MATH 5330: Computational Methods of Linear Algebra

Lecture Note 2: The Gaussian Elimination and LU Decomposition

Xianyi Zeng
Department of Mathematical Sciences, UTEP

1 The Gaussian elimination

The method now known as *The Gaussian Elimination* (GE) first appeared about two thousand years ago; the modern notation was, however, devised by Carl F. Gauss and later adopted by "*hand computers*" to solve the normal equations of least-squares problems.

In essence, it is a technique that allows one to solve a small linear system $A\boldsymbol{x}=\boldsymbol{b}$, or equivalently computing A^{-1}, by hand. For example, let us consider the next linear system due to W. S. Ericksen [1]:

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 5 \\ 3 & 6 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

The recipe starts with aligning A with $\boldsymbol{b}$ as $[A\,|\,\boldsymbol{b}]$, and then uses row operations to transform the A part into the identity matrix:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 3 & 4 & 5 & 0 \\ 3 & 6 & 10 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & -3 \\ 0 & 3 & 7 & -3 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & -3 \\ 0 & 0 & 1 & 6 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 0 & -5 \\ 0 & 1 & 0 & -15 \\ 0 & 0 & 1 & 6 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & -15 \\ 0 & 0 & 1 & 6 \end{array}\right].$$

In the end, we obtain $[I\,|\,\boldsymbol{x}]$ if no row exchange occurs. The similar technique can be used to compute the inverse of a matrix A, particularly we use row operations to transform $[A\,|\,I]$ to $[I\,|\,B]$; and if no row exchange occurs we have $B=A^{-1}$.

The main observation here is that: triangular matrices are easy to invert and any non-singular matrix can be reduced to a upper triangular matrix purely by row operations. To this end, let's first consider reducing a general matrix $A=[a_{ij}]$ to an upper triangular form. Define $A^{(0)}=A$ and the k-th step in the GE process transform $A^{(k-1)}$ to $A^{(k)}=[a_{ij}^{(k)}]$, such that:

$$a_{ij}^{(k)}=0 \quad \forall j\le k \text{ and } j<i\,. \tag{1.1}$$

For now we assume there is no row exchange, hence the k-th step involves using the k-th row of $A^{(k-1)}$ (more specifically $a_{kk}^{(k-1)}$) to eliminate all the non-zero elements $a_{ik}^{(k-1)}, i>k$. More precisely:

$$a_{ij}^{(k)}=\begin{cases} a_{ij}^{(k-1)} & i\le k\,, \\ a_{ij}^{(k-1)}-\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}a_{kj}^{(k-1)} & i>k\,. \end{cases} \tag{1.2}$$

It is not difficult to check that if $A^{(k-1)}$ satisfies (1.1), then $A^{(k)}$ obtained from (1.2) also obeys

(1.1). In the matrix notation, (1.1) summarizes as:

$$A^{(k)} = L^{(k)} A^{(k-1)}\,,\ \text{where } L^{(k)} = \begin{bmatrix} 1 & & & & & & \\ \vdots & \ddots & & & & & \\ 0 & \cdots & 1 & & & & \\ 0 & \cdots & -l_{k+1,k} & 1 & & & \\ 0 & \cdots & -l_{k+2,k} & 0 & 1 & & \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \\ 0 & \cdots & -l_{nk} & 0 & 0 & \cdots & 1 \end{bmatrix} \ \text{and}\ l_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}\,. \tag{1.3}$$

Let the process continue until the n-th step, and we eventually obtain:

$$A^{(n)} = L^{(n)} A^{(n-1)} = \cdots = L^{(n)} L^{(n-1)} \cdots L^{(1)} A\,. \tag{1.4}$$

By applying (1.1) to $k=n$ we see that $A^{(n)}$ is upper triangular, which is also denoted by U. Since each $L^{(k)}$ is lower-triangular, their product is also so and we denote its inverse by L. Finally, we obtain the LU decomposition of the matrix A:

$$A = (L^{(n)} L^{(n-1)} \cdots L^{(1)})^{-1} A^{(n)} = LU\,. \tag{1.5}$$

To compute L, we first note that each $L^{(k)}$ of (1.3) is called an *atomic triangular matrix* and it satisfies:

$$(L^{(k)})^{-1} = \begin{bmatrix} 1 & & & & & & \\ \vdots & \ddots & & & & & \\ 0 & \cdots & 1 & & & & \\ 0 & \cdots & l_{k+1,k} & 1 & & & \\ 0 & \cdots & l_{k+2,k} & 0 & 1 & & \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \\ 0 & \cdots & l_{nk} & 0 & 0 & \cdots & 1 \end{bmatrix} \Rightarrow L = (L^{(1)})^{-1} (L^{(2)})^{-1} \cdots (L^{(n)})^{-1} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{bmatrix};$$

hence no additional computational cost is needed for obtaining L.

