

Solving a Linear System of Equations $Ax = b$

Reading: Cheney & Kincaid, Chapter 2, Section 8.1

Norms

The norm of a vector or matrix is a measure of its size.

- Typical vector norms:

Let $v = [v_1, v_2, \dots, v_n]^T$ be a real vector.

$$\|v\|_1 = \sum_{i=1}^n |v_i|, \quad \|v\|_\infty = \max_i |v_i|, \quad \|v\|_2 = \left(\sum_{i=1}^n v_i^2 \right)^{1/2}.$$

inner product
 $\langle \cdot, \cdot \rangle$

\downarrow
norm

2-norm
 $\| \cdot \|_2$ of a vector,
using $\|x\| = \sqrt{x \cdot x}$

- Typical matrix norms:

Induced norm

It is generated by a simpler object,
having a norm defined on it.

Let $A = (a_{ij})$ be an $m \times n$ real matrix.

The p -norm of vector induces p -norms on matrices through

1. The p -norm: $\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$, $p = 1, 2, \infty$. We can show

NOT EXAMINABLE

$$= \max_{x, \|x\|_p=1} \|Ax\|_p$$

$$\|A\|_1 = \max_j \sum_{i=1}^m |a_{ij}|, \quad \|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|, \quad \|A\|_2 = (\text{the largest eigenvalue of } A^T A)^{1/2}.$$

2. The Frobenius norm: $\|A\|_F = (\sum_{ij} a_{ij}^2)^{1/2}$.

\Rightarrow Easier to compute

not induced

i.e. It cannot be generated from a corresponding vector norm in this way

Gaussian Elimination with No Pivoting (GENP)

Problem: $Ax = b$, where A is a nonsingular $n \times n$ matrix.

GENP has two phases:

- Forward elimination: transform $Ax = b$ to an upper triangular system.
- Back substitution: solve the upper triangular system.

GENP Algorithm: Given A and b , solve $Ax = b$.

for $k = 1 : n - 1$

 for $i = k + 1 : n$

$$m_{ik} \leftarrow a_{ik}/a_{kk}$$

 for $j = k + 1 : n$

$$a_{ij} \leftarrow a_{ij} - m_{ik} * a_{kj}$$

 end

$$b_i \leftarrow b_i - m_{ik} * b_k$$

end

end

$$x_n \leftarrow b_n/a_{nn}$$

for $k = n - 1 : -1 : 1$

$$x_k \leftarrow (b_k - \sum_{j=k+1}^n a_{kj} * x_j)/a_{kk}$$

end

① The given algorithm operates directly on A , hence destroys the original matrix. Matlab parses A by value, so you still have the access to the original. However, if your method parses by reference and you still wish to preserve A , you need to make a copy of it.

② The algorithm only operates and modify the upper triangle of A , including the diagonal.

Example of Gaussian Elimination

$$\left(\begin{array}{ccc} 1 & 2 & 2 \\ 2 & 0 & 1 \\ -1 & 0 & 3 \end{array} \right) \xrightarrow{\substack{R_2 - 2R_1 \rightarrow R_2 \\ R_3 + R_1 \rightarrow R_3}} \left(\begin{array}{ccc} 1 & 2 & 2 \\ 0 & -4 & -3 \\ 0 & 2 & 5 \end{array} \right) \xrightarrow{R_3 + \frac{1}{2}R_2 \rightarrow R_3} \left(\begin{array}{ccc} 1 & 2 & 2 \\ 0 & -4 & -3 \\ 0 & 0 & 3.5 \end{array} \right)$$

Gaussian Elimination with no pivoting (row reduce)

$$\left(\begin{array}{ccc} 1 & 2 & 2 \\ 1 & 2 & 1 \\ -1 & 0 & 3 \end{array} \right) \xrightarrow{\substack{R_2 - R_1 \rightarrow R_2 \\ R_3 + R_1 \rightarrow R_3}} \left(\begin{array}{ccc} 1 & 2 & 2 \\ 0 & 0 & -1 \\ 0 & 2 & 5 \end{array} \right) \quad \text{GELP breaks down.}$$

↓ $R_2 \leftrightarrow R_3$

$$\left(\begin{array}{ccc} 1 & 2 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -1 \end{array} \right)$$

Gaussian Elimination with partial pivoting (row reduce + swap)

③ You can also store m_{ik} . They fit into a triangle matrix (excluding the diagonals).

