高等数值算法与应用(2)

－ 线性方程组求解 (chap2) －

喻文健

Outline

- 高斯消去法与LU分解
- 误差分析
- 稀疏矩阵与条带状矩阵
- Google的PageRank
- 对称正定矩阵的Cholesky分解

高斯消去法与LU分解

- 解线性方程组的基本概念与方法
- 排列阵、上/下三角阵的求解
- 基于部分主元LU分解的方法
- 相关Matlab程序

求解线性方程组

- 方程 $$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$
  - 若$m>n$, 超定方程组, 一般无解 (但有最小二乘解)
  - 若$m<n$, 一般有无穷多解, 需加其他条件 (约束优化, 等)
  - m=n，线性方程组求解问题, 矩阵形式为

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \quad , \quad \boldsymbol{A} \in \mathbb{R}^{n\times n}, \boldsymbol{b} \in \mathbb{R}^n$$

  - 复习: 奇异/非奇异矩阵, 解的存在性与唯一性, …
  - $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$, 但实际上，计算$\boldsymbol{A}^{-1}$不但没必要也很不明智
  - 例: 7x=21, x=7⁻¹*21=0.142857*21=2.99997

而且计算量大

求解线性方程组

- Matlab反斜线算符
  - \ ：backward slash
  - 线性方程组 AX=B，求解命令为X=A\B
  - 由于A在左边，\也称为"左除"算符
  - 求解方程组YA=B的命令：Y=B/A. 称/为"右除"
  - 左除运算与右除运算有何关系?
- 求解线性方程组的一个例子

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix} \Longleftrightarrow \begin{aligned} 10x_1 - 7x_2 \quad\quad &= 7 \\ -3x_1 + 2x_2 + 6x_3 &= 4 \\ 5x_1 - x_2 + 5x_3 &= 6 \end{aligned}$$

  - 用第1个方程消其他方程中的x_1

求解线性方程组

乘子的相反数

- 求解线性方程组的一个例子　　（第1个方程乘0. 3，−0. 5）
  - 第1个方程x_1的系数称为主元(pivot)，即矩阵对角元
  - 当前列其他系数除以主元，得到"乘子": −0.3, 0.5

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix} \Rightarrow \begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix}$$

  - 用第2、第3个方程可消去其中一个的x_2. 考虑到−0.1较小，不适合当主元，交换第2、3个方程（选主元）

消元时，乘子为−0. 04

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.1 \end{pmatrix} \Rightarrow \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.2 \end{pmatrix}$$

最后一个方程 $6.2x_3 = 6.2 \Rightarrow x_3 = 1$

求解线性方程组

- 求解线性方程组的一个例子
  - 将$x_3=1$代回前面方程，依次解出x_2, x_1 $\Rightarrow$ $x=\begin{pmatrix}0\\-1\\1\end{pmatrix}$

$$\underbrace{\begin{pmatrix}10 & -7 & 0\\ 0 & 2.5 & 5\\ 0 & 0 & 6.2\end{pmatrix}}_{U}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}7\\2.5\\6.2\end{pmatrix}$$

  - 用原始方程验证解的正确性

$$\underbrace{\begin{pmatrix}10 & -7 & 0\\ -3 & 2 & 6\\ 5 & -1 & 5\end{pmatrix}}_{A}\begin{pmatrix}0\\-1\\1\end{pmatrix}=\begin{pmatrix}7\\4\\6\end{pmatrix}$$

  - 消去过程的描述(仅看A): $M_2P_2M_1A=U$

$$\Rightarrow P_2M_1A=M_2^{-1}U \Rightarrow P_2M_1P_2P_2A=M_2^{-1}U$$

$$\Rightarrow P_2A=\underline{(P_2M_1P_2)^{-1}M_2^{-1}}U$$

  - 行交换矩阵P_2为

$$P_2=\begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{pmatrix}$$

由乘子填充
执行行交换

$$L=\begin{pmatrix}1 & 0 & 0\\ 0.5 & 1 & 0\\ -0.3 & -0.04 & 1\end{pmatrix}$$

  - 矩阵A与L, U的关系：$PA=LU$

称为**部分主元LU分解 (P排列阵)**

排列阵与上(下)三角阵

- 排列阵(permutation matrix)：单位阵经一系列行、列交换而得. 每行(列)上有且仅有一个1，其余为0
  - 例:
    $$P=\begin{pmatrix}0&0&0&1\\1&0&0&0\\0&0&1&0\\0&1&0&0\end{pmatrix}$$
  - PA ~ 对A作行重列
  - AP ~对A作列重列
  - 用向量p表示单位阵各行的新顺序，易得排列阵
  - I(p, :) 就是排列阵P
  - 注意":"的用法
  - A(p, :)实现矩阵A的行重排
  - 求解系数为排列阵的方程

