II Efficient direct methods for the solution of $AX=B$

① Operation count

A rough estimate of the efficiency & speed of execution of an algorithm can be obtained by counting the # of FP operations performed by the algorithm.

Examples • Matrix multiplication $C = AB$ (A, B, C are $N \times N$ matrices)

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj}$$

→ for each coefficient c_{ij} we need N additions & N multiplications

There are N^2 c_{ij} coeffs so the total FP op count is $O(N^3)$

• Back-substitution (from $\left(\nabla\right)\left(x\right) = \left(B\right)$)

Mult/Div	Add/Subs	
1	0	$x_N = \dfrac{b_N}{a_{NN}}$
2	1	$x_{N-1} = \dfrac{b_{N-1} - a_{N-1,N}\, x_N}{a_{N-1,N-1}}$
3	2	$x_{N-2} = \dfrac{b_{N-2} - a_{N-2,N-1}\, x_{N-1} - a_{N-2,N}\, x_N}{a_{N-2,N-2}}$
⋮		⋮
N	$N-1$	$x_1 = \ldots$
↓ a total of $\dfrac{N(N+1)}{2}$	↓ a total of $\dfrac{N(N-1)}{2}$	⇒ a grand total of $O(N^2)$ ops. (though * and / are more costly than + and −).

see numerical recipes:

- • GJ elimination $O(N^3 + N^2 M)$ (M = # of RHS vectors)
- • GE with backsubs. : $O\left(\dfrac{N^3}{3} + \dfrac{1}{2}N^2 M + \dfrac{1}{2}N^2 M\right)$

Note that if $N = M$ (cf for calculating the inverse) then the op. count is $O(N^3)$.

(2) LU decomposition

- The disadvantage of all the above methods is that the RHS must be known in advance.

- If it isn't, then one could calculate A^{-1} and store A^{-1} (an $O(N^3)$) operation, and then for any new RHS we multiply $A^{-1}B$ to get the answer (an $O(N^2M)$ op count). where M is the # of RHS vectors.

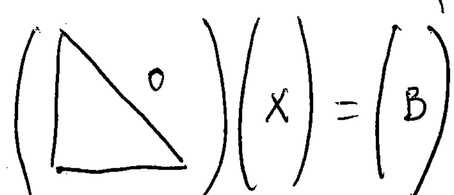
But there is a better way of doing this!

- Suppose that instead we can write $A = LU$ where L is lower triangular and U is upper triangular, in a # of operations (much) smaller than N^3,

- then to solve $AX = B$ we need to solve

$$LUX = B \iff \begin{cases} LY = B \\ UX = Y \end{cases}$$

⊕ $LY = B$ is called a "forward substitution" problem



which is essentially similar to back-substitution with a lower-triangular matrix.

⊕ $UX = Y$ is a standard back-substitution

So both have op counts $O(N^2M)$.

$\Rightarrow$ Case 1: GJ to get A^{-1}, then $A^{-1}B$: (B is $N \times M$)

$$O(N^3) + O(N^2M).$$

Case 2 : LU

$$\left(< O(N^3) \right) + O(2N^2 M)$$

$\Rightarrow$ the second option (if we can indeed make the first step $< O(N^3)$) will be more efficient, and more versatile.

- How to perform an LU decomposition?

$$LU = A \iff a_{ij} = \sum l_{ik} u_{kj} \quad \forall i,j$$

with
$$L = \begin{pmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ l_{N1} & l_{N2} & & & l_{NN} \end{pmatrix}$$

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1N} \\ 0 & u_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & u_{NN} \end{pmatrix}$$

$\rightarrow$ there are a total of $\frac{N(N+1)}{2} + \frac{N(N+1)}{2}$ unknown coefficients $\{l_{ij}\}$ and $\{u_{ij}\}$, and a total of N^2 equations for these coefficients. This is an under-determined system so we can arbitrarily choose N of the unknowns to begin with.

Crout's Algorithm has the following property

- sets $l_{ii} = 1 \quad \forall i$
- solves for the other unknowns
- stores the L and U matrices into the original A matrix as

$$\begin{pmatrix} u_{11} & \cdots & & u_{1N} \\ l_{21} & \ddots & & \\ \vdots & & \ddots & \\ l_{N1} & \cdots & l_{N,N-1} & u_{NN} \end{pmatrix}$$

Idea behind the algorithm

in the equation $$a_{ij} = \sum_{k=1}^{N} l_{ik} u_{kj}$$ $l_{ik} = 0$ if $k > i$, $u_{kj} = 0$ if $k > j$

so $$= \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj}$$

Step 1 so suppose $i = 1$

$$a_{1j} = \sum_{k=1}^{1} l_{1k} u_{kj} = l_{11} u_{1j} = u_{1j}$$ (since we assume $l_{11} = 1$)

$$\Rightarrow \quad u_{1j} = a_{1j}$$

then suppose $j = 1$

$$a_{i1} = \sum_{k=1}^{1} l_{ik} u_{k1} = l_{i1} u_{11}$$

$$\Rightarrow \quad l_{i1} = \frac{a_{i1}}{u_{11}}$$ which we can calculate since we know u_{11}

