Numerical Analysis Hilary Term 2023
Lecture 2: Gaussian Elimination and LU factorisation

In lecture 1 we treated Lagrange interpolation. A traditional, more straightforward approach (worse for computation!) would be to express the interpolating polynomial as $p_n(x) = \sum_{i=0}^{n} c_i x^i$ and find the coefficients c_i by a linear system of equations:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}.$$

(The matrix here is known as the *Vandermonde* matrix, and nonsingular iff $\{x_i\}$ are distinct.) This is a linear algebra problem, which is the subject we will discuss in the next lectures. We start with solving linear systems.

Setup: Given a square n by n matrix A and vector with n components b, find x such that

$$Ax = b.$$

Equivalently find $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ for which

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned} \tag{1}$$

Lower-triangular matrices: the matrix A is **lower triangular** if $a_{ij} = 0$ for all $1 \le i < j \le n$. The system (1) is easy to solve if A is lower triangular.

$$\begin{array}{lcl} a_{11}x_1 & = b_1 \implies x_1 = \dfrac{b_1}{a_{11}} & \Downarrow \\ a_{21}x_1 + a_{22}x_2 & = b_2 \implies x_2 = \dfrac{b_2 - a_{21}x_1}{a_{22}} & \Downarrow \\ \quad\vdots & & \Downarrow \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ii}x_i & = b_i \implies x_i = \dfrac{b_i - \displaystyle\sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}} & \Downarrow \\ \quad\vdots & & \Downarrow \end{array}$$

This works if, and only if, $a_{ii} \neq 0$ for each i. The procedure is known as **forward substitution**.

Computational work estimate: one floating-point operation (flop) is one scalar multiply/division/addition/subtraction as in $y = a * x$ where a, x and y are computer representations of real scalars.[1]

[1] This is an abstraction: e.g., some hardware can do $y = a * x + b$ in one FMA flop ("Fused Multiply and Add") but then needs several FMA flops for a single division. For a trip down this sort of rabbit hole, look up the "Fast inverse square root" as used in the source code of the video game "Quake III Arena".

Hence the work in forward substitution is 1 flop to compute x_1 plus 3 flops to compute x_2 plus ... plus $2i-1$ flops to compute x_i plus ... plus $2n-1$ flops to compute x_n, or in total

$$\sum_{i=1}^{n}(2i-1) = 2\left(\sum_{i=1}^{n} i\right) - n = 2\left(\tfrac{1}{2}n(n+1)\right) - n = n^2 + \text{lower order terms}$$

flops. We sometimes write this as $n^2 + O(n)$ flops or more crudely $O(n^2)$ flops.

Upper-triangular matrices: the matrix A is **upper triangular** if $a_{ij} = 0$ for all $1 \le j < i \le n$. Once again, the system (1) is easy to solve if A is upper triangular.

$$\begin{array}{rcll}
 & \vdots & & \Uparrow \\
a_{ii}x_i + \cdots + a_{in-1}x_{n-1} + a_{1n}x_n & = b_i & \Longrightarrow \quad x_i = \dfrac{b_i - \sum\limits_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} & \Uparrow \\
 & \vdots & & \Uparrow \\
a_{n-1n-1}x_{n-1} + a_{n-1n}x_n & = b_{n-1} & \Longrightarrow \quad x_{n-1} = \dfrac{b_{n-1} - a_{n-1n}x_n}{a_{n-1n-1}} & \Uparrow \\
a_{nn}x_n & = b_n & \Longrightarrow \quad x_n = \dfrac{b_n}{a_{nn}}. & \Uparrow
\end{array}$$

Again, this works if, and only if, $a_{ii} \neq 0$ for each i. The procedure is known as **backward** or **back substitution**. This also takes approximately n^2 flops.

For computation, we need a reliable, systematic technique for reducing $Ax = b$ to $Ux = c$ with the same solution x but with U (upper) triangular $\Longrightarrow$ Gauss elimination.