1:n的一个重排序

```
>> p = [4 1 3 2];
>> I = eye(4, 4); P= I(p, :)
```

```
>> P*A
>> A(p, :)    %计算速度快, 省内存
```

$$Px=b \Longrightarrow x=P^Tb$$

P为正交阵 ？！

排列阵与上(下)三角阵

- 系数矩阵为上(下)三角阵的方程
  - 矩阵的LU分解得: 上三角阵U，单位下三角阵L

$$U = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix} \qquad L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & -0.04 & 1 \end{pmatrix}$$

  - 解$Ux=b$的两种"回代"(backsubs)

```
x = zeros(n, 1);
for k = n: -1: 1
   j = k+1:n;
   x(k)=(b(k)-U(k, j)*x(j))/ U(k,k)
end
```

用U的行和解出的部分x作内积

```
x = zeros(n, 1); b;
for k= n: -1: 1
   x(k) = b(k)/ U(k, k);        x
   i = 1: k-1;
   b(i) = b(i) - x(k) *U(i, k)
end
```

从b中依次减去U某列的倍数

先LU分解，再解x: $PA=LU \Longrightarrow x=U^{-1}(L^{-1}Pb)$

部分主元LU分解程序

- 为什么必须选主元？
  - 保证算法不中断; 防止小主元导致的误差传播

乘子$m_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$ 主元

- lutx程序
  - “原地存储”方式

(KIJ version)

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

```
function [L,U,p] = lutx(A)
[n,n] = size(A);
p = (1:n)';
for k = 1:n-1
   [r,m] = max(abs(A(k:n,k)));
   m = m+k-1;
   if (A(m,k) ~= 0)
      if (m ~= k)
        A([k m],:) = A([m k],:);
        p([k m]) = p([m k]);
      end
      i = k+1:n;
      A(i,k) = A(i,k)/A(k,k);
      j = k+1:n;
      A(i,j) = A(i,j) - A(i,k)*A(k,j);
   end
end
```

最大值,位置索引

交换两行

算乘子

更新未消去部分

部分主元LU分解程序

- Matlab中的程序
  - 前一页程序加两行算L, U:
  - 使用向量/矩阵运算提高效率
  - **lu**: 部分主元LU分解; lutx为其简化版
  - \: 解单个/多个右端项的线性方程组
  - bslashtx.m是"\"的简化版，它检测三种特殊情况做处理
  - 调用forward, backsubs求解下三角、上三角线性方程组
  - <u>请自己看bslashtx程序的源码!</u>
  - lugui是NCM中一个演示程序, 试验4种选主元策略: 手工, 对角线, 部分主元, 全主元

```
L = tril(A, -1) + eye(n, n);
U = triu(A);
```

稀疏阵 → UMFPACK/CHOLMOD

稠密阵 → Lapack+BLAS

(Basic Linear Algebra Subpro

```
>> lugui
```

看演示网站第2.4个演示, *操作与思考*

误差分析

- 残差小、误差大的情况
- 敏感性分析-矩阵条件数
- 舍入误差对结果的影响

残差小、误差大的情况

- $\boldsymbol{x}_*$为解方程$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$得到的数值解, ≠准确解$\boldsymbol{x}$
  - 误差: $\boldsymbol{e}=\boldsymbol{x}_*-\boldsymbol{x}=-(\boldsymbol{x}-\boldsymbol{x}_*)$
  - 残差向量(residual, 剩余): $\boldsymbol{r}=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_*=\boldsymbol{A}(\boldsymbol{x}-\boldsymbol{x}_*)$
  - 常常用残差评估计算准确度. 但可能残差小、误差大
  - 例: 在3位有效数字(10进制)的机器上求解

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix}$$

交换行, 消元

$$\begin{pmatrix} 0.913 & 0.659 \\ 0 & 0.001 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} 0.254 \\ 0.001 \end{pmatrix}$$

解出:

$$x_2=\frac{0.001}{0.001}=1.00$$

$$x_1=\frac{0.254-0.659x_2}{0.913}$$

$$=-0.443$$ (保留3位)