Taking a look at the complexity of this algorithm, the operation (1.2) involves $(n-k)+(n-k)^2$ multiplications/divisions and $(n-k)^2$ summations/subtractions. Thus the complexity (counted as the number of *floating point operations*) of the LU decomposition is:

$$\sum_{k=1}^{n} \left[(n-k) + 2(n-k)^2\right] \sim \frac{2}{3} n^3\,.$$

A^{-1} can be computed from the inverses of L and U. Clearly, L^{-1} is already known, and we look at U^{-1} here. Let $W=[w_{ij}]$ be U^{-1}, then W is also upper triangular and starting from the last row of $WU=I$ and moving backward:

$$\begin{bmatrix} w_{11} & \cdots & w_{1,n-1} & w_{1n} \\ & \ddots & \vdots & \vdots \\ & & w_{n-1,n-1} & w_{n-1,n} \\ & & & w_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & \cdots & u_{1,n-1} & u_{1n} \\ & \ddots & \vdots & \vdots \\ & & u_{n-1,n-1} & u_{n-1,n} \\ & & & u_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

we obtain:

$$\begin{gathered} w_{nn}u_{nn} = 1\,; \\ w_{n-1,n-1}u_{n-1,n-1} = 1\,, \quad w_{n-1,n-1}u_{n-1,n} + w_{n-1,n}u_{nn} = 0\,; \\ \vdots \end{gathered}$$

and the elements of W are computed by:

$$\begin{aligned} &\text{for } k = n, n-1, \cdots, 1 : \\ &\quad w_{kk} = \frac{1}{u_{kk}}\,; \\ &\quad \text{for } l = k+1, \cdots, n : \\ &\quad\quad w_{kl} = -w_{ll}\sum_{j=k}^{l-1} w_{kj}u_{jl}\,. \end{aligned} \tag{1.6}$$

This process involves:

$$\sum_{k=1}^{n}\left[1 + \sum_{l=k+1}^{n}(l-k+1)\right] = \sum_{k=1}^{n}\frac{1}{2}(n-k+1)(n-k+2) = \sum_{k=1}^{n}\frac{1}{2}k(k+1) \sim \frac{1}{6}n^3$$

multiplications/divisions, and:

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

summations. Hence the complexity of computing U^{-1} is $\sim \frac{1}{3}n^3$. Computing L^{-1} is similar to that of U^{-1}, except that all the diagonal elements of L is unity; and the complexity is also $\sim \frac{1}{3}n^3$.

The diagonal elements of L are unities, but those of U are not. However, we can write $U = D\tilde{U}$, so that D is a diagonal matrix containing the diagonal entries of U, and $\tilde{U} = D^{-1}U$ is an upper triangular matrix with unity diagonal elements. In this case we call $A = LD\tilde{U}$ the LDU decomposition, and whenever appropriate we will also denote it by $A = LDU$.

Theorem 1.1. *If A is non-singular and the LDU decomposition exists, then it is also unique.*

Proof. Let $A = L_1D_1U_1$ and $A = L_2D_2U_2$ be two LDU decompositions, then we have:

$$L_1D_1U_1 = L_2D_2U_2 \quad \Rightarrow \quad L_2^{-1}L_1 = D_2U_2U_1^{-1}D_1^{-1}\,,$$

where we have used the fact that if A is non-singular, both D_1 and D_2 are non-singular. It is not difficult to check that the left hand side is a lower-triangular matrix with unity diagonal entries whereas the right hand side is an upper-triangular matrix. Hence the only way that the equality can hold is that both sides are diagonal matrices and particularly for the left hand side we have $L_2^{-1}L_1 = I$ or equivalently $L_1 = L_2$.

Similarly, we can also write down:

$$D_2^{-1}L_2^{-1}L_1D_1 = U_2U_1^{-1}$$

and deduce that $U_2 = U_1$. Finally, we have $D_1 = L_1^{-1}AU_1^{-1} = L_2^{-1}AU_2^{-1} = D_2$. □

2 Analysis of the LU decomposition and pivoting

Theorem 1.1 assumes the existence of the LDU decomposition. Now let us investigate more about this issue. Particularly, if the Gaussian elimination process in Section 1 succeeds without any trouble, the LDU decomposition exists by construction. The process can only break down if some $a_{kk}^{(k-1)} = 0$ in (1.2) (with possible the exception for the case $k = n$, if the LU decomposition not the LDU one is sought).

