高等数值算法与应用（五）

Advanced Numerical Algorithms & Applications

计算机科学与技术系 喻文健

内容概要

- 稀疏矩阵线性方程组的直接解法
  - 稀疏矩阵概述与数据结构
    - 稀疏矩阵
    - 数据结构
    - **Matlab**的有关命令
  - 稀疏矩阵线性方程组的直接解法
    - 基本问题、概念: **fill-in**, 矩阵对应的图
    - 简单的,不需选主元的系数矩阵
    - 使**fill-in**”最少”的优化技术

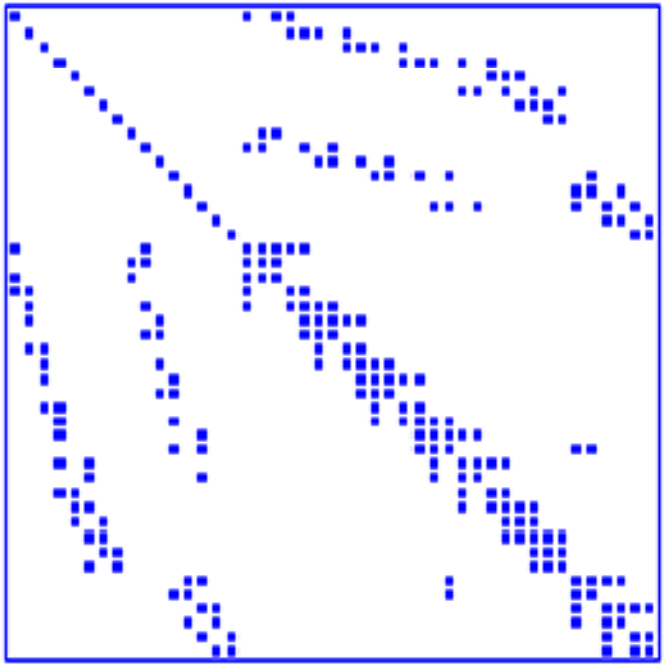
Sparse Matrix and Data Structure

With several slides from Y. Saad

Wenjian Yu

什么是稀疏矩阵?

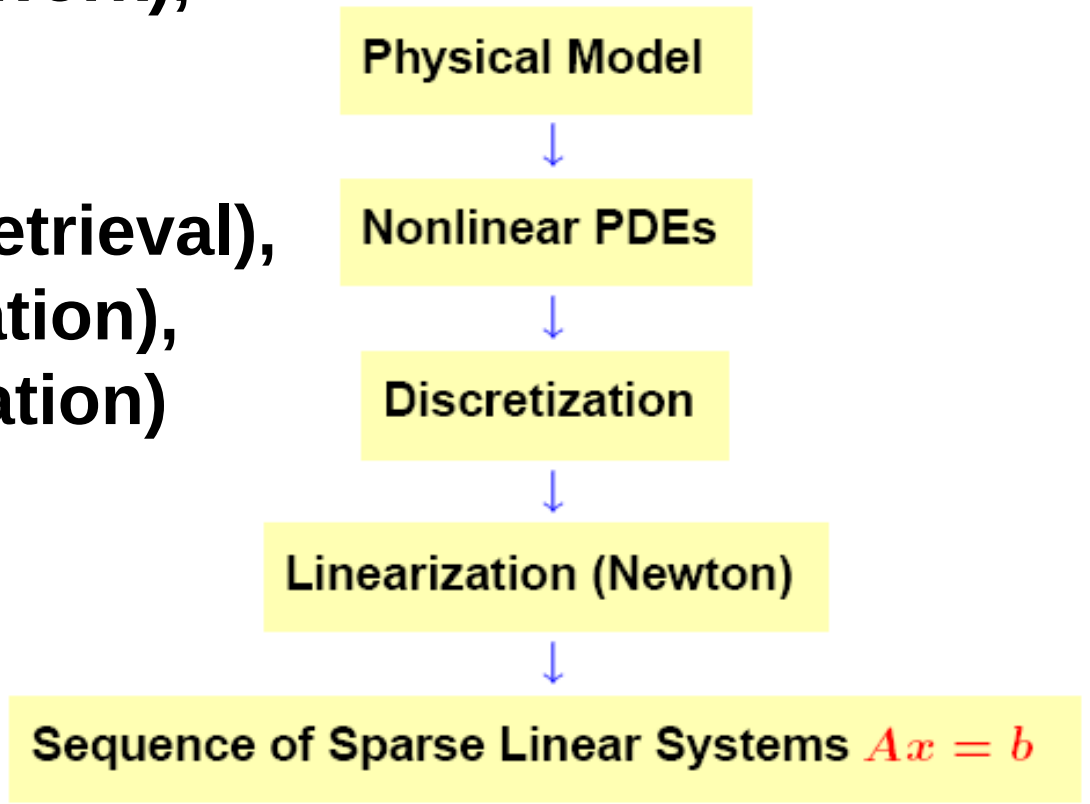
- 几种定义/描述
  - 模糊的定义: 包含很少的非零元素(nonzero entry)的矩阵
  - 从实用角度: 如果一个 $m \times n$ 矩阵仅有 $O(\min(m, n))$ 个非零元素. 这意味着每行/列中非零元数目大约为常数
  - **J. Wilkinson's definition**
    - **"matrices that allow special techniques to take advantage of the large number of zero elements."**
    - 强调特殊技术节约计算/存储量
  - 其他定义: 非零元数目关于矩阵维数m, 或n缓慢增长



稀疏矩阵的非零元分布图

什么是稀疏矩阵?

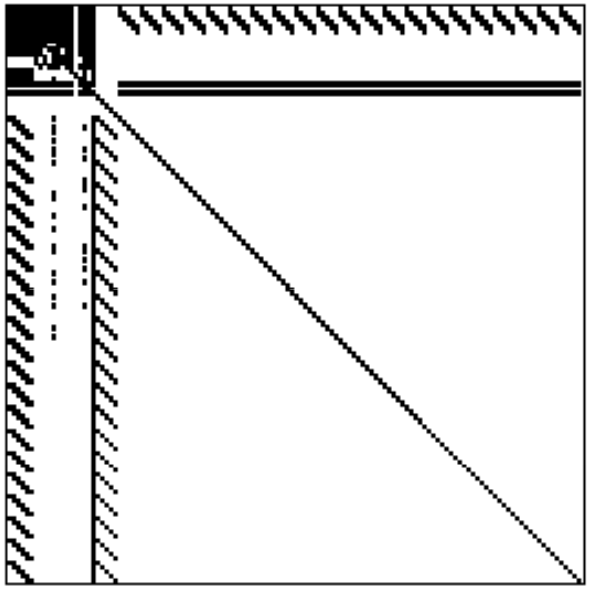
- 稀疏矩阵的应用领域
  - **结构工程(structural engineering),**
    计算流体力学(computational fluid dynamics),
    油藏数值仿真(reservoir simulation),
    电力网络(electrical network),
    优化问题(optimation),
    Google PageRank,
    信息抽取(information retrieval),
    电路仿真(circuit simulation),
    器件仿真(device simulation)



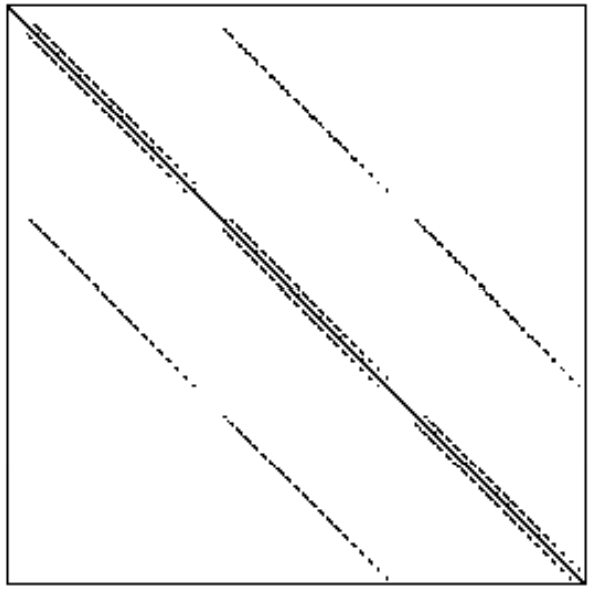
什么是稀疏矩阵?

- 稀疏矩阵技术的目的
  - 通过不存储/操作零矩阵元素，高效率地执行矩阵运算
  - 例子：将两个n阶稠密方阵相加需要$O(n^2)$次运算; 将稀疏阵A和B相加，需要O(nnz(A)+nnz(B))次运算, nnz(X)代表矩阵X中非零元的数目
  - 对于典型的偏微分方程(PDE)的离散方法, 有限差分(FDM)或有限元法(FEM), 其生成矩阵的nnz(X)为O(n)
  - 保证矩阵的稀疏性对提高相关计算的效率很关键
  - 注意: 如果处理得当，很多矩阵进行分解得到的矩阵L和U可能是稀疏的，但逆矩阵A^{-1}通常是稠密的
  - 按非零元分布特点, 稀疏矩阵分为两类: 结构化的(structured), 非结构化的(unstructured)

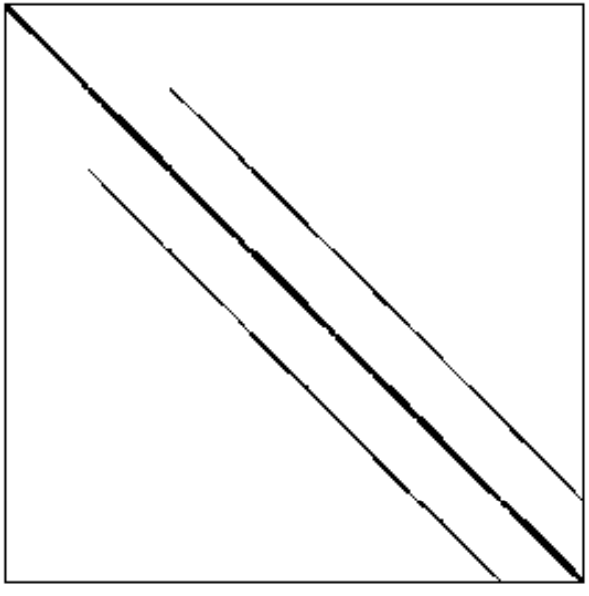
Nonzero patterns of a few sparse matrices



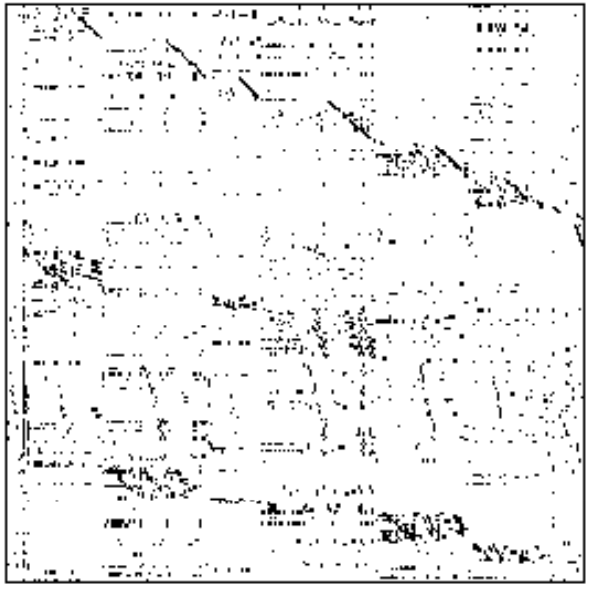
ARC130



SHERMAN5



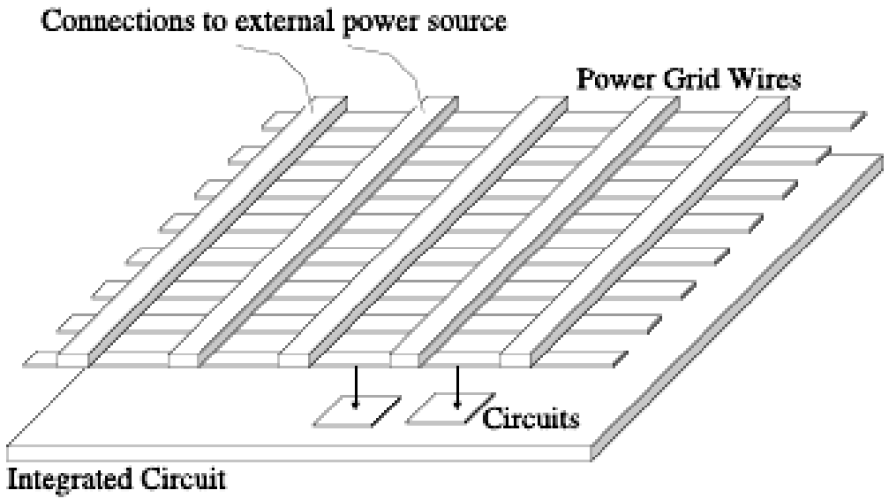
PORES3



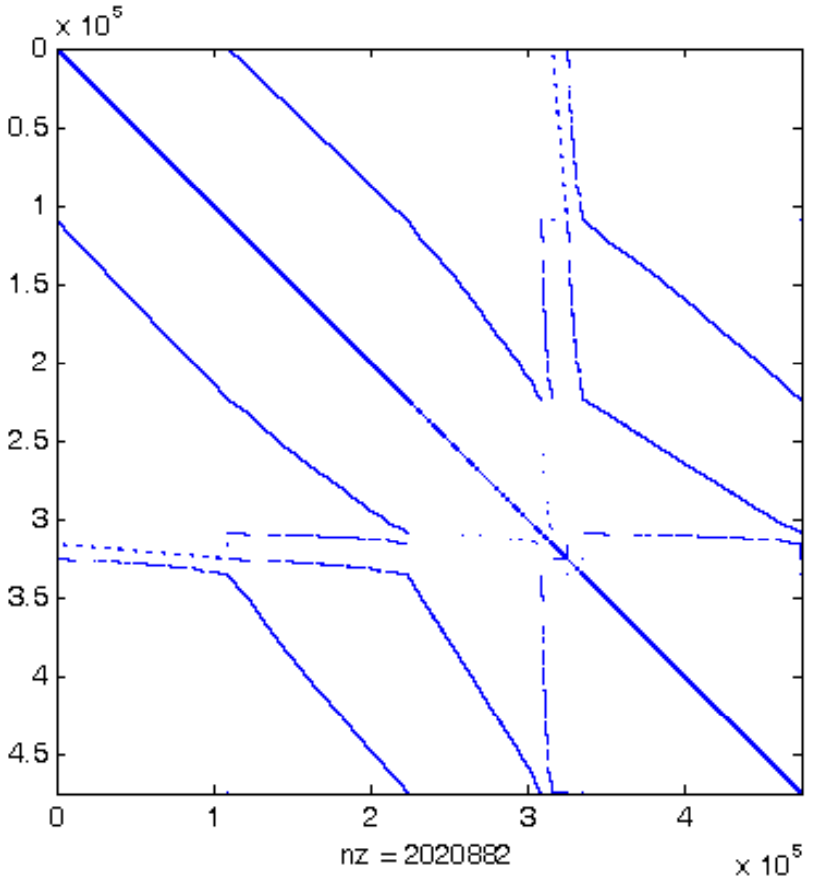
BP_1000

一个实际例子

- 集成电路供电网仿真
  - DC分析：求解电阻网络电路中的节点电压
  - 一个来自IBM的例子
    - **n= 474,524**
    - **nnz= 2,020,882**
    - 在**Matlab**中, 占**26M**内存
    - 高效的稀疏线性方程组求解算法, **t= 15s**
    - 进行**Cholesky**分解，内存不够!



