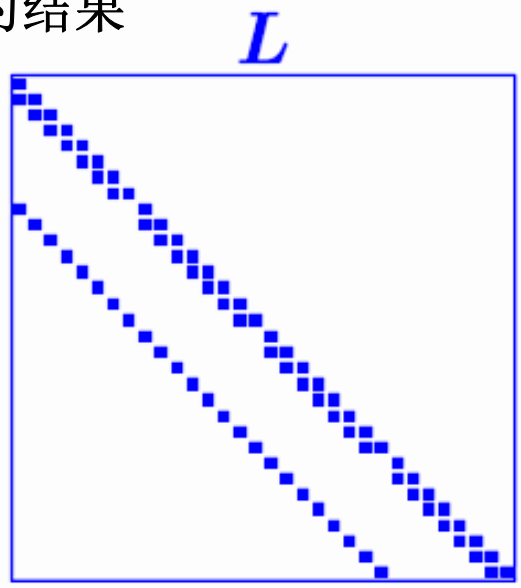
准确的LU分解会成什么样子？

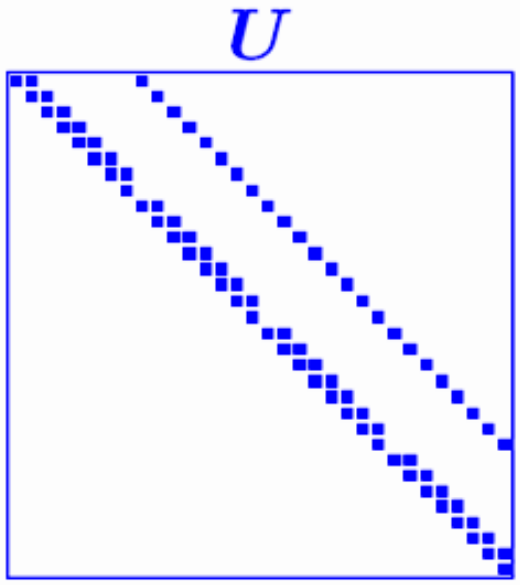




L·U的结果是什么？

ILU(0)的结果





Floating point operations
10^{-7} initial residual

GMRES without preconditioning

Matrix	Iters	Kflops	Residual	Error
F2DA	95	3841	0.32E-02	0.11E-03
F3D	67	11862	0.37E-03	0.28E-03
ORS	205	9221	0.33E+00	0.68E-04

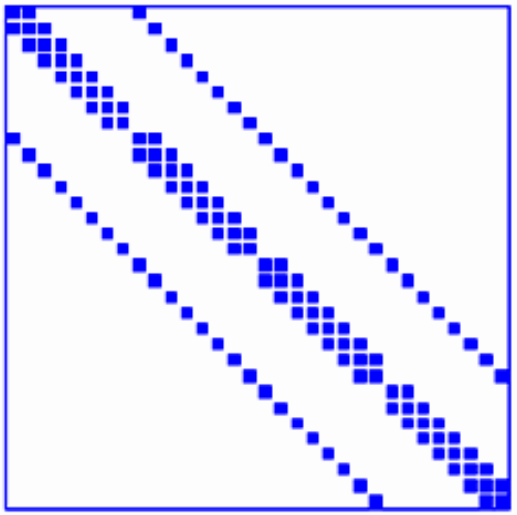
GMRES with SSOR preconditioning

Matrix	Iters	Kflops	Residual	Error	*N*
F2DA	38	1986	0.76E-03	0.82E-04	**1024**
F3D	20	4870	0.14E-02	0.30E-03	**1024**
ORS	110	6755	0.31E+00	0.68E-04	**4096**
F2DB	300	15907	0.23E+02	0.66E+00	**1030**
FID	300	99070	0.26E+02	0.51E-01	**3079**

GMRES with ILU(0) preconditioning

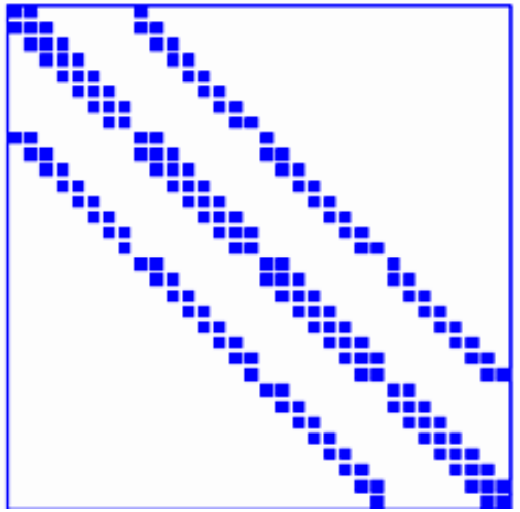
Matrix	Iters	Kflops	Residual	Error
F2DA	28	1456	0.12E-02	0.12E-03
F3D	17	4004	0.52E-03	0.30E-03
ORS	20	1228	0.18E+00	0.67E-04
F2DB	300	15907	0.23E+02	0.67E+00
FID	206	67970	0.19E+00	0.11E-03

