高等数值算法与应用（二）

Advanced Numerical Algorithms & Applications

计算机科学与技术系 喻文健

Lecture 1

- 误差来源：计算之前的、计算过程中的
- Absolute Error & Relative Error
- Data Error & Computational Error
- Truncation Error & Rounding Error

  计算过程中产生的两种误差

- Forward Error & Backward Error

  就是最终误差　　最终误差在输入上的等效

- Sensitivity & Conditioning

  条件数用来度量问题对输入误差的放大倍数

$$= \frac{|\text{relative change in solution}|}{|\text{relative change in input data}|}$$

  对问题而言；**sensitive = ill-conditioned**

$$\frac{|[f(\hat{x}) - f(x)]/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|}$$

- Stability & Accuracy

  对算法而言；稳定指计算过程中误差对结果的影响不大**;** 以**BEA**论，邻近问题的准确解

  导致最终计算准确的条件：不敏感的问题＋稳定的算法

- Rounding Error: Overflow, Underflow, Rounding ($\varepsilon_{\text{mach}}$), Cancellation

内容概要

- 矩阵分解及其应用
  - 六大分解简介
  - 矩阵与线性方程组求解基本理论
  - **LU**分解, **Cholesky**分解及其应用
  - **QR**分解与线性最小二乘问题
  - 特征值分解、奇异值分解**(SVD)**及其应用
  - 稀疏矩阵的直接解法

“The Big-Six” Matrix Decomposition

Wenjian Yu

“六大”矩阵分解 – “十大算法”，有用且稳定

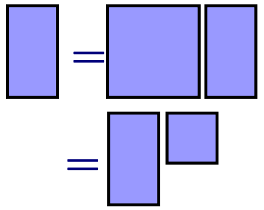
- Cholesky分解 $A = R^{\mathrm{T}} R$ $A = LL^T$
  - A对称正定方阵 $A = LDL^{\mathrm{T}}$

高斯(1777-1855)提出一种消去过程来化简二次型

- 选主元的LU分解 $PA = LU$
  - A非奇异方阵 $PAQ = LU$

Jacobi(1804-1851)提出对双线性函数 $\varphi(x, y)$的分解

- QR分解 $A = QR$
  - 任意A矩阵



1907, Schmidt

- 特征值分解（谱分解） $A = X\Lambda X^{-1}$ $A = Q\Lambda Q^T$
  - A非亏损方阵，或实对称方阵

1829, Cauchy

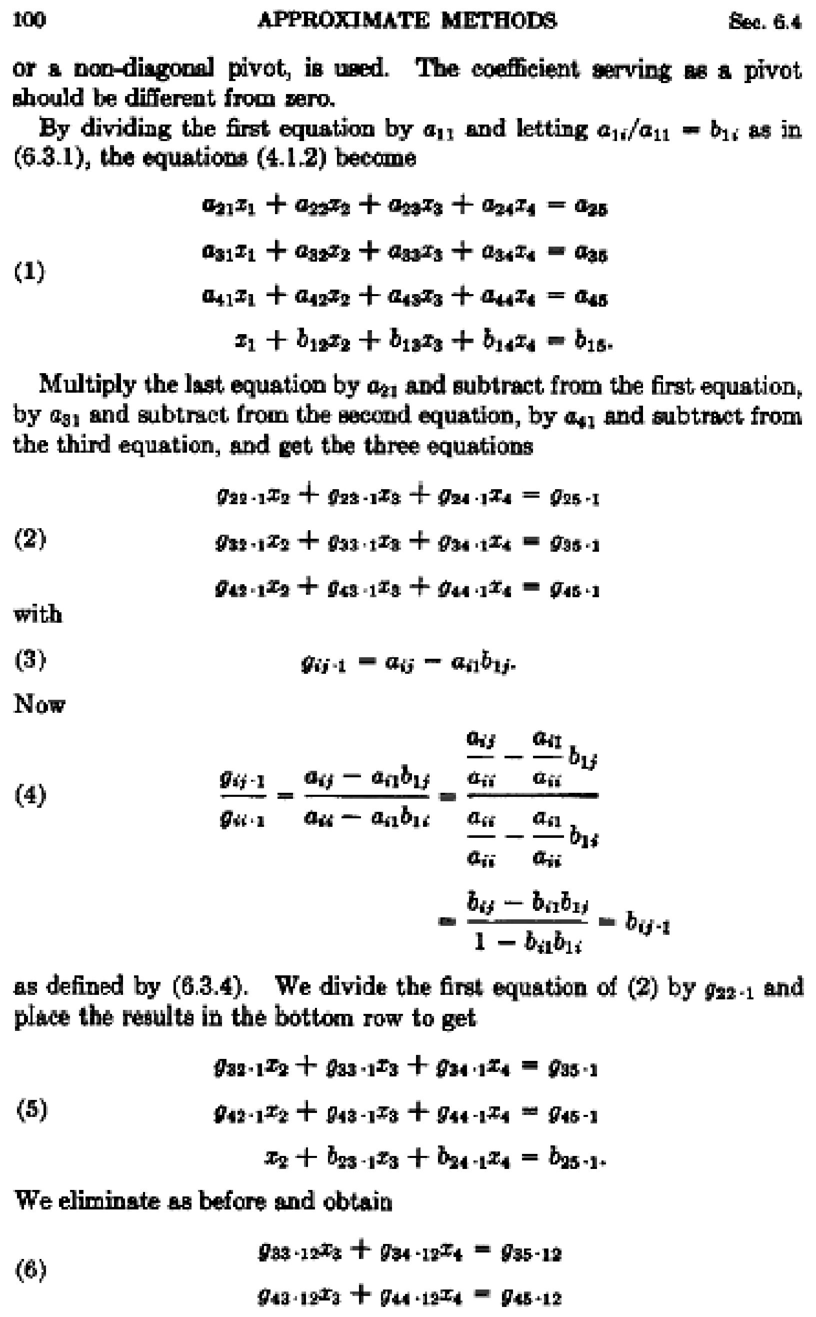
- Schur分解 $A = QRQ^T$ $A = QRQ^H$
  - A实方阵，R拟上三角阵；A复矩阵，R上三角阵

1909, Schur

- 奇异值分解 $A = U\Sigma V^T$
  - 任意A矩阵

1873, Beltrami
1874, Jordan

P. S. Dwyer, *Linear Computations*, 1951



100 APPROXIMATE METHODS Sec. 6.4

or a non-diagonal pivot, is used. The coefficient serving as a pivot should be different from zero.