Such a breakdown, however, may happen no matter whether A is singular or not. For example, the GE process will break down immediately if $a_{11} = 0$, no matter what other entries are. A fix to this issue is to allow row exchanges, also known as "pivoting". To explain the idea, let us suppose we are at the k-th step computing $A^{(k)}$ from $A^{(k-1)}$. If $a_{kk}^{(k-1)} = 0$, one thing we can do is to identify another row such that $a_{lk}^{(k-1)} \neq 0, l > k$ and exchange the k-th row and the l-th row. This process can be illustrated in matrix form as:

$$A^{(k-1)} \longrightarrow \tilde{A}^{(k-1)} = P^{(k)} A^{(k-1)} ,$$

where $P^{(k)} = [p_{ij}^{(k)}]$ is a special permutation matrix that differs from I in four locations:

$$p_{kl}^{(k)} = p_{lk}^{(k)} = 1 , \quad p_{kk}^{(k)} = p_{ll}^{(k)} = 0 ; \quad p_{ij}^{(k)} = \delta_{ij} \text{ , o.w. } . \tag{2.1}$$

By construction, the kk-th element of $\tilde{A}^{(k-1)}$ is non-zero and we can proceed according to (1.2) with $\tilde{A}^{(k-1)}$ instead of $A^{(k-1)}$. Denoting the resulting multiplier again by $L^{(k)}$, we have:

$$A^{(k)} = L^{(k)} P^{(k)} A^{(k-1)} ,$$

and eventually:

$$U = A^{(n)} = L^{(n)} P^{(n)} L^{(n-1)} P^{(n-1)} \cdots L^{(1)} P^{(1)} A . \tag{2.2}$$

This form is rather complicated comparing to the LU decomposition. However, fundamentally if we knew ahead which rows we'll exchange in the end, we can do all these operations at the beginning and then apply the GE process without worrying about pivoting again. This implies that with the help of pivoting, we can obtain the next factorization of A:

$$U = L^{-1} P^{-1} A \quad \Rightarrow \quad A = PLU . \tag{2.3}$$

Here P (as well as P^{-1}) is the product of a chain of special permutation matrices, which in essence indicates a re-ordering of the rows when it is left-multiplied to a matrix A. The second equation of (2.3) is known as the PLU decomposition; and similarly we can also construct the PLDU factorization if no breakdown occurs.

In general, we can choose any two rows to exchange during the pivoting process as long as a non-zero diagonal entry is found. In practice, however, it is almost exclusively to choose l such that:

$$l = \underset{l \geq k}{\operatorname{argmax}} \left| a_{lk}^{(k-1)} \right| , \tag{2.4}$$

i.e., the largest (in absolute value) element in the k-th column below the diagonal. The benefit of doing so is that the magnitude of each l_{ik} will be no larger than one, c.f. (1.3). We will see later such a strategy leads to smaller accumulated roundoff errors in the final factorization. In subsequent sections, we assume (2.4) by default and still refer to this specific pivoting strategy as pivoting.

Theorem 2.1. *If A is non-singular, then it has a PLDU decomposition.*

Proof. To show the existence, we just need to show that the previous GE process with pivoting will not break down. Particularly, if it breaks down at the k-th stage, on the one hand we know that:

$$a_{ij}^{(k-1)} = 0\,, \quad \forall i \geq k \text{ and } j \leq k\,.$$

Hence $A^{(k-1)}$ is singular. On the other hand, we also have:

$$A^{(k-1)} = L^{(k-1)} P^{(k-1)} \cdots L^{(1)} P^{(1)} A\,,$$

and each $L^{(l)}$ and $P^{(l)}$, $l \leq k-1$ are non-singular; thus the only possibility is that A is singular, a contradiction. □

Remark 1. *The PLDU factorization is in general not unique. In fact, if A is non-singular there usually exist many permutation matrix P such that the GE process of PA does not fail; then each such a P will leads to a different PLDU factorization of the same matrix A.*

Remark 2. *Here we only consider using row exchanges to avoid break down of the GE algorithm, hence the strategy discussed before is also called partial pivoting. Another strategy also allows column exchanges simultaneously with row ones, and it is known as complete pivoting. The result of the complete pivoting is a factorization like $A = PLUQ$ or $A = PLDUQ$, where Q is also a permutation matrix.*

Finally, we note that if the complete pivoting is allowed, an argument similar to the proof of Theorem 2.1 will show that if the process breaks down at the k-th state, then all the elements $a_{ij}^{(k-1)}$, $i \geq k$ are zero. Hence the rank of $A^{(k-1)}$ is $k-1$, or equivalently:

$$\text{rank} A = k - 1\,. \tag{2.5}$$

In this case when A is singular, we can still write $A = PLDUQ$ where L and U are triangular matrices with unity diagonal entries, but D is singular and $\text{rank} D = \text{rank} A$. We can always choose P and Q properly, so that the first $\text{rank} D$ diagonal entries of D are non-zero.

3 The Cholesky decomposition

Let A be a non-singular symmetric matrix, and we suppose again that no pivoting is applied and the GE process does not break. Hence by construction the factorization $A = LDU$ exists and by Theorem 1.1 it is also unique. Because $A = A^t = U^t D L^t$ is also an LDU factorization, by uniqueness we must have:

$$L = U^t$$

or equivalently $A = LDL^t$.

In the case when A is positive-definite all the diagonal entries of D are positive, and we can write $D = D^{\frac{1}{2}} D^{\frac{1}{2}}$. Defining $C = LD^{\frac{1}{2}}$ we arrive at:

$$A = CC^t\,, \tag{3.1}$$

where C is a lower-triangular matrix. (3.1) is known as the *Cholesky decomposition* of the symmetric positive-definite matrix A; and it is easy to show that if it exists it is unique.