准确解为: $\boldsymbol{x}=\begin{pmatrix} 1 \\ -1 \end{pmatrix}$

若不知道准确解, 计算残差

$$\boldsymbol{r}=\begin{pmatrix} 0.217-((0.780)(-0.443)+(0.563)(1.00)) \\ 0.254-((0.913)(-0.443)+(0.659)(1.00)) \end{pmatrix}=\begin{pmatrix} -0.000460 \\ -0.000541 \end{pmatrix}$$

误差分析

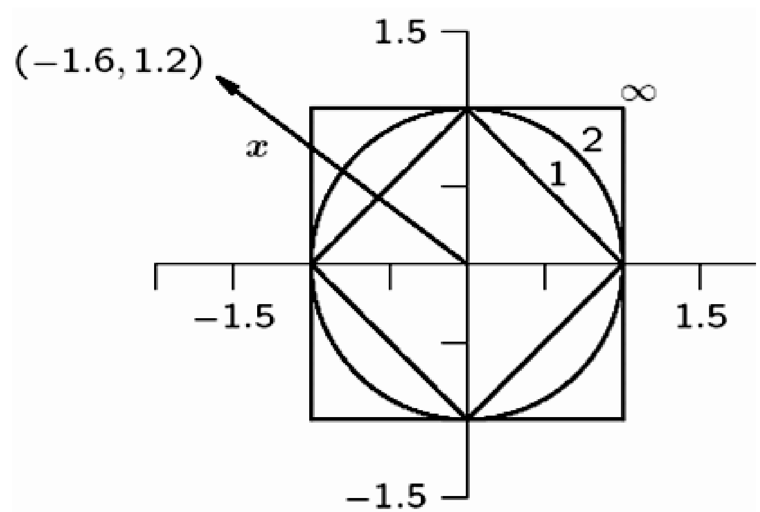
- 数值求解方程$\boldsymbol{Ax} = \boldsymbol{b}$
  - 上例：舍入误差造成解误差大，但残差很小 (相对量)
  - 改用6位有效数字重算(减小舍入误差)，可得很准确的解
  - 另一个不可忽视的原因：系数矩阵接近奇异
  - 一个重要事实: ***采用部分选主元的高斯消去法，可以保证解对应的残差$\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_*$相对较小***
- 线性方程组求解的误差分析
  - 1.系数矩阵与右端项上有数值扰动 (敏感性)
  - 2.线性方程组是精确的, 考虑计算中的舍入误差
  - 对情况1, 与$\boldsymbol{A}$矩阵的近奇异性有关(病态矩阵 vs. 单位阵) 严格讨论，需使用**矩阵条件数**的概念

范数与矩阵条件数

$$\|\boldsymbol{x}\|_p = (\sum_{i=1}^{n}|x_i|^p)^{1/p}, \ p \geq 1$$

- 常用的向量范数
  - 1−范数：$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n}|x_i|$ **(曼哈顿范数)**
  - 2−范数：$\|\boldsymbol{x}\|_2 = (\sum_{i=1}^{n}|x_i|^2)^{1/2}$ **(欧氏距离)**
  - ∞−范数：$\|\boldsymbol{x}\|_\infty = \max_{1\leq i\leq n}|x_i|$ **(“最大”范数)**
  - 例子: 在二维坐标系中按不同范数绘出的“单位圆”
  - norm命令(默认2范数)

```
>> x = (1:4) /5
>> norm1= norm(x, 1)
>> norm2= norm(x)
>> norminf= norm(x, inf)
```



范数与矩阵条件数

- 矩阵条件数
  - 不同的$\boldsymbol{x}$, $\boldsymbol{Ax}$的范数可能差别很大
  - 有两个极端情况: $M = \max \frac{\|Ax\|}{\|x\|}$, $m = \min \frac{\|Ax\|}{\|x\|}$
  - 其比值反映$\boldsymbol{A}$的性质, 称为"矩阵条件数"

$\|\boldsymbol{A}\|$ 与向量范数相容:
$\|\boldsymbol{Ax}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{x}\|$

$$\kappa(A) = \frac{\max \frac{\|Ax\|}{\|x\|}}{\min \frac{\|Ax\|}{\|x\|}}$$

对于奇异阵, m=0,
条件数为无穷大

- 敏感性分析
  - 改变右端项, $A(x+\delta x) = b + \delta b$ $\Rightarrow$ $\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\delta b\|}{\|b\|}$
  - 矩阵条件数是<u>数据传递误差</u>的上限情况 (可达到, 见pp.58)
  - 对于系数矩阵上的扰动，有类似结果