Step 2: now suppose $i = 2$

$$a_{2j} = \sum_{k=1}^{2} l_{2k} u_{kj} = l_{21} u_{1j} + \overset{1}{\cancel{l_{22}}} u_{2j}$$

$$\Rightarrow \quad u_{2j} = a_{2j} - l_{21} u_{1j}$$

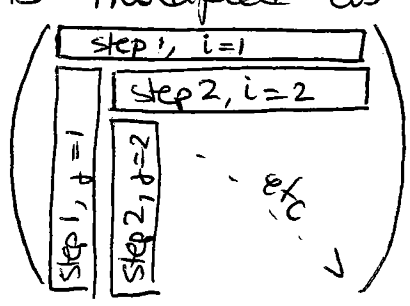
and $j = 2$

$$a_{i2} = \sum_{k=1}^{2} l_{ik} u_{k2} = l_{i1} u_{12} + l_{i2} u_{22}$$

$$\Rightarrow \quad l_{i2} = \frac{a_{i2} - l_{i1} u_{12}}{u_{22}}$$

so it is possible to construct the LU decomposition directly into the A matrix following these steps, so that A is modified as

A :



⇒ The unpivoted Crout algorithm is simply written as

For $k = 1, N$ ← sweep through step #

$$\left\lfloor\ \begin{array}{l} \text{For } j = k, N \\ \quad \lfloor\ u_{kj} = a_{kj} - \sum_{m=1}^{k-1} l_{km} u_{mj} \qquad \leftarrow \text{assume } i=k \\ \text{For } i = k+1, N \qquad \leftarrow \text{assume } j=k \\ \quad \lfloor\ l_{ik} = \frac{1}{u_{kk}} \left[a_{ik} - \sum_{m=1}^{k-1} l_{im} u_{mk} \right] \end{array} \right.$$

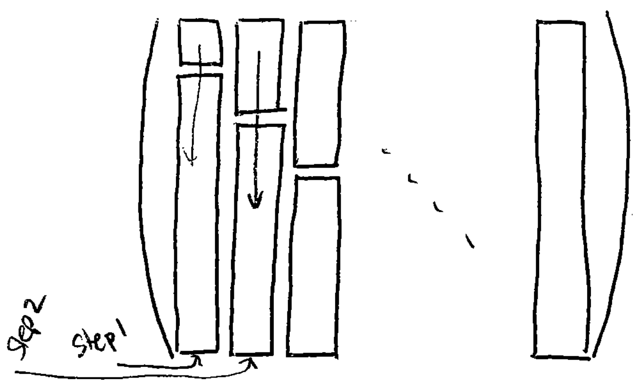
<u>Note</u>: in practise, u_{ij} and l_{ij} are directly stored into a_{ij}

<u>Pivoting for LU</u>

- As in the case of standard GJ elimination, pivoting is essential to the stability of the algorithm.

- However, since LU is done independently of the RHS, the permutations of the lines in the A matrix must be <u>recorded</u> for later use.

- Moreover, the ordering of the operations in Crout's algorithm does <u>not</u> facilitate pivoting. Another algorithm, which simply re-orders the operations, does allow pivoting:



⇒ since the calculation of l_{ij} only requires the knowledge of u_{kj} with $k<j$ we postpone the calculat of the other u_{kj} until later and change the order of ops

For $j = 1, N$ ← sweep over columns

$$\left\lfloor\ \begin{array}{l} \text{For } i = 1, j \\ \quad \lfloor\ u_{ij} = a_{ij} - \sum_{m=1}^{i-1} l_{im} u_{mj} \\ \text{For } i = j+1, N \\ \quad \lfloor\ l_{ij} = \frac{1}{u_{jj}} \left[a_{ij} - \sum_{m=1}^{j-1} l_{im} u_{mj} \right] \end{array} \right.$$

→ Now it is possible to pivot the algorithm

Idea: The pivot is selected so that the denominator u_{jj} is the largest possible. (for the same reasons as before). when calculating the l_{ij} coefficient

LU algorithm with pivoting

① For $i = 1, N$

$$\left\lfloor scale(i) = \max |a(i,j)|_{j=1,N} \right.$$

② For $j = 1, N$

- For $i = 1, j-1$

$$\left\lfloor u_{ij} = a_{ij} - \sum_{m=1}^{i-1} l_{im} u_{mj} \right.$$

Note: this calculates all of the u_{ij} in column j except the diagonal element u_{jj} which we must select by pivoting

- For $i = j, N$

$$\left\lfloor l_{ij} = a_{ij} - \sum_{m=1}^{j-1} l_{im} u_{mj} \right.$$

Note: this doesn't calculate l_{ij} just yet, but only the part in the [ ]. The largest of these will be the pivot u_{jj} (Note that it has the right formula for being u_{jj}).

- $$p = \max \left| \frac{l_{ij}}{scale(i)} \right|_{i=j,N}$$ ← select pivot

- switch line j with line containing the pivot (p becomes u_{jj}). RECORD the SWITCH for RHS.