The next question is whether the Cholesky decomposition exists or not, with or without any pivoting strategies. By assuming that A is positive-definite, it is automatically non-singular; but the fact that the GE process will not breakdown without any pivoting is not as obvious. To obtain an idea, we first consider the case when the GE process breaks down in the first step, meaning that $a_{11}=0$. This is, however, not allowed, since it indicates:

$$\boldsymbol{e}_1^t A \boldsymbol{e}_1 = 0\,,$$

contradicting the positive-definiteness assumption.

To proceed, we would like to modify the GE algorithm a little bit. That is, instead of computing the entries of L and U, we would like to directly compute the elements of C – which at least saves the storage cost by a half. Writing down (3.1) more explicitly:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & \cdots & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & & & \\ c_{12} & c_{22} & & \\ \vdots & \vdots & \ddots & \\ c_{1n} & c_{2n} & \cdots & c_{nn} \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ & c_{22} & \cdots & c_{2n} \\ & & \ddots & \vdots \\ & & & c_{nn} \end{bmatrix}.$$

We thusly deduce:

$$\begin{aligned} & a_{11} = c_{11}^2\,; \\ & a_{12} = c_{11}c_{12}\,, \quad a_{22} = c_{12}^2 + c_{22}^2\,; \\ & a_{13} = c_{11}c_{13}\,, \quad a_{23} = c_{12}c_{13} + c_{22}c_{23}\,, \quad a_{33} = c_{13}^2 + c_{23}^2 + c_{33}^2\,; \\ & \qquad \vdots \end{aligned}$$

and obtain the following algorithm:

```
for $k = 1,2,\cdots,n$ :                                        (3.2)
    for $l = 1,2,\cdots,k-1$ :
        $c_{lk} = \frac{1}{c_{ll}}\left(a_{lk} - \sum_{j=1}^{l-1} c_{jl}c_{jk}\right)$ ;
    $c_{kk} = \sqrt{a_{kk} - \sum_{l=1}^{k-1} c_{lk}^2}$ .
```

Showing that this algorithm will not break is equivalent to show that the quantity under the square root operator is always positive for all iterations, which we will prove by induction.

Proof. For $k=1$, we just need to show $a_{11}>0$, which is equivalent to:

$$\boldsymbol{e}_1^t A \boldsymbol{e}_1 > 0\,.$$

Now we suppose that the algorithm proceeds until the k-th iteration ($k>1$). Let us denote the upper-left $(k-1)\times(k-1)$ and $k\times k$ sub-matrices of A by A_{k-1} and A_k, respectively; and we also denote the upper-left $(k-1)\times(k-1)$ sub-matrix of C, which is already constructed, by C_{k-1}. Then both A_{k-1} and A_k are symmetric positive-definite and $A_{k-1}=C_{k-1}C_{k-1}^t$. With $c_{ll}, l=1,\cdots,k-1$ available, we can compute $c_{lk}, l<k$ and define:

$$\boldsymbol{a}_k=\begin{bmatrix} a_{1k}\\ a_{2k}\\ \cdots\\ a_{k-1,k}\end{bmatrix}, \quad \text{and} \quad \boldsymbol{c}_k=\begin{bmatrix} c_{1k}\\ c_{2k}\\ \cdots\\ c_{k-1,k}\end{bmatrix}, \quad \text{we have} \quad \boldsymbol{a}_k=C_{k-1}\boldsymbol{c}_k\,.$$

It follows that:

$$A_k=\begin{bmatrix} A_{k-1} & \boldsymbol{a}_k\\ \boldsymbol{a}_k^t & a_{kk}\end{bmatrix}=\begin{bmatrix} C_{k-1}C_{k-1}^t & C_{k-1}\boldsymbol{c}_k\\ \boldsymbol{c}_k^tC_{k-1}^t & a_{kk}\end{bmatrix}.$$

Let $\boldsymbol{x}^t=[-\boldsymbol{c}_k^tC_{k-1}^{-1}\ 1]$, by the positive-definiteness of A_k we have $\boldsymbol{x}^tA_k\boldsymbol{x}>0$, or equivalently:

$$a_{kk}-\boldsymbol{c}_k^t\boldsymbol{c}_k>0\,,$$

which indicates that the algorithm succeeds in the k-th iteration as well. □

We can study the computational cost with the algorithm 3.2: It has:

$$\sum_{k=1}^{n}\left[\sum_{l=1}^{k-1}l+(k-1)\right]\sim\frac{1}{6}n^3$$

multiplications/divisions; and it has:

$$\sum_{k=1}^{n}\left[\sum_{l=1}^{k-1}(l-1)+(k-1)\right]\sim\frac{1}{6}n^3$$

summations/subtractions; in addition, there are n square root operations. Because the cost of computing the square root is independent of n, we conclude that the complexity of the Cholesky decomposition (3.2) is $\sim\frac{1}{3}n^3$, half of that for LU decomposition of general matrices.