敏感性分析不
考虑具体解法

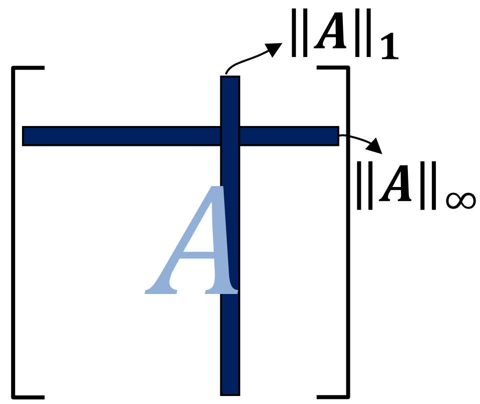
范数与矩阵条件数

- 矩阵条件数的性质

$$\kappa(A) = \frac{\max \frac{\|Ax\|}{\|x\|}}{\min \frac{\|Ax\|}{\|x\|}}$$

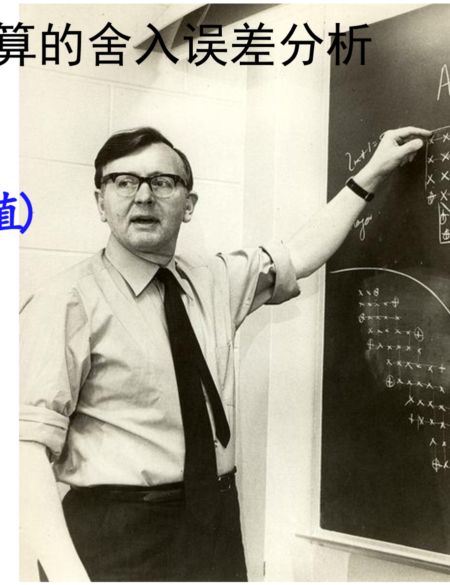
  - $\kappa(A) \geq 1$
  - $\kappa(P) = 1$ (排列阵)　$\kappa(\boldsymbol{A})$描述奇异程度,而行列式不能
  - $\kappa(cA) = \kappa(A)$　$\det(\begin{bmatrix} 0.1 & & \\ & \ddots & \\ & & 0.1 \end{bmatrix}_{100})=10^{-100}$
  - $\kappa(D) = \frac{\max |d_{ii}|}{\min |d_{ii}|}$
  - Matlab中计算/估算条件数: cond(A), cond(A, 1), condest(A)
- 矩阵范数　$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$
  - $\|\boldsymbol{A}\| = \max \frac{\|\boldsymbol{Ax}\|}{\|\boldsymbol{x}\|}$　注意, $\min \frac{\|\boldsymbol{Ax}\|}{\|\boldsymbol{x}\|} = \frac{1}{\|\boldsymbol{A}^{-1}\|}$
  - $\|\boldsymbol{A}\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$, $\|\boldsymbol{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$
  - $\|\boldsymbol{A}\|_2$的计算涉及奇异值分解SVD

```
>> norm2= norm(A)
```



舍入误差对结果的影响

J. H. Wilkinson



矩阵计算的舍入误差分析

- 高斯消去法的舍入误差分析
  - 设数值解$\boldsymbol{x}_*$精确满足$(\boldsymbol{A}+\boldsymbol{E})\boldsymbol{x}_* = \boldsymbol{b}$
  - 进行LU分解的解法：$|\boldsymbol{E}| \le n\varepsilon_{\text{mach}}(3|\boldsymbol{A}| + 5|\hat{\boldsymbol{L}}||\hat{\boldsymbol{U}}|)$ (绝对值)
  - $\dfrac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|} \lesssim n\rho\varepsilon_{\text{mach}}$，　其中$\rho = \dfrac{\max_{i,j}|\hat{u}_{ij}|}{\max_{i,j}|a_{ij}|}$ (增长因子)
  - 不选主元,ρ可任意大, 部分选主元, ρ一般≤10
  - 高斯消去, 残差: $b - Ax_* = Ex_*$

from Golub's

两个重要结论

理论上限2^{n-1}

$$\|b - Ax_*\| \le \|E\|\|x_*\| \Rightarrow \frac{\|b - Ax_*\|}{\|A\|\|x_*\|} \le \frac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|} \lesssim n\rho\varepsilon_{\text{mach}}$$

相对残差

- 高斯消去, 解误差: $x - x_* = A^{-1}(b - Ax_*)$

$$\|x - x_*\| \le \|A^{-1}\|\|E\|\|x_*\| \Rightarrow \frac{\|x - x_*\|}{\|x_*\|} \lesssim n\rho\underbrace{\|A\|\|A^{-1}\|}_{\text{条件数}\kappa(\boldsymbol{A})}\varepsilon_{\text{mach}}$$

