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

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

inner product $\langle \cdot, \cdot \rangle$ induces 2-norm $\|\cdot\|_2$ of a vector, using $\|x\| = \langle x, x \rangle^{\frac{1}{2}}$

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

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

  2. The Frobenius norm: $\|A\|_F = (\sum_{ij} a_{ij}^2)^{1/2}$. not induced

     $\Rightarrow$ Easier to compute

     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 ← a_ik / a_kk
        for j = k + 1 : n
            a_ij ← a_ij − m_ik * a_kj
        end
        b_i ← b_i − m_ik * b_k
    end
end
x_n ← b_n / a_nn
for k = n − 1 : −1 : 1
    x_k ← (b_k − Σ_{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 passes A by value, so you still have the access to the original. However, if your method passes 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

$$\begin{pmatrix} \boxed{1} & 2 & 2 \\ 2 & 0 & 1 \\ -1 & 0 & 3 \end{pmatrix} \xrightarrow[R_3 + R_1 \to R_3]{R_2 - 2R_1 \to R_2} \begin{pmatrix} 1 & 2 & 2 \\ 0 & \boxed{-4} & -3 \\ 0 & 2 & 5 \end{pmatrix} \xrightarrow{R_3 + \frac{1}{2}R_2 \to R_3} \begin{pmatrix} 1 & 2 & 2 \\ 0 & -4 & -3 \\ 0 & 0 & 3.5 \end{pmatrix}$$

Gaussian Elimination with no pivoting (row reduce)

$$\begin{pmatrix} \boxed{1} & 2 & 2 \\ 1 & 2 & 1 \\ -1 & 0 & 3 \end{pmatrix} \xrightarrow[R_3 + R_1 \to R_3]{R_2 - R_1 \to R_2} \begin{pmatrix} 1 & 2 & 2 \\ 0 & \boxed{0} & -1 \\ 0 & 2 & 5 \end{pmatrix}$$

GENP breaks down.

$$\downarrow \quad R_2 \leftrightarrow R_3$$

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -1 \end{pmatrix}$$

Gaussian Elimination with partial pivoting (row reduce + swap)

---

② 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.

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

part of U.

part of L.

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

implicit in the above storage.

[Verify] $$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -\frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 2 \\ 0 & -4 & -3 \\ 0 & 0 & 3.5 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 0 & 1 \\ -1 & 0 & 3 \end{pmatrix}$$

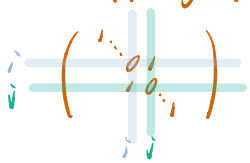
This is because each row reduction step can be represented as a multiplication.

[Elementary Matrices]

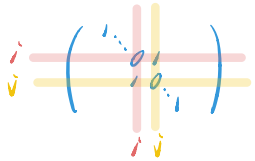
① Multiplication of a row by a scalar

$\begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & \alpha & \\ & & & \ddots \\ & & & & 1 \end{pmatrix}$ $\alpha \neq 0$ $\alpha R_i \to R_i$ inverse $= \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & \frac{1}{\alpha} & \\ & & & \ddots \\ & & & & 1 \end{pmatrix}$

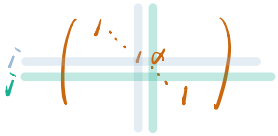
② Swapping (permuting) 2 rows



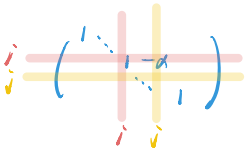
$R_i \leftrightarrow R_j$ inverse $=$



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



$R_i + \alpha R_j \to R_i$ inverse $=$



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

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

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

Back to the original example

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

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

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

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

Then, $L_3 L_2 L_1 A = U$

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

Here, $L_1^{-1} L_2^{-1} L_3^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{5} & 1 \end{pmatrix}$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{5} & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -\frac{1}{5} & 1 \end{pmatrix}$$

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

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

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

which aligns the # steps with the original GENP.