ILU(1)



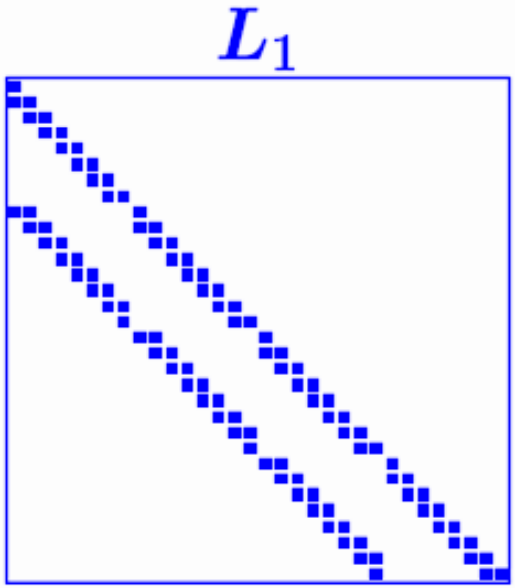
如何使ILU结果更接近准确LU分解?

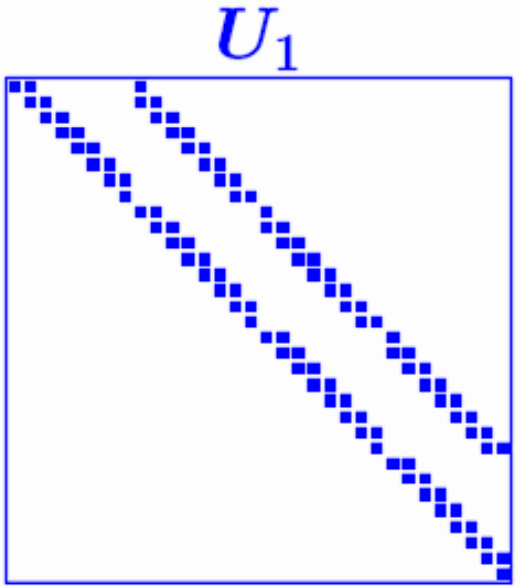
定义更接近准确LU分解的非零元pattern



增大的非零元pattern

根据这个pattern，对A进行ILU分解，得到ILU(1)的结果





Higher order ILU factorization

- Higher accuracy incomplete Choleski: for regularly structured problems, IC(p) allows p additional diagonals in L. 考虑上一页的例子

一般的ILU(p)怎么定义？

- Can be generalized to irregular sparse matrices using the notion of level of fill-in 是每个元素的属性，并随GE的过程不断更新

- Initially $Lev(a_{ij}) = \begin{cases} 0 & \text{for } a_{ij} \neq 0 \\ \infty & \text{for } a_{ij} == 0 \end{cases}$ 永远是0
- At a given step i of Gaussian elimination: (IKJ-GE的第5步进行更新)

$$Lev(a_{kj}) = \min\{Lev(a_{kj}); Lev(a_{ki}) + Lev(a_{ij}) + 1\}$$

- ILU(p) Strategy = drop anything with level of fill-in exceeding p. 超过*p*

* Increasing level of fill-in usually results in more accurate ILU and...

* ...typically in fewer steps and fewer arithmetic operations.

ALGORITHM : **ILU(p)**

For $i = 2, \ldots, N$ **Do**

For each $k = 1, \ldots, i-1$ **and if** $a_{ij} \neq 0$ **do**

Compute $a_{ik} := a_{ik} / a_{kk}$

Compute $a_{i,*} := a_{i,*} - a_{ik} a_{k,*}$. **For** $j = k+1, \ldots, n$ **Do:**

Update the levels of $a_{i,*}$

Replace any element in row i **with** $lev(a_{ij}) > p$ **by zero.**

EndFor

EndFor

- The algorithm can be split into a symbolic and a numerical phase.

Level-of-fill ➤ in Symbolic phase 符号分解阶段计算level of fill

ILU with threshold – generic algorithms

ILU(p) factorizations are based on structure only and not numerical values ➡ potential problems for non M-matrices.

ILU(0), ILU(p)并不总能进行！

- 另一种不完全分解思路：带阈值的不完全**LU**分解**(ILUT)**

补充：什么是 **M-matrix**？

DEFINITION 1.4 *A matrix is said to be an M-matrix if it satisfies the following four properties:*

1. $a_{i,i} > 0$ for $i = 1, \ldots, n$.
2. $a_{i,j} \leq 0$ for $i \neq j$, $i, j = 1, \ldots, n$.
3. A is nonsingular.
4. $A^{-1} \geq 0$.

$\Longleftrightarrow$ 3. $\rho(B) < 1$, where $B = I - D^{-1}A$.