舍入误差对结果的影响

- 高斯消去法的舍入误差分析　　增长因子ρ很小
  - 一般情况下，部分主元高斯消去(LU分解)的相对残差很小
  - 若问题不病态(矩阵条件数不很大)，且使用部分主元高斯消去法，则将得到很准确的解
  - 若不选主元，则相对残差、解的误差都可能很大
  - 更一般的两个结论：

$$\frac{\|b - Ax_*\|}{\|A\|\|x_*\|} \le \frac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|}$$

相对残差大，说明方程求解算法不稳定

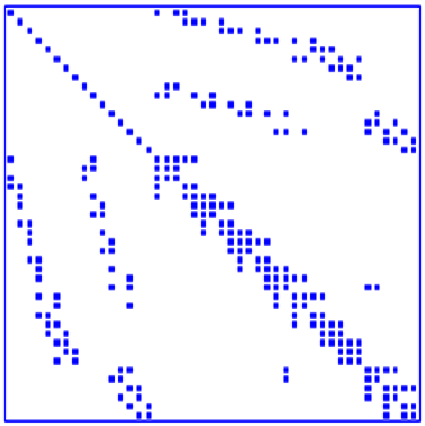
$$\frac{\|x - x_*\|}{\|x_*\|} \le \frac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|}\|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$$

解的误差受算法稳定性与问题敏感性共同影响

稀疏矩阵与条带状矩阵

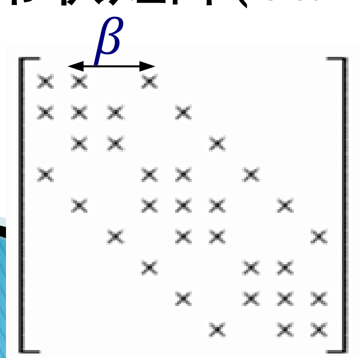
- 基本概念
- 存储格式与生成矩阵
- 带状矩阵方程的求解

基本概念



- 稀疏矩阵 (sparse matrix)
  - 存在大量零元素的矩阵　非零元数目为O(n)
  - Wilkinson's definition: "***matrices that allow special techniques to take advantage of the large number of zero elements***."
  - 强调采用特殊技术提高计算与处理效率 (不处理零元素)
  - 用二维数组存储的矩阵, 即使有些零元素, 也不算稀疏矩阵
  - 非零元的数目、稀疏度
  - 带状矩阵(band matrix):

>>density= nnz(A)/prod(size(A))
>>sparsity= 1- density



带宽: 非零元到主对角线的最大距离

>>[i, j]= find(A)
>>bandwidth= max(abs(i-j))

(有些书也称之为"半带宽")

存储格式

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

- 存储稀疏矩阵的数据结构
  - 三元组 (COO格式)

aa	1	2	5	3	4	6	7	8	9	10	11
row	1	1	2	2	2	3	3	3	4	4	5
col	1	3	4	1	2	1	3	5	2	4	5

    - 非零元的排列顺序可任意;　冗余: 连续的相同行编号
  - 压缩稀疏行 (CSR格式)　　关键是方便操作某行

aa	1	2	5	3	4	6	7	8	9	10	11
col	1	3	4	1	2	1	3	5	2	4	5
prow	1	3	6	9	11	12					

    - 按第1行, 第2行, … 顺序存非零元; prow为各行第一个元素位置

压缩稀疏列 (CSC格式)

aa	1	3	6	4	9	2	7	5	10	8	11
row	1	2	3	2	4	1	3	2	4	3	5
pcol	1	4	6	8	10	12					

存储格式

- Matlab中的稀疏矩阵
  - 采用**压缩稀疏列**(CSC)方式存储
  - 相反的矩阵格式转换
  - 若阶数很大，稠密格式A无法存可直接用sparse命令生成
  - (相当于用COO格式输入数据)
  - spdiags生成带状稀疏矩阵
  - spy看矩阵的非零元分布

```
>>S = sparse(A)
```

```
>>A = full(S)
```

```
>>S = sparse(i, j, x, m, n)
```

生成的S满足 $\begin{cases} [i,j,x] = \mathrm{find}(S) \\ [m,n] = \mathrm{size}(S) \end{cases}$

```
>>S = spdiags([a b c], [-1, 0, 1], n, n)
```

whos看变量类型

```
>>whos
 Name       Size        Bytes  Class     Attributes
  A         20x20        3200  double
  S         20x20         528  double    sparse
```

