Solving a Linear System of Equations $Ax = b$

Norms

Norm is a measure of size of a vector or matrix.

- Typical vector norms:

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

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

- Typical matrix norms:

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

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

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

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

Gaussian Elimination with No Pivoting (GENP)

Problem: $Ax = b$, where 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

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 $/$.

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

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).

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:

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

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, we can solve two triangular systems $Ly = b$ and $Ux = y$ to obtain the solution x. The MATLAB program for the LU factorization will be presented in class.

Gaussian Elimination with Partial Pivoting (GEPP)

Problem: $Ax = b$, where A: nonsingular $n \times n$ matrix.
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) $>> 1$, then GENP will usually give unnecessary poor results.

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
```

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

$$PA = LU$$

where P is a permutation matrix, L is an $n \times n$ unit lower triangular matrix, and U is an $n \times n$ upper triangular matrix, cf. Chap 8 of Cheney & Kincaid. Once this factorization is available, we can solve two triangular systems $Ly = Pb$ and $Ux = y$ to obtain the solution x. The MATLAB program for computing the LU factorization with partial pivoting can easily be obtained by modifying the above code.

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);
```

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 $\kappa(A) \geq 1$.

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 $i = 2 : n$
    $mult \leftarrow a_{i-1}/d_{i-1}$
    $d_i \leftarrow d_i - mult * c_{i-1}$
    $b_i \leftarrow b_i - mult * b_{i-1}$
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).