Finally, we note that although no pivoting is needed in computing the Cholesky decomposition, allowing certain pivoting strategy can improve the *robustness* of the algorithm, especially when A is ill-conditioned. When pivoting is used, we do not want to destroy the symmetry structure of the matrix; hence we look at the counterpart of the PLUQ or PLDUQ factorization and make sure that $P=Q^t$. In this case, we have:

$$A=PCC^tP^t\,, \tag{3.3}$$

where P is a permutation matrix and C is non-singular lower-triangular.

4 Roundoff error analysis

Modern computers use floating point numbers to represent real numbers. In this framework, only a subset $\mathcal{F}$ of real numbers can be represented exactly:

$$f=\pm.d_1d_2\cdots d_t\times\beta^e\,,\quad 0\le d_i<\beta\,,\quad d_1\ne0\,,\quad L\le e\le U\,. \tag{4.1}$$

The first part $.d_1\cdots d_t$ is called the *significand* and the number e is called the *exponent.* The significand is represented in the basis of β, for example $\beta=10$ corresponds to decimal numbers and $\beta=2$ corresponds to binary numbers. Thus for any nonzero $f\in\mathcal{F}$, the smallest and the largest magnitudes of f are:

$$f_{\min}\le|f|\le f_{\max}\,,\quad f_{\min}=\beta^{L-1}\,,\ \text{ and }\ f_{\max}=\beta^{U}(1-\beta^{-t})\,. \tag{4.2}$$

Define the set $\mathcal{G}$ as:

$$\mathcal{G}=\{x\in\mathbb{R}:\ f_{\min}\le|x|\le f_{\max}\}\cup\{0\}\,, \tag{4.3}$$

then the floating point operator fl: $\mathcal{G}\to\mathcal{F}$ is defined as the nearest representable floating point number (ties are handled by rounding away from zero). We have for all $x\in\mathcal{G}$:

$$fl(x)=x(1+\epsilon)\,,\quad |\epsilon|\le\epsilon_0=\frac{1}{2}\beta^{1-t}\,. \tag{4.4}$$

Double-precision system on typical machines have $(\beta,t,L,U)=(2,52,-1023,1024)$, which makes $\epsilon_0=2^{-52}\approx2\times10^{-16}$ (try the `eps` function in `Matlab`).

For this reason, we cannot expect all the entries of A or all the component of $\boldsymbol{b}$ are precisely represented, and have to accept the fact that the closest floating-point representations of A and $\boldsymbol{b}$ are $A+\delta A$ and $\boldsymbol{b}+\delta b$, respectively:

$$|\delta a_{ij}|\le\epsilon_0|a_{ij}|\,,\quad |\delta b_i|\le\epsilon_0|b_i|\,.$$

To make the notations more compact, we define $fl(A)$ by the matrix $[fl(a_{ij}0]$, $|A|$ by the matrix $[|a_{ij}|]$, and the notation $A\le B$ means $a_{ij}\le b_{ij}$, $\forall i,j$; and similarly for vectors. We can express the previous inequalities as:

$$|fl(A)-A|\le\epsilon_0|A|\,,\quad |fl(\boldsymbol{b})-\boldsymbol{b}|\le\epsilon_0|\boldsymbol{b}|\,. \tag{4.5}$$

In the rest, we may use $fl(A)-A$ and δA interchangeably.

It is not very difficult to check that the estimates (4.5) carry to several norms:

$$||\delta A||_\infty\le\epsilon_0||A||_\infty\,,\quad ||\delta A||_1\le\epsilon_0||A||_1\,,\quad ||\delta\boldsymbol{b}||_p\le\epsilon_0||\boldsymbol{b}||_p\,. \tag{4.6}$$

The next question is, if we apply some algorithm to the matrix $fl(A)$, what will be error in the final results. For example, let B be the result of applying the GE algorithm to compute the inverse of $fl(A)$, how different it will be from A^{-1}? The answer to such questions is rather complicated, so we start with simple operations to prepare our mindset for an analysis of the Gaussian elimination.

First of all, computing units are built in a way such that for the operation *op*, being anything in $\{+,\ -,\ \times,\ \div\}$, if $a,b\in\mathcal{F}$ and $a\ op\ b\in\mathcal{G}$, then:

$$|fl(a\ op\ b)-(a\ op\ b)|\le\epsilon_0|a\ op\ b|\,. \tag{4.7}$$

Next, we consider computing the dot product of two floating point vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ using the following algorithm:

```
$s=0\,,$                                  (4.8)
for $i=1,2,\cdots,n$ :
  $s=s+x_iy_i\,.$
```