- For $i = j+1, N$

$$\left\lfloor l_{ij} = \frac{l_{ij}}{u_{jj}} \right.$$ ← divide by u_{jj} to complete the calculation of l_{ij}

Note: in practise, l_{ij} and u_{ij} are directly stored into a_{ij}

Advantages & application of LU decomposition

- the operation count for LU is $O(\frac{N^3}{3})$, so factor of 3 faster than GJ elimination.

- A^{-1} is never calculated. if A^{-1} is needed, then we need to perform a backsubstitution on I.

- if det(A) is required

$$\det(A) = \det(L)\det(U) = 1 \cdot \det(U)$$
$$= \prod_i u_{ii}$$

→ LU decomposition provides a very fast way of calculating the determinant of a matrix. (note that the sign of the determinant must be updated according to the permutations if pivoting was used).

(3) Choleski decomposition for real symmetric positive definite matrices

When the matrix A is real and symmetric AND positive definite then LU decomposition can be accelerated using Cholesky factorization.

Definitions.
- A symmetric matrix satisfies $A^T = A$ $(a_{ij} = a_{ji})$
- A positive definite matrix satisfies

$$\forall \underline{v}, \quad v^T A v > 0$$

These properties are often satisfied by matrices arising from physical systems (see later).

LU decomposition of a symmetric matrix implies

$$A^T = A$$

$$\Rightarrow (LU)^T = LU$$

$$\Rightarrow U^T L^T = LU$$

But the transpose of an upper triangular matrix is a lower triangular one, so symmetry of A implies that we can choose U as

$$U^T = L \qquad (L^T = U)$$

$\Rightarrow$ The original system of equations

$$\sum_{k}^{\min(i,j)} l_{ik} u_{kj} = a_{ij}$$

can now be simplified to

$$\sum_{k}^{\min(i,j)} l_{ik} l_{jk} = a_{ij}$$

$\Rightarrow$ now we have $\frac{N(N+1)}{2}$ equations (only $\frac{N(N+1)}{2}$ of the N^2 coefficients of A are independent since A is symmetric) for $\frac{N(N+1)}{2}$ variables (L has $\frac{N(N+1)}{2}$ coeffs).

$\Rightarrow$ Writing out the equations we get

$i=1$: $\quad l_{11}^2 = a_{11} \qquad (j=1)$

$i=2$: $\quad l_{21} l_{11} = a_{21} \qquad (j=1)$

$\quad l_{21} l_{21} + l_{22} l_{22} = a_{22} \qquad (j=2=i)$

$i=3$: $\quad l_{31} l_{11} = a_{31} \qquad (j=1)$

$\quad l_{31} l_{21} + l_{32} l_{22} = a_{32} \qquad (j=2)$

$\quad l_{31} l_{31} + l_{32} l_{32} + l_{33} l_{33} \qquad (j=3=i)$

and so forth

so we can always calculate the l_{ij} coefficients using the following algorithm:

do $j = 1, N$

- $$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$$ (can be stored in a_{jj})

- do $i = j+1, N$

  $$l_{ij} = \frac{1}{l_{jj}} \left[a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right]$$ (can be stored in a_{ij})

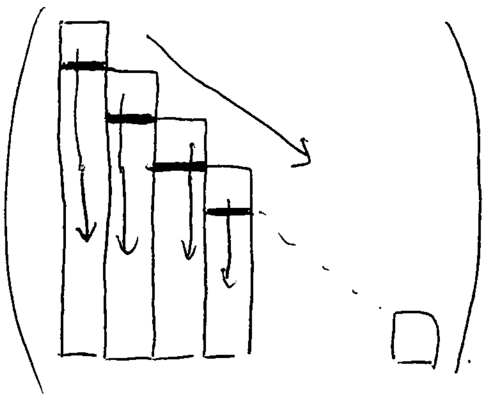
⇒ this solves, in order,

$(j=1)$
$$\begin{cases} l_{11} = \sqrt{a_{11}} \\ l_{21} = \frac{1}{l_{11}} [a_{21}] \\ l_{31} = \frac{1}{l_{11}} [a_{31}] \\ \vdots \\ l_{N1} = \frac{1}{l_{11}} [a_{N1}] \end{cases}$$

$(j=2)$
$$\begin{cases} l_{22} = \sqrt{a_{22} - l_{21}^2} \\ l_{32} = \frac{1}{l_{22}} [a_{32} - l_{31} l_{21}] \\ l_{42} = \frac{1}{l_{22}} [a_{42} - l_{41} l_{21}] \\ l_{N2} = \frac{1}{l_{22}} [a_{N2} - l_{N1} l_{21}] \end{cases}$$

$\vdots$ etc

⇒ treats the matrix in the following order.



Note : • Because the matrix is positive definite, det A > 0.
→ the diagonal elements of L are also positive definite

• In fact, if a diagonal element is v. small, it implies det A small ⇒ the system is near singular, in which case the method above is not appropriate.

⇒ Unless A is near singular, pivoting is not necessary; if A is near singular, the Cholesky method will fail.