集成电路供电线网的一部分



稀疏矩阵的数据结构

- 基本观点
  - 存储稀疏矩阵的数据结构对高效率的运算非常关键
  - 基本的矩阵运算(例如矩阵与向量乘法)的算法设计依赖于存储矩阵的数据结构
  - 很多种数据结构，其中一些仅对稀疏线性方程组的迭代解法有用，对直接解法无用
  - 常用的数据结构
    - **COO: coordinate,** 三元组
    - **CSR, CSC:** 压缩稀疏行(列), 及其变种
    - **DIA:** 对角线
    - **BSR:** 分块稀疏行

The coordinate format (COO)

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

AA	JR	JC
12.	5	5
9.	3	5
7.	3	3
5.	2	4
1.	1	1
2.	1	4
11.	4	4
3.	2	1
6.	3	1
4.	2	2
8.	3	4
10.	4	3

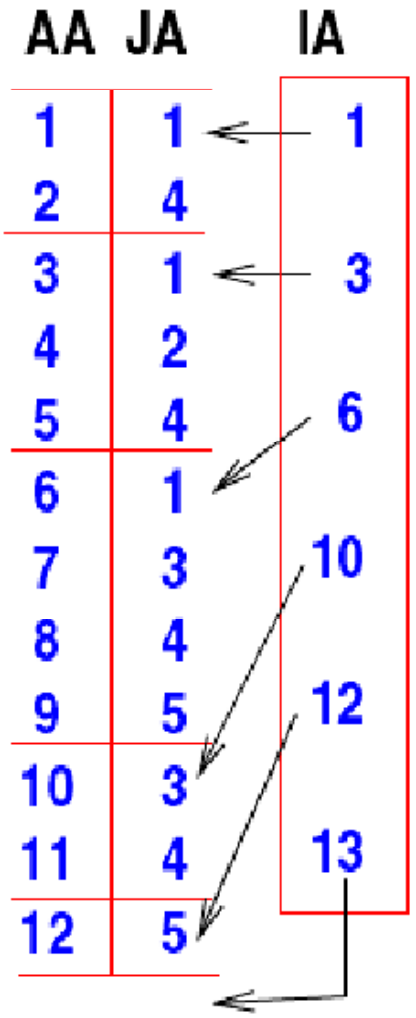
- Simplest data structure -
- Used in many packages as 'entry' format

一种类似结构：十字交叉链表

按行、列组织矩阵非零元；用于早期的线性方程组直接解法，但效率较低。

Compressed Sparse Row (CSR) format

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$



- IA(j) points to beginning or row j in arrays AA, JA
- Related formats: Compressed Sparse Column format, Modified Sparse Row format (MSR).
- Used predominantly in Fortran & portable codes [e.g. Metis] – what about C?

CSR (CSC) format - C-style

*** CSR: Collection of pointers of rows & array of row lengths**

```
typedef struct SpaFmt {
/*-----------------------------------------
| C-style CSR format - used internally
| for all matrices in CSR/CSC format
|----------------------------------------*/
  int n;          /* size of matrix          */
  int *nzcount;   /* length of each row      */
  int **ja;       /* to store column indices */
  double **ma;    /* to store nonzero entries */
} SparMat;
```

ma[i][*] == entries of i-th row (col.); 没有规定排列顺序

ja[i][*] == col. (row) indices,

nzcount[i] == number of nonzero elmts in row (col.) i

与标准的**CSR**有何不同？

ja, ma是指针数组(二维变长)，数组**nzcount**的长度

The Diagonal (DIA) format

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

$$\mathrm{DIAG}(i,j) \leftarrow a_{i,i+\mathrm{ioff}(j)}$$

$$DA = \begin{pmatrix} * & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & * \\ 11 & 12. & * \end{pmatrix}$$

IOFF = -1 0 2

离主对角线的偏差

BLOCK MATRICES

Block sparse row (BSR)

$$A = \begin{pmatrix} 1. & 2. & 0. & 0. & 3. & 4. \\ 5. & 6. & 0. & 0. & 7. & 8. \\ 0. & 0. & 9. & 10. & 11. & 12. \\ 0. & 0. & 13. & 14. & 15. & 16. \\ 17. & 18. & 0. & 0. & 20. & 21. \\ 22. & 23. & 0. & 0. & 24. & 25. \end{pmatrix}$$

也可以按存成行

$$AA = \begin{array}{|cc|cc|cc|} \hline 1. & 3. & 9. & 11. & 17. & 20. \\ 5. & 7. & 13 & 15 & 22. & 24. \\ 2. & 4. & 10. & 12. & 18. & 21. \\ 6. & 8. & 14. & 16. & 23. & 25. \\ \hline \end{array}$$

JA = 1 5 3 5 1 5

IA = 1 3 5 7

- Each column in AA holds a 2 x 2 block. JA(k) = col. index of (1,1) entries of k-th block. AA may be declared as AA(2,2,6)

Sparse matrices – data structure in C

- **Recall:**

```
typedef struct SpaFmt {
/*----------------------------------------
| C-style CSR format - used internally
| for all matrices in CSR format
|----------------------------------------*/
  int n;
  int *nzcount;  /* length of each row */
  int **ja;      /*  to store column indices  */
  double **ma;   /*  to store nonzero entries */
} CsMat, *csptr;
```

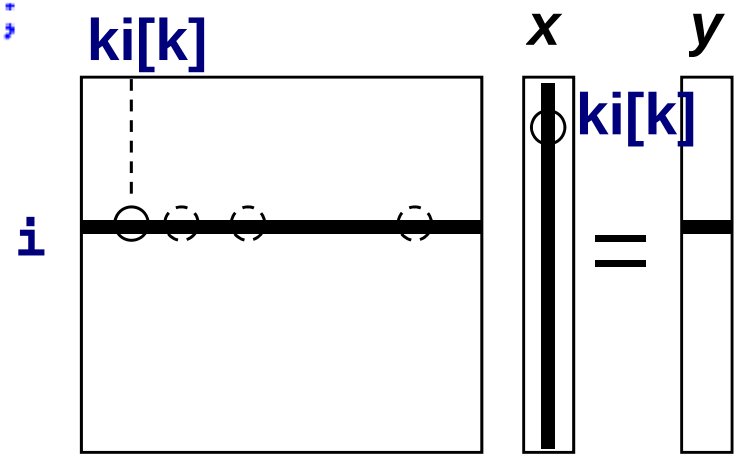
- **Can store rows of a matrix (CSR)**
- **or its columns (CSC)**
- **How to perform the operation $y = A * x$ in each case?**

矩阵与向量的乘法

Matvec – row version

```
void matvec( csptr mata, double *x, double *y )
{
    int i, k, *ki;
    double *kr;
    for (i=0; i<mata->n; i++) {
        y[i] = 0.0;
        kr = mata->ma[i];      // 第i行的非零元素值
        ki = mata->ja[i];      // 第i行非零元的列号
        for (k=0; k<mata->nzcount[i]; k++)
            y[i] += kr[k] * x[ki[k]];
    }
    return;
```

CSR存储格式



- Uses sparse dot products

计算量：nnz(A)次乘法

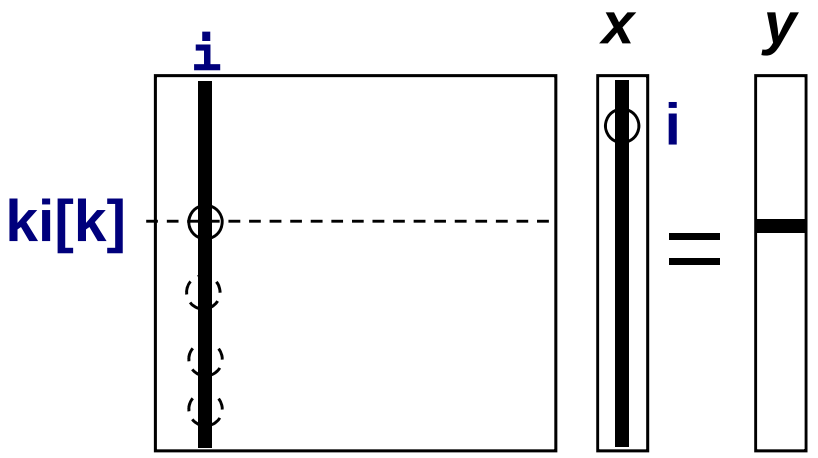
Matvec – Column version

```
void matvecC( csptr mata, double *x, double *y )
{
  int n = mata->n, i, k, *ki;
  double *kr;
  for (i=0; i<n; i++)
    y[i] = 0.0;
  for (i=0; i<n; i++) {
    kr = mata->ma[i];          // 第i列的非零元素值
    ki = mata->ja[i];          // 第i列非零元的行号
    for (k=0; k<mata->nzcount[i]; k++)
      y[ki[k]] += kr[k] * x[i];
  }
  return;
```

CSC存储格式

- Uses sparse SAXPY

Scalar x vector + vector
标量乘以向量x+向量y



与稀疏矩阵有关的Matlab命令

- 生成或转换成稀疏矩阵 (输入数据结构COO, 内部CSC)
  - sparse(X) or sparse(i,j,s,m,n); X = full(A)
- 显示矩阵非零元分布： spy(X)
- 构造特殊的稀疏矩阵
  - speye(n,m); spones(pattern); spdiags(B, d, m, n)
- 随机稀疏矩阵生成器
  - sprand(S) or sprand(m, n, density)
- 统计非零元数目，获取非零元信息
  - nnz(X); nonzeros(A), find(X)
- 其他辅助： whos; \, lu, qr, svd等各种运算都支持稀疏矩阵

参考资料

[见网络学堂/教学资源]

- ***Direct Methods for Sparse Linear Systems*, T. A. Davis, SIAM, Philadelphia, Sept. 2006.**
- ***Iterative Method for Sparse Linear Systems*, Y. Saad, SIAM, Philadelphia, 2000.**
- ***Sparse Matrix Computations*, CSE Department, University of Minnesota, 2009.**
- Matrix market

  "http://math.nist.gov/MatrixMarket/"
- Florida collection

  "http://www.cise.ufl.edu/research/sparse/matrices/"
- SUPERLU web-page

  "http://crd.lbl.gov/∼xiaoye/SuperLU"

Direct Method for Solving Sparse Matrix

With several slides from T. Davis

Wenjian Yu

内容概要

- 稀疏矩阵的直接解法
  - 思想: 高斯消去法(LU, Cholesky分解)的应用
  - 采用稀疏存储结构，如何处理填入(fill in)?
  - 简单的, 不需选主元的几种情况
    - 结构化矩阵: 条带状矩阵(band matrix)
    - 非结构化矩阵:上 / 下三角矩阵 — lsolve算法
    - 非结构化矩阵: 对称正定矩阵
    - 非结构化矩阵: 非对称矩阵
  - 使Fill-in”最少”的优化技术
    - 矩阵重排列
    - 同时考虑数值选主元

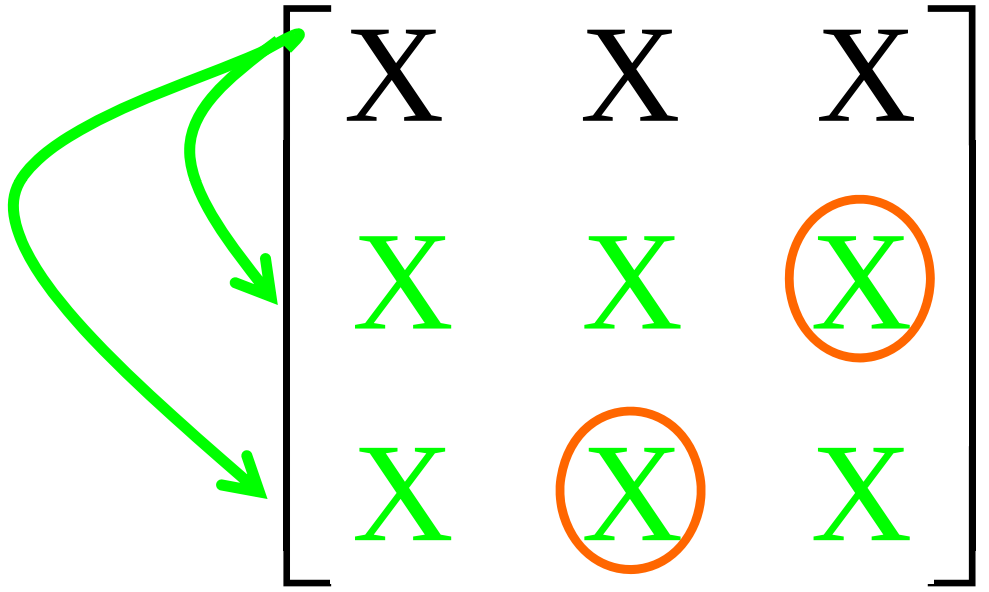
对大规模矩阵，fill-in
可能使内存溢出!