所有元素≥0

Jacobi Iteration: $D^{-1}(D-A)$
若**A**对角占优则**Jacobi**收敛，即*B*矩阵谱半径**<1**

很多工程中遇到的矩阵均为**M-matrix**，比如电路分析的例子

Two broad approaches:

First approach [derived from direct solvers]: use any (direct) sparse solver and incorporate a dropping strategy.

扔掉数值小的矩阵元素（设值为0）

Second approach : [derived from 'iterative solvers' viewpoint]

结合IKJ-GE

1. use a (row or colum) version of the (i, k, j) version of GE;
2. apply a drop strategy for the elment l_{ik} as it is computed;
3. perform the linear combinations to get a_{i*}. Use full row expansion of a_{i*};

   矩阵的第i行
4. apply a drop strategy to £ll-ins.

ALGORITHM: ILUT (based on IKJ version of GE)

1. For $i = 1, \ldots, n$ Do:
2. $w := a_{i*}$;取第i行放到工作数组中
3. For $k = 1, \ldots, i-1$ and when $w_k \neq 0$ Do:
4. $w_k := w_k / a_{kk}$;L部分
5. Apply a dropping rule to w_k
6. If $w_k \neq 0$ then
7. $w := w - w_k * u_{k*}$ **For $j = k+1, \ldots, n$ Do:**
8. EndIf
9. EndDo
10. Apply a dropping rule to row w ;控制fill-in
11. $l_{i,j} := w_j$ for $j = 1, \ldots, i-1$
12. $u_{i,j} := w_j$ for $j = i, \ldots, n$
13. $w := 0$
14. EndDo

ILU with threshold: ILUT(k, ϵ)

- Do the i, k, j version of Gaussian Elimination (GE).
- During each i-th step in GE, discard any pivot or £ll-in whose value is below $\epsilon \|row_i(A)\|$. 第5, 10行
- Once the i-th row of $L + U$, (L-part + U-part) is computed retain only the k largest elements in both parts. 第10行

➨ Advantages: controlled £ll-in. Smaller memory overhead.

➨ Easy to implement – much more so than preconditioners derived from direct solvers.

➨ can be made quite inexpensive.

Computational results:

Matrix	Iters	Kflops	Residual	Error
F2DA	18	964	0.47E-03	0.41E-04
F3D	14	3414	0.11E-02	0.39E-03
ORS	6	341	0.13E+00	0.60E-04
F2DB	130	7167	0.45E-02	0.51E-03
FID	59	19112	0.19E+00	0.11E-03

Table 10.3 *A test run of GMRES(10)-ILUT(1, 10^{-4}) preconditioning.*

ILU(0):

Iters	Kflops
28	1456
17	4004
20	1228
300	15907
206	67970

Matrix	Iters	Kflops	Residual	Error
F2DA	7	478	0.13E-02	0.90E-04
F3D	9	2855	0.58E-03	0.35E-03
ORS	4	270	0.92E-01	0.43E-04
F2DB	10	724	0.62E-03	0.26E-03
FID	40	14862	0.11E+00	0.11E-03

Table 10.4 *A test run of GMRES(10)-ILUT(5, 10^{-4}) preconditioning.*

ILUT仍然可能无法完成: 遇到0主元

解决办法：**ILUTP (pivoting)**

另一种有效的ILU－Crout-based ILUT

通过比较**A(i,j)**值直接求出**L,U**矩阵元素

计算顺序很重要！

$$\begin{bmatrix} 1 & & & \\ \vdots & \ddots & & \\ l_{k1} & \cdots & 1 & \\ \vdots & & \vdots & \ddots \\ l_{n1} & \cdots & l_{n,k} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & \cdots & u_{1k} & \cdots & u_{1n} \\ & \ddots & \vdots & & \vdots \\ & & u_{k,k} & \cdots & u_{k,n} \\ & & & \ddots & \vdots \\ & & & & u_{n,n} \end{bmatrix}$$

1. $u_{1j} = a_{1j}, (j = 1, 2, \cdots, n)$

2. $l_{j1} = a_{j1} / u_{11}, (j = 2, 3, \cdots, n)$

计算**U**的第**k**行，**L**的第**k**列元素 $(k = 2, 3, \cdots, n)$

3. $u_{k,j} = a_{k,j} - \sum_{i=1}^{k-1} l_{k,i} u_{i,j}, (j = k, k+1, \cdots, n)$

一般的**LU**分解算法
可把***l***, ***u***都换成***a***

4. $l_{j,k} = (a_{j,k} - \sum_{i=1}^{k-1} l_{j,i} u_{i,k}) / u_{k,k}, (j = k+1, \cdots, n,\ k \neq n)$

Crout-based ILUT (ILUTC)

Remind the IKJ variant of Gauss Eliminate:

For $i = 2, \ldots, n$ **Do:**

  For $k = 1, \ldots, i-1$ **Do:**

    $a_{ik} := a_{ik} / a_{kk}$

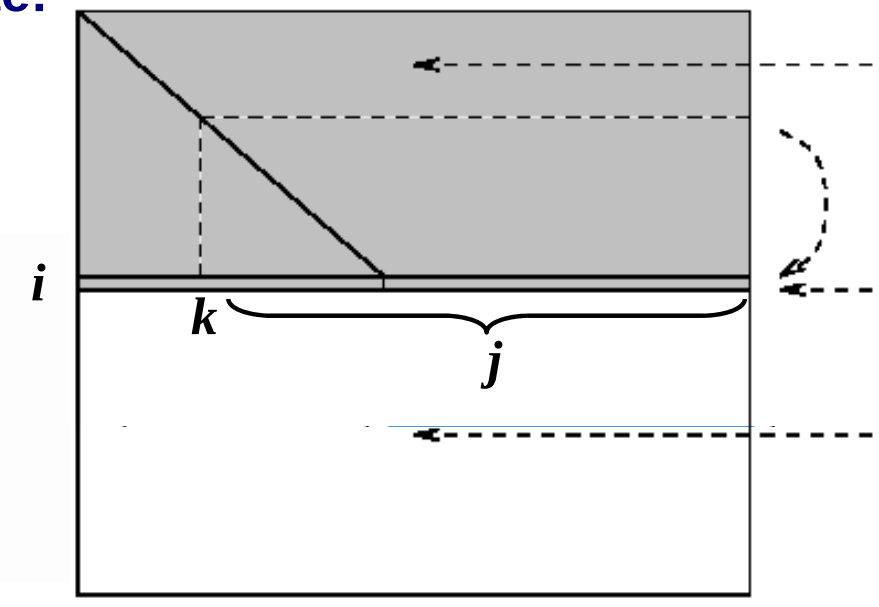
    For $j = k+1, \ldots, n$ **Do:**

      $a_{ij} := a_{ij} - a_{ik} * a_{kj}$

    EndDo

  EndDo

EndDo



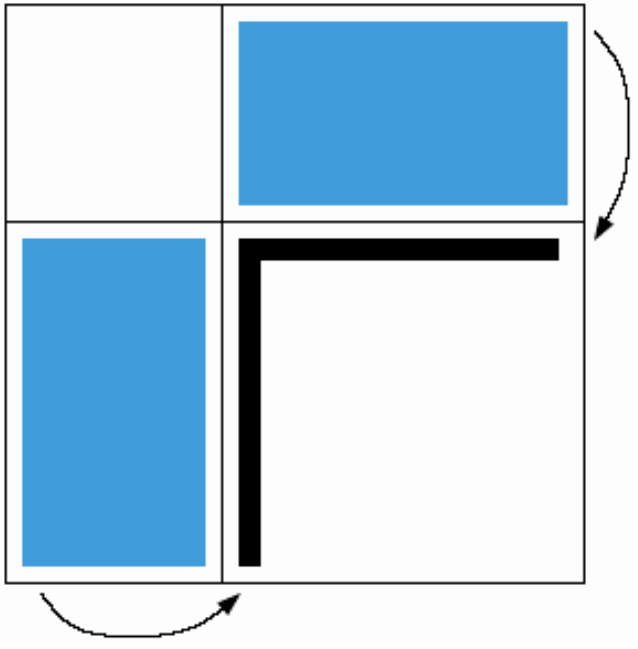
必须从左到右依次找非零元！
由于ILUT中非零元分布动态变化，带来很大计算量。

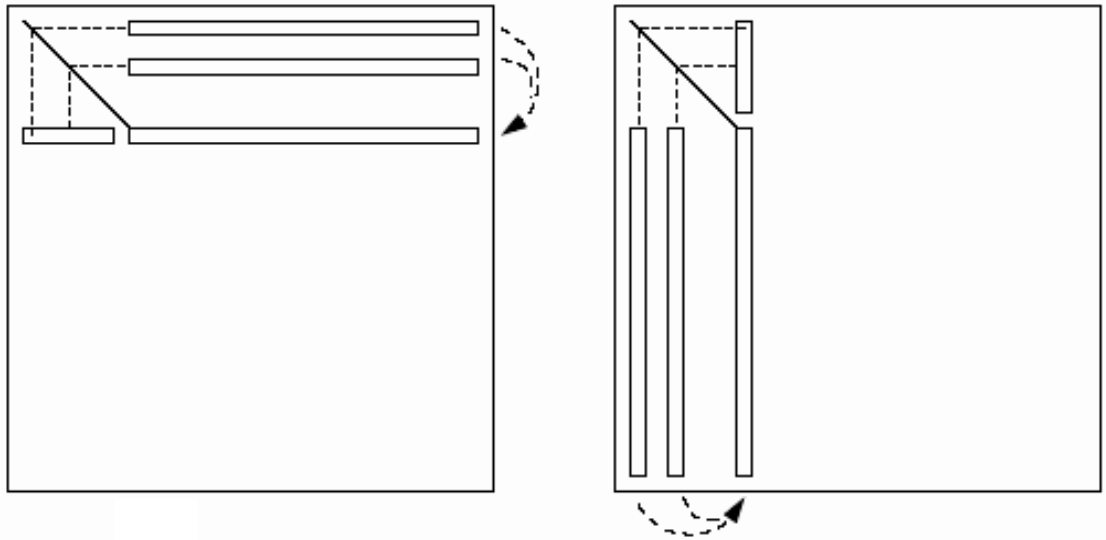
Crout versions of LU compute the k-th row of U the k-th column of L at the k-th step.