By dividing the first equation by a_{11} and letting $a_{1i}/a_{11} = b_{1i}$ as in (6.3.1), the equations (4.1.2) become

$$(1)\qquad \begin{aligned} a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= a_{25}\\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= a_{35}\\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= a_{45}\\ x_1 + b_{12}x_2 + b_{13}x_3 + b_{14}x_4 &= b_{15}. \end{aligned}$$

Multiply the last equation by a_{21} and subtract from the first equation, by a_{31} and subtract from the second equation, by a_{41} and subtract from the third equation, and get the three equations

$$(2)\qquad \begin{aligned} g_{22\cdot1}x_2 + g_{23\cdot1}x_3 + g_{24\cdot1}x_4 &= g_{25\cdot1}\\ g_{32\cdot1}x_2 + g_{33\cdot1}x_3 + g_{34\cdot1}x_4 &= g_{35\cdot1}\\ g_{42\cdot1}x_2 + g_{43\cdot1}x_3 + g_{44\cdot1}x_4 &= g_{45\cdot1} \end{aligned}$$

with

$$(3)\qquad g_{ij\cdot1} = a_{ij} - a_{i1}b_{1j}.$$

Now

$$(4)\qquad \frac{g_{ij\cdot1}}{g_{ii\cdot1}} = \frac{a_{ij} - a_{i1}b_{1j}}{a_{ii} - a_{i1}b_{1i}} = \frac{\dfrac{a_{ij}}{a_{ii}} - \dfrac{a_{i1}}{a_{ii}}b_{1j}}{\dfrac{a_{ii}}{a_{ii}} - \dfrac{a_{i1}}{a_{ii}}b_{1i}}$$

$$= \frac{b_{ij} - b_{i1}b_{1j}}{1 - b_{i1}b_{1i}} = b_{ij\cdot1}$$

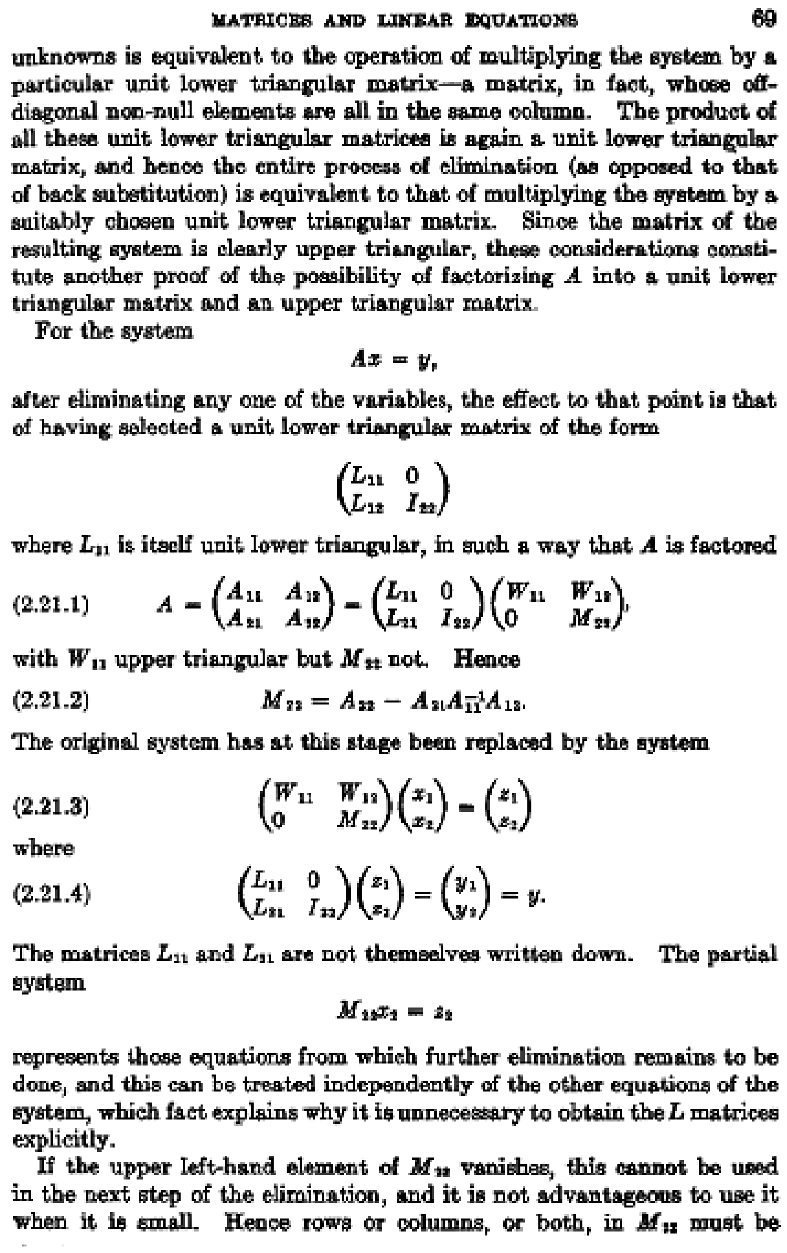
as defined by (6.3.4). We divide the first equation of (2) by $g_{22\cdot1}$ and place the results in the bottom row to get

$$(5)\qquad \begin{aligned} g_{32\cdot1}x_2 + g_{33\cdot1}x_3 + g_{34\cdot1}x_4 &= g_{35\cdot1}\\ g_{42\cdot1}x_2 + g_{43\cdot1}x_3 + g_{44\cdot1}x_4 &= g_{45\cdot1}\\ x_2 + b_{23\cdot1}x_3 + b_{24\cdot1}x_4 &= b_{25\cdot1}. \end{aligned}$$

We eliminate as before and obtain

$$(6)\qquad \begin{aligned} g_{33\cdot12}x_3 + g_{34\cdot12}x_4 &= g_{35\cdot12}\\ g_{43\cdot12}x_3 + g_{44\cdot12}x_4 &= g_{45\cdot12} \end{aligned}$$

A. S. Householder, *Principles of Numerical Analysis*, 1954



MATRICES AND LINEAR EQUATIONS 69

unknowns is equivalent to the operation of multiplying the system by a particular unit lower triangular matrix—a matrix, in fact, whose off-diagonal non-null elements are all in the same column. The product of all these unit lower triangular matrices is again a unit lower triangular matrix, and hence the entire process of elimination (as opposed to that of back substitution) is equivalent to that of multiplying the system by a suitably chosen unit lower triangular matrix. Since the matrix of the resulting system is clearly upper triangular, these considerations constitute another proof of the possibility of factorizing A into a unit lower triangular matrix and an upper triangular matrix.

For the system

$$Ax = y,$$

after eliminating any one of the variables, the effect to that point is that of having selected a unit lower triangular matrix of the form

$$\begin{pmatrix} L_{11} & 0 \\ L_{12} & I_{22} \end{pmatrix}$$

where L_{11} is itself unit lower triangular, in such a way that A is factored

$$(2.21.1)\qquad A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{22} \end{pmatrix} \begin{pmatrix} W_{11} & W_{12} \\ 0 & M_{22} \end{pmatrix},$$

with W_{11} upper triangular but M_{22} not. Hence

$$(2.21.2)\qquad M_{22} = A_{22} - A_{21}A_{11}^{-1}A_{12}.$$

The original system has at this stage been replaced by the system

$$(2.21.3)\qquad \begin{pmatrix} W_{11} & W_{12} \\ 0 & M_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

where

$$(2.21.4)\qquad \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{22} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = y.$$

The matrices L_{11} and L_{21} are not themselves written down. The partial system

$$M_{22}x_2 = z_2$$

represents those equations from which further elimination remains to be done, and this can be treated independently of the other equations of the system, which fact explains why it is unnecessary to obtain the L matrices explicitly.

If the upper left-hand element of M_{22} vanishes, this cannot be used in the next step of the elimination, and it is not advantageous to use it when it is small. Hence rows or columns, or both, in M_{22} must be

为什么这些矩阵分解这么重要?