Let s_p be the (floating-point) outcome of the p-th iteration, we thusly have $s_1 = fl(x_1y_1) = x_1y_1(1+\delta_1)$ and for $p \geq 2$:

$$s_p = fl(s_{p-1} + fl(x_py_p)) = (s_{p-1} + x_py_p(1+\delta_p))(1+\epsilon_p)\,,$$

where all δ_p and ϵ_p have magnitudes no larger than ϵ_0. It follows that:

$$fl(\boldsymbol{x}\cdot\boldsymbol{y}) = fl(s_n) = \sum_{i=1}^{n} x_iy_i(1+\gamma_i)\,,\quad \gamma_i = (1+\delta_i)\prod_{j=\max(i,2)}^{n}(1+\epsilon_j) - 1\,. \tag{4.9}$$

There are two conclusions that we can draw here. First, noticing that $|\gamma_i| \leq n\epsilon_0 + O(\epsilon_0^2)$, we obtain an estimate on the roundoff error of the inner product:

$$|fl(\boldsymbol{x}\cdot\boldsymbol{y}) - \boldsymbol{x}\cdot\boldsymbol{y}| \leq n\epsilon_0|\boldsymbol{x}|\cdot|\boldsymbol{y}| + O(\epsilon_0^2)\,. \tag{4.10}$$

Second, we notice that a tighter bound for γ_i is $|\gamma_i| \leq \min(n-i+2, n)\epsilon_0 + O(\epsilon_0^2)$, with the bound becoming smaller for larger i. Hence it is advantageously to order the items x_iy_i in a way such that the largest is left to the last, for the purpose of reducing the roundoff error in the final result.

A direct corollary of (4.10) is:

$$|fl(AB) - AB| \leq n\epsilon_0|A||B| + O(\epsilon_0^2)\,, \tag{4.11}$$

where A and B are floating-point $n\times n$ matrices.

Now let's look at the Gaussian elimination process. To begin with, if we can perform all the operations exactly and the only errors are in the initial data A and $\boldsymbol{b}$, follow the last exercise from previous lecture we have:

$$\frac{||\delta\boldsymbol{x}||_\infty}{||\boldsymbol{x}||_\infty} \leq 4\epsilon_0\kappa_\infty(A)\,,$$

if $\epsilon_0\kappa_\infty(A) < 1/2$. Hence a bottom line is that we cannot expect accurate solution of $A\boldsymbol{x} = \boldsymbol{b}$, if the matrix A itself is very ill-conditioned such that $\epsilon_0\kappa_\infty(A) \sim O(1)$.

Then we assume that both A and $\boldsymbol{b}$ are floating-point quantities, and try to derive an error bound on the roundoff error resulting from the GE process. For now, let us assume that no pivoting is used and the GE process in Section 1 is successful. Let $\hat{L}\hat{U} = A + E$ be the result of the GE process, then we can obtain a *posteriori* estimate:

$$|E| \leq 2n\epsilon_0(|A| + |\hat{L}||\hat{U}|) + O(\epsilon_0^2)\,. \tag{4.12}$$

Proof. We use the method of induction and (4.12) clearly holds for $n = 1$. Now suppose that it holds for all $(n-1)\times(n-1)$ floating point matrices and we consider an $n\times n$ matrix A. Write:

$$A = \begin{bmatrix} \alpha & \boldsymbol{u}^t \\ \boldsymbol{v} & B \end{bmatrix},$$

then in the first step of the algorithm we'd like to compute:

$$A = \begin{bmatrix} 1 & 0 \\ \boldsymbol{z} & I \end{bmatrix}\begin{bmatrix} \alpha & \boldsymbol{u}^t \\ 0 & B - \boldsymbol{z}\boldsymbol{u}^t \end{bmatrix},\quad \boldsymbol{z} = \frac{1}{\alpha}\boldsymbol{v}\,.$$

Let $\hat{\boldsymbol{z}} = fl(\boldsymbol{z})$, the actual computations are:

$$\hat{L}\hat{U} = \begin{bmatrix} 1 & 0 \\ \hat{\boldsymbol{z}} & \hat{L}_1 \end{bmatrix} \begin{bmatrix} \alpha & \boldsymbol{u}^t \\ 0 & \hat{U}_1 \end{bmatrix},$$

where $\hat{L}_1\hat{U}_1 = \hat{A}_1 + E_1$ and $\hat{A}_1 = fl(B - fl(\hat{\boldsymbol{z}}\boldsymbol{u}^t)) = B - \hat{\boldsymbol{z}}\boldsymbol{u}^t + F$. By induction:

$$|E_1| \le 2(n-1)\epsilon_0(|\hat{A}_1| + |\hat{L}_1||\hat{U}_1|) + O(\epsilon_0^2).$$

It follows that:

$$E = \hat{L}\hat{U} - A = \begin{bmatrix} 0 & 0 \\ \alpha\hat{\boldsymbol{z}} - \boldsymbol{v} & F + E_1 \end{bmatrix} \quad \Rightarrow \quad |E| \le \begin{bmatrix} 0 & 0 \\ \epsilon_0|\boldsymbol{v}| & |F| + |E_1| \end{bmatrix}.$$