Computational pattern

Black = part computed at step k

Blue = part accessed





ALGORITHM : **Crout LU Factorization (dense case)**

1. **For** $k = 1 : n$ **Do :**
2. **For** $i = 1 : k-1$ **and if** $a_{ki} \neq 0$ **Do :**
3. $a_{k,k:n} = a_{k,k:n} - a_{ki} * a_{i,k:n}$
4. **EndDo**
5. **For** $i = 1 : k-1$ **and if** $a_{ik} \neq 0$ **Do :**
6. $a_{k+1:n.k} = a_{k+1:n,k} - a_{ik} * a_{k+1:n,i}$
7. **EndDo**
8. $a_{ik} = a_{ik}/a_{kk}$ **for** $i = k+1, \ldots, n$
9. **EndDo**

*i*的取值并不需要按顺序！
因此，非零元不需要排序

ALGORITHM : **ILUC - Crout version of ILU**

1. **For** $k = 1 : n$ **Do :**
2. **Initialize row** z: $z_{1:k-1} = 0, \quad z_{k:n} = a_{k,k:n}$ 行工作数组
3. **For** $\{i \mid 1 \leq i \leq k-1 \; and \; l_{ki} \neq 0\}$ **Do :**
4. $z_{k:n} = z_{k:n} - l_{ki} * u_{i,k:n}$
5. **EndDo**
6. **Initialize column** w: $w_{1:k} = 0, \quad w_{k+1:n} = a_{k+1:n,k}$ 列工作数组
7. **For** $\{i \mid 1 \leq i \leq k-1 \; and \; u_{ik} \neq 0\}$ **Do :**
8. $w_{k+1:n} = w_{k+1:n} - u_{ik} * l_{k+1:n,i}$
9. **EndDo**
10. **Apply a dropping rule to row** z
11. **Apply a dropping rule to column** w
12. $u_{k,:} = z; \quad l_{:,k} = w/u_{kk}, \quad l_{kk} = 1$
13. **Enddo**

Main advantages:

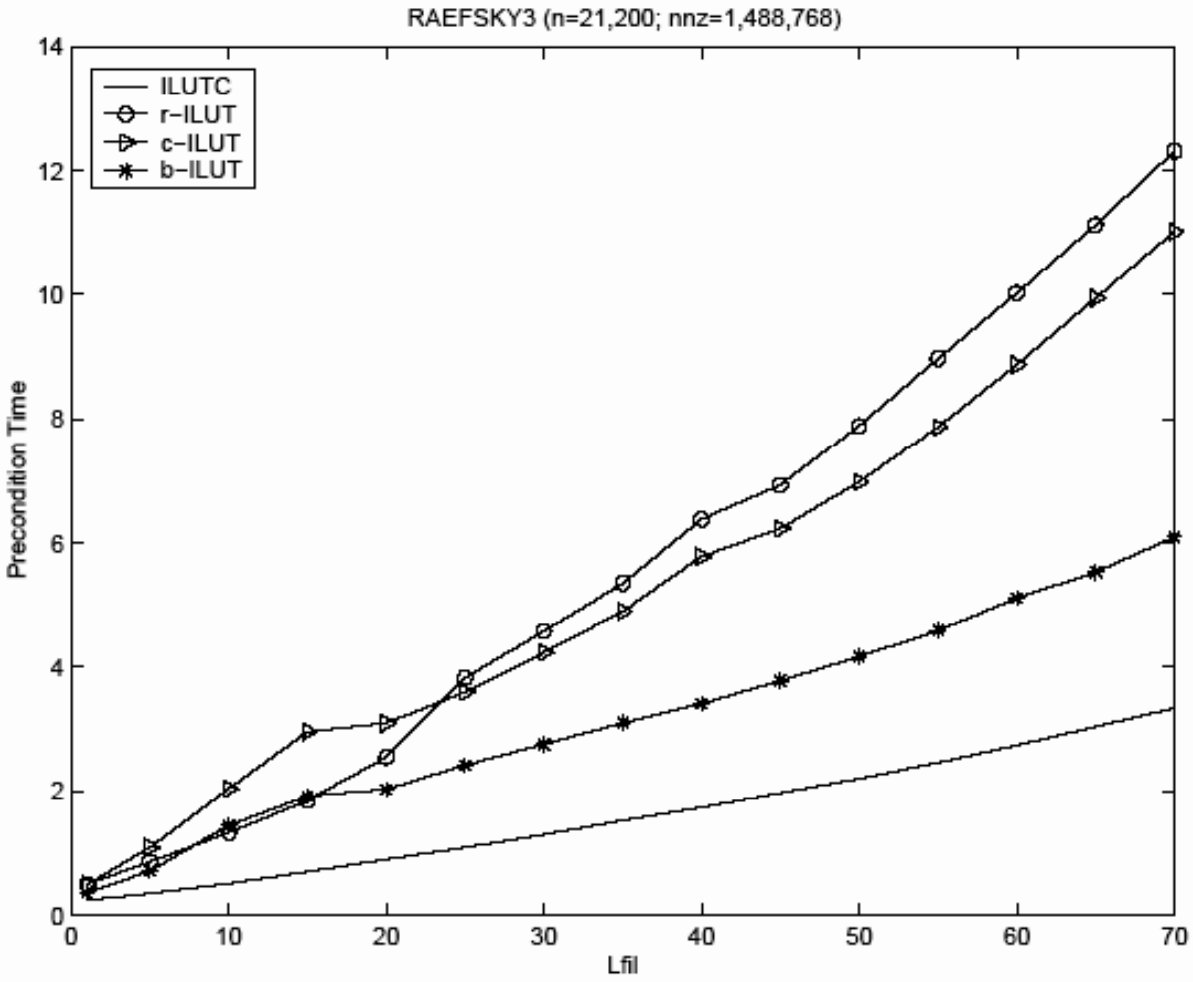
1. Less expensive than ILUT (avoids sorting)
2. Allows better techniques for dropping

