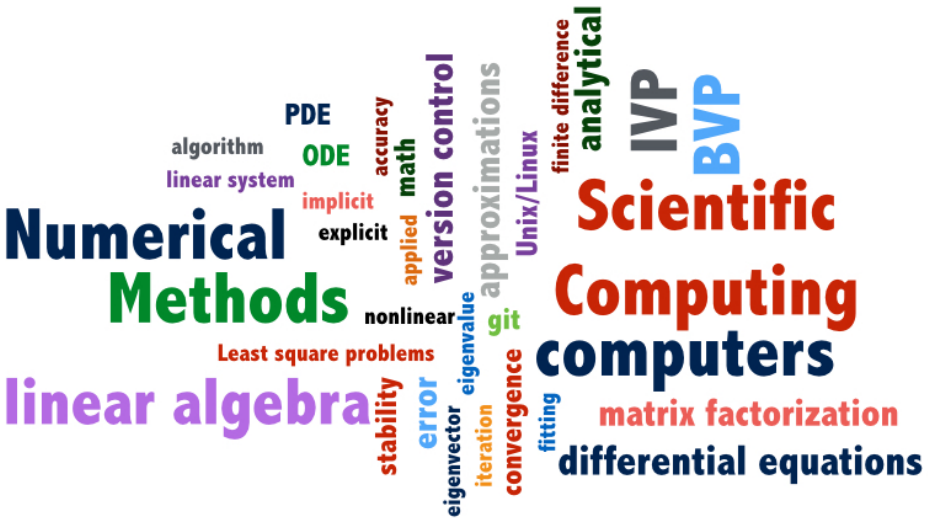
AMS 213B Lecture Notes

Instructor: Prof. Dongwook Lee (dlee79@ucsc.edu)

MWF, 11:00 am – 12:10 pm
Jack Baskin Engineering classroom 169
Spring, 2016



Contents

1 Systems of Linear Equations **3**

Chapter 1

Systems of Linear Equations

There are many relationships in nature that are *linear*, meaning that effects can be described by a matrix-vector notation, or a linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where $\mathbf{A}$ is a $m \times n$ matrix, $\mathbf{x}$ is an n-vector, and $\mathbf{b}$ is an m-vector. This form of liner equation tells us that if we know *causes* $\mathbf{x}$ then we can predict the resulting *effects* $\mathbf{b}$.

More interestingly, we we often want to know is the "reverse" of the process: if we know the effect $\mathbf{b}$ then we should like to be able to determine the corresponding cause vector $\mathbf{x}$. Numerical methods for accomplishing this task is our primary goal in numerical linear algebra.

What about when relationships are nonlinear? In this case, we often seek for an approximated solution that is *linear locally*, whereby we makes use of the linear theory.

One can identify different types of linear problems in three different approaches:

- Solutions of well-posed linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{A}$ a $n \times n$ square matrix, and $\mathbf{x}$ and $\mathbf{b}$ are n-vectors – this is a topic in this chapter.
- Approximate solutions of overdetermined linear systems with $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{A}$ a $m \times n$ matrix ($m > n$), $\mathbf{x}$ is an n-vector, and $\mathbf{b}$ is an m-vector – this is a topic in Chapter 3.
- Eigenvalue and eigenvector problems $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ – this is a topic in Chapter 4.

Studying the systems of linear equations is not only very important in its own sake mathematically, but also crucial in solving various types of discrete solutions in ODEs and PDEs. Learning stable, accurate, fast and efficient numerical algorithms for linear algebra therefore will be a fundamental resource in both pure mathematics and applied mathematics.

The goal will be for you to develop a set of useful routines for solving a wide range of linear algebra problems. Let's begin with reviewing basic concepts of vectors, matrices, and their relations.

1. Review of Basic Linear Algebra

1.1. Existence and uniqueness

An $n \times n$ matrix $\mathbf{A}$ is said to be *nonsingular* if it satisfies any one of the following equivalence conditions:

1. $\mathbf{A}$ has an inverse $\mathbf{A}^{-1}$ such that $\mathbf{AA^{-1}} = \mathbf{I}$, where $\mathbf{I}$ is an identity matrix.
2. $\det(\mathbf{A}) \neq 0$.
3. $\text{rank}(\mathbf{A}) = n$ (the rank of a matrix is the maximum number of linearly independent rows or columns it contains).
4. For any nonzero vector $\mathbf{x} \neq 0$, $\mathbf{Ax} \neq \mathbf{0}$ (i.e., $\mathbf{A}$ does not annihilate non-zero vector).

Otherwise, the matrix is said to be *singular*. For a given square matrix $\mathbf{A}$ and $\mathbf{b}$, the possibilities of solution $\mathbf{x}$ are summarized as follows:

- Unique solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ if $\mathbf{A}$ is nonsingular and $\mathbf{b}$ is arbitrary.
- Infinitely many solutions if $\mathbf{A}$ is singular and $\mathbf{b} \in \text{span}(\mathbf{A}) = \{\mathbf{Ax} : \mathbf{x} \in \mathbb{R}^n\}$ (why? Hint: Assume $\mathbf{A}(\mathbf{x} + \gamma\mathbf{z}) = \mathbf{0}$ for any scalar γ and for nonzero $\mathbf{z}$).
- No solution if $\mathbf{A}$ is singular and $\mathbf{b} \notin \text{span}(\mathbf{A})$.

Definition: The p-norm (or l^p-norm) of an n-vector $\mathbf{x}$ is defined by

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} \left| x_i \right|^p \right)^{\frac{1}{p}}. \tag{1.2}$$

Important special cases are:

- 1-norm:
$$||\mathbf{x}||_1 = \sum_{i=1}^{n} \left| x_i \right| \tag