Here we have used the estimate that:

$$|\alpha\hat{\boldsymbol{z}} - \boldsymbol{v}| = |\alpha(fl(\alpha^{-1}\boldsymbol{v}) - \alpha^{-1}\boldsymbol{v})| \le |\alpha\epsilon_0(\alpha^{-1}\boldsymbol{v})| = \epsilon_0|\boldsymbol{v}|.$$

For the last two terms, we first estimate $|F|$ and $|\hat{A}_1|$:

$$\begin{aligned} |F| &\le |fl(B - fl(\hat{\boldsymbol{z}}\boldsymbol{u}^t)) - (B - fl(\hat{\boldsymbol{z}}\boldsymbol{u}^t))| + |fl(\hat{\boldsymbol{z}}\boldsymbol{u}^t) - (\hat{\boldsymbol{z}}\boldsymbol{u}^t)| \\ &\le \epsilon_0|B - fl(\hat{\boldsymbol{z}}\boldsymbol{u}^t)| + \epsilon_0|\hat{\boldsymbol{z}}||\boldsymbol{u}|^t + O(\epsilon_0^2) \le 2\epsilon_0(|B| + |\hat{\boldsymbol{z}}||\boldsymbol{u}|^t) + O(\epsilon_0^2), \end{aligned}$$

and

$$|\hat{A}_1| \le |B - \hat{\boldsymbol{z}}\boldsymbol{u}^t| + |F| \le (1 + 2\epsilon_0)(|B| + |\hat{\boldsymbol{z}}||\boldsymbol{u}|^t) + O(\epsilon_0^2).$$

Hence

$$|E_1| \le 2(n-1)\epsilon_0[(1+2\epsilon_0)(|B| + |\hat{\boldsymbol{z}}||\boldsymbol{u}|^t) + |\hat{L}_1||\hat{U}_1|] + O(\epsilon_0^2) = 2(n-1)\epsilon_0(|B| + |\hat{\boldsymbol{z}}||\boldsymbol{u}|^t + |\hat{L}_1||\hat{U}_1|) + O(\epsilon_0^2),$$

and

$$|F| + |E_1| \le 2n(|B| + |\hat{\boldsymbol{z}}||\boldsymbol{u}|^t + |\hat{L}_1||\hat{U}_1|) + O(\epsilon_0^2).$$

Consequently

$$\begin{aligned} |E| &\le 2n\epsilon_0 \begin{bmatrix} 0 & 0 \\ |\boldsymbol{v}| & |B| + |\hat{\boldsymbol{z}}||\boldsymbol{u}|^t + |\hat{L}_1||\hat{U}_1| \end{bmatrix} + O(\epsilon_0^2) \\ &\le 2n\epsilon_0 \left\{ \begin{bmatrix} |\alpha| & |\boldsymbol{u}|^t \\ |\boldsymbol{v}| & |B| \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ |\hat{\boldsymbol{z}}| & |\hat{L}_1| \end{bmatrix} \begin{bmatrix} |\alpha| & |\boldsymbol{u}|^t \\ 0 & |\hat{U}_1| \end{bmatrix} \right\} + O(\epsilon_0^2). \end{aligned}$$

□

Next we would like to derive a *a priori* estimate on E; and for this purpose we need to estimate $|\hat{L}|$ and $|\hat{U}|$. To this end, we consider the L^∞-norm and have:

$$||E||_\infty \le 2n\epsilon_0(||A||_\infty + ||\hat{L}||_\infty ||\hat{U}||_\infty) + O(\epsilon_0^2).$$

At the moment, let us assume $l = \max_{i,j}|\hat{l}_{ij}|$ and define a *growth factor* ρ by:

$$\rho = \max_{i,j,k} \frac{|\hat{a}_{ij}^{(k)}|}{||A||_\infty}. \tag{4.13}$$

Then we have the estimates:

$$||\hat{L}||_\infty \leq nl\,, \quad ||\hat{U}||_\infty \leq n\rho||A||_\infty\,;$$

these leads to the following estimate:

$$||E||_\infty \leq 2n(n^2l\rho+1)\epsilon_0||A||_\infty + O(\epsilon_0^2)\,. \tag{4.14}$$

Remark 3. *Before going into improving (4.14), we'd like to motivate why we estimate $\hat{L}$ and $\hat{U}$ differently. The major reason is that we have kept the diagonal elements of L to be 1; and it is easy to verify that if A is replaced by βA for some scalar β, L will remain the same whereas U becomes βU. This is reflected in the previous estimates: the norm of A only appears in the estimate for $\hat{U}$.*

There is at least one drawback of (4.14): the number l cannot be bounded by $||A||_\infty$. However, this issue is easily fixed if we allow row pivoting. Noticing that row exchanges do not affect the build up of roundoff errors, we can employ the pivoting of Section 2 and obtain $l=1$. In this case (4.14) is improved as:

$$||E||_\infty \leq 2n(n^2\rho+1)\epsilon_0||A||_\infty + O(\epsilon_0^2)\,. \tag{4.15}$$

Finally, the growth factor ρ is also kind of annoying. However, it is fairly difficult to improve the estimate further. The main reason is that ρ is somehow affected by the conditioning of the matrix A (note that $||A^{-1}||_\infty$ is absent from previous estimates) and one typically expects ρ to be very large for ill-conditioned systems. Nevertheless, for many practical problems people find out that ρ almost remains constant as the size (n) of the problem grows.