稀疏矩阵的“填入元” – 例1

系数矩阵的非零元结构

$$\begin{bmatrix} X & X & X \\ X & X & 0 \\ X & 0 & X \end{bmatrix}$$

一次高斯消去步骤后



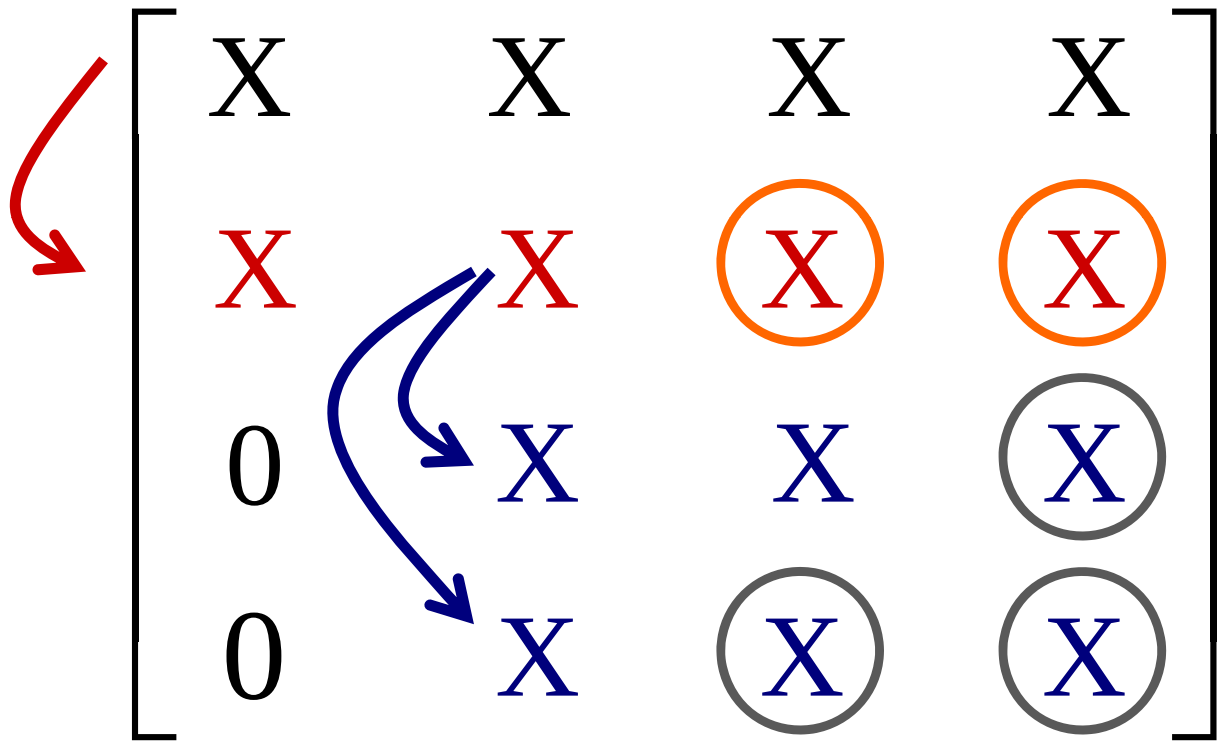
X—非零元

(X)—填入元(**Fill-in**)

Cholesky分解的填入元情况是一样的!

稀疏矩阵的“填入元” – 例2

消去过程中“填入元”的传播



一般的非对称矩阵

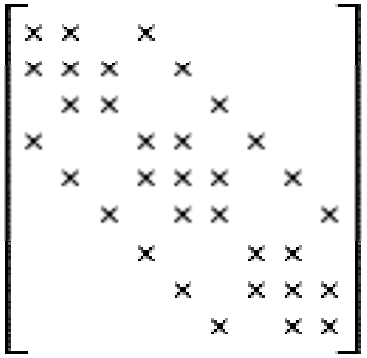
第1步的“填入元”导致第2步的“填入元”

1. 满足什么条件在(i,j)处发生fill-in? (观察主元所在的第k行、k列)
2. 证明: 对”结构对称”矩阵，消去过程发生fill-in的位置也是对称的

简单系数矩阵的直接解法—条带状矩阵

定义: $a_{ij} = 0$ for all $|i-j| > \beta$

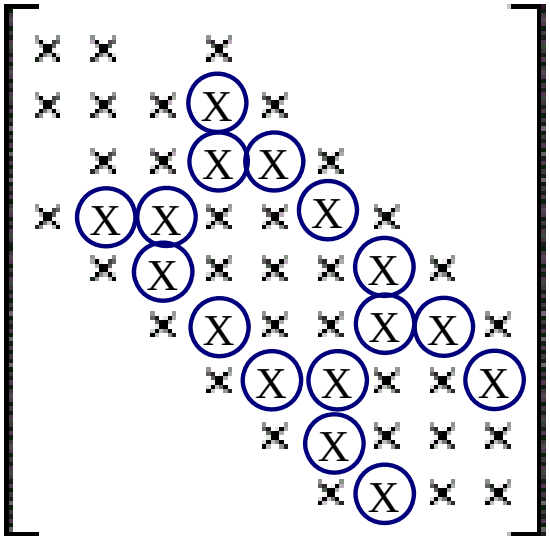
β is the *bandwidth* of A

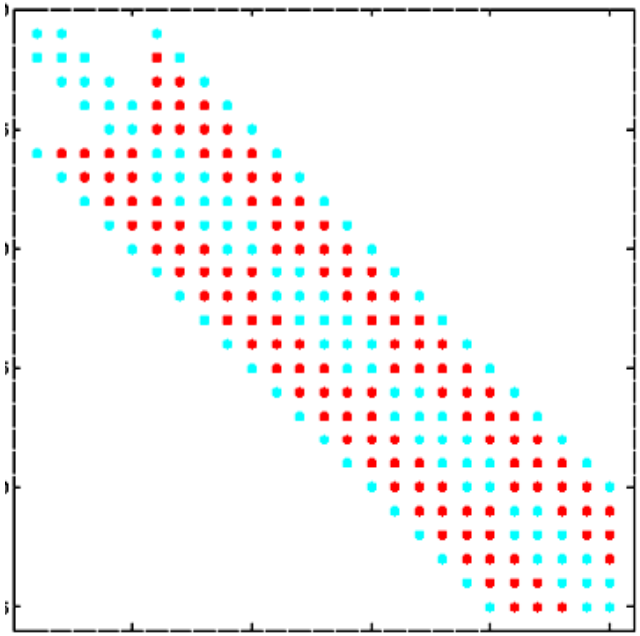


典型存储方式：**DIA**

高斯消去过程的填入情况 —— **LU分解后L, U的非零元分布**

- 若不选主元
  - 带宽不增大
- 若选主元
  - 带宽可能增大
  - 但不会翻倍？





对怎样的系数矩阵，高斯消去时可不选主元？

条带状矩阵

对怎样的矩阵**A**做**LU**分解不需选主元？

- 按行对角占优(diagonally dominant)

$$A \in \mathbb{R}^{n\times n},\ |a_{ii}| \geq \sum_{j=1,j\neq i}^{n} |a_{ij}|$$

且至少一个i使>成立

按列对角占优
按行严格对角占优
按列严格对角占优

- 定理：对于按行(列)对角占优的矩阵***A***，不选主元的高斯消元过程(即**LU**分解)是稳定的 **[T. Davis'06]**
- 对于对称正定矩阵，也可不选主元，做**Cholesky**分解

在不选主元的情况下，对带状矩阵为系数矩阵的线性方程组，求解算法可以非常简单、高效。

- 重点讨论三对角矩阵

Tridiagonal Matrices

Consider tridiagonal matrix, for example

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ m_2 & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & m_{n-1} & 1 & 0 \\ 0 & \cdots & 0 & m_n & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 \\ 0 & d_2 & c_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & d_{n-1} & c_{n-1} \\ 0 & \cdots & \cdots & 0 & d_n \end{bmatrix}$$

若不需要选主元，采用直接**LU**分解算法的思想构造三对角阵**LU**分解的算法：

L^{-1}, U^{-1}, A^{-1}?

$d_1 = b_1$
for $i = 2$ **to** n
$\quad m_i = a_i / d_{i-1}$
$\quad d_i = b_i - m_i c_{i-1}$
end

线性复杂度

对一般的条带状矩阵，若带宽β较小，且不需选主元，可不考虑**fill-in**(即认为带宽内稠密)，**LU**分解的时间和空间复杂度分别为$O(\beta^2 n)$和$O(\beta n)$

非结构化稀疏矩阵需用“图”表示

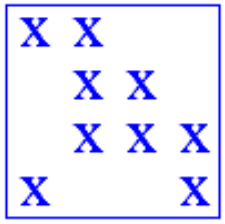
- **Graph theory is a fundamental tool in sparse matrix techniques.**

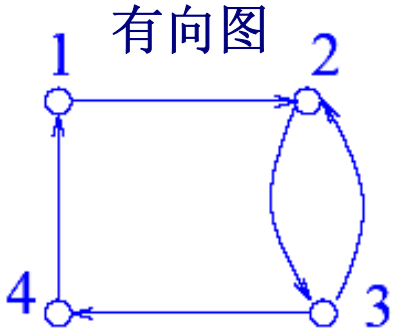
Graph $G = (V, E)$ of an $n \times n$ matrix A defined by

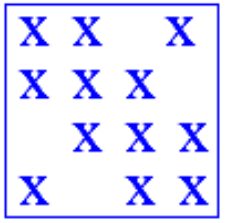
Vertices $V = \{1, 2, ..., N\}$.

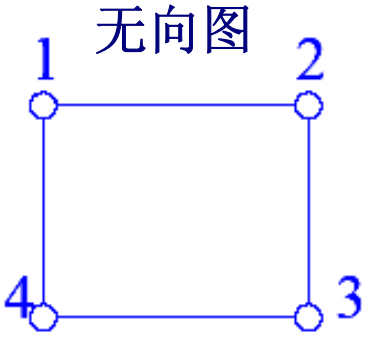
Edges $E = \{(i, j) | a_{ij} \neq 0\}$. 通常矩阵的对角元≠0, 因此不表示

- **Graph is undirected if matrix has symmetric structure: $a_{ij} \neq 0$ iff $a_{ji} \neq 0$.**









邻接图<—>邻接矩阵

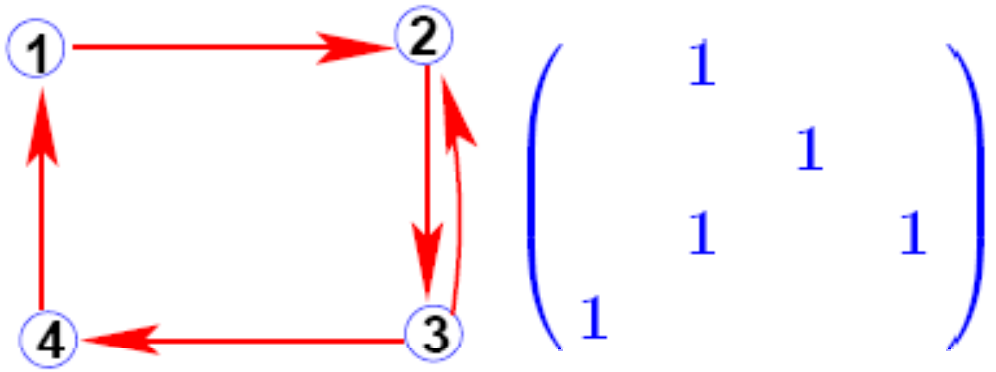
图的基本术语

- 若$(u, v) \in E$, 则称v与u相邻　　adjacent to
- 若图是有向的，(u, v)是从顶点u的出边，是到顶点v的入边　(outgoing/incoming edge)
- $Adj(i) = \{j \mid j \text{ adjacent to } i\}$　　邻点集合
- 顶点v的度(degree): 连到v的边数(有向图, 出度/入度)
- $|Adj(i)| = deg(i)$
- 若$V' \subseteq V$且$E' \subseteq E$，　$G' = (V', E')$为$G = (V, E)$的子图
- 路径(path)为一有序列的点$w_0, w_1, \ldots, w_k$，　满足$(w_i, w_{i+1}) \in E$, $i = 0, \ldots, k-1$. 路径的长度为边的数目.
- 封闭的路径为环(circle)，　即满足$w_k = w_0$的路径
- 无环图acyclic, 森林forest, 树tree, 有向无环图DAG…

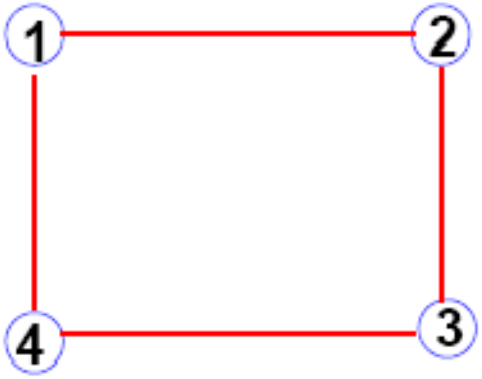
图的数据结构

- 图的表示
  - 邻接矩阵***A*** **(adjacency matrix)**

$$a_{i,j} = \begin{cases} 1 \text{ if } (i,j) \in E \\ 0 \text{ Otherwise} \end{cases}$$



$$\begin{pmatrix} & 1 & & \\ & & 1 & \\ & 1 & & 1 \\ 1 & & & \end{pmatrix}$$



$$\begin{pmatrix} & 1 & & 1 \\ 1 & & 1 & \\ & 1 & & 1 \\ 1 & & 1 & \end{pmatrix}$$

  - 数据结构：实际上，图就是“一个没有数值的稀疏矩阵”
  - 因此，可使用任何稀疏矩阵存储结构，只需忽略矩阵元素的信息 **(COO, CSR)**

怎样根据**A**的图得到A^2, A^k的图？

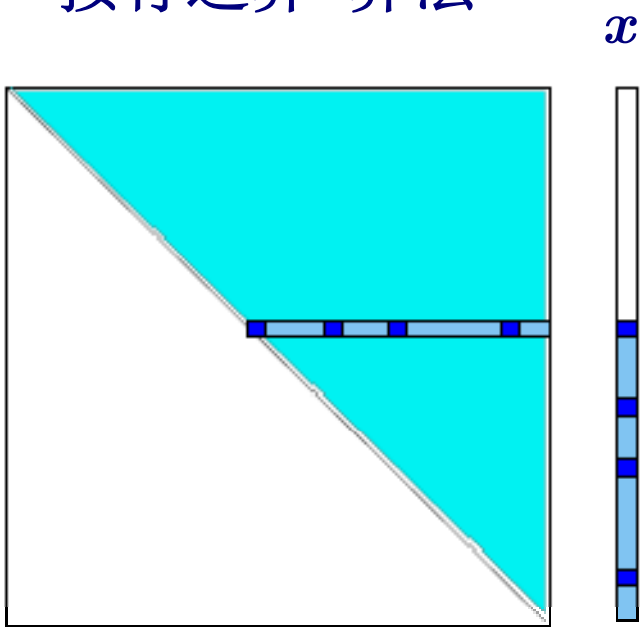
非结构化的上三角稀疏阵

- 没有选主元的问题(对角元≠0), 也没有fill-in
- 稀疏存储结构对算法的影响: 两种情况

先看稠密矩阵例子:

$$\begin{pmatrix} 2 & 4 & 4 \\ 0 & 5 & -2 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix}$$