- Notice that the updates to the k-th row of U (resp. the k-th column of L) can be made in any order.

- Operations in Lines 4 and 8 are sparse vector updates (must be done in sparse mode)..

Comparison with standard techniques

构造预条件的时间



基于**IKJ version**，但采用二叉树对非零元排序

ILUC最有效

Level of fill-in

Precondition time vs. L£l for ILUC (solid), row-ILUT (circles), column-ILUT (triangles) and r-ILUT with Binary Search Trees (stars)

总结ILU预条件

- 构造思路1: 原始矩阵非零元pattern + fill-in级别
  - ILU(0), IC(0), 针对多对角矩阵的ILU(p), 一般的ILU(p)
  - 缺点: 未考虑矩阵元素的数值; 数值稳定性、中断
- 构造思路2: 考虑分解过程中元素数值大小
  - ILUT(k, ε): 基于IKJ版本的高斯消元(逐行得到L, U)
  - 丢弃规则: 若某元素< ε·原矩阵A该行范数; 每行最多保留前k个较大的元素(或比原始A此行非零元数目多k)

可能比ILU(0)非零元更少

  - ILUTP(k, ε, ε_p): 带行选主元，避免ILUT算法中断(0主元)和舍入误差累计; 仅当 $\varepsilon_p |a_{ij}| > |a_{ii}|$，$a_{ij}$才是主元候选

三个参数不一定都要

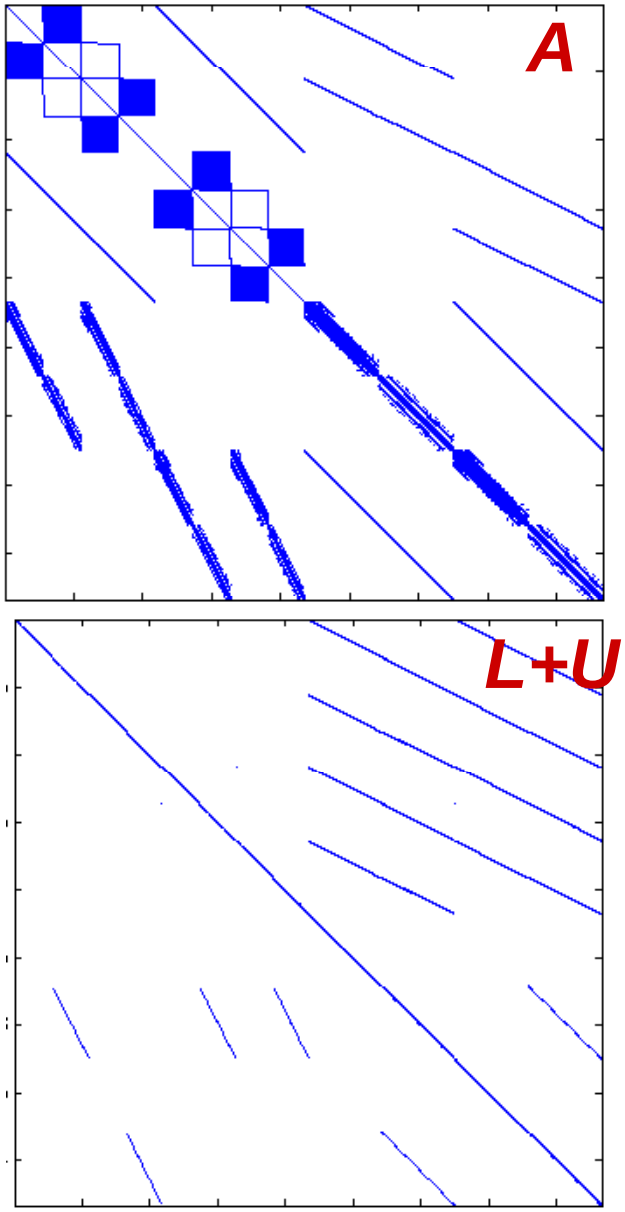
  - ILUC: 基于Crout版本LU分解(每步算U一行, L一列)，丢弃规则类似ILUT，但更灵活; 算法效率比ILUT高
  - ILUC仍可能中断，ILUC与选主元结合？