- 一些历史
  - 这些分解方法的思想在1909年前都已确立
  - **von Numann, Householder, Wilkinson**的贡献
- 矩阵分解的好处
  - 简化问题，导出算法

    例如：求解**$Ax=b$, $A^Tx=b$,** 计算**$x^TA^{-1}x$**
  - 计算的复用，降低计算代价

    将高斯消去法分成两步，第一步**LU**分解代价高，可复用
  - 发现不同算法的内在统一性，简化误差分析

    高斯消去法的五种主要形式
  - 许多矩阵分解可以更新，节省计算量

    例如：求**$A+xx^T$**的**Cholesky**分解
  - 根据矩阵分解，可开发标准、有效的程序包

Fundamentals of Matrix and the Solution of Linear Equations

Wenjian Yu

- 本节主要内容
  - 向量范数、矩阵范数
  - 矩阵的条件数
  - 线性方程组求解的敏感性分析
  - 残差（residual）与算法稳定性

Vector Norms

模

Magnitude, modulus, or absolute value for scalars generalizes to *norm* for vectors

We will **often use** p-norms, defined by

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

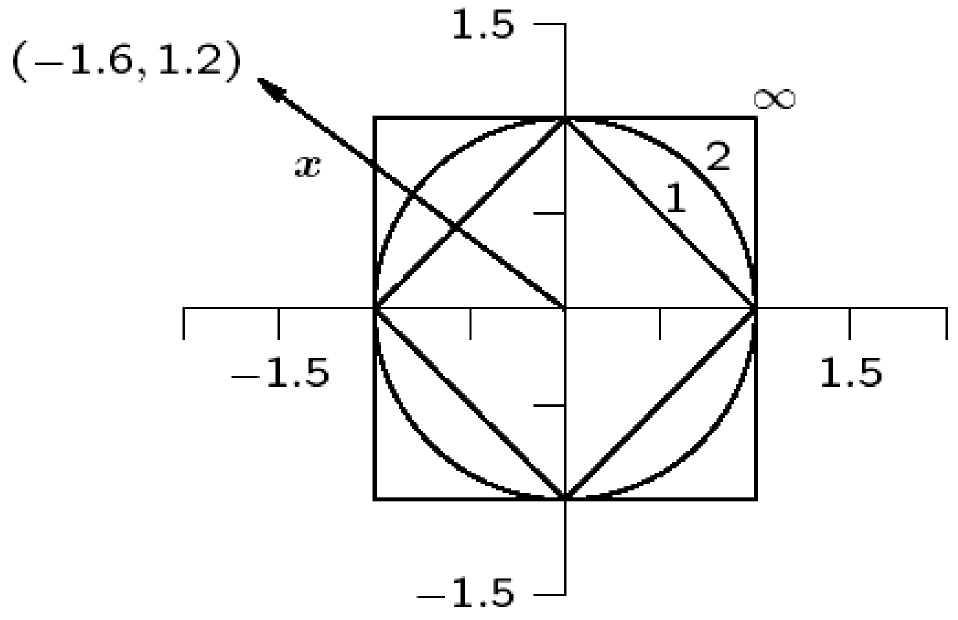
for integer $p > 0$ and n-vector $\boldsymbol{x}$

Important special cases:

- 1-norm: $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$ ***Manhattan norm***
- 2-norm: $\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$ ***Euclidean norm***
- ∞-norm: $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$

Vector Norms, continued

Drawing shows unit sphere in two dimensions for each of these norms:



按几种范数定义出的单位圆

Norms have following values for vector shown:

$$\|x\|_1 = 2.8, \quad \|x\|_2 = 2.0, \quad \|x\|_\infty = 1.6$$

In general, for any vector x in $\mathbb{R}^n$,

$$\|x\|_1 \ge \|x\|_2 \ge \|x\|_\infty$$

$\|\boldsymbol{x}\|_1 \le \sqrt{n}\|\boldsymbol{x}\|_2$, $\|\boldsymbol{x}\|_2 \le \sqrt{n}\|\boldsymbol{x}\|_\infty$, and $\|\boldsymbol{x}\|_1 \le n\|\boldsymbol{x}\|_\infty$

All *p*-norms are equivalent, except for differing by a constant

Properties of Vector Norms

For any vector norm,

正定性、齐次性、
三角不等式

1. $\|\boldsymbol{x}\| > 0$ if $\boldsymbol{x} \neq \boldsymbol{o}$

2. $\|\gamma\boldsymbol{x}\| = |\gamma| \cdot \|\boldsymbol{x}\|$ for any scalar γ

三角不等式

3. $\|\boldsymbol{x}+\boldsymbol{y}\| \leq \|\boldsymbol{x}\|+\|\boldsymbol{y}\|$ (triangle inequality)

In more general treatment, these properties taken as *definition* of vector norm

例：$\|\boldsymbol{x}\| = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x})^{1/2}$

$\boldsymbol{A}$为对称正定矩阵

Useful variation on triangle inequality:

$$\big|\,\|\boldsymbol{x}\| - \|\boldsymbol{y}\|\,\big| \leq \|\boldsymbol{x} - \boldsymbol{y}\|$$

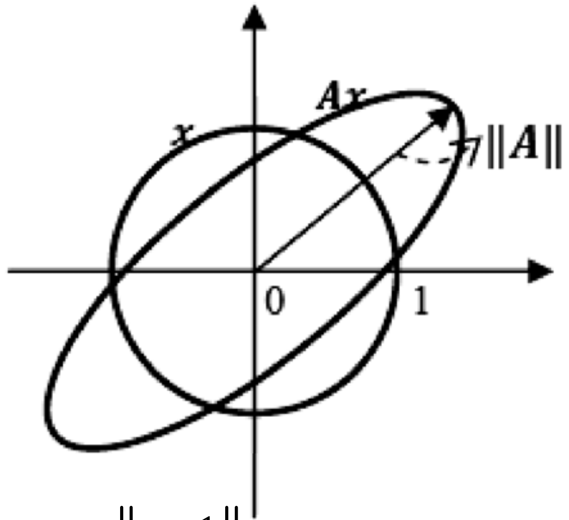
Matrix Norms

Matrix norm corresponding to given vector norm defined by

$$\|A\| = \max_{x \neq o} \frac{\|Ax\|}{\|x\|}$$

算子范数

向量范数诱导出的范数

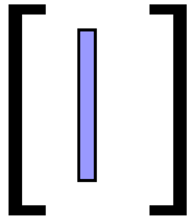


Norm of matrix measures maximum stretching matrix does to any vector in given vector norm

$$\|A^{-1}\| = ?$$

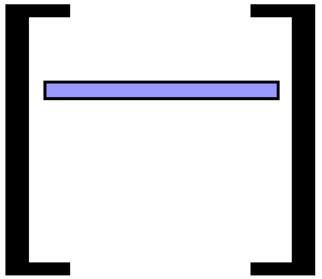
Matrix norm corresponding to vector 1-norm is maximum absolute column sum,

$$\|A\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$



Matrix norm corresponding to vector ∞-norm is maximum absolute row sum,

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$



How to remember?

How to prove?