Example

$$\begin{bmatrix} 3 & -1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \end{bmatrix}.$$

Multiply first equation by 1/3 and subtract from the second $\Longrightarrow$

$$\begin{bmatrix} 3 & -1 \\ 0 & \frac{7}{3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 12 \\ 7 \end{bmatrix}.$$

Gauss(ian) Elimination (GE): this is most easily described in terms of overwriting the matrix $A = \{a_{ij}\}$ and vector b. At each stage, it is a systematic way of introducing zeros into the lower triangular part of A by subtracting multiples of previous equations (i.e., rows); such (elementary row) operations do not change the solution.

for columns $j = 1, 2, \ldots, n-1$

for rows $i = j+1, j+2, \ldots, n$

$$\begin{aligned} \text{row } i &\leftarrow \text{row } i - \frac{a_{ij}}{a_{jj}} * \text{row } j \\ b_i &\leftarrow b_i - \frac{a_{ij}}{a_{jj}} * b_j \end{aligned}$$

end

end

Example.

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 3 \\ 2 & -2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \\ 2 \end{bmatrix} : \text{ represent as } \left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 1 & 2 & 3 & 11 \\ 2 & -2 & -1 & 2 \end{array}\right]$$

$$\Longrightarrow \quad \begin{array}{l} \text{row } 2 \leftarrow \text{row } 2 - \frac{1}{3}\text{row } 1 \\ \text{row } 3 \leftarrow \text{row } 3 - \frac{2}{3}\text{row } 1 \end{array} \left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 0 & \frac{7}{3} & \frac{7}{3} & 7 \\ 0 & -\frac{4}{3} & -\frac{7}{3} & -6 \end{array}\right]$$

$$\Longrightarrow \quad \text{row } 3 \leftarrow \text{row } 3 + \tfrac{4}{7}\text{row } 2 \left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 0 & \frac{7}{3} & \frac{7}{3} & 7 \\ 0 & 0 & -1 & -2 \end{array}\right]$$

Back substitution:

$$\begin{aligned} x_3 &= 2 \\ x_2 &= \frac{7 - \frac{7}{3}(2)}{\frac{7}{3}} = 1 \\ x_1 &= \frac{12 - (-1)(1) - 2(2)}{3} = 3. \end{aligned}$$

Cost of Gaussian Elimination: note, row $i \leftarrow$ row $i - \dfrac{a_{ij}}{a_{jj}} *$ row j is

for columns $k = j+1, j+2, \ldots, n$

$$a_{ik} \leftarrow a_{ik} - \frac{a_{ij}}{a_{jj}} a_{jk}$$

end

This is approximately $2(n-j)$ flops as the **multiplier** a_{ij}/a_{jj} is calculated with just one flop; a_{jj} is called the **pivot**. Overall therefore, the cost of GE is approximately

$$\sum_{j=1}^{n-1} 2(n-j)^2 = 2\sum_{l=1}^{n-1} l^2 = 2\frac{n(n-1)(2n-1)}{6} = \frac{2}{3}n^3 + O(n^2)$$

flops. The calculations involving b are

$$\sum_{j=1}^{n-1} 2(n-j) = 2\sum_{l=1}^{n-1} l = 2\frac{n(n-1)}{2} = n^2 + O(n)$$

flops, just as for the triangular substitution.

LU factorization:

The basic operation of Gaussian Elimination, row $i \leftarrow$ row $i + \lambda *$row j, can be achieved by pre-multiplication by a special lower-triangular matrix

$$M(i,j,\lambda) = I + \begin{bmatrix} 0 & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} \\ \leftarrow i \\ \\ \end{matrix}$$
$$\qquad\qquad\qquad\quad \begin{matrix} \uparrow \\ j \end{matrix}$$

where I is the identity matrix.