MATLAB中的Krylov子空间迭代法

- **[x,flag,relres,iter,resvec] = gmres/pcg/bicg…**
  - **gmres(A,b,restart,tol,maxit,M1,M2,x0)**
  - **pcg(A,b,tol,maxit,M1,M2,x0), bicg, …**
  - **relres是** $\|b - Ax\|/\|b\|$ **, iter是迭代步数, resvec是各迭代步残差的向量, restart是重启动参数m, tol是循环终止阈值**
- **不完全LU分解: [L,U] = ilu(A,setup)**
  - **setup.type: 'nofill'—ILU(0); 'crout'—ILUC; 'ilutp'—ILUTP**
  - **setup.droptol:控制ILUTP和ILUC丢弃规则的ε, ~ 0.xx**
  - **setup.thresh:控制ILUTP选主元的ε_p参数, ∈[0, 1]**
  - **ILUTP最稳定，采用两个参数选项**

MATLAB中的Krylov子空间迭代法

- **GMRES+ILUTP与"\"比较**
  - 矩阵A：**n=43320，nnz=731869**
  - **ilutp(0.2, 0.5), nnz(L+U)=248441**
  - 时间：**0.36+28.7=29.1s, 201**次迭代
  - **"\"时间：~ 57.7s**
  - 误差：$\|x-x^*\|_1/\|x^*\|_1=0.013$　　分量误差：
    $\|x-x^*\|_\infty/\|x^*\|_\infty=0.025$　　最大值处**1.1%**；
    最大误差处**3.1%**



Matlab命令:

```
setup.type='ilutp';
setup.droptol=0.2;
setup.thresh=0.5;
[L U P]=ilu(A, setup); spy(L+U);    % PA ≈ LU
tic; [L U]=ilu(A, setup); toc        % A ≈ LU
tic; [x, flag, relres, iter]=gmres(A, b, 500, 1e-3, 10, L, U); toc
```

其他预条件

- Jacobi –> block Jacobi
  - Index set S={1,2,…, n} is partitioned as $\cup S_i$, then

$$m_{i,j} = \begin{cases} a_{i,j} & \text{if } i \text{ and } j \text{ are in the same index subset} \\ 0 & \text{otherwise.} \end{cases}$$

  - Preconditioner is block-diagonal matrix (after ordering)
  - 变种: ”mesh neighbor method”，直接得到M^{-1}
- ILU(0) –> D-ILU (仅改变D)

$$M = (D - E)D^{-1}(D - F)$$

  - With identical form of SSOR, but different D
- ILUT –> ILUS (sparse skyline结构，针对对称矩阵)
- General block factorization methods
- 可参考: Richard Barrett, et al., *Templates for the Solution of Linear Systems Building Blocks for Iterative Methods*, SIAM Press, 1995 , http://www.netlib.org/templates

Summary of Krylov Methods

Wenjian Yu

Summary of operations

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve
JACOBI			1^a	
GS		1	1^a	
SOR		1	1^a	
CG	2	3	1	1
GMRES	$i+1$	$i+1$	1	1
BiCG	2	5	1/1	1/1
QMR	2	$8+4^{bc}$	1/1	1/1
CGS	2	6	2	2
Bi-CGSTAB	4	6	2	2

Table 2.1: Summary of Operations for Iteration i. "a/b" means "a" multiplications with the matrix and "b" with its transpose.

[a]This method performs no real matrix vector product or preconditioner solve, but the number of operations is equivalent to a matrix-vector multiply.
[b]True **SAXPY** operations + vector scalings.
[c]Less for implementations that do not recursively update the residual.