方程求解

- Matlab中对稀疏矩阵处理
  - 大多数矩阵运算和函数可直接用于稀疏矩阵
  - 例如："\"会判断稀疏矩阵，并用高效的特殊算法求解
  - 非零元数目nnz(S)是影响运算时间与存储量的主要因素
- 三对角系数矩阵的方程求解

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_1 & b_2 & c_2 & & & \\ & a_2 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}$$

```
function x = tridisolve(a,b,c,d)
x = d;  n = length(x);
for j = 1:n-1
   mu = a(j)/b(j);
   b(j+1) = b(j+1) - mu*c(j);
   x(j+1) = x(j+1) - mu*x(j);
end
x(n) = x(n)/b(n);
for j = n-1:-1:1
   x(j) = (x(j)-c(j)*x(j+1))/b(j);
end
```

  - 若需选主元, 用spdiags生成A再用"\"算符

Matlab中处理稀疏矩阵小结

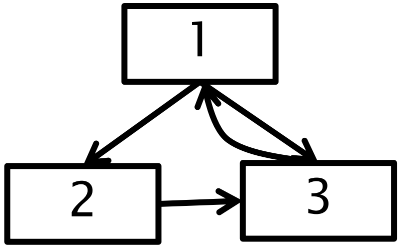
- 生成或转换成稀疏矩阵
  - sparse(X) or sparse(i, j, s, m, n); sparse(m, n)
- 显示矩阵非零元分布： spy(X)
- 构造特殊的稀疏矩阵
  - speye(n, m); spones(pattern); spdiags(B, d, m, n)
- 随机稀疏矩阵生成器
  - sprand(S) or sprand(m, n, density)
- 统计非零元数目，获取非零元信息
  - nnz(X); nonzeros(A), find(X) (索引, 三元组); nzmax (存储空间)
- 其他： \, lu, qr, svd等各种运算都支持稀疏矩阵

Google的PageRank

- PageRank及其数学问题
- 迭代计算的收敛性
- 三种计算方法
- 实用的幂法程序

PageRank™的计算

- Google网络搜索
  - PageRank技术是其创立之初的关键创新之一
  - 搜索分两步: 找到匹配关键词的网页, 对它们排序显示
  - L. Page和S. Brin于1998年提出PageRank算法
  - 给出网页信息可靠/重要性的指标(PageRank), 带来突破
  - 怎样的网页PageRank高? 被推荐, 被链接到
- 数学模型
  - n: 网页的总数, $\boldsymbol{G} = (g_{ij})_{n\times n}$: 网页链接矩阵
  - 若网页$\boldsymbol{j}$链接到网页$\boldsymbol{i}$, 则$g_{ij} = 1$, 否则$g_{ij} = 0$



$$\boldsymbol{G} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

大规模、非对称、稀疏阵

PageRank™的计算

- 数学模型(续)
  - $\boldsymbol{G}$的列元素之和$c_j = \sum_i g_{ij}$, 等于网页j的"出度"
  - 访问网页的Markov过程, 到达网页的极限概率定义为它的PageRank (按概率p沿链接跳, 按概率1-p随便跳)
  - 设当前在网页j、下一步到网页i的条件概率为a_{ij}:

    i在j的链接上: $p \cdot 1/c_j + (1-p) \cdot 1/n$ 乘g_{ij}

    i不在j的链接上: $(1-p) \cdot 1/n$ 乘$(1-g_{ij})$

    $$a_{ij} = \frac{p g_{ij}}{c_j} + \underbrace{\frac{1-p}{n}}_{=\delta}$$

    或0
  - 若$c_j = 0$呢? $a_{ij} = 1/n$
  - a_{ij}组成转移概率矩阵$\boldsymbol{A}$:

    $$\begin{cases} \boldsymbol{A} = p\boldsymbol{GD} + \delta \boldsymbol{e}\boldsymbol{e}^T, \ \boldsymbol{D} = \mathrm{diag}(1/c_j) \\ \boldsymbol{A} = p\boldsymbol{GD} + \boldsymbol{e}\boldsymbol{f}^T, \ f_j = \begin{cases} \delta, & c_j \neq 0 \\ 1/n, & c_j = 0 \end{cases} \end{cases}$$
  - 设$\boldsymbol{x}^{(k)}$为第k次跳转后在各个网页的概率, 则