Example: $n = 4$,

$$M(3,2,\lambda) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \lambda & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } M(3,2,\lambda)\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a \\ b \\ \lambda b + c \\ d \end{bmatrix},$$

i.e., $M(3,2,\lambda)A$ performs: row 3 of $A \leftarrow$ row 3 of $A + \lambda *$ row 2 of A and similarly $M(i,j,\lambda)A$ performs: row i of $A \leftarrow$ row i of $A + \lambda *$ row j of A.

So GE for e.g., $n = 3$ is

$$\underset{l_{32} = \frac{a_{32}}{a_{22}}}{M(3,2,-l_{32})} \cdot \underset{l_{31} = \frac{a_{31}}{a_{11}}}{M(3,1,-l_{31})} \cdot \underset{l_{21} = \frac{a_{21}}{a_{11}}}{M(2,1,-l_{21})} \cdot \underset{\text{(upper triangular)}}{A = U = (\ \urcorner\ )}.$$

The l_{ij} are called the **multipliers**.

Be careful: each multiplier l_{ij} uses the data a_{ij} and a_{ii} that *results from the transformations already applied*, not data from the original matrix. So l_{32} uses a_{32} and a_{22} that result from the previous transformations $M(2,1,-l_{21})$ and $M(3,1,-l_{31})$.

Lemma. If $i \neq j$, $(M(i,j,\lambda))^{-1} = M(i,j,-\lambda)$.

Proof. Exercise.

Outcome: for $n = 3$, $A = M(2,1,l_{21}) \cdot M(3,1,l_{31}) \cdot M(3,2,l_{32}) \cdot U$, where

$$M(2,1,l_{21}) \cdot M(3,1,l_{31}) \cdot M(3,2,l_{32}) = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} = \underset{\text{(lower triangular)}}{L = (\ \llcorner\ )}.$$

This is true for general n:

Theorem. For any dimension n, GE can be expressed as $A = LU$, where $U = (\ \urcorner\ )$ is upper triangular resulting from GE, and $L = (\ \llcorner\ )$ is unit lower triangular (lower triangular with ones on the diagonal) with l_{ij} = multiplier used to create the zero in the (i, j)th position.

Most implementations of GE therefore, rather than doing GE as above,

$$\begin{array}{rll} \text{factorize} & A = LU & (\approx \frac{1}{3}n^3 \text{ adds} + \approx \frac{1}{3}n^3 \text{ mults}) \\ \text{and then solve} & Ax = b & \\ \text{by solving} & Ly = b & \text{(forward substitution)} \\ \text{and then} & Ux = y & \text{(back substitution)} \end{array}$$

Note: this is much more efficient if we have many different right-hand sides b but the same A.

Pivoting: GE or LU can fail if the pivot $a_{ii} = 0$. For example, if

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

GE fails at the first step. However, we are free to reorder the equations (i.e., the rows) into any order we like. For example, the equations

$$\begin{array}{c} 0 \cdot x_1 + 1 \cdot x_2 = 1 \\ 1 \cdot x_1 + 0 \cdot x_2 = 2 \end{array} \quad \text{and} \quad \begin{array}{c} 1 \cdot x_1 + 0 \cdot x_2 = 2 \\ 0 \cdot x_1 + 1 \cdot x_2 = 1 \end{array}$$

are the same, but their matrices

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

have had their rows reordered: GE fails for the first but succeeds for the second $\Longrightarrow$ better to interchange the rows and then apply GE.

Partial pivoting: when creating the zeros in the jth column, find

$$|a_{kj}| = \max(|a_{jj}|, |a_{j+1j}|, \ldots, |a_{nj}|),$$

then swap (interchange) rows j and k.