Properties of Matrix Norms

Any matrix norm satisfies: 范数的普遍定义

1. $\|\boldsymbol{A}\| > 0$ if $\boldsymbol{A} \neq \boldsymbol{O}$

2. $\|\gamma \boldsymbol{A}\| = |\gamma| \cdot \|\boldsymbol{A}\|$ for any scalar γ

3. $\|\boldsymbol{A}+\boldsymbol{B}\| \leq \|\boldsymbol{A}\| + \|\boldsymbol{B}\|$

例：**Frobenius范数**

$$\|A\|_F = \left(\sum_{j=1}^{n}\sum_{i=1}^{n}|a_{i,j}|^2\right)^{1/2}$$

Matrix norms we have defined also satisfy

4. $\|\boldsymbol{AB}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{B}\|$ 对方阵的要求

5. $\|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{x}\|$ for any vector $\boldsymbol{x}$ 相容性条件，诱导范数满足的条件

Condition Number of Matrix

Condition number of square nonsingular matrix A defined by

$$\operatorname{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

其值与采用的范数有关，但对相关分析影响不大

回忆条件数的普遍定义

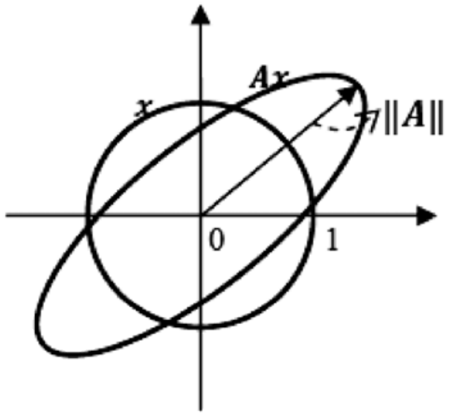
$$\left| \text{结果的相对误差} \right| = \operatorname{cond} \times \left| \text{输入数据的相对误差} \right|$$

By convention, $\operatorname{cond}(A) = \infty$ if A singular

Since

$$\|A\| \cdot \|A^{-1}\| = \left(\max_{x \neq o} \frac{\|Ax\|}{\|x\|} \right) \cdot \left(\min_{x \neq o} \frac{\|Ax\|}{\|x\|} \right)^{-1},$$

condition number measures ratio of maximum stretching to maximum shrinking matrix does to any nonzero vectors



矩阵的行列式不能指示是否近奇异

Large $\operatorname{cond}(A)$ means A nearly singular

矩阵变换造成单位圆的扭曲、变形程度

矩阵条件数的定义：$\left(\max_{x\neq o}\frac{\|Ax\|}{\|x\|}\right)\cdot\left(\min_{x\neq o}\frac{\|Ax\|}{\|x\|}\right)^{-1}$

Properties of Condition Number

1. For any matrix A, $\text{cond}(A)\geq 1$

2. For identity matrix, $\text{cond}(I)=1$

3. For any matrix A and scalar γ,

$$\text{cond}(\gamma A)=\text{cond}(A)$$

4. For any diagonal matrix $D=\text{diag}(d_i)$,

$$\text{cond}(D)=(\max|d_i|)/(\min|d_i|)$$

一般问题的条件数可能小于**1**

若采用**2-**范数，则 **cond(*Q*)=1, *Q***为正交阵 **cond(*QA*)=cond(*AQ*)= cond(*A*)**

Matrix_cond.m

Computing Condition Number

Definition of condition number involves matrix inverse, so nontrivial to compute

Computing condition number from definition would require much more work than computing solution whose accuracy to be assessed

比需要评价精度的求解过程更费事

In practice, condition number estimated inexpensively as byproduct of solution process

副产品

Matrix norm $\|\boldsymbol{A}\|$ easily computed as maximum absolute column sum (or row sum, depending on norm used)

关键：

Estimating $\|\boldsymbol{A}^{-1}\|$ at low cost more challenging

详细见课本例**2.7, p. 48-49**

若使用**2-**范数，条件数通过奇异值分解**(SVD)**得到

- MATLAB中相关函数
  - cond(A)或cond(A, 2)用于计算2-条件数，它调用函数svd(A)，适合于较小的矩阵。
  - cond(A, 1)计算1-条件数，它调用函数inv(A)，运算量小于cond(A, 2)。
  - cond(A, inf)计算∞-条件数，它调用函数inv(A)，等同于计算cond(A’, 1)。
  - condest(A)估算1-条件数，它使用lu(A)以及Higham和Tisseur 2000年提出的一个算法，特别适合于大型稀疏矩阵。
  - rcond(A)估算1-条件数的倒数，它使用lu(A)以及由LINPACK和LAPACK项目组开发的一个较老的算法。

线性方程组求解问题的敏感性

考虑右端项变化的误差分析

Condition number yields error bound for computed solution to linear system

Let x be solution to $Ax = b$, and let $\hat{x}$ be solution to $A\hat{x} = b + \Delta b$

If $\Delta x = \hat{x} - x$, then

$$b + \Delta b = A(\hat{x}) = A(x + \Delta x) = Ax + A\Delta x,$$

which leads to bound

$$\|\Delta x\| = \|A^{-1}\Delta b\| \le \|A^{-1}\| \cdot \|\Delta b\|$$

$$\|b\| = \|Ax\| \le \|A\| \cdot \|x\|$$

$$\frac{\|\Delta x\|}{\|x\|} \le \text{cond}(A)\frac{\|\Delta b\|}{\|b\|}$$

for possible relative change in solution due to relative change in right-hand side b

相对误差放大倍数的上限，与一般条件数定义保持一致

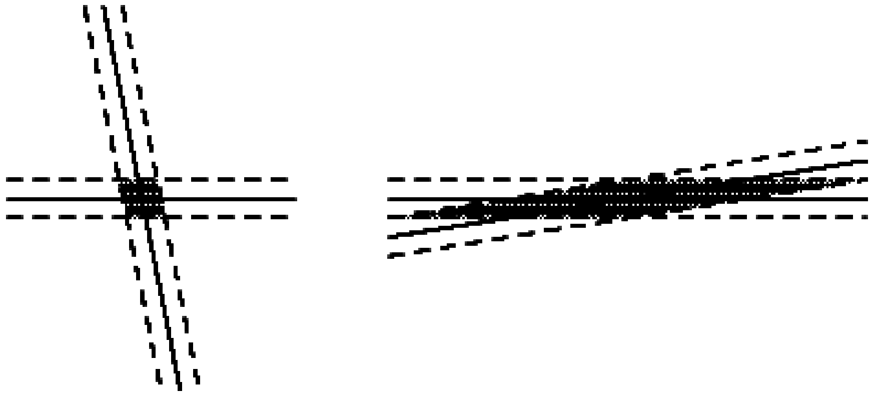
线性方程组求解问题的敏感性

Similar result holds for relative change in matrix: if $(A+E)\hat{x} = b$, then

$$\frac{\|\Delta x\|}{\|\hat{x}\|} \le \text{cond}(A)\frac{\|E\|}{\|A\|}$$

$$A\Delta x = -E\hat{x} \Rightarrow \Delta x = -A^{-1}E\hat{x}$$

In two dimensions, uncertainty in intersection point of two lines depends on whether lines nearly parallel



线性方程组解的敏感性的几何解释**(2x2**矩阵**)**

线性方程组求解：两直线求交点

左右两图分别反映了良态问题和病态问题两种情况。

线性方程组求解问题的敏感性

If input data accurate to machine precision, then bound for relative error in computed solution given by

IEEE标准的机器精度
～10^{-7} (单精度浮点)
～10^{-16} (双精度浮点)

数据传递误差

$$\frac{\|\hat{x} - x\|}{\|x\|} \le \text{cond}(A)\, \epsilon_{\text{mach}}$$

Computed solution loses about $\log_{10}(\text{cond}(A))$ decimal digits accuracy relative to accuracy of input