If you rename them as $m_{ik} = l_{ik}$, they become the non-zero and non-diagonal values of L.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & \ddots & & a_{2n} \\ \vdots & & \ddots & a_{nn} \end{pmatrix} \xrightarrow{\text{GELP}} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ l_{21} & u_{22} & \cdots & \\ \vdots & & \ddots & \\ l_{m1} & & & u_{mn} \end{pmatrix}$$

part of U.
part of L.

$$\Rightarrow U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & \ddots & & \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & u_{mn} \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & \\ \vdots & & \ddots & \\ l_{m1} & l_{m2} & \cdots & 1 \end{pmatrix}$$

implicit in the above storage.

[Verify] $\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 2 \\ 0 & -4 & -3 \\ 0 & 2 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 0 & 1 \\ -1 & 0 & 3 \end{pmatrix}$

This is because each row reduction step can be represented as a multiplication.

[Elementary Matrices]

① Multiplication of a row by a scalar

$$\begin{pmatrix} 1 & & \\ & \alpha & \\ & & 1 \end{pmatrix} \quad \alpha \neq 0 \quad \alpha R_i \rightarrow R_i$$

$$\text{inverse} = \begin{pmatrix} 1 & & \\ & \frac{1}{\alpha} & \\ & & 1 \end{pmatrix}$$

② Swapping (permuting) 2 rows

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

$$R_i \leftrightarrow R_j$$

$$\text{inverse} = \begin{pmatrix} 1 & & \\ j & & \\ i & & 1 \end{pmatrix}$$

③ Adding a (non-zero) multiple of one row to another

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

$$R_i + \alpha R_j \rightarrow R_i$$

$$\text{inverse} = \begin{pmatrix} 1 & & \\ & 1 & \\ & -\alpha & \\ & & 1 \end{pmatrix}$$

When we perform GEPP, we only use row operations of type ③

For GEPP, we use row operations of type ② and ③

Operation ① is practically not used, other than polishing results to a particular form. e.g. LDU decomposition, where L, U are unitary and D is a diagonal, which is a variant of LU decomposition.

Back to the original example

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 0 & 1 \\ -1 & 0 & 3 \end{pmatrix} \xrightarrow{R_2 - 2R_1 \rightarrow R_2} \begin{pmatrix} 1 & 2 & 2 \\ 0 & -4 & -3 \\ -1 & 0 & 3 \end{pmatrix} \xrightarrow{R_3 + R_1 \rightarrow R_3} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -4 & -3 \\ 0 & 2 & 5 \end{pmatrix}$$

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

$$\xrightarrow{R_3 + \frac{1}{2}R_2 \rightarrow R_3} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -4 & -3 \\ 0 & 0 & 3.5 \end{pmatrix} = U$$

$$L_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & 1 \end{pmatrix}$$

Then, $L_3 L_2 L_1 A = U$

$$A = \underbrace{L_1^{-1} L_2^{-1} L_3^{-1}}_L U$$

$$\begin{aligned} \text{Here, } L_1^{-1} L_2^{-1} L_3^{-1} &= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -\frac{1}{2} & 1 \end{pmatrix} \end{aligned}$$

You can take a short cut by using L_1 & L_2 together

$$L_2 L_1 = \tilde{L}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$L_3 = \tilde{L}_2$$

which aligns the # steps with the original G.E.N.P.