For example,

$$\begin{bmatrix} a_{11} & \cdot & a_{1j-1} & a_{1j} & \cdot & \cdot & \cdot & a_{1n} \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & a_{j-1j-1} & a_{j-1j} & \cdot & \cdot & \cdot & a_{j-1n} \\ 0 & \cdot & 0 & a_{jj} & \cdot & \cdot & \cdot & a_{jn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{kj} & \cdot & \cdot & \cdot & a_{kn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{nj} & \cdot & \cdot & \cdot & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & \cdot & a_{1j-1} & a_{1j} & \cdot & \cdot & \cdot & a_{1n} \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & a_{j-1j-1} & a_{j-1j} & \cdot & \cdot & \cdot & a_{j-1n} \\ 0 & \cdot & 0 & a_{kj} & \cdot & \cdot & \cdot & a_{kn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{jj} & \cdot & \cdot & \cdot & a_{jn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{nj} & \cdot & \cdot & \cdot & a_{nn} \end{bmatrix}$$

Property: GE with partial pivoting cannot fail if A is nonsingular.

Proof. If A is the first matrix above at the jth stage,

$$\det[A] = a_{11} \cdots a_{j-1j-1} \cdot \det \begin{bmatrix} a_{jj} & \cdot & \cdot & \cdot & a_{jn} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{kj} & \cdot & \cdot & \cdot & a_{kn} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{nj} & \cdot & \cdot & \cdot & a_{nn} \end{bmatrix}.$$

Hence $\det[A] = 0$ if $a_{jj} = \cdots = a_{kj} = \cdots = a_{nj} = 0$. Thus if the pivot $a_{k,j}$ is zero, A is singular. So if A is nonsingular, all of the pivots are nonzero. (Note: actually a_{nn} can be zero and an LU factorization still exist.)

The effect of pivoting is just a permutation (reordering) of the rows, and hence can be represented by a permutation matrix P.

Permutation matrix: P has the same rows as the identity matrix, but in the pivoted order. So

$$PA = LU$$

represents the factorization—equivalent to GE with partial pivoting. E.g.,

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} A$$

has the 2nd row of A first, the 3rd row of A second and the 1st row of A last.

Matlab example:

```
>> A = rand(5,5)
A =
      0.69483      0.38156      0.44559       0.6797      0.95974
       0.3171      0.76552      0.64631       0.6551      0.34039
      0.95022       0.7952      0.70936      0.16261      0.58527
     0.034446      0.18687      0.75469        0.119      0.22381
      0.43874      0.48976      0.27603      0.49836      0.75127
>> exactx = ones(5,1);  b = A*exactx;
>> [LL, UU] = lu(A)  % note "psychologically lower triangular" LL
LL =
      0.73123     -0.39971      0.15111            1            0
      0.33371            1            0            0            0
            1            0            0            0            0
     0.036251        0.316            1            0            0
      0.46173      0.24512     -0.25337      0.31574            1
UU =
      0.95022       0.7952      0.70936      0.16261      0.58527
            0      0.50015      0.40959      0.60083      0.14508
            0            0      0.59954    -0.076759      0.15675
            0            0            0      0.81255      0.56608
            0            0            0            0      0.30645
```

```
>> [L, U, P] = lu(A)
L =
            1            0            0            0            0
      0.33371            1            0            0            0
     0.036251        0.316            1            0            0
      0.73123     -0.39971      0.15111            1            0
      0.46173      0.24512     -0.25337      0.31574            1
U =
      0.95022       0.7952      0.70936      0.16261      0.58527
            0      0.50015      0.40959      0.60083      0.14508
            0            0      0.59954    -0.076759      0.15675
            0            0            0      0.81255      0.56608
            0            0            0            0      0.30645
P =
     0     0     1     0     0
     0     1     0     0     0
     0     0     0     1     0
     1     0     0     0     0
     0     0     0     0     1

>> max(max(P'*L - LL))  % we see LL is P'*L
ans =
     0

>> y = L \ (P*b);  % now to solve Ax = b...
>> x = U \ y
x =
            1
            1
            1
            1
            1

>> norm(x - exactx, 2)  % within roundoff error of exact soln
ans =
   3.5786e-15
```