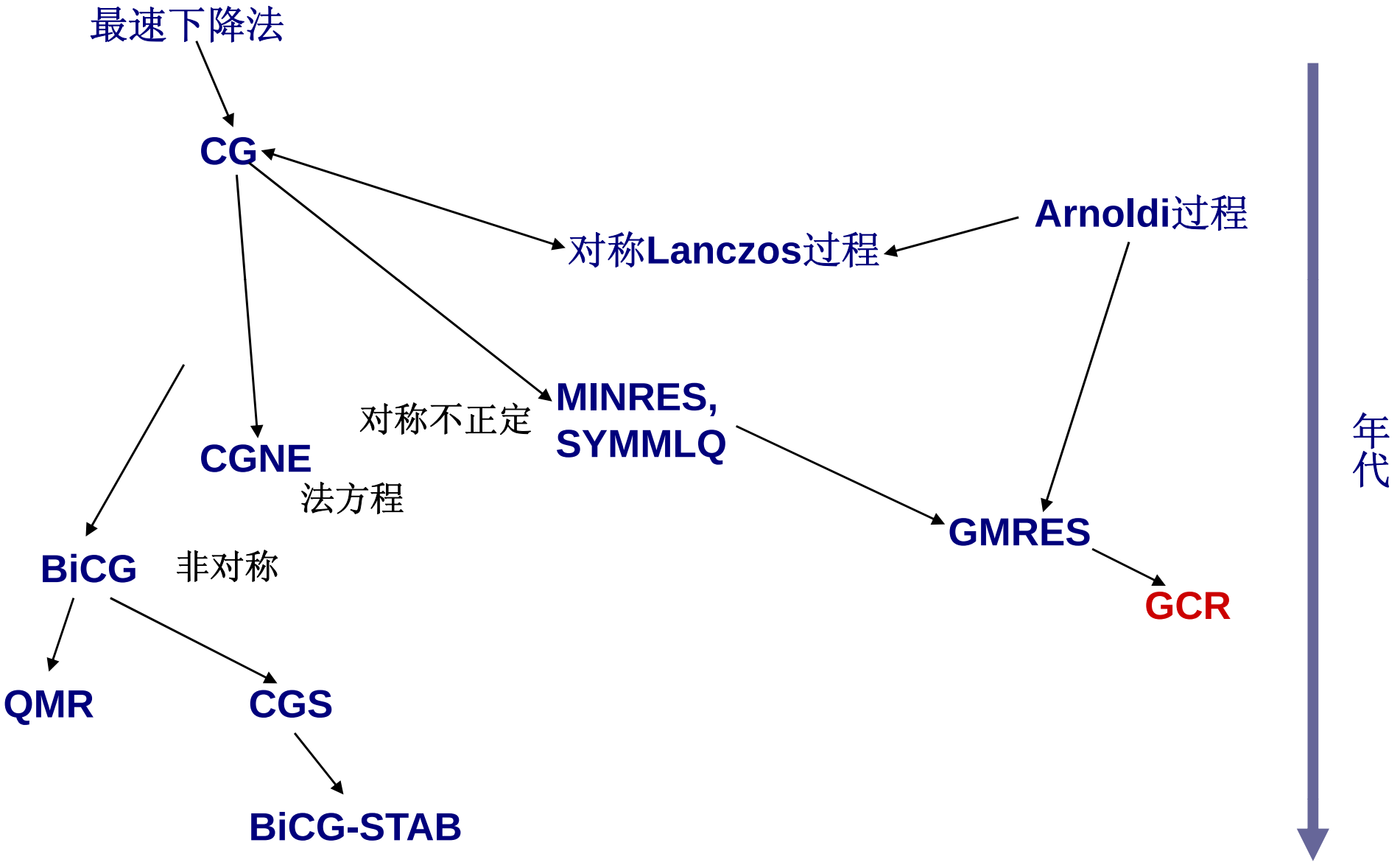
BLAS:

Saxpy – scalar times vector plus vector

Summary of Krylov subspace method

	Subspace	Characteristics	
Projection	***K, L***	**Matrix type**	**Variations**
Arnoldi Procedure	$L_m=K_m=\mathcal{K}_m(A, r_0)$		FOM, FOM(m), DIOM
GMRES	$L_m=AK_m$	Non-symmetric	GMRES(m), QGMRES, GCR
Lanczos(Sym-Arn)	$L_m=K_m$		
D-Lanczos	$L_m=K_m$	symmetric	
CG	$L_m=K_m$	symmetric	
Lanczos Bi-Orthogonalization	$K_m=\mathcal{K}_m(A, v_1)$ $L_m=\mathcal{K}_m(A^T, w_1)$		
BiCG	$K_m=\mathcal{K}_m(A, r_0)$ $L_m=\mathcal{K}_m(A^T, r_0)$	Non-symmetric	QMR, CGS, BiCGSTAB

Historical view of algorithm evolvement



直接法 VS 迭代法

- 直接法——“方法思路简单，实现复杂”
  - 基于**Gauss**消元**/LU**分解，鲁棒性好，结果准确
  - 无法处理大规模或矩阵稠密的问题**(**内存、时间**)**
  - 不能根据应用特殊性减少计算量，比如已有一个近似解，准确度要求不太高，无显式的系数矩阵
  - 算法实现很复杂，考虑**fill reduce,** 选主元，等等

便于形成独立软件包

- 迭代法——“方法思路复杂，实现简单”
  - 基于**Krylov**子空间的投影方法，鲁棒性较差
  - 处理大规模或矩阵稠密问题的唯一选择**(**省内存**)**
  - 能灵活利用各种应用要求，减小计算量
  - 算法实现较简单，系数矩阵不需修改**(**甚至不生成**)**

不便于形成独立软件包

直接法**+**迭代法：构造预条件，加快迭代法收敛速度