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -\frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Forward substitution
(start from y_1)

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & -4 & 6 \\ 0 & 0 & 3.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = y = \begin{pmatrix} 1 \\ 0 \\ 4 \end{pmatrix}$$

Backward substitution
(start from x_3)

The quantities a_{kk} are referred to as the pivot elements, and m_{ik} are referred to as the multipliers.

Cost of GENP:

1 flop = 1 elementary operation: $+$, $-$, $*$, or $/$.

$$\sum_{k=1}^{n-1} (1 + 2(n-k) + 2)(n-k) + 1 + \sum_{k=1}^{n-1} (1 + (n-k) + (n-k-1) + 1) \approx \frac{2}{3}n^3.$$

Here we have ignored the lower order terms.

MATLAB file genp.m for solving $Ax = b$

```
function x = genp(A,b)
% genp.m Gaussian elimination with no pivoting
%
% input: A is an n x n nonsingular matrix
%         b is an n x 1 vector
% output: x is the solution of Ax=b.
%
n = length(b);
for k = 1:n-1
    for i = k+1:n
        mult = A(i,k)/A(k,k);
        A(i,k+1:n) = A(i,k+1:n)-mult*A(k,k+1:n);
        b(i) = b(i) - mult*b(k);
    end
end
x = zeros(n,1);
x(n) = b(n)/A(n,n);
for k = n-1:-1:1
    x(k) = (b(k) - A(k,k+1:n)*x(k+1:n))/A(k,k);
end
```

Note: To make the code run fast, the above code uses two for-loops instead of three in the forward elimination stage. Actually the second for-loop can be eliminated too (the modified code will be presented in class).

LU Factorization

It can be shown that GENP actually produces the so called LU factorization:

$$A = LU$$

where $L = (l_{ik})$ is an $n \times n$ unit lower triangular matrix and U is an $n \times n$ upper triangular matrix:

$$l_{ik} = m_{ik} \text{ for } n \geq i > k \geq 1, \quad l_{kk} = 1 \text{ for } 1 \leq k \leq n, \quad l_{ik} = 0 \text{ for } 1 \leq i < k \leq n,$$

$$u_{ij} = a_{ij} \text{ for } 1 \leq i \leq j \leq n, \quad u_{ij} = 0 \text{ for } n \geq i > j \geq 1.$$

Although the running time and storage are the same, LU can be more advantageous to have. For example when you need to solve the system $Ax=b$ with the same A but multiple RHS, the original GENT has $O(n^3)$ running time but LU $x=b$ has $O(n^2)$.

Here a_{ij} is the final a_{ij} obtained by GENT, not the original given a_{ij} . For details, see Chap 8 of Cheney & Kincaid. Once the LU factorization is available, to solve $\underline{Ax = b}$, we can solve two triangular systems

$$\begin{array}{l} \text{forward substitution} \\ \rightarrow Ly = b, \quad Ux = y \end{array} \quad \begin{array}{l} \text{backward} \\ \text{substitution} \end{array} \quad \begin{array}{l} \text{LU}x = b \\ y \end{array}$$

to obtain the solution x . The LU factorization can be used for other purposes, e.g., computing A^{-1} . The MATLAB program for the LU factorization will be presented in class.

Gaussian Elimination with Partial Pivoting (GEPP)

The difficulties with GENT:

In the k -th step of forward elimination,

- if $a_{kk} = 0$, GENT will break down.
- if a_{kk} is (relatively) small, i.e., some multipliers (in magnitude) $\gg 1$, then GENT will usually give unnecessary poor results.

Some examples will be given in class to illustrate the difficulties.

In order to overcome the difficulties, in the k -th step of forward elimination, we choose the largest element in magnitude from $a_{kk}, a_{k+1,k}, \dots, a_{nk}$ as a pivot element:

$$|a_{qk}| = \max\{|a_{kk}|, |a_{k+1,k}|, \dots, |a_{nk}|\} \quad (\text{say})$$

then interchange row k and row q of A , and interchange b_k and b_q as well. This process is called **partial pivoting**. The resulting algorithm is called GEPP.