"按行运算"算法



For $i = n : -1 : 1$ do:

$t := b_i$

For $j = i + 1 : n$ do

$t := t - a_{ij}x_j$

End

$x_i = t/a_{ii}$

End

上三角稀疏矩阵(CSR存储格式)

- 求解稀疏上三角矩阵的C语言程序(按行运算)

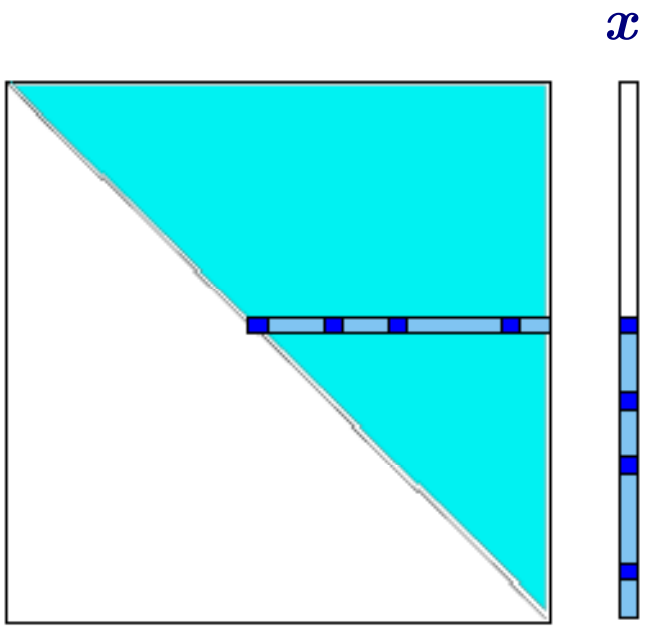
```
void Usol(csptr mata, double *b, double *x)
{
  int i, k, *ki;
  double *ma;
  for (i=mata->n-1; i>=0; i--) {
    ma = mata->ma[i];
    ki = mata->ja[i];
    x[i] = b[i] ;
// Note: diag. entry avoided
    for (k=1; k<mata->nzcount[i]; k++)
      x[i] -= ma[k] * x[ki[k]];
    x[i]  /= ma[0];
  }
  return;
}
```

矩阵采用CSR存储格式，且假设存储行的数组第一个为对角元

这里假设右端项b稠密！

上三角稀疏矩阵(CSC存储格式)

- CSC存储格式使”按行运算”麻烦
  - 需连续地取矩阵一行元素
  - 改为采用”按列运算”的算法
  - 稠密矩阵的”按列运算”算法



For $j = n : -1 : 1$ do:

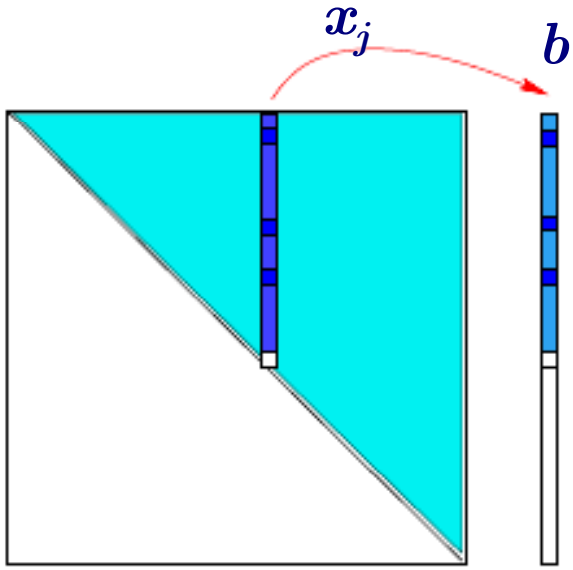
$x_j = b_j / a_{jj}$

For $i = 1 : j - 1$ do

$b_i := b_i - x_j * a_{ij}$

End

End



修改了b向量

上三角稀疏矩阵(CSC存储格式)

- 求解稀疏上三角矩阵的C语言程序(按列运算)

```
void UsolC(csptr  U   , double *b, double *x)
{
   int i, k, *ja;
   double *ma;
   for (i=   U->n-1; i>=0; i--) {
          ja = U->ja[i];
          ma = U->ma[i];
          x[i]=b[i]/ma[0];
// Note: diag. entry avoided
          for( j = 1; j < U->nzcount[i]; j++ )
              b[ja[j]] -= ma[j] * x[i];
   }
  return;
}
```

矩阵采用CSC存储格式，且假设列数组中第一个元素为对角元

浮点运算次数为nnz(U). 若右端项b稀疏，还可能更少！

非结构化的下三角稀疏阵

- 采用CSC存储格式，”按列运算”求解算法
  - 右端项**b**稠密

$$Lx = b$$

$x = b$

for $j = 1{:}n$

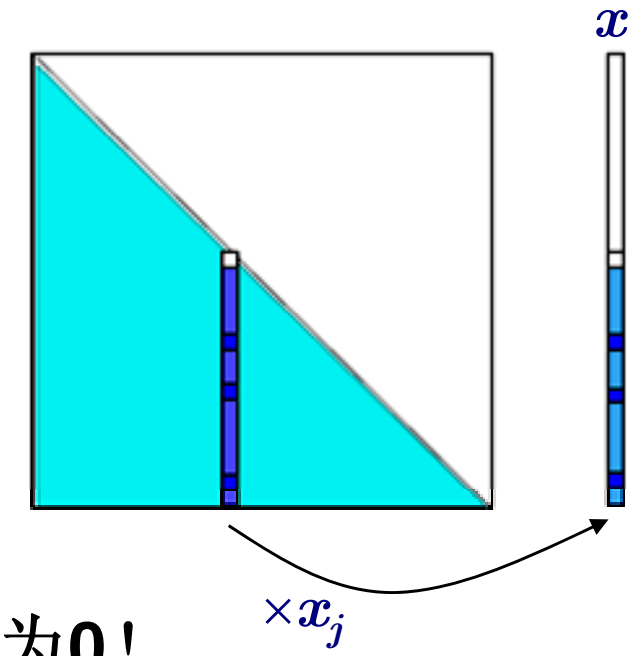
  $x_j = x_j / l_{jj}$

  for $i = j{+}1{:}\ n$, and $l_{ij} \neq 0$

    $x_i = x_i - l_{ij} x_j$

  end

end



  - 若右端项**b**为稀疏向量呢？ x_j可能为**0**！
  - 增加：If $(x_j \neq 0)$

下三角稀疏矩阵(右端项稀疏)

