

CSE/Math 455
Lecture # 9

We are still looking at solving

$$Ax = b.$$

Our algorithm is Gaussian elimination with partial pivoting. It factors A into

$$A = PLU$$

P , permutation matrix
 L , lower triangular matrix
 U , upper triangular matrix

P^T represented as $\mathbf{p} = (p_1, p_2, \dots, p_n)^T$ integer permutation vector.

```
function [p, L, U]=goodfact(A)
n= length(A);
p = 1:n;
for k = 1:n - 1
    Find  $i_{max}$  such that
     $|a_{i_{max},k}| \max_{k \leq i \leq n} |a_{ik}|$ 
    % MATLAB statement
    % [z, imax]= max(abs(A(k:n,k))); imax = imax + k-1;
    if  $i_{max} > k$ 
        temp = p(k); p(k) = p(imax); p(imax) = temp;
        atemp = A(k,:); A(k,:) = A(imax,:); A(imax,:) = temp;
    end;
    ii = k + 1:n;
    % Compute multipliers
    A(ii, k) = A(ii, k)/A(k, k);
    % Update matrix
    jj = k + 1:n ;
    A(ii, jj) = A(ii, jj) - A(ii, k) * A(k, jj);
end;
U=triu(A); L = eye(n)+ tril(L,-1);
```

To solve the linear system (mathematically to the left, MATLAB to the right)

$$\begin{aligned} Ly &= P^T \mathbf{b}, & \mathbf{y} &= L \setminus \mathbf{b}(\mathbf{p}) \\ U\mathbf{x} &= \mathbf{y}., & \mathbf{x} &= U \setminus \mathbf{y} \end{aligned}$$

Gaussian elimination with partial pivoting “almost always” solves linear systems as accurately as can be expected in floating point arithmetic.

The following result was stated by Reid(1971) but was mostly proved by Wilkinson (1961).

Suppose that the LU decomposition of $P^T A$ exists where P is a row permutation. Then in floating point arithmetic with machine unit ε_M , the computed L and U satisfy

$$P^T(A + E) = LU$$

where

$$E = (e_{ij}), \quad |e_{ij}| \leq 3.01 \max_{\ell, k} |a_{\ell, j}^{(k)}| \varepsilon_M$$

and $a_{\ell, j}^{(k)}$ is the value of $a_{\ell, j}$ after k elimination steps.

In plain English, if the elements of the matrix do not grow a great deal, Gaussian elimination is stable.

Partial pivoting is a heuristic to prevent this growth. In theory, the upper bound is

$$\max_{\ell, k} |a_{\ell, j}^{(k)}| \leq 2^{n-1} \max_i |a_{ij}|.$$

The 2^{n-1} factor is almost never achieved. More realistic growth is about $O(n)$.

Here is a famous example where 2^{n-1} growth in Gaussian elimination occurs. It is due to Wilkinson (1965).

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix}.$$

Partial pivoting (breaking ties by accepting first maximum) will not change the row ordering. The LU factorization is

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 & 16 \end{pmatrix}.$$

I emphasize that this is a pathological example. An m-file to generate it is in www.cse.psu.edu/~barlow/cse455/wilk61.m. A $n = 75$, this example will yield L and U such that $A - LU$ is $O(1)$ in MATLAB.

Notice that the algorithm is called “partial” pivoting. Complete pivoting produces

$$A = PLUQ$$

where P , L , and U are as before and Q is another permutation matrix. Each step of complete pivoting produces (i_{max}, j_{max}) such that

$$|a_{i_{max}, j_{max}}| = \max_{k \leq i \leq n, k \leq j \leq n} |a_{ij}|.$$

Do both a row and column exchange to put $a_{i_{max}, j_{max}}$ in the (k, k) position. Complete pivoting is “safe.” There we have

$$\max_{(i,j,k)} |a_{ij}^{(k)}| \leq f(n-1) \max_{(i,j)} |a_{ij}|$$

where

$$f(n) = n^{1/2} (2 \cdot 3^{1/2} \dots n^{1/n-1})^{1/2} = O(n^{1/2(1+\ln n)})$$

I did not give this function in class and it is not that important that you know it, I just want to show you that it exists and grows “modestly.” The searching and data movement involved in complete pivoting makes it far more expensive than partial pivoting, thus it is seldom done.

In general, we expect that Gaussian elimination with partial pivoting will yield a solution $\hat{\mathbf{x}}$ to

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

$$|e_{ij}| \leq c_0(n)\varepsilon_M \max_{(\ell,k)} |a_{\ell,k}|, \quad |r_i| \leq c_0(n)\varepsilon_M \max_{(\ell)} |b_\ell|$$

and $c_0(n)$ is a “modestly” growing function.

That implies that

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

is small, about the size of machine precision. Thus the solution “fits” the original equation!

However, we need to know how close $\hat{\mathbf{x}}$ is to the solution of

$$A\mathbf{x} = \mathbf{b}.$$

For instance, consider the Hilbert matrices given by

$$H = (h_{ij}), \quad h_{ij} = 1/(i + j - 1).$$

The MATLAB command

`H=hilb(n)`

produces the Hilbert matrix of order n . Run the following code

```
x=ones(12,1); % A vector of 12 ones.
H=hilb(12); % The 12 x 12 Hilbert matrix
b=A*x;
[L,U]=lu(H); % Factor H
y = L\b ; % in this option (the usual one) MATLAB incorporates P into L
x1 = U\y ;
err=x1-x ; % Will be big
r=b-A*x1; % Will be small
```

Even though x_1 is far from x it comes close to satisfying the linear system. Why?

To understand this, we need to define a measure for the size of $\hat{\mathbf{x}} - \mathbf{x}$. We will need to know the size of a vector. We will also need to know the size of a matrix. That leads to matrix and vector norms. Next time!