GEPP Algorithm: Given A and b , solve $Ax = b$.

for $k = 1 : n - 1$

determine q such that

$$|a_{qk}| = \max\{|a_{kk}|, |a_{k+1,k}|, \dots, |a_{nk}|\}$$

for $j = k : n$

do interchange: $a_{kj} \leftrightarrow a_{qj}$

end

do interchange: $b_k \leftrightarrow b_q$

for $i = k + 1 : n$

$$m_{ik} \leftarrow a_{ik}/a_{kk}$$

for $j = k + 1 : n$

$$a_{ij} \leftarrow a_{ij} - m_{ik} * a_{kj}$$

end

$$b_i \leftarrow b_i - m_{ik} * b_k$$

end

end

$$x_n \leftarrow b_n/a_{nn}$$

for $k = n - 1 : -1 : 1$

$$x_k \leftarrow (b_k - \sum_{j=k+1}^n a_{kj} * x_j)/a_{kk}$$

end

Cost: $\frac{2}{3}n^3$ flops + $\frac{1}{2}n^2$ comparisons.

MATLAB file gepp.m for solving $Ax = b$

```
function x = gepp(A,b)
% genp.m GE with partial pivoting
% input: A is an n x n nonsingular matrix
%         b is an n x 1 vector
% output: x is the solution of Ax=b.
n = length(b);
for k = 1:n-1
    [maxval, maxindex] = max(abs(A(k:n,k)));
    q = maxindex+k-1;
    if maxval == 0, error('A is singular'), end
    A([k,q],k:n) = A([q,k],k:n);
    b([k,q]) = b([q,k]);
    i = k+1:n;
    A(i,k) = A(i,k)/A(k,k);
    A(i,i) = A(i,i) - A(i,k)*A(k,i);
    b(i) = b(i) - A(i,k)*b(k);
end
x = zeros(n,1);
x(n) = b(n)/A(n,n);
for k = n-1:-1:1
    x(k) = (b(k) - A(k,k+1:n)*x(k+1:n))/A(k,k);
end
```

MATLAB built-in command for solving $Ax = b$: $x = A \setminus b$. It uses GEPP.

LU Factorization with Partial Pivoting

It can be shown that GEPP actually produces the so called LU factorization with partial pivoting:

$$PA = LU$$

where $P = P_{n-1} \dots P_1 P_1$ with P_k being the permutation matrix used in the k -th step of the forward elimination process (if no row permutation occurs in the k -th step, $P_k = I$), L is an $n \times n$ unit lower triangular matrix, whose elements are the permuted multipliers, and U is an $n \times n$ upper triangular matrix obtained at the end of the forward elimination process, cf. Section 8.1 of Cheney & Kincaid. Once this factorization is available, to solve $Ax = b$, we can solve two triangular systems

$$Ly = Pb, \quad Ux = y$$

to obtain the solution x . The MATLAB program for computing the LU factorization with partial pivoting can easily be obtained by modifying `gepp.m`.

$$A = \begin{pmatrix} 1 & -6.5 & 0 \\ 2 & 1 & 2 \\ 4 & -2 & 6 \end{pmatrix}$$

\downarrow

$R_1 \leftrightarrow R_3$

$$\begin{pmatrix} 4 & -2 & 6 \\ 2 & 1 & 2 \\ 1 & -6.5 & 0 \end{pmatrix}$$

\downarrow

$R_2 - \frac{1}{2}R_1 \rightarrow R_2$
 $R_3 - \frac{1}{4}R_1 \rightarrow R_3$

$$\begin{pmatrix} 4 & -2 & 6 \\ 0 & 2 & -1 \\ 0 & -6 & -1.5 \end{pmatrix}$$

\downarrow

$R_2 \leftrightarrow R_3$

$$\begin{pmatrix} 4 & -2 & 6 \\ 0 & -6 & -1.5 \\ 0 & 2 & -1 \end{pmatrix}$$

\downarrow

$R_3 + \frac{1}{3}R_2 \rightarrow R_3$

$$\begin{pmatrix} 4 & -2 & 6 \\ 0 & -6 & -1.5 \\ 0 & 0 & -0.5 \end{pmatrix} = U$$

$$P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & 0 & 1 \end{pmatrix}$$

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

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{3} & 0 \end{pmatrix}$$

$$L_2 P_2 L_1 P_1 A = U$$

We'd like to have all the L_i 's next to each other in order to construct the lower triangular matrix, which could be easily used to solve equations through forward substitution.