For example using 3-digit precision for problem with cond $> 10^3$, which yields no correct digits in solution

采用双精度浮点数，在一定程度上可克服条件数大带来的问题

说明两点

前面的解相对误差分析仅能控制最大分量

1. Normwise analysis bounds relative error in *largest* components of solution; relative error in smaller components can be much larger

对于系数差别很大的方程

Componentwise error bounds can be obtained, but somewhat more complicated

2. Conditioning of system affected by scaling

后面将介绍

Ill-conditioning can result from poor scaling as well as near singularity

Rescaling can help former, but not latter

Residual

残差、剩余

Residual vector of approximate solution $\hat{x}$ to linear system $Ax = b$ defined by

$$r = b - A\hat{x}$$

In theory, if A is nonsingular, then $\|\hat{x} - x\| = 0$ if, and only if, $\|r\| = 0$, but they are not necessarily small simultaneously

并不同时变小或变大

Since $r = -A\Delta x \Rightarrow \Delta x = -A^{-1}r$

$$\frac{\|\Delta x\|}{\|\hat{x}\|} \le \text{cond}(A) \frac{\|r\|}{\|A\| \cdot \|\hat{x}\|},$$

定义

small relative residual implies small relative error only if A well-conditioned

若相对残差很小，且问题良态，则解准确。

Residual, continued

If computed solution $\hat{x}$ exactly satisfies

向后误差分析

$$(A+E)\hat{x} = b,$$

then

相对残差：
$$\frac{\|r\|}{\|A\| \; \|\hat{x}\|} \le \frac{\|E\|}{\|A\|},$$

so large *relative residual* implies large backward error in matrix, and algorithm used to compute solution is unstable

若算法稳定，则相对残差一定小；
反过来，相对残差小，不一定算法稳定、不一定解准确

用“相对残差”作后验误差分析：

1. 若相对残差大，说明算法不稳定；

2. 若已知问题条件数很小，相对残差小则说明解准确

例：用4位十进制运算解方程

$$Ax = \begin{bmatrix} 0.913 & 0.659 \\ 0.457 & 0.330 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.254 \\ 0.127 \end{bmatrix} = b$$

解得 $\hat{x} = \begin{bmatrix} -0.0827 \\ 0.5 \end{bmatrix}$， $r = b - A\hat{x} = \begin{bmatrix} 0.0051 \\ -0.2061 \end{bmatrix} \times 10^{-3}$

$\|r\|_1 = 2.1 \times 10^{-4}$ ，相对残差也很小，但准确解 $x = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, **Cond(A)$_1$=1.6x10^4**

Pivoted LU Factorization and Cholesky Factorization

Wenjian Yu

- 本节主要内容
  - 高斯消去过程与部分选主元**LU**分解
  - 算法的稳定性与程序实现
  - 计算复用与矩阵更新
  - 提高解的准确度
  - 对称正定矩阵的**Cholesky**分解
  - 对称不定矩阵的处理
  - 有关软件与程序

高斯消去法的思路

- To solve linear system, transform it into one whose solution is same but easier to compute
- 怎样的线性方程组易于求解？
  - If one equation in system involves only one component of solution, that component can be computed by division
  - If another equation in system involves only one additional solution component we can solve for it by substitution
  - … … only one new component per equation … …
  - 系数矩阵是triangular matrix !
- 怎样变换线性方程组而不改变解？
  - Pre-multiply (from left) both sides of linear system ***Ax=b*** by any nonsingular matrix ***M***

怎样选取**M**矩阵以达到目的？

排列矩阵与初等消去矩阵

置换阵

- Permutation matrix *P* has one 1 in each row and column and zeros elsewhere.
  - *P*是正交矩阵，*P*的乘积也是排列阵
  - *PA* reorders rows of *A*
  - $\|\boldsymbol{P}\|=1$, cond(*P*) =1, cond(*PA*)=cond(*A*)
  - How about *AP*?

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

- Elementary elimination matrix *M* has form like
  - *M*为单位下三角阵，对角线下方除一列元素外均为零
  - *MA*的效果？
  - 将*M*对角线下方非零那列元素取相反数则得到其逆矩阵M^{-1}

$$\begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{bmatrix}$$

Gaussian Elimination

$\boldsymbol{M}_k$ called elementary elimination matrix, with form

$$\begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{bmatrix}$$

$\boldsymbol{m}_{k+i}$为相应元素与主元的比值

消元中用$\boldsymbol{m}_{k+i}$乘以当前行加到相应行上

To reduce general linear system $\boldsymbol{Ax} = \boldsymbol{b}$ to upper triangular form, first choose $\boldsymbol{M}_1$, with a_{11} as pivot, to annihilate first column of $\boldsymbol{A}$ below first row 主元

System becomes $\boldsymbol{M}_1\boldsymbol{Ax} = \boldsymbol{M}_1\boldsymbol{b}$, but solution unchanged

Next choose $\boldsymbol{M}_2$, using a_{22} as pivot, to annihilate second column of $\boldsymbol{M}_1\boldsymbol{A}$ below second row. System becomes $\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{Ax} = \boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{b}$, but solution still unchanged

Process continues for each successive column until all subdiagonal entries have been zeroed

Resulting upper triangular linear system

$$\boldsymbol{MAx} = \boldsymbol{M}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{Ax} = \boldsymbol{M}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{b} = \boldsymbol{Mb}$$

can be solved by back-substitution to obtain solution to original linear system $\boldsymbol{Ax} = \boldsymbol{b}$

LU Factorization

乘积结果为下三角部分合并

Product $L_k L_j$ unit lower triangular if $k < j$, so

$$L = M^{-1} = M_1^{-1} \cdots M_{n-1}^{-1} = L_1 \cdots L_{n-1}$$

unit lower triangular

$\boxed{M_k^{-1} \text{same as } M_k \text{ except signs of multipliers reversed}}$ L_k

By design, $U = MA$ upper triangular

So $A = LU$, with L unit lower triangular and U upper triangular 主对角线上全1

Thus, $Ax = b$ becomes $LUx = b$, and can be solved by forward-substitution in lower triangular system $Ly = b$, followed by back-substitution in upper triangular system $Ux = y$

在高斯消元过程中，对b操作的最终结果为y

$y = Mb$, transformed right hand side in Gaussian elimination

Gaussian elimination and LU factorization are two ways of expressing same solution process

Example: Gaussian Elimination

Use Gaussian elimination to solve linear system

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}$$

To annihilate subdiagonal entries of first column of $\boldsymbol{A}$, $\boldsymbol{M}_1\boldsymbol{A} =$

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix}$$

$$\boldsymbol{M}_1\boldsymbol{b} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$$

To annihilate subdiagonal entries of second column, $M_2M_1A =$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix}$$