    $\boldsymbol{x}^{(k+1)} = \boldsymbol{A}\boldsymbol{x}^{(k)}$为再跳一次后, 在各网页的概率

PageRank™的计算

- 数学模型(续)
  - PageRank: 随机”冲浪”过程访问网页的**极限**概率

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{A}\boldsymbol{x}^{(k)} \qquad \lim_{k\to\infty} \boldsymbol{x}^{(k)} = ?$$

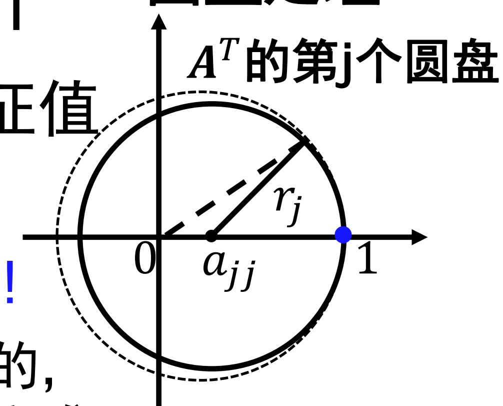
它存在吗?

- 迭代过程收敛性分析

$$\begin{cases} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{x} \\ \sum_{i=1}^{n} x_i = 1 \end{cases}$$

  - 数学本质: 求1对应的特征向量$\boldsymbol{x}$
  - 矩阵$\boldsymbol{A}$的特点: $a_{ij} > 0$, 第j列元素之和为1
  - $|\lambda(\boldsymbol{A})| \le \|\boldsymbol{A}\| = 1$; $\boldsymbol{A}^T \boldsymbol{e} = \boldsymbol{e}$ $\Longrightarrow$ 1是主特征值
  - 幂法的迭代过程一定收敛!
- 计算中不需对$\boldsymbol{x}^{(k)}$规格化;
  实际用时不形成矩阵$\boldsymbol{A}$

“圆盘定理”



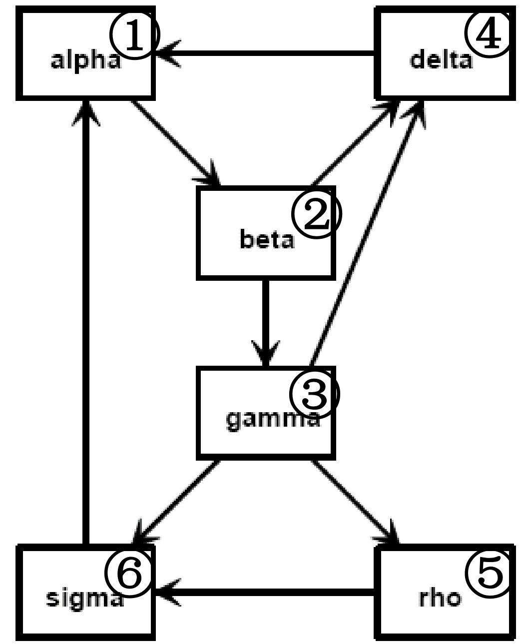
唯一主特征值!

可证明是单的,
因此极限概率唯一

PageRank™的计算

- 一个小型的例子(设G无全零列)

```
>> i= [2 3 4 4 5 6 1 6 1];
>> j= [1 2 2 3 3 3 4 5 6];
>> n=6; G= sparse(i, j, 1, n, n);
```



$(e^T x = 1)$

甚至不生成G

- 幂法: $A = pGD + \delta e e^T \Rightarrow x^{(k+1)} = pGDx^{(k)} + \delta e$

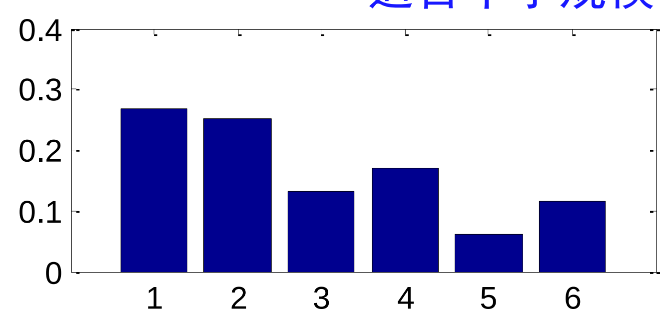
```
>> c= sum(G);
>> D= spdiags(1./c, 0, n, n);
>> x= p*G*D*x+ (1-p)/n;
```

$Ax = x \Rightarrow x = pGDx + \delta e$

- 变为方程求解的解法 $\Rightarrow (I - pGD)x = \delta e$ 保留了稀疏性，适合中小规模

```
>> I= speye(n,n); e= ones(n,1);
>> x= (I- p*G*D) \ ((1-p)/n*e);
```