We'd like to unmix the P_i 's and L_i 's

$$L_i P_i = P_i \tilde{L}_i \text{ for some } \tilde{L}_i \text{ having the same non-zero structures as } L_i.$$

$$\begin{aligned}
 P_2^{-1} L_1 P_2 &= P_2 L_1 P_2 = \text{swap} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 \\ -\frac{1}{4} & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix} \\
 &= \tilde{L}_2
 \end{aligned}$$

$$\begin{aligned}
 P_2^{-1} L_1 P_2 &= P_2 (I + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \\ -\frac{1}{2} & 0 & 0 \end{pmatrix}) P_2 \\
 &= P_2 I P_2 + P_2 \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \\ -\frac{1}{2} & 0 & 0 \end{pmatrix} P_2 \\
 &= I + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \\ -\frac{1}{2} & 0 & 0 \end{pmatrix} P_2 \\
 &= I + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \\ -\frac{1}{2} & 0 & 0 \end{pmatrix}
 \end{aligned}$$

$$\begin{aligned}
 L_2 P_2 L_1 P_1 A &= U \\
 \Rightarrow L_2 \tilde{L}_1 P_2 P_1 A &= U
 \end{aligned}$$

Let $\tilde{L}_2 = L_2$ (just for consistency of notation)

$$\Rightarrow \underbrace{P_2 P_1 A}_{P} = \underbrace{\tilde{L}_1^{-1} \tilde{L}_2^{-1}}_{L} U$$

$$\begin{aligned}
 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} A &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{pmatrix} U \\
 \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} A &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix} U
 \end{aligned}$$

[How to deal with longer sequence]

$$\begin{aligned}
 L_{n-1} P_{n-1} \cdots L_2 P_2 L_2 P_2 L_1 P_1 A &= U \\
 \Rightarrow \tilde{L}_{n-1} \tilde{L}_{n-2} \cdots \tilde{L}_2 \tilde{L}_1 P_{n-1} P_{n-2} \cdots \tilde{P}_2 \tilde{P}_1 \tilde{P}_1 A &= U
 \end{aligned}$$

where

$$\tilde{L}_{n-1} = L_n$$

$$Z_{n-2} = P_{n-1} L_{n-1} P_{n-1}$$

\vdots

$$Z_1 = P_{n-1} P_{n-2} \cdots \underbrace{P_2 L_1 P_2 P_3 \cdots P_{n-1}}_{\text{induction.}}$$

$L_i^{(n)}$ same structure as L_1 .
same structure as L_1 .

[Example]

$$\text{take } \tilde{L}_3 = L_3$$

$$L_3 P_3 \tilde{L}_2 P_2 L_1 P_1 A = U$$

$$\text{take } \tilde{L}_2 = P_3 \tilde{L}_2 P_3$$

$$\tilde{L}_3 \tilde{L}_2 P_3 P_2 L_1 P_1 A = U$$

$$\text{take } \tilde{L}_1^{(1)} = P_2 L_1 P_2$$

$$L_3 \tilde{L}_2 P_3 \tilde{L}_1^{(1)} P_2 P_1 A = U$$

$$\text{take } \tilde{L}_1 = \tilde{L}_1^{(1)} = P_2 \tilde{L}_1^{(1)} P_2$$

$$\tilde{L}_3 \tilde{L}_2 \tilde{L}_1 P_3 P_2 P_1 A = U$$

High-level GEPP & LU with PP are better than GENP (or LU without pivoting)

- ① Some matrices cannot be solved by GENP.
- ② Small pivots occurred as a result of cancelation, using those pivots allows the propagate the increased uncertainty around them.