Example Continued

$$M_2M_1b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

The original system is reduced to triangular system

$$\begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

which can now be solved by back-substitution to obtain $x = [-1 \quad 2 \quad 2]^T$

$$\boxed{L_1 = M_1^{-1}}$$

To write out LU factorization explicitly,

$$L = L_1L_2 =$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix}$$

So that,

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix}$$

选主元的高斯消去法

消元中的乘子应越小越好
避免舍入误差放大

- Why and how
  - Zero pivot causes break down of elimination procedure
  - 选主元: choose some entry with <u>large magnitude</u> in the unreduced submatrix, permute it into diagonal pivot position
  - 部分(列)选主元，全选主元
- 部分选主元的矩阵描述

$$U = M_{n-1}P_{n-1}\cdots M_2P_2M_1P_1A \qquad (1)$$

  - 矩阵P_i为初等交换阵，通过交换单位阵的第**i**行和第**j**行**(j>i)**得到。P_i为特殊的排列阵，其逆矩阵为自身
  - P_iM的效果为将矩阵M的第**i**行和第**j**行交换，MP_i的效果为将矩阵M的第**i**列和第**j**列交换
  - 矩阵M_i为初等消去矩阵

选主元的高斯消去法

- 部分选主元的矩阵描述

$$M_{n-1}^{-1}U = P_{n-1}M_{n-2}P_{n-1}P_{n-1}P_{n-2}\cdots M_1P_1A \quad (2)$$

  - $P_{n-1}M_{n-2}P_{n-1}$等价于将矩阵M_{n-2}的第**n-1**行和某个第**j**行**(j>n-1)**进行交换，然后对结果矩阵实施第**n-1**列和第**j**列交换，由于矩阵M_{n-2}的结构，易知 $P_{n-1}M_{n-2}P_{n-1}$ 仍是一个仅第**n-2**列有非对角线非零元的初等消去矩阵，记 $\widetilde{M}_{n-2}$

$$\widetilde{M}_{n-2}^{-1}M_{n-1}^{-1}U = P_{n-1}P_{n-2}M_{n-3}P_{n-3}\cdots M_1P_1A$$

$$=(P_{n-1}P_{n-2}M_{n-3}P_{n-2}P_{n-1})P_{n-1}P_{n-2}P_{n-3}\cdots M_1P_1A \quad (3)$$

  - 同理，$\widetilde{M}_{n-3} = P_{n-1}P_{n-2}M_{n-3}P_{n-2}P_{n-1}$ 为仅第**n-3**列有非对角线非零元的初等消去矩阵

$$\widetilde{M}_{n-3}^{-1}\widetilde{M}_{n-2}^{-1}M_{n-1}^{-1}U = P_{n-1}P_{n-2}P_{n-3}M_{n-4}P_{n-4}\cdots M_1P_1A$$

  - ……

选主元的高斯消去法

- 部分选主元的矩阵描述

$$\widetilde{M}_1^{-1}\cdots\widetilde{M}_{n-2}^{-1}M_{n-1}^{-1}U=P_{n-1}P_{n-2}\cdots P_1A$$

  - 记 $L_i=\widetilde{M}_i^{-1}, i=1,\ldots,n-2;\ L_{n-1}=M_{n-1}^{-1},$

$$L_1L_2\cdots L_{n-1}U=P_{n-1}P_{n-2}\cdots P_1A$$

  - 矩阵L_i为仅在对角线下方第**i**列有非零元的初等消去矩阵，$L=L_1L_2\cdots L_{n-1}$为单位下三角矩阵
  - $P=P_{n-1}P_{n-2}\cdots P_1$为排列矩阵

$$LU=PA \qquad (4)$$

  - 实际编程时，用一个整型数组Ip即可表示排列阵**P**，Ip[i]为它第i行的**1**所处的列编号。则Ip的初始值为(1,2,…,n)，第i步选主元时交换第i和第j_i行，为反映矩阵**P**的变化，需将Ip[i]和Ip[j_i]的值进行交换。
  - 得到P矩阵的Matlab命令为：I= eye(n); P=I(Ip, :);

选主元的高斯消去法

例(不显式进行行交换的部分主元消去过程)：不显式进行行交换，用高斯部分主元消去法分解矩阵

$$\boldsymbol{A}=\begin{bmatrix}1&2&2\\4&4&2\\4&6&4\end{bmatrix}.$$

解：将$\boldsymbol{A}$存储为二维数组 A，下面说明 A 中数据的变化过程。

第一步消去过程，p= [2, 1, 3]，乘数 m_{21}=-A(p[2], 1)/A(p[1], 1)=-1/4, m_{31}=-A(p[3], 1)/A(p[1], 1)=-1，这里利用 p[i]来索引行交换后矩阵的第 i 行，使用同样的方法更新其他元素，得到

$$\boldsymbol{A}=\begin{bmatrix}1/4&1&3/2\\4&4&2\\1&2&2\end{bmatrix}$$

取矩阵(i,j)位置元素变为A(p[i], j)

第二步消去过程，考察 A(p[2], 2)和 A(p[3], 2)，需交换第二、三行，得到 p= [2, 3, 1]，乘数 m_{32}=-A(p[3], 2)/A(p[2], 2)=-1/2，类似地更新 A 未消去部分的其他元素，得到

$$\boldsymbol{A}=\begin{bmatrix}1/4&1/2&1/2\\4&4&2\\1&2&2\end{bmatrix}.$$

L的元素为A(p[2], 1); A(p[3], 1:2)

最后，根据 A 和 p 的值，可以得到矩阵$\boldsymbol{L}$和$\boldsymbol{U}$。

$$\boldsymbol{L}=\begin{bmatrix}1&0&0\\1&1&0\\1/4&1/2&1\end{bmatrix},\boldsymbol{U}=\begin{bmatrix}4&4&2\\0&2&2\\0&0&1/2\end{bmatrix}.$$

U的元素为A(p[1], 1:3); A(p[2], 2:3); A(p[3], 3)

高斯消去法、LU分解

存储情况

- U存在于矩阵A的上三角部分, 组成L的倍乘因子覆盖 A的严格下三角部分
- 选主元时，不实际进行行交换，附加数组Ip便记录了新、旧行号的关系

算法复杂度

- **LU分解~n^3/3** 次浮点乘法、浮点加减法，向前、向后回代 **~ n^2** 次运算
- 任何情况下，避免直接计算**A**$^{-1}$，从性能、准确度考虑

$$PA = LU, \quad \Longrightarrow \quad x = U^{-1}L^{-1}Pb$$

LU分解的算法稳定性

- 假设计算解 $\hat{x}$ 满足 $(A+E)\hat{x}=b$, 则矩阵上的向后误差 $\frac{\|E\|}{\|A\|}$反映求解算法的稳定性
- **Wilkinson在[J. ACM, 1961]**对高斯消元法进行了仔细的向后误差分析，得出如下结论**(**做了简化**)**

$$\frac{\|E\|}{\|A\|} \le \rho n \varepsilon_{\text{mach}}$$

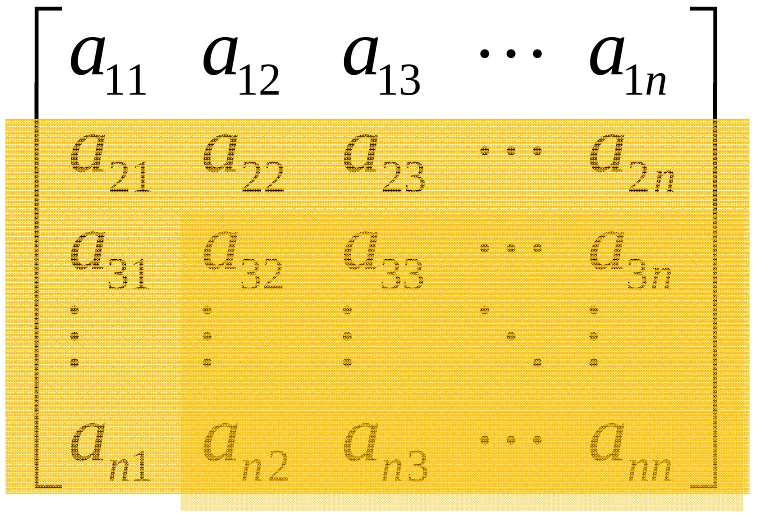
其中增长因子 ρ是***U***最大元素与 ***A***最大元素的比值；
增长因子是反映舍入误差的放大的关键参数，对各种直接解法的稳定性分析都有效。