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -\frac{1}{5} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Forward substitution
(start from y_1)

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & -4 & 3 \\ 0 & 0 & 3.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = y = \begin{pmatrix} 1 \\ 0 \\ 4 \end{pmatrix}$$

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 GENP has $O(n^3)$ running time but $LUx=b$ has $O(n^2)$.

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

forward substitution → $Ly = b$, backward $Ux = y$ ($LUx = b$, y)

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 GENP:
In the k-th step of forward elimination,

- if $a_{kk} = 0$, GENP will break down.
- if a_{kk} is (relatively) small, i.e., some multipliers (in magnitude) $\gg 1$, then GENP 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 $a_{kk}, a_{k+1,k}, \ldots, a_{nk}$ as a pivot element:

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

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
    $|a_{qk}| = \max\{|a_{kk}|, |a_{k+1,k}|, \ldots, |a_{nk}|\}$
  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\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`.

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

$R_1 \leftrightarrow R_3$ $\quad P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$

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

$R_2 - \frac{1}{2} R_1 \to R_2$, $R_3 - \frac{1}{4} R_1 \to R_3$ $\quad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & 0 & 1 \end{pmatrix}$

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

$R_2 \leftrightarrow R_3$ $\quad P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

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

$R_3 + \frac{1}{3} R_2 \to R_3$ $\quad L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{3} & 0 \end{pmatrix}$

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

$L_2 P_2 L_1 P_1 A = U$

We'd like to have all the L_i 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

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

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

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

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

$$= \tilde{L}_2$$

$$P_2^{-1} L_1 P_2 = P_2 \left(I + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \\ -\frac{1}{5} & 0 & 0 \end{pmatrix} \right) P_2$$

$$= P_2 I P_2 + P_2 \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \\ -\frac{1}{5} & 0 & 0 \end{pmatrix} P_2$$

$$= I + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{5} & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \end{pmatrix} P_2$$

$$= I + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{1}{4} & 0 & 0 \\ -\frac{1}{5} & 0 & 0 \end{pmatrix}$$

$$L_2 P_2 L_1 P_1 A = U$$

$$\Rightarrow L_2 \tilde{L}_1 P_2 P_1 A = U$$

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

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

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} A = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{5} & 1 & 0 \\ \frac{1}{4} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{5} & 1 \end{pmatrix} U$$

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} A = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{5} & 1 & 0 \\ \frac{1}{4} & -\frac{1}{5} & 1 \end{pmatrix} U$$

[How to deal with longer sequence]

$$L_{n-1} P_{n-1} \cdots L_2 P_2 L_2 P_2 L_1 P_1 A = U$$

$$\Rightarrow \tilde{L}_{n-1} \tilde{L}_{n-2} \cdots \tilde{L}_2 \tilde{L}_2 \tilde{L}_1 P_{n-1} P_{n-2} \cdots \tilde{P}_2 \tilde{P}_2 \tilde{P}_1 A = U$$

where

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

$$\tilde{L}_{n-2} = P_{n-1} L_{n-1} P_{n-1}$$

$$\vdots$$

$$\tilde{L}_1 = P_{n-1} P_{n-2} \cdots \underbrace{P_2 L_1 P_2}_{L_1^{(1)}} P_2 \cdots P_{n-1}$$

induction.

L_1^{n-2} same structure as L_1.

$L_1^{(1)}$ same structure as L_1.

[Example]

take $\tilde{L}_3 = L_3$

$L_3 P_3 L_2 P_2 L_1 P_1 A = U$

take $\tilde{L}_2 = P_3 L_2 P_3$

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

take $L_1^{(1)} = P_2 L_1 P_2$

$L_3 \tilde{L}_2 P_3 L_1^{(1)} P_2 P_1 A = U$

take $\tilde{L}_1 = L_1^{(2)} = P_3 L_1^{(1)} P_3$

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

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 occured as a result of cancelation, using those pivot 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] = 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

$$(A + E)x_c = b, \tag{1}$$

  where **usually**

$$\|E\| \approx \epsilon\|A\|, \tag{2}$$

  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 $\kappa(A) \geq 1$.