MATLAB file lupp.m for computing the LU factorization of A with partial pivoting

```

function [L,U,P] = lupp(A)
% lupp.m LU factorization with partial pivoting
% input: A is an n x n nonsingular matrix
% output: Unit lower triangular L, upper triangular U,
%          permutation matrix P such that PA = LU
n = size(A,1);
P = eye(n);
for k = 1:n-1,
    [maxval, maxindex] = max(abs(A(k:n,k)));
    q = maxindex + k - 1;
    if maxval == 0, error('A is singular'), end
    A([k,q], :) = A([q,k], :);
    P([k,q], :) = P([q,k], :);
    i = k+1:n
    A(i,k) = A(i,k)/A(k,k);
    A(i,i) = A(i,i) - A(i,k)*A(k,i);
end
L = tril(A,-1) + eye(n);
U = triu(A);

```

MATLAB built-in function for computing the factorization: $[L,U,P] = \text{lu}(A)$

Some Theoretical Results about GEPP

Residual vector: $r = b - Ax_c$, where x_c is the computed solution of $Ax = b$ by an algorithm. In the following, the norm $\|\cdot\|$ can be $\|\cdot\|_1$, $\|\cdot\|_2$, or $\|\cdot\|_\infty$.

- We can show that if we use GEPP, then the computed solution x_c satisfies

$$(A + E)x_c = b, \quad (1)$$

where **usually**

$$\|E\| \approx \epsilon \|A\|, \quad (2)$$

with ϵ being the machine epsilon. So x_c exactly solves a nearby problem. We say GEPP is usually **numerically stable**

- If (1) and (2) hold, we can show

$$\|r\| \lesssim \epsilon \|A\| \cdot \|x_c\|,$$

$$\frac{\|x_c - x\|}{\|x\|} \lesssim \epsilon \|A\| \cdot \|A^{-1}\|,$$

where $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ is called the condition number of $Ax = b$. It can be shown that $\kappa(A) \geq 1$.

Notes:

- The size of residual is usually relatively small compared with the product of the size of A and the size of x_c .
- Let $\epsilon \approx 10^{-t}$ and $\kappa(A) \approx 10^p$. Then usually x_c has approximately $t - p$ accurate decimal digits. If $\kappa(A)$ is large, we say the problem $Ax = b$ is **ill-conditioned**.

Conclusion: The accuracy of a computed solution of the linear system depends on (i) the stability of the algorithm (ii) the condition number of the problem.

Solving Tridiagonal Systems by GENP

Algorithm for solving

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

for $i = 2 : n$

```

mult ←  $a_{i-1}/d_{i-1}$ 
 $d_i \leftarrow d_i - mult * c_{i-1}$ 
 $b_i \leftarrow b_i - mult * b_{i-1}$ 

```

end

$x_n \leftarrow b_n/d_n$

for $i = n - 1 : -1 : 1$

```
 $x_i \leftarrow (b_i - c_i * x_{i+1})/d_i$ 
```

end

Cost: $8n$ flops.

Storage: store only a_i, c_i, d_i and b_i by using 4 1-dimensional arrays. Do not use a 2-dimensional array to store the whole matrix.

Diagonally Dominant Matrices

Def: Let $A = (a_{ij})_{n \times n}$. A is strictly diagonally dominant by column if

$$|a_{jj}| > \sum_{i=1, i \neq j}^n |a_{ij}|, \quad j = 1 : n.$$

A is strictly diagonally dominant by row if

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}|, \quad i = 1 : n.$$

We can show

- If a tridiagonal A is strictly diagonally dominant by column, then partial pivoting is not needed, i.e., GENP and GEPP will give the same results. (exercise)
- If a tridiagonal A is strictly diagonally dominant by row, then GENP will not fail (see C&K, pp. 282-283).