- 为简化，假设L对角元为1
  - “按列运算”求解算法求解 $Lx = b$

```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

稀疏向量

  - 时间复杂度$O(n+flop)$，n可能> flop

考虑稀疏b, flop可能远小于nnz(L)

  - 一个极端例子: 假设n很大, 只有$b_n \neq 0$, 此时仍需要做n次判断: If (x(j) ≠ 0)
  - 如果能先知道x的非零元分布$\mathcal{X}$就好了
  - 替换：for each $j \in \mathcal{X}$ do

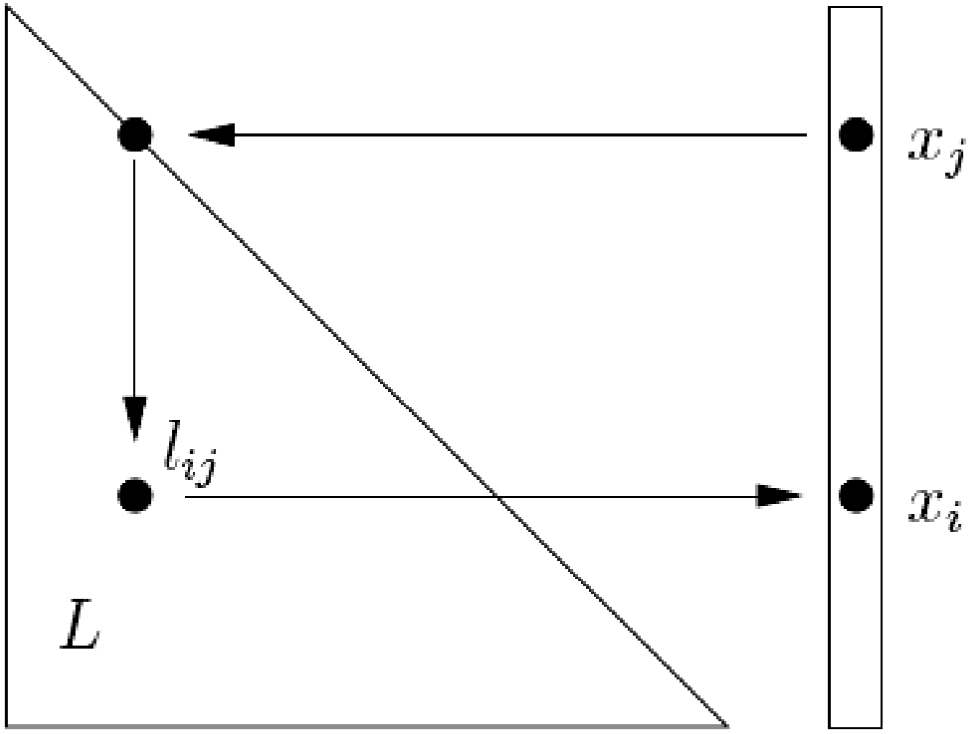
注意$\mathcal{X}$中元素的顺序

下三角稀疏矩阵(右端项稀疏) 一求x

- 什么条件使$x_i \neq 0$？ 解 $Lx = b$

$$x_i = b_i - \sum_{j=1}^{i-1} l_{ij} x_j$$

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$



- let $G(L)$ have an edge $j \rightarrow i$ if $l_{ij} \neq 0$
- let $\mathcal{B} = \{i \,|\, b_i \neq 0\}$ and $\mathcal{X} = \{i \,|\, x_i \neq 0\}$
- then $\mathcal{X} = \text{Reach}_{G(L)}(\mathcal{B})$

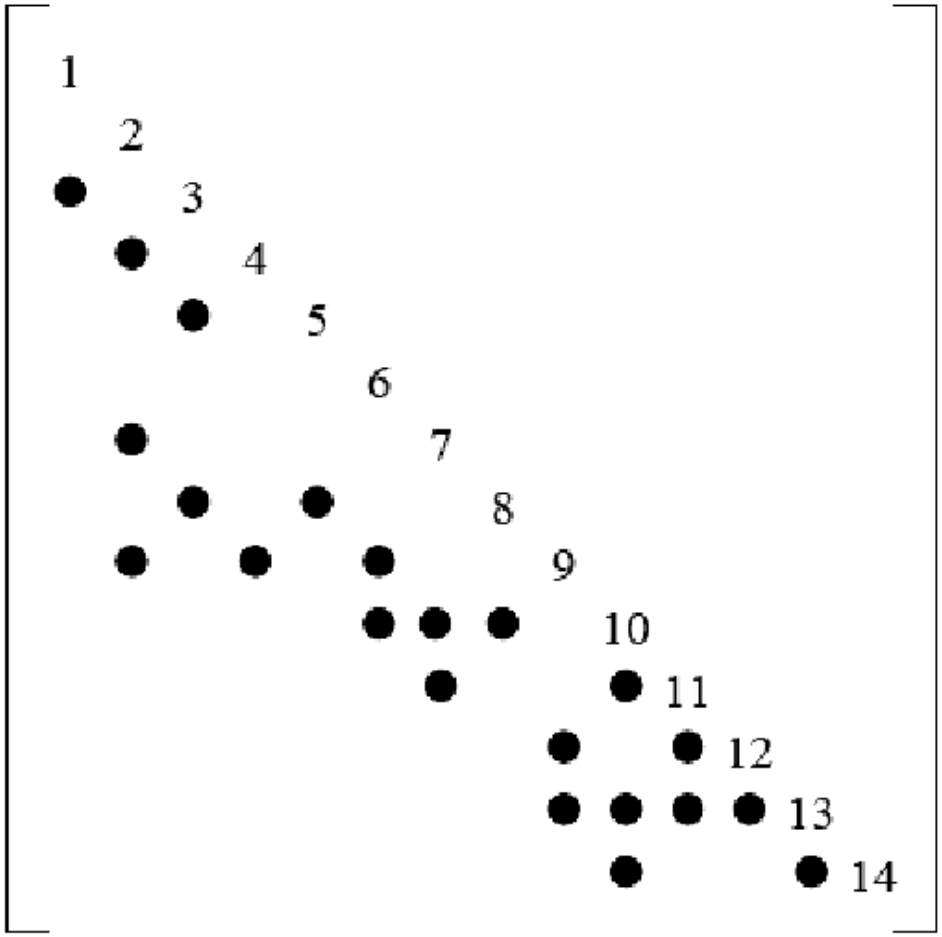
注意图的定义(适合CSC格式，找邻节点方便)

从集合$\mathcal{B}$的点出发的路径上的所有点

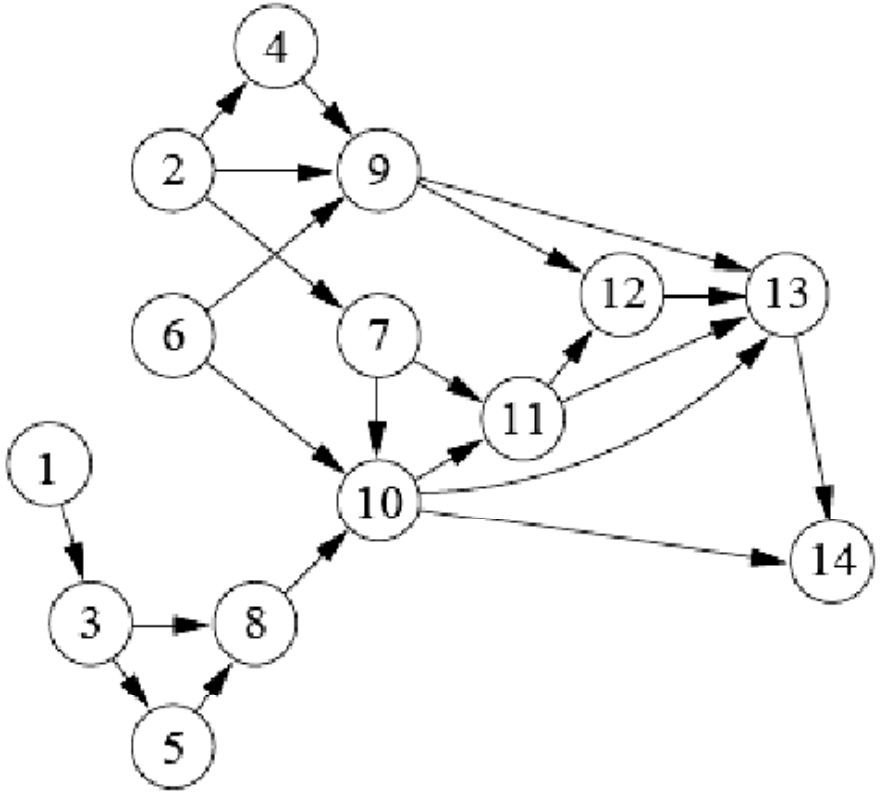
根据矩阵图求x

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$

Lower triangular matrix L



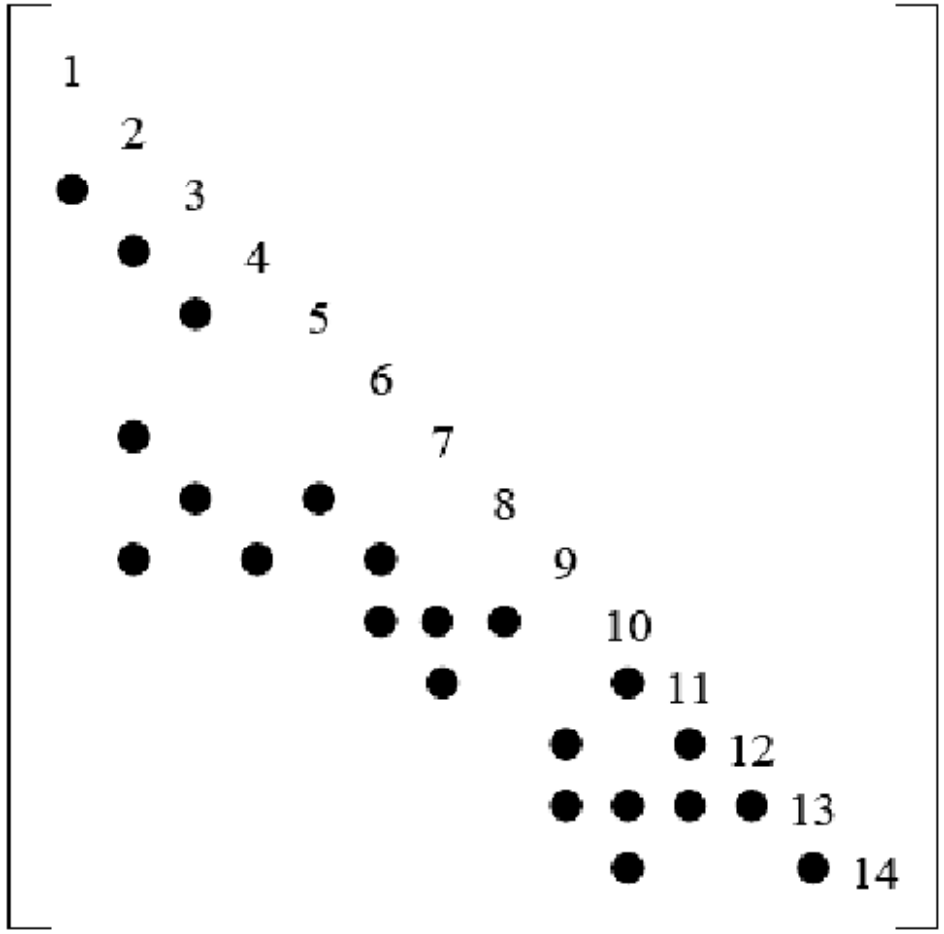
Graph G_L



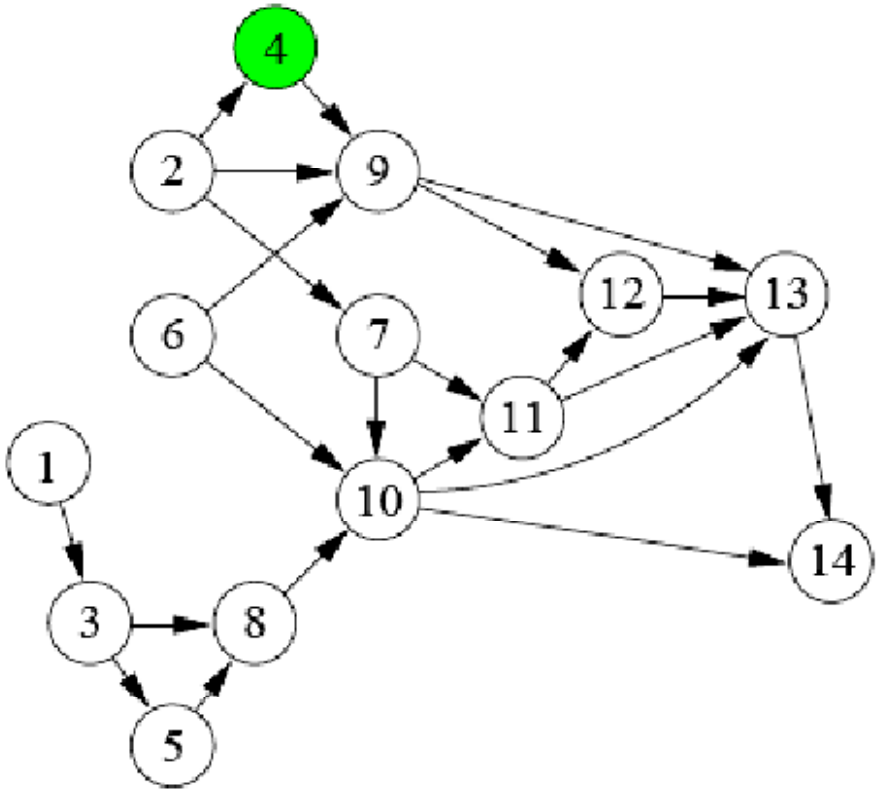
根据矩阵图求x

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$

Lower triangular matrix L



Graph G_L



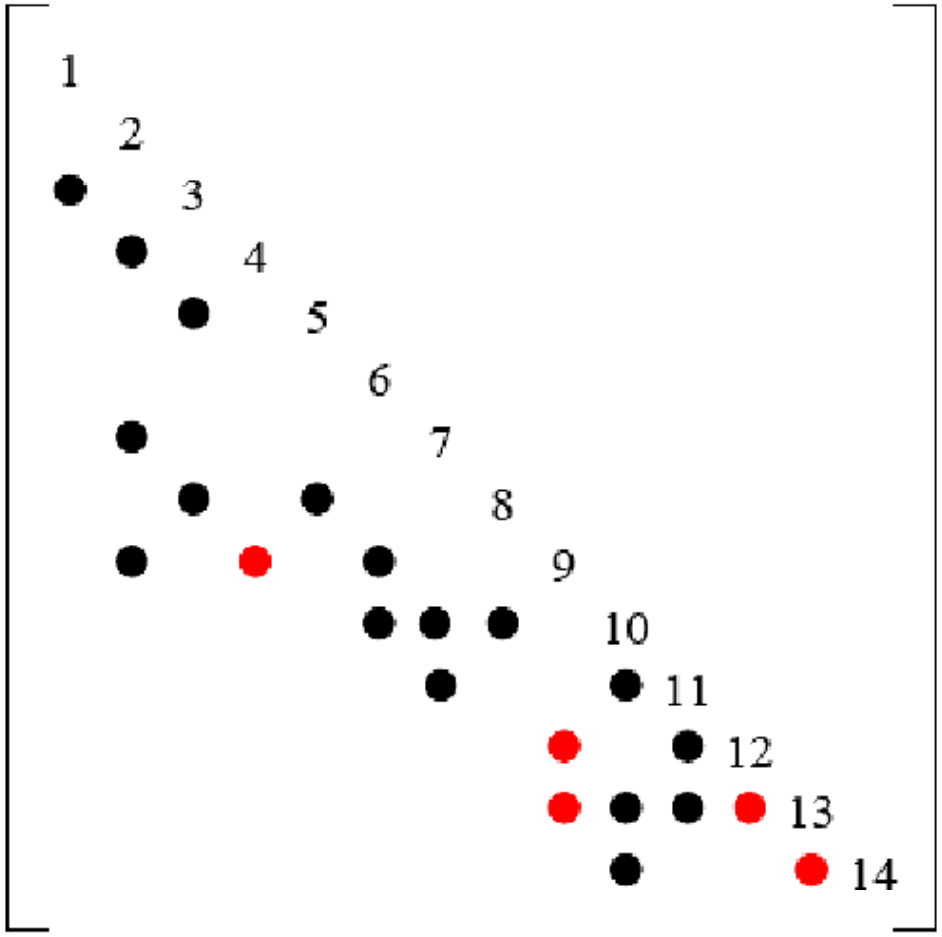
If $\mathcal{B} = \{4\}$