- 解:



- 逆迭代法(奇异方程): x=(I - A)\e
  (部分主元高斯消去解近奇异矩阵) x= x/sum(x)

一个实用的幂法程序

```
function [x,cnt] = pagerankpow(G)
[n,n] = size(G);
for j = 1:n
   L{j} = find(G(:,j));   %某列非零元
   c(j) = length(L{j});  %网页的出度
end
p = .85;  delta = (1-p)/n;
x = ones(n,1)/n;
z = zeros(n,1);  cnt = 0;
while max(abs(x-z)) > .0001
   z = x;  x = zeros(n,1);
   for j = 1:n
      if c(j) == 0,  x = x + z(j)/n;
      else
         x(L{j}) = x(L{j}) + z(j)/c(j);
      end
   end
   x = p*x + delta;   cnt = cnt+1;
end
```

(单元数组cell)

- 习题2.27
- 迭代中不使用矩阵A, G
- z, x分别为前一步和当前的迭代解

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{z}$$

网页j出度

$$a_{ij} = \begin{cases} p\, g_{ij}/c_j + \delta, & c_j \neq 0 \\ 1/n = p/n + \delta, & c_j = 0 \end{cases}$$

$$\boldsymbol{A} = p\widetilde{\boldsymbol{G}} + \delta \boldsymbol{e}\boldsymbol{e}^T \Rightarrow \boldsymbol{x} = p\widetilde{\boldsymbol{G}}\boldsymbol{z} + \delta \boldsymbol{e}$$

$$\widetilde{\boldsymbol{G}}\boldsymbol{z} = [\tilde{\boldsymbol{g}}_1 \dots \tilde{\boldsymbol{g}}_n]\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \sum_{j=1}^{n} z_j \tilde{\boldsymbol{g}}_j$$

- 更多关于pagerank, 看"教学资源"–《Matrix Methods in Data Mining》第12章

Cholesky分解

- 分解定理
- 分解算法

Cholesky分解定理

1.如果实对称阵*A*各阶顺序主子式≠0，则它可唯一分解为：

$$A = LDL^T ,$$

其中*L*为单位下三角阵，*D*为对角阵

2.若矩阵*A*同时正定，则存在非奇异下三角阵*L*，

$$A = LL^T$$

若限定*L* 的对角元>0, 则此分解唯一

实对称阵正定 $\Leftrightarrow$

- $\forall \boldsymbol{x} \neq \boldsymbol{0}$, 二次型$\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} > 0$
- $\boldsymbol{A}$的各阶顺序主子式都大于0
- $\boldsymbol{A}$的所有特征值都大于0

In 2x2 case, for example,

$$\begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{bmatrix}$$

which implies

$$l_{11} = \sqrt{a_{11}}, \quad l_{21} = a_{21} / l_{11}, \quad l_{22} = \sqrt{a_{22} - l_{21}^2}$$

Cholesky分解算法

可根据LU分解算法的kij版本进行修改，考虑对称性；

原地存储: 仅使用A的下三角部分，L的结果将其覆盖

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n1} & \cdots & a_{nn} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}$$

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & l_{nn} \end{bmatrix} = A$$

1. For k=1 to n

1'. $a_{kk} = \sqrt{a_{kk}}$

2. For i= k+1 to n

3. $a_{ik} = a_{ik} / a_{kk}$

4. For j= k+1 to i

5. $a_{ij} = a_{ij} - a_{ik} \cdot a_{jk}$

6. End

7. End

8. End

注意：L不是单位下三角

(没有选主元)

第2种算法

For k =1 to n

$$a_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} a_{kj}^2}$$

For $i = k+1$ to n

$$a_{ik} = (a_{ik} - \sum_{j=1}^{k-1} a_{ij} a_{kj}) / a_{kk}$$

End

End

每步计算都求得L中的某个元素!

对称正定矩阵的Cholesky分解算法

- 算法复杂度：仅需要$n^3/6$次乘法运算和差不多的加法运算，是LU分解的一半计算量
- 对于对称正定矩阵，可证明：
- 所有n个平方根运算的操作数都为正数，算法可行
- **Cholesky**算法是稳定的**(**对舍入误差不敏感**)**，不需要选主元
- **Matlab**命令为**chol,** 缺省得到上三角矩阵**(R'*R=A)**，可用于检测矩阵正定性

```
>>R = chol(S)
>>L = chol(S, 'lower')
```

- 仅使用矩阵***A***上**(**下**)**三角部分的元素，因此下**(**上**)**三角部分元素没有用