5 Block matrices and sparse systems

In the last section we briefly discuss two related topics with the LU decomposition. Firstly, the Gaussian elimination process can be extended easily to *block matrices*. Taking the 2×2 matrix as an example, the LU decomposition is given by:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a_{12}a_{11}^{-1} & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ 0 & a_{22}-a_{12}a_{11}^{-1}a_{21} \end{bmatrix},$$

if $a_{11}\neq0$. Now let us suppose that A_{11}, A_{12}, A_{21}, and A_{22} are $m\times m$, $m\times n$, $n\times m$, and $n\times n$ matrices, respectively; and A_{11} is invertible. Then we can construct the following block LU decomposition similar to the previous example:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I_{m\times m} & 0 \\ A_{12}A_{11}^{-1} & I_{n\times n} \end{bmatrix} \begin{bmatrix} A_{11} & A_{21} \\ 0 & A_{22}-A_{12}A_{11}^{-1}A_{21} \end{bmatrix};$$

note that in the block case, the order of the matrix-multiplications cannot be changed. The block-version of the factorization is very useful, for example, in discretizing system of partial differential equations.

The second topic is on sparse matrices, i.e., matrices with the majority of its entries being zero. Such matrices frequently appear, again, in the numerical discretization of differential equations. Because it is generally a waste of storage if we allocate a full two-dimensional array to store a sparse matrix, a common practice is to only store the information of the non-zero elements. For example, let $A=[a_{ij}]$ be a sparse matrix then in a compressed sparse row storage the following information[1] are stored: For each i such that there exists at least one non-zero a_{ij}, we store the array: $\{(j,a_{ij})\ :\ a_{ij}\neq 0\}$. With such a format, row manipulations are fairly simple but column operations are much more difficult. Hence, complete pivoting is rarely employed for such matrices; whereas row pivoting can be efficiently implemented.

Another issue with the sparse matrix is that A^{-1} is generally not sparse. To this end, people have been choosing to ignore any elements of A^{-1} during the Gaussian elimination process, at the location where $a_{ij}=0$. The result is a matrix B, which is clearly different from A^{-1}; but the hope is that B is reasonably close to A^{-1} so that BA is close to I. In this case, people use B as a pre-conditioner to the linear system with matrix A (see the next lectures). Sometimes there are simply too few non-zeroes in matrix A, and people design algorithms to allow creating new non-zero locations (called *filling in*) in constructing B. Generally, the more filling-ins are allowed, the closer B is to A^{-1} but the more computational cost is required.

Exercises

Exercise 1. *If the purpose is to use the Gaussian elimination to solve the linear system $A\boldsymbol{x}=\boldsymbol{b}$ for some non-singular matrix A, we do not have to compute $A^{-1}=U^{-1}L^{-1}$ explicitly. Actually, once $A=LU$ is available, we can first solve for a vector $\boldsymbol{y}$ such that $L\boldsymbol{y}=\boldsymbol{b}$, which is known as back substitution. Then we solve for the solution $\boldsymbol{x}$ by $U\boldsymbol{x}=\boldsymbol{y}$. How many floating point operations are involved in these two solves?*

Now we extend the discussion to solving the linear systems with multiple right hand sides. Particularly, we look for solutions $\boldsymbol{x}_1,\cdots,\boldsymbol{x}_m$ such that $A[\boldsymbol{x}_1\ \boldsymbol{x}_2\ \cdots\ \boldsymbol{x}_m]=[\boldsymbol{b}_1\ \boldsymbol{b}_2\ \cdots\ \boldsymbol{b}_m]$. Determine the complexity in leading terms of n and m (you also need to take into account of the computational cost associated with computing the LU decomposition of A).

Exercise 2. *In Section 2, we define a permutation matrix P as the product of a chain of simple matrices that are specified by (2.1). Here, we provide a more straightforward definition of a permutation matrix $P=[p_{ij}]$ as any matrix that has exact 1 one and $n-1$ zeroes in each row and in each column; here $n\times n$ is the dimension of the matrix P. Show that if P is a permutation matrix, then it is invertible and its inverse is given by P^t.*

Exercise 3. *Show (4.11). Next, using this estimate to show that for the same matrices A and B:*

$$||fl(AB)-AB||_p\leq n\epsilon_0||A||_p||B||_p+O(\epsilon_0^2)\,. \tag{5.1}$$

for $p=1$ and $p=\infty$.

References

[1] W. S. Ericksen. Inverse pairs of matrices with integer elements. SIAM J. Numer. Anal., 17(3):474–477, 1980.

[1]In a equivalent sense.