遍历所有路径，标记经过的点

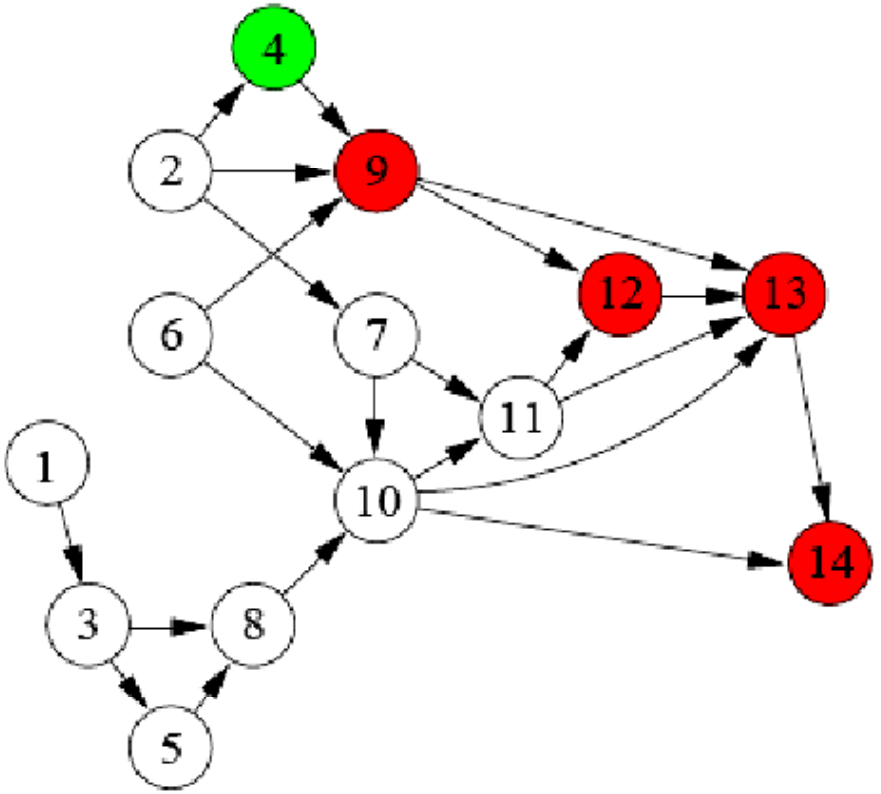
根据矩阵图求$\mathcal{X}$

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$

Lower triangular matrix L



Graph G_L



If $\mathcal{B} = \{4\}$
then $\mathcal{X} = \{4, 9, 12, 13, 14\}$

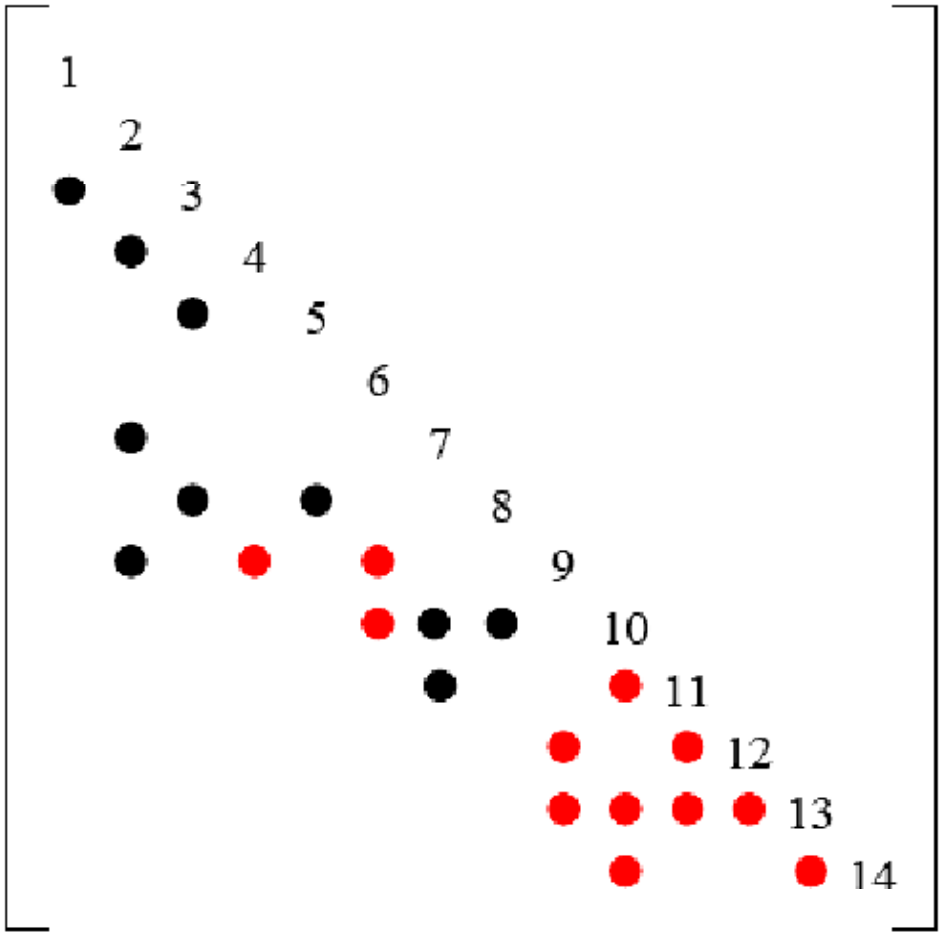
遍历所有路径，标记经过的点

遍历某顶点的出边，相当于矩阵的某列元素，矩阵采用**CSC**存储

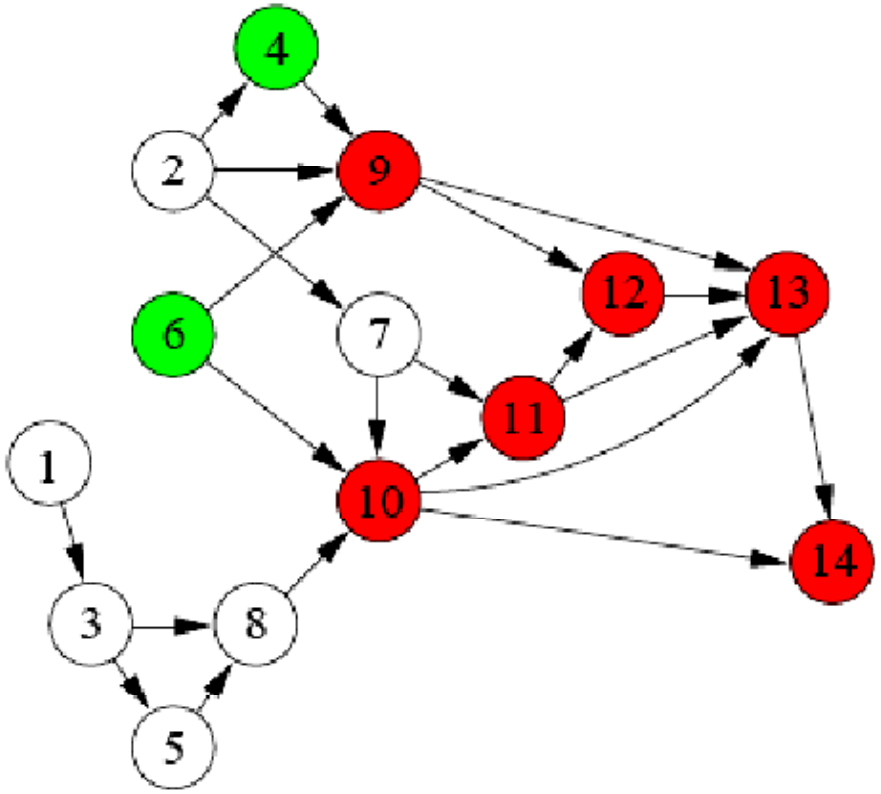
根据矩阵图求$\mathcal{X}$

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$

Lower triangular matrix L



Graph G_L



If $\mathcal{B} = \{4, 6\}$
then $\mathcal{X} = \{6, 10, 11, 4, 9, 12, 13, 14\}$

注意: 采用DFS进行后序遍历入栈, 则出栈的顺序为$\mathcal{X}$的正确顺序 (DAG的拓扑顺序)

解下三角稀疏矩阵的算法(右端项稀疏)

function x = lsolve(L,b)　　求解单位下三角方程 $Lx = b$

$\mathcal{X} = \text{Reach}(L, \mathcal{B})$

x = b

for each j in $\mathcal{X}$　　(按节点的拓扑顺序)

x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)

function $\mathcal{X} = \text{Reach}(\text{L}, \mathcal{B})$

for each i in $\mathcal{B}$ do

if (node i is unmarked) dfs(i)

function dfs(j)

mark node j

for each i in $\mathcal{L}_j$ do

if (node i is unmarked) dfs(i)

push j onto stack for $\mathcal{X}$

j的邻点集合
(CSC存储结构)

进一步完善，可使时间复杂度降为***O*(|b|+*flop*)**;
可能比***n*, nnz(*L*)**小得多
实际编程要解除递归!

DFS后序遍历入栈

Lsolve算法的简单Matlab程序实现