- 不选主元LU分解，ρ可能很大，$\rho\to\infty$，因此算法不稳定；
- 部分选主元，$\rho \le 2^{n-1}$，但一般很小，大多数情况下<10，因此算法是稳定的。
- 怎么证明$\rho \le 2^{n-1}$? 构造极端的例子

LU分解的编程实现

```
1. For $k$=1 to $n$-1            ← 选主元语句插入
2.     For $i$= $k$+1 to $n$
3.         $a_{ik} := a_{ik}/a_{kk}$            行倍乘因子
4.         For $j$= $k$+1 to $n$
5.             $a_{ij} := a_{ij} - a_{ik}a_{kj}$
6.         End
7.     End
8. End
```

(KIJ version)



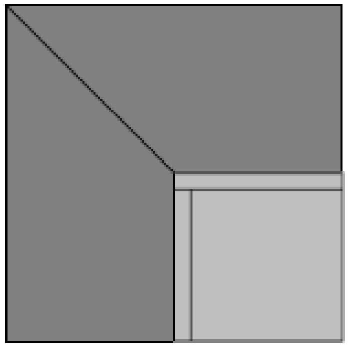
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

```
for ________
    for ________
        for ________
            $a_{ij} = a_{ij} - (a_{ik}/a_{kk})a_{kj}$
        end
    end
end
```

算法变种

- 三重循环变量k, i, j可变换顺序，最多可得到6种算法实现
- 直接三角分解+L对角线为1/U对角线为1 (Doolittle/Crout分解)，选主元变化
- 对不同的矩阵结构(稀疏存储)、不同的计算机系统结构(缓存、多CPU、向量化)，各个变种算法性能不同

LU分解的五种主要形式

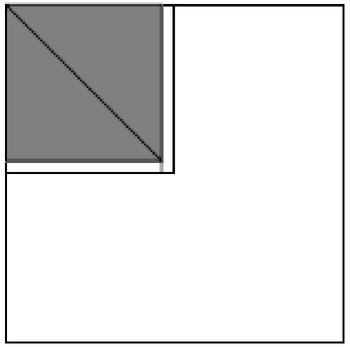


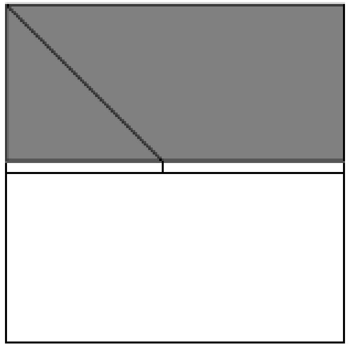
白色区域：原矩阵元素

灰色区域：分解结果矩阵元素

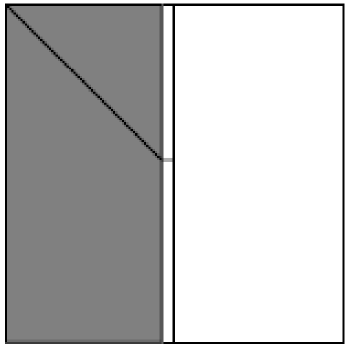
浅灰色区域：分解过程的中间结果

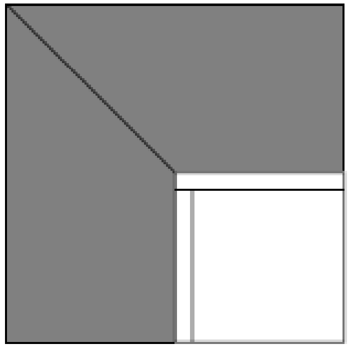
边界条状区域：当前处理元素





若标量描述是不同的算法，但都是相同的矩阵分解，只需做一次数值分析(稠密矩阵)





编程实现的一个细节

回代过程

Solve a upper triangular system $Ux = b$:

$$x_n = b_n / u_{nn}, \qquad x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \Big/ u_{ii}, \quad i = n-1, \cdots, 1.$$

Algorithm

```
For $j = n$ to 1                    {loop backwards over columns}
    If $u_{jj} = 0$ then stop       {stop if matrix is singular}
    $x_j := b_j / u_{jj}$           {compute solution component}
    For $i = 1$ to $j-1$
        $b_i := b_i - u_{ij} x_j$   {update right-hand side}
    End
End
```

标量乘向量

```
For $i = j+1$ to $n$
    $b_j := b_j - u_{ji} x_i$
End
$x_j := b_j / u_{jj}$
```

两个向量的内积

这种编程实现上的差别可能会导致程序性能的差别，特别是对于大规模的问题（依赖于所用的编程语言和计算机系统）

计算复用与矩阵更新

- 修改问题的求解
  - 多右端项问题

$$AX = B,\ \ PA = LU \Longrightarrow X = U^{-1}L^{-1}PB$$

    - LU分解矩阵A，针对B的每列做回代
  - 系数矩阵的秩1改变**(rank-one update)**

Sherman-Morrison formula $(A - uv^T)^{-1} = A^{-1} + A^{-1}u(1 - v^T A^{-1}u)^{-1}v^T A^{-1}$

$$x = (A - uv^T)^{-1}b = A^{-1}b + A^{-1}u(1 - v^T A^{-1}u)^{-1}v^T A^{-1}b$$

Steps: 1. Solve $Az = u$ for z, so $z = A^{-1}u$

2. Solve $Ay = b$ for y, so $y = A^{-1}b$

3. Compute $x = y + ((v^T y)/(1 - v^T z))z$

$\boxed{\mathcal{O}(n^2)\ \text{work}}$

提高解的准确度

- Rescaling系数调整
  - 行调整、列调整，影响矩阵条件数，也影响选主元
  - 尽量使矩阵元素大小差不多，有可能改善条件数
  - 系数大小极不平衡的例子： $\begin{bmatrix} 1 & 0 \\ 0 & \varepsilon \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \varepsilon \end{bmatrix}$ **Cond(A)=1/ε**

    右端扰动**[0, -ε]^T**，解由**[1, 1]^T**变成**[1, 0]^T.** 若进行行调整，则变成良态
  - 对一般的情况，如何调整不是很显然的。
- Iterative Refinement迭代求精
  - For approximate solution x_0, compute residual $r_0 = b - Ax_0$
  - Now solve $Az_0 = r_0$ and take $x_1 = x_0 + z_0$ as new solution

    $$Ax_1 = A(x_0 + z_0) = Ax_0 + Az_0 = (b - r_0) + r_0 = b$$
  - Repeat the process to refine solution successively to convergence
  - 保存原矩阵以及**L, U**
  - 在有些情况下有效果，同时应注意舍入误差**(抵消)**

Cholesky分解定理

1.如果对称矩阵***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，则存在实的非奇异下三角矩阵***L***，

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$$

若限定***L*** 对角线元素**>0**，则此分解唯一。

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；此外，仅取**A**的下三角部分，**L**的结果将其覆盖。

两点区别：**L**不是单位下三角；
A为对称矩阵

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

```
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
```

课本算法2.7

```
for $k$ =1 to n
    $a_{kk} = \sqrt{a_{kk}}$
    for $i = k+1$ to $n$
        $a_{ik} = a_{ik} / a_{kk}$
    end
    for $j = k+1$ to $n$
        for $i = j$ to $n$
            $a_{ij} = a_{ij} - a_{ik} \cdot a_{jk}$
        end
    end
end
```

Cholesky分解算法

采用直接三角分解的方式推导算法，同时仅取**A**的下三角部分，**L**的结果将其覆盖。

- 注意列方程的顺序
- 计算出变量的顺序

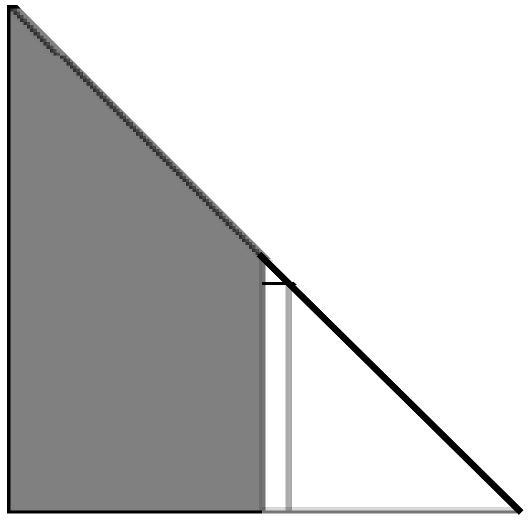
$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 &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

```
for $j$ =1 to n
    $a_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} a_{jk}^2}$
    for $i = j+1$ to $n$
        $a_{ij} = (a_{ij} - \sum_{k=1}^{j-1} a_{ik} a_{jk}) / a_{jj}$
    end
end
```

若 $A = LL^T$，则

$$\boxed{a_{jj} = l_{j1}^2 + \cdots + l_{jj}^2}$$

矩阵元素计算过程的图示



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
- 仅使用矩阵***A***下三角部分的元素，因此上三角部分元素可不存储
- **Matlab**命令为**chol(A)**
- 算法的稳定性**(**考虑**LU**分解**)**

只利用**A**的一半信息;
缺省得到上三角矩阵**(R'*R=A)**;
用于检查对称矩阵的正定性.

$$A = L_0 U = L_0 D^{1/2} D^{1/2} L_0^T = LL^T \Longrightarrow \begin{array}{l} U = D^{1/2} L^T \\ U^T = L D^{1/2} \end{array}$$

增长因子 $$\rho = \frac{max\{|U^T(i,j)|\}}{max\{|a_{ij}|\}} = \frac{max\{|l_{ij} l_{jj}|\}}{max\{|a_{ij}|\}} \le \frac{max\{l_{ij}^2\}}{max\{a_{ii}\}} \le 1$$

算法稳定！

对称矩阵的LDL^{T}分解与对称不定矩阵

- 若不正定，**Cholesky**分解会发生中断
- 考虑**LU**分解的一个变种

$$A = LDL^T$$

$U = DL^T$, **D**为对角阵

因为**LU**分解唯一，此分解也唯一

- **L**非对角线元素可能很大(增长因子>>1)，算法不稳定，需考虑选主元

$$PAP^T = LDL^T$$

P为排列阵，选主元是做对称交换

- 仅仅对称选主元仍可能使增长因子>>1，算法不稳定
- 实际采用的为**Aasen**算法或**Bunch**和**Parlett**给出的算法

$$PAP^T = LDL^T$$

D为三对角矩阵，或含**1x1**或**2x2**子块的分块对角阵。 能保证**L**非对角元素不大于**1**，使算法稳定性

- 分解的计算量~ $n^3/3$ flops

Matlab有关命令的说明

- **lu命令**
  - 部分选主元**LU**分解，均能处理“稀疏”矩阵
  - Y=lu(A); 丢弃排列阵**P**的信息
  - [L, U]=lu(A); **A=LU, L**为经行排列的单位下三角阵
  - [L, U, P]=lu(A); **P**为矩阵，若加参数**’vector’**，则为向量
  - [L,U,P,Q] = lu(A), [L,U,P,Q,R] = lu(A), 针对稀疏矩阵的算法
- **chol命令**
  - R = chol(A); 仅读**A**上三角部分形成对称矩阵，再分解
  - L = chol(A,‘lower’); 仅读取**A**下三角部分
  - 得到的对称矩阵若不正定，则出错退出
  - [R,p] = chol(A); 若**A**正定，返回**p=0**；否则分解到第**p**行中断，则R^TR=A(1:q,1:q), q=p-1.
  - [R,p,S] = chol(A), 等等，针对稀疏矩阵

线性方程组求解的程序包/软件

- 一般(非稀疏)的系数矩阵
  - 部分主元LU分解，回/前代过程
  - 前缀s,d,c,z
  - 矩阵条件数估计
- 特殊类型系数矩阵
  - 更有效的存储方式

Source	Factor	Solve	Condition estimate
LINPACK	sgefa	sgesl	sgeco
LAPACK	sgetrf	sgetrs	sgecon
NR	ludcmp	lubksb	
MATLAB	lu	\	rcond/condest
…	**…**	**…**	**…**

Source	Symmetric positive define	Symmetric indefinite	General banded
LINPACK	spofa/sposl	ssifa/ssisl	sgbfa/sgbsl
LAPACK	spotrf/spotrs	ssytrf/ssytrs	sgbtrf/sgbtrs
NR	choldc/cholsl		bandec/banbsk
Matlab	chol	\	\

- LINPACK
  - Widely used / benchmark to evaluate computer performance
  - LAPACK is revision; both are available from ***Netlib***

- Basic Linear Algebra Subprograms (BLAS)
  - LINPACK and LAPACK are based on low-level BLAS
  - 封装 BLAS encapsulate basic operations on vectors & matrices so they can be optimized for given computer architecture
  - Generic Fortran versions available from *Netlib*; optimized versions provided by computer vendors
  - Key to good performance is data reuse. Level-3 BLAS have more opportunity for data reuse, and hence higher performance.

级别	**TOMS #**	运算量	例程序	功能
1	**539**	$\mathcal{O}(n)$	**saxpy** **sdot** **snrm2**	**Scalar x vector + vector** **Inner product** **Euclidean vector norm**
2	**656**	$\mathcal{O}(n^2)$	**sgemv** **strsv** **sger**	**Matrix-vector product** **Triangular solution** **Rank-one update**
3	**679**	$\mathcal{O}(n^3)$	**sgemm** **strsm** **ssyrk**	**Matrix-matrix product** **Multiple triang. solutions** **Rank-k update**

Algori. No. 679 in *ACM Trans. on Math. Soft.*

Assignment

- 阅读课本第二章有关内容
- 阅读文献: **(“教学资源”--” top 10 algorithm”)**
- G. W. Stewart, “The decompositional approach to matrix computation,” Computing in Science and Engineering, Vol. 2, No. 1, pp. 50-59, 2000
- 作业题见网络学堂