```
function x= lsolve(L, b)
% L, b为稀疏矩阵, 向量
n= size(L,1);
global stack marked ptr_s;
marked= zeros(n, 1);   //节点标记
stack= zeros(n, 1);    //栈
ptr_s= 0;              //栈顶指针
bb= find(b);           //b中非零元位置
reach(bb);             //结果存于stack
x= zeros(n, 1);
x(bb)= b(bb);
for i=ptr_s:-1:1,    //按出栈顺序
   j=stack(i);
   x(j+1:n)=x(j+1:n)-L(j+1:n, j)*x(j);
end
```

```
function reach(bb)
for i=1:size(bb,1),     //bb为列向量
   ii= bb(i);
   if ~marked(ii),
      dfs(ii);          //深度优先遍历
   end
end
```

```
function dfs(j)
marked(j)=1;
adj=find(L(:, j));      //找邻节点
for i=1:size(adj, 1),
   if ~marked(adj(i)),
      dfs(adj(i));      //递归调用
   end
end
ptr_s= ptr_s+1;     //入栈
stack(ptr_s)=j;
```

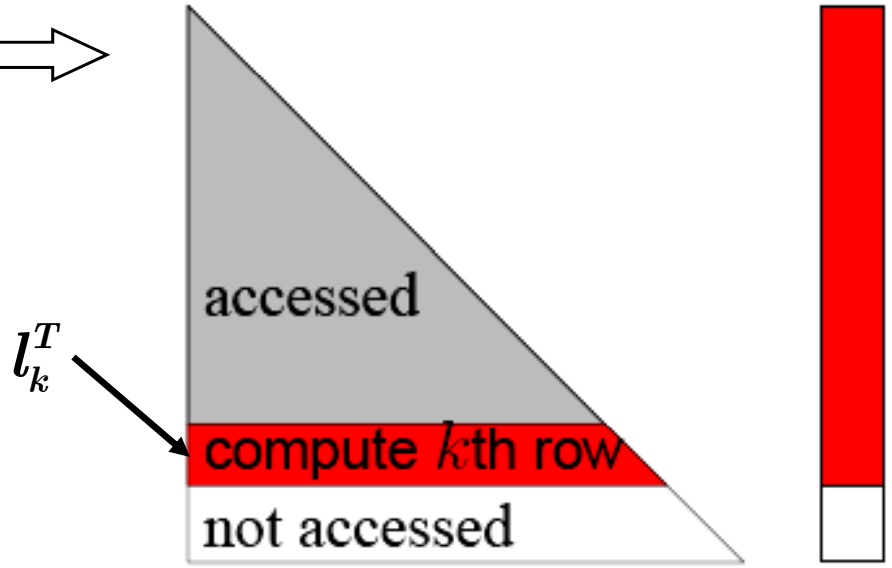
非结构化的对称正定稀疏阵

- 将**Cholesky**分解转化为求解一系列**Lx=b**

$$\begin{bmatrix} L_{11} & \\ \mathbf{l}_{12}^{\mathbf{T}} & \mathbf{l}_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & \mathbf{l}_{12} \\ & \mathbf{l}_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} \\ a_{12}^T & \mathbf{a}_{22} \end{bmatrix}$$

1. factorize $L_{11}L_{11}^T = A_{11}$ (递归)
2. solve $L_{11}l_{12} = a_{12}$ for l_{12}
3. $\mathbf{l}_{22} = \sqrt{\mathbf{a}_{22} - \mathbf{l}_{12}^{\mathbf{T}}\mathbf{l}_{12}}$

Up-looking方法



- 非递归算法

for k = 1:n

solve $L_{k-1}l_k = a_k(1:k-1)$ for l_k

$l_{k,k} = \sqrt{a_{k,k} - l_k^T l_k}$

end

L_{k-1} 为**k-1**阶顺序主子阵,
l_k 为第**k**行除对角元部分

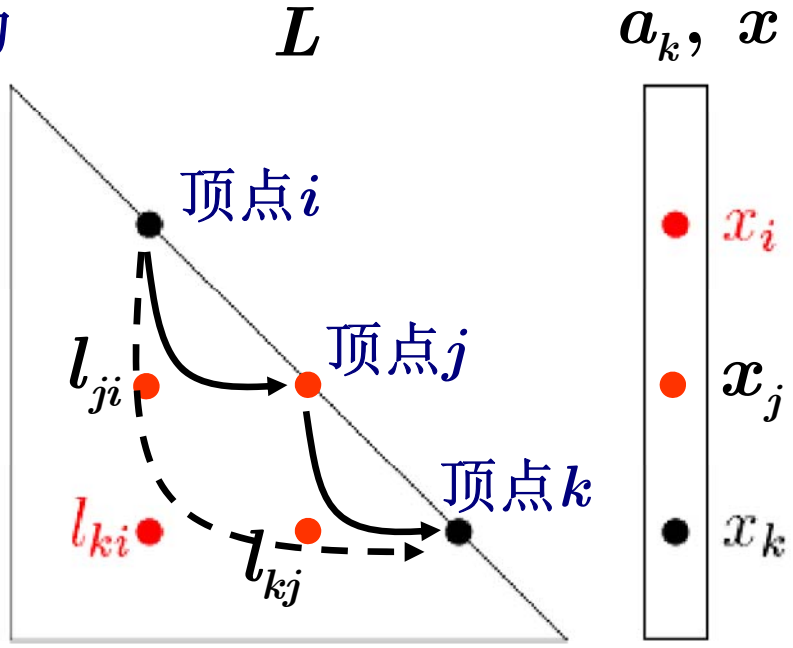
若**A**稀疏, 关键问题是: 预先了解**L**的非零元分布, 开辟存储空间

非结构化的对称正定稀疏阵

- 利用lsolve算法进行分解 $LL^T = A$
  - 考虑计算L的第k行: solve $L_{k-1} l_k = a_k(1:k-1)$
    - G_{k-1}: 矩阵 L_{k-1} 的图
    - $\mathcal{A}_k$: a_k 上三角部分向量中的非零元位置
    - $\mathcal{L}_k$: L 第k行的非零元位置, 则 $\mathcal{L}_k = \mathrm{Reach}_{G_{k-1}}(\mathcal{A}_k)$
  - 1.符号分解: 逐行得到L的非零元分布，创建数据结构
    - $(G_1, \mathcal{A}_2) \to \mathcal{L}_2 \to G_2, (G_2, \mathcal{A}_3) \to \mathcal{L}_3 \to G_3, \ldots\ldots$
  - 2.数值分解: 根据lsolve算法具体计算L元素的值
- 使用消去树使符号分解复杂度最低
  - 每次DFS遍历: $O(|\mathcal{L}_k| + e)$ (e为遍历的边数)
  - 剪掉L图的一些边得到树$\mathcal{T}$, DFS复杂度降到 $O(|\mathcal{L}_k|)$

对称正定稀疏阵的符号分解

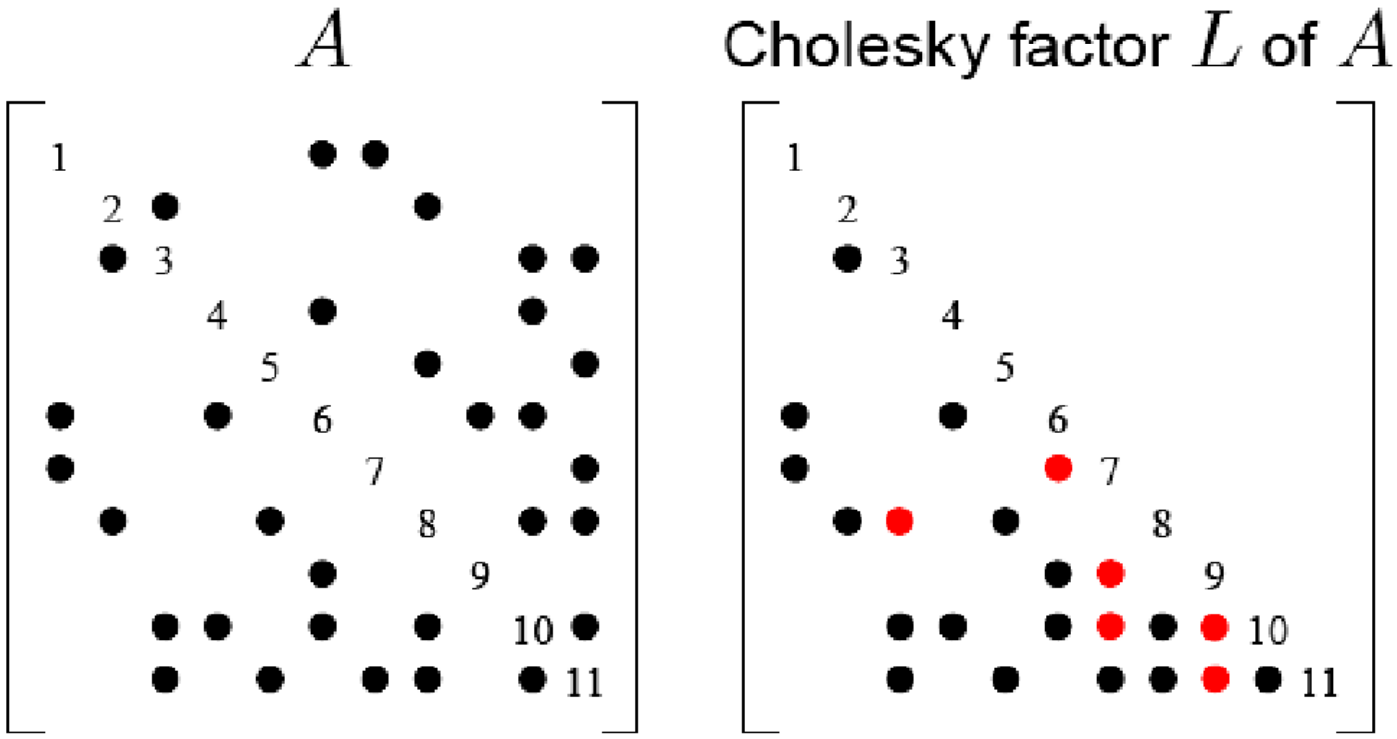
- **消去树(elimination tree)**
  - 符号分解: $(G_{k-1}, \mathcal{A}_k) \rightarrow \mathcal{L}_k \rightarrow G_k$, $\mathcal{L}_k = \mathrm{Reach}_{G_{k-1}}(\mathcal{A}_k)$
  - 设$a_k(i) \neq 0$, 从顶点i开始图的遍历
  - 若$l_{ji} \neq 0$, 到达顶点j, $x_j \neq 0$, 即$l_{kj} \neq 0$, 说明j到k有条边
  - $a_k(i) \neq 0$,即 $x_i \neq 0$, 也表明顶点i到k有条边, 即 l_{ki}
  - 对后续计算$\mathcal{L}_{k+1} = \mathrm{Reach}_{G_k}(\mathcal{A}_{k+1})$ l_{ki}对应的边实际上是多余的, 可在图中删除



  - 修剪后的图中, 顶点i只有一条出边(到j), j是满足$l_{ji} \neq 0$的最小数. 图G经修剪后成为"消去树"$\mathcal{T}$. j是i的双亲顶点

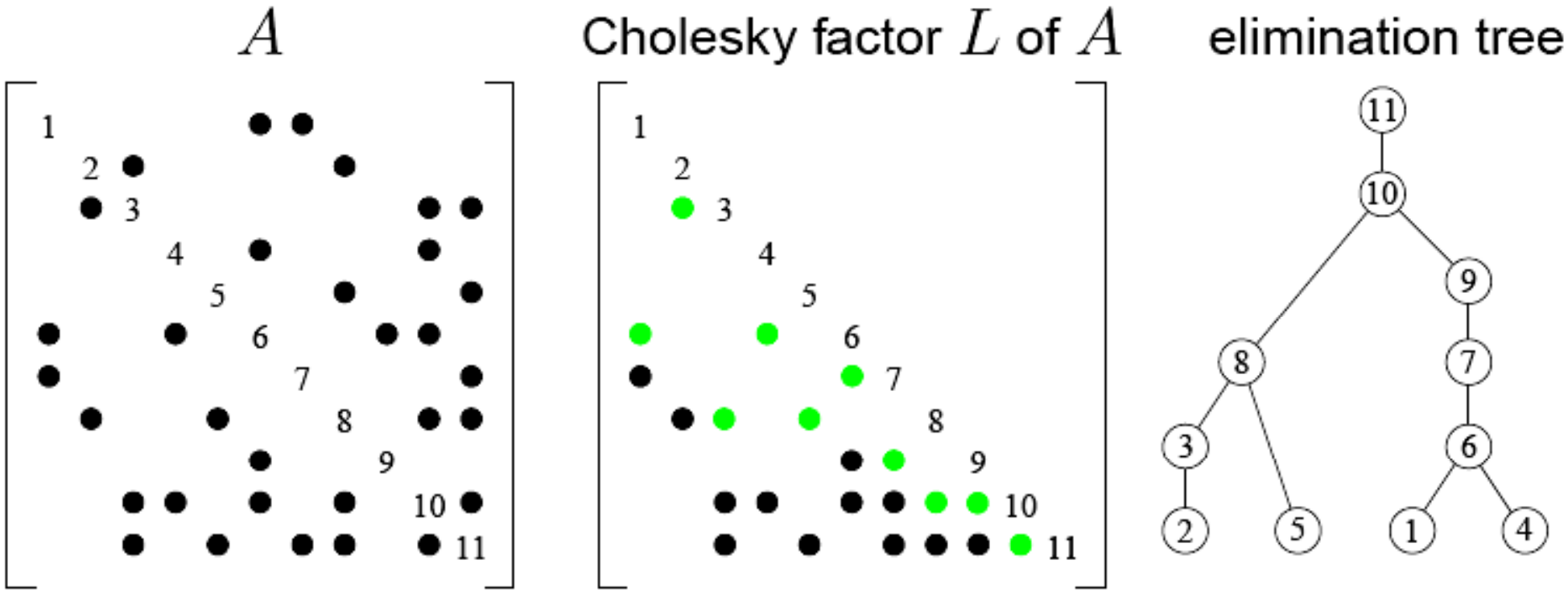
对称正定稀疏阵的符号分解

- 消去树(e-tree)—例子



对称正定稀疏阵的符号分解

- 消去树(e-tree)—例子



通过构建“消去树”，$\mathcal{L}_{k+1} = \mathrm{Reach}_{\mathcal{T}_k}(\mathcal{A}_{k+1})$的计算复杂度减小为$O(|\mathcal{L}_{k+1}|)$

(由于树的顶点数目和边数目的关系)

求解对称正定稀疏阵

- 符号分解 **(Symbolic analysis)**
  - 在稀疏矩阵的数值分解之前进行
  - 除了利用**lsolve**算法，常规做法是不算数值的矩阵分解
  - 依据原矩阵的非零元分布**(pattern)**，计算分解后因子矩阵的非零**pattern**，适合于求解多个**pattern**一样的线性方程组
- 多种数值分解算法

  算法实现有很多技巧
  - **Up-looking**
  - **Left-looking, supernodal**
  - **Right-looking, mutilfrontal**

  } 对符号分解的要求不同；不同的加速算法
- **Matlab**命令
  - **etree, chol, symbfact, cholmod**

非结构化的非对称稀疏阵—LU分解

- **Left-looking算法将LU分解转为求解一系列Lx=b**

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & l_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & u_{12} & U_{13} \\ & u_{22} & u_{23} \\ & & U_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} & A_{13} \\ a_{21} & a_{22} & a_{23} \\ A_{31} & a_{32} & A_{33} \end{bmatrix}$$

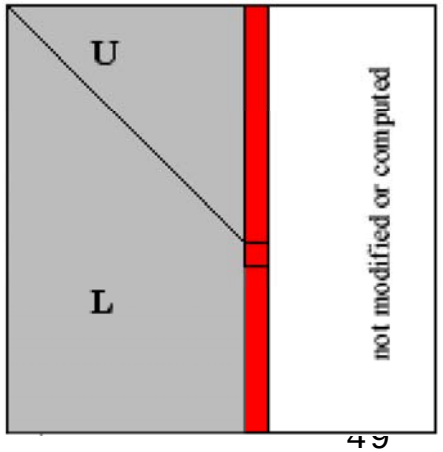
  - 根据L, U的前k-1列算它们的第k列

$$\begin{cases} L_{11}u_{12} = a_{12} \\ l_{21}u_{12} + u_{22} = a_{22} \\ L_{31}u_{12} + l_{32}u_{22} = a_{32} \end{cases}$$

写成一个下三角方程

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$

  - 则, $u_{12} = x_1, u_{22} = x_2, l_{32} = x_3 / u_{22}$
  - 注意初始第1列的计算
  - 利用Isove算法先符号分解, 再数值分解
  - 假设不需要"数值"选主元



非对称的稀疏矩阵—选主元的GPLU算法

- **GPLU算法(Gilbert/Peierls LU)**

$$\begin{cases} L_{11}u_{12} = a_{12} \\ l_{21}u_{12} + u_{22} = a_{22} \\ L_{31}u_{12} + l_{32}u_{22} = a_{32} \end{cases}$$

写成一个下三角方程

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$

  - 则, $u_{12} = x_1, u_{22} = x_2, l_{32} = x_3 / u_{22}$
  - 选主元: 避免l_{32}中元素>1, 交换x_2, x_3中元素
  - 算法：**Lu_left**

$$PA = LU$$

```
L= speye(n); U= speye(n); P= speye(n);
For k= 1: n
   x= L \ (P*A(:, k));        //符号分解(消去树, DFS), 数值分解
   find pivot position i from x(k:n);
   swap rows i and k of L, P, and x;
   U(1:k, k)= x(1:k);
   L(k+1:n, k)= x(k+1:n)/U(k,k)
end
```

这里省略严格的证明

参考资料

- **T. A. Davis @ University of Florida**

  - **http://www.cise.ufl.edu/~davis/**
  - **书: Direct Methods for Sparse Linear Systems, SIAM Press, 2006, (CSparse)**
  - **Package: SuiteSparse**
    http://www.cise.ufl.edu/research/sparse/
  - **Matrix collection**

Matlab “\”的内部算法

检查矩阵A的特点	求解算法
对角阵	对角线元素取倒数
稀疏的条带状方阵	“追赶法”，LAPACK软件包
上三角、或下三角矩阵	回代法
对三角矩阵重新排列行得到的结果	重排序后用回代法
对称矩阵、正对角线元素	试Cholesky算法，LAPACK软件包，若为稀疏进行MD排序
上Hessenberg矩阵	消为上三角矩阵再求解
一般的方阵、但是稀疏矩阵	UMFPACK软件包，2004年开发的稀疏矩阵直接解法，U. Florida, T. Davis
一般的方阵、非稀疏矩阵	LAPACK软件包
不是方阵	用Householder变换做QR分解，同命令qr，得到最小二乘解; 若稀疏，用Givens旋转

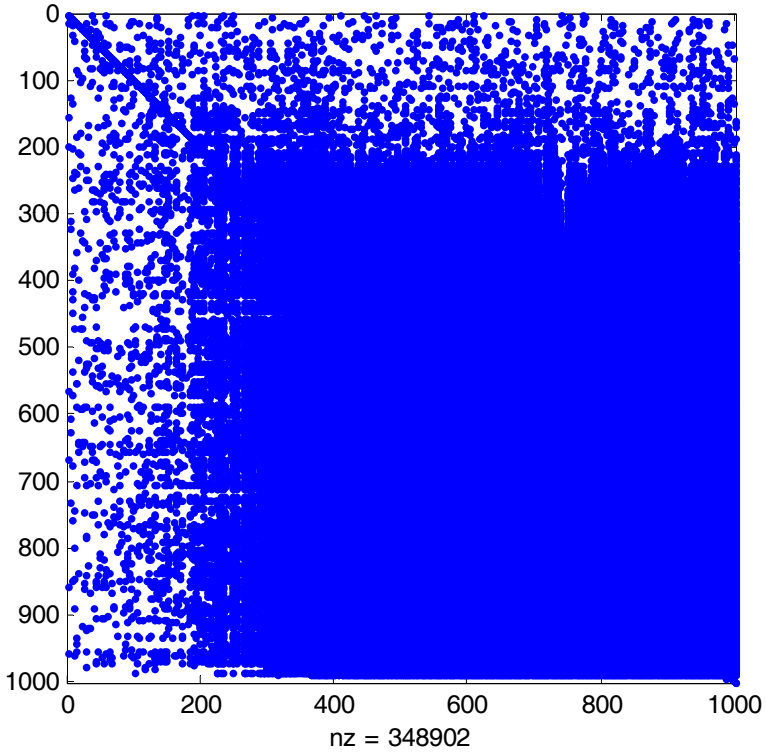
Matrix Reordering and General Discussion

Wenjian Yu

一般稀疏矩阵的“填入”

>>A= sprand(1000, 1000, 0.005);
>>spy(A);

>> [L U P]=lu(A);
>> spy(L+U);



对一般的非结构化稀疏矩阵，**Fill-in**是个大问题！

矩阵重排序可减少“填入”

例1：

$$\begin{bmatrix} X & X & X \\ X & X & 0 \\ X & 0 & X \end{bmatrix} \Rightarrow \begin{bmatrix} X & X & X \\ X & X & X \\ X & X & X \end{bmatrix}$$

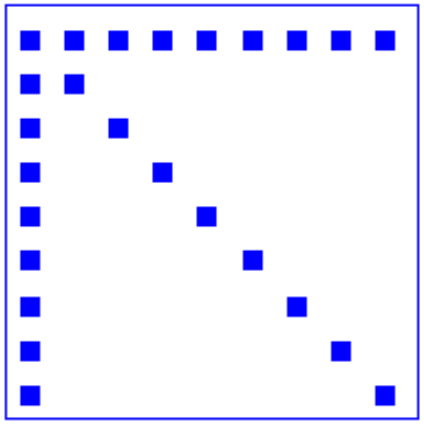
Fill-ins

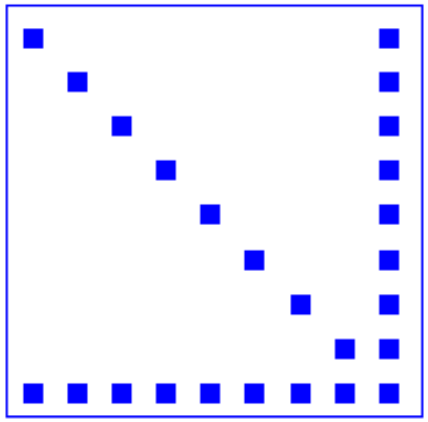
交换1, 2行/列

$$\begin{bmatrix} X & X & 0 \\ X & X & X \\ 0 & X & X \end{bmatrix}$$

No fill-in!

思考: 怎么做对称重排, 使得没有**fill-in**?





矩阵重排序——减少“填入”

- **Reordering**是减少稀疏矩阵运算的关键
- 在常规符号分解的过程中对矩阵重排序

$$Ax = b \longrightarrow \begin{cases} PAx = Pb \\ APy = b \\ PAQy = Pb \end{cases}$$

(Q=P^T, 为对称重排)

  - “怎样重排序使填入元最少？”是**NP-hard**问题!
  - **Markowitz**算法**(1957)**
  - 性能**:** 可能比最优情况多**5%**的填入，但运行速度快
  - 思想**:** 在高斯消元的每一步，对**fill-in**的数目进行估计，进行矩阵重排列**(**类似于选主元**)**使当前消元步产生的**fill-in**最少。

Permutations for sparsity

"I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were 'sparse' in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care"

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics

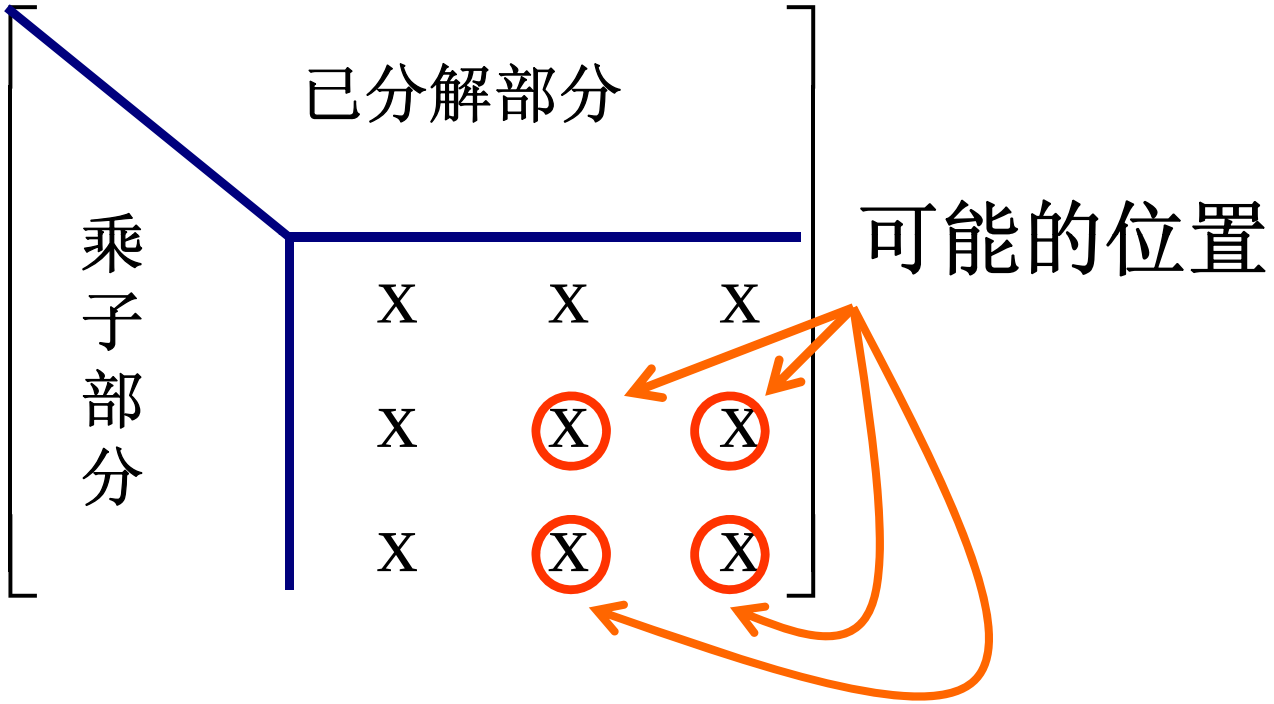
Courtesy of J. Gilbert at UCSB

重排序算法－Markowitz算法

填入元可能出现在哪里？

填入: **0 —> 非0**

两个必要条件:
1. 主元当前列下面的元素非零
2. 主元当前行右边元素非零



填入元数目估计 = (未分解部分当前行非零元数目 -1) ×
(未分解部分当前列非零元数目 -1)
≡ Markowitz乘积

对未分解部分任一元素，
可算**Markowitz**乘积

重排序算法—Markowitz算法

算法1：列重排 $APy = b$

For $i = 1$ to n

Find a_{ij} $(j \geq i)$ with min Markowitz Product

Swap column $j \neq i$ and column i

Eliminate with the new row i and determine fill-ins

End

算法2：行列对称重排 $PAP^T y = Pb$

For $i = 1$ to n

Find a_{jj} $(j \geq i)$ with min Markowitz Product

Swap rows $j \neq i$ and columns $j \neq i$

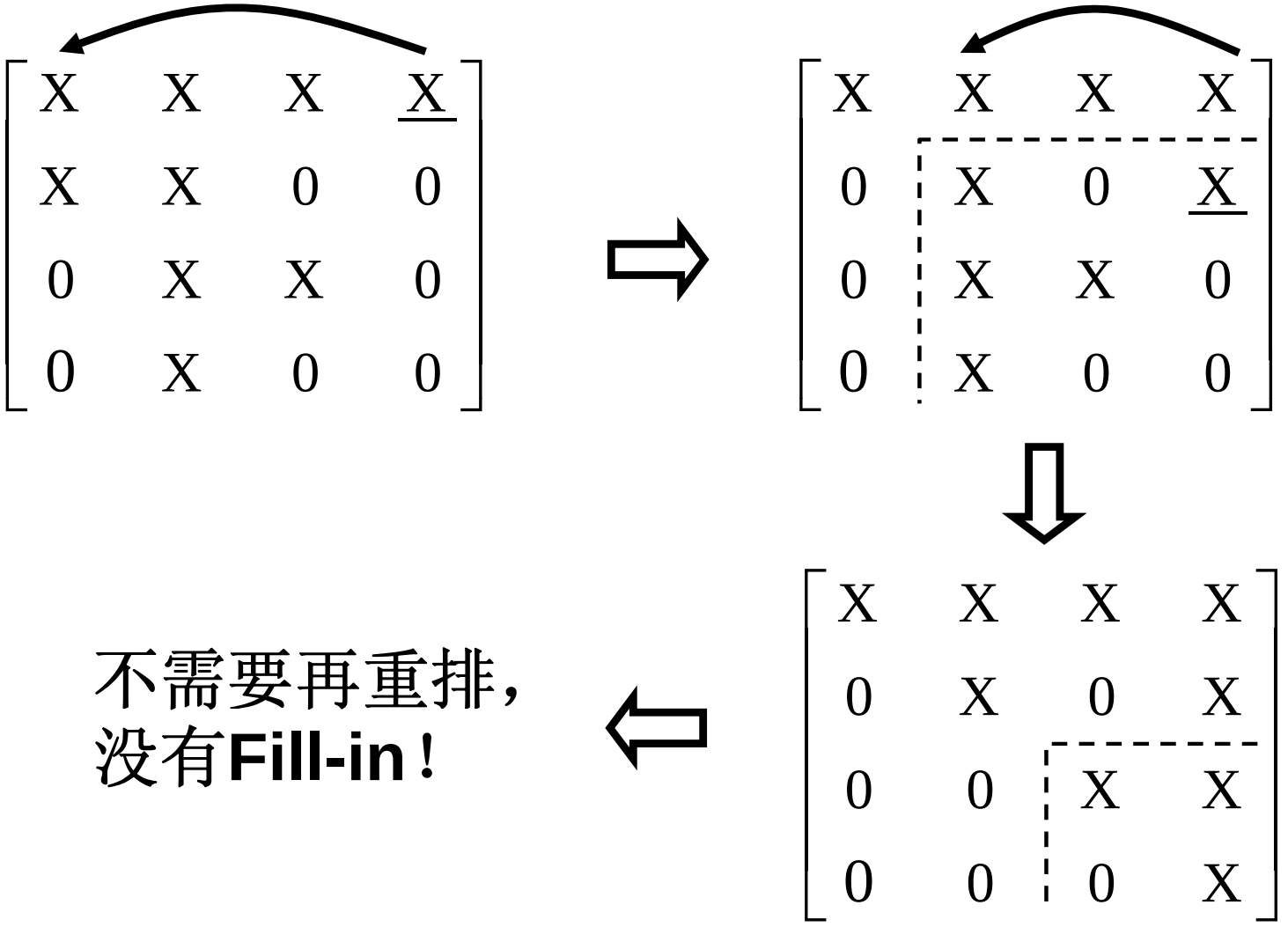
Eliminate with the new row i and determine fill-ins

End

对称正定矩阵;
对角占优矩阵

采用合适的数据结构(压缩稀疏行/列, **CSR/CSC**),
以及编程技巧，计算代价并不大。

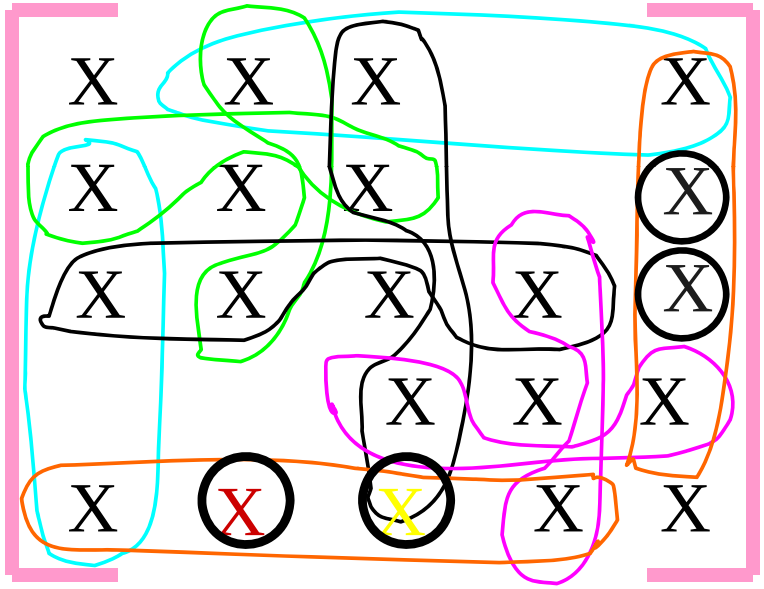
列重排减少“填入元”一例

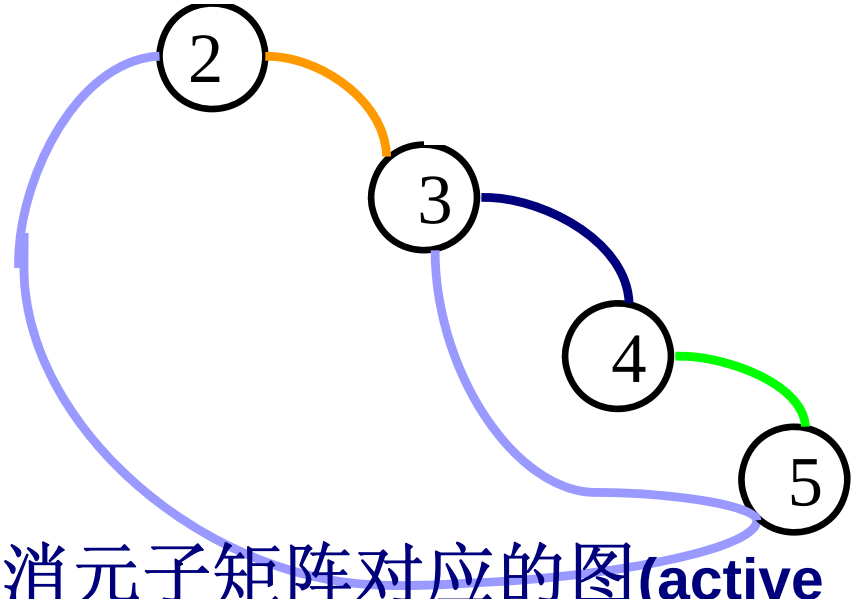


- 各列M-乘积不同的原因：该列未分解部分的非零元数目
- 列重排后，交换行进行“数值”选主元 (对fill-in也有影响)

Markowitz算法——最小度排序

- **Minimal Degree (MD) Ordering—图的解释**
  - 对称矩阵，**M-乘积==顶点度的平方**





  - 消去一行后，图的更新
  - **MD算法：在消元的第i步，对未消元子矩阵对应的图(active graph)上找度最小的顶点，设它对应的行为主元行**
  - 非对称矩阵，**M-乘积的含义？**

Heuristic fill-reducing matrix reorderings

- **Nested dissection**:
  - Find a separator, number it *last*, proceed recursively
  - Theory: approx optimal separators => approx optimal fill and flop count
  - Practice: often wins for very large problems
- **Minimum degree**:
  - Eliminate row/col with fewest nzs, add fill, repeat
  - Hard to implement efficiently – current champion is "Approximate Minimum Degree" [Amestoy, Davis, Duff]
  - Theory: can be suboptimal even on 2D model problem
  - Practice: often wins for medium-sized problems
- **Banded orderings** (Reverse Cuthill-McKee, Sloan, . . .):
  - Try to keep all nonzeros close to the diagonal
  - Theory, practice: often wins for "long, thin" problems
- The best modern general-purpose orderings are ND/MD hybrids.

Courtesy of J. Gilbert

Fill-reducing reorderings in Matlab

- **Symmetric approximate minimum degree**:
  - p = amd(A); //another one: symamd
  - symmetric permutation: chol(A(p,p)) often sparser than chol(A)
- **Symmetric nested dissection:**
  - not built into Matlab
  - several versions in meshpart toolbox (course web page references)
- **Nonsymmetric approximate minimum degree:**
  - p = colamd(A);
  - column permutation: lu(A(:,p)) often sparser than lu(A)
  - also for QR factorization
- **Reverse Cuthill-McKee**
  - p = symrcm(A);
  - A(p,p) often has smaller bandwidth than A
  - similar to Sparspak RCM

Courtesy of J. Gilbert

The four essential stages of a solve $Ax = b$

A对称正定

1. Reordering: $A \longrightarrow A := PAP^T$

- Preprocessing: uses graph only [Min. deg, AMD, Nested Dissection]

2. Symbolic Factorization: Build static data structure.

- Exploits 'elimination tree', uses graph only.
- Also: 'supernodes'

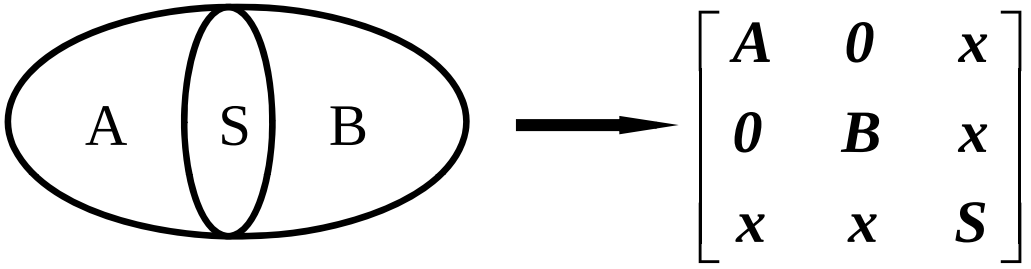
3. Numerical Factorization: Actual factorization $A = LL^T$

- Pattern of L is known. Uses static data structure. Exploits supernodes (blas3)

4. Triangular solves: Solve $Ly = b$ then $L^T x = y$

一般矩阵同时考虑数值选主元

- **Nested dissection**
  - 基于图操作，找最小割集(将图分割开)
  - 对**A,B**重复此操作



- 数值选主元 (一般矩阵)
  - 目的是控制**L, U**中数值的增大，使算法稳定
  - 对稀疏矩阵通常放松要求(为了控制稀疏性, 或可并行化)
  - 原始要求: $a_{kk}^{(k)} \geq a_{ik}^{(k)},\ i > k$
  - 阈值选主元: $a_{kk}^{(k)} \geq \mu a_{ik}^{(k)},\ i > k,\ \mu \approx 0.1$
  - 在满足阈值条件的范围内做**fill-reduce permutation**
  - 符号分解阶段进行 $LU = PAQ$ **GPLU算法**
  - **P:** 数值选主元, **Q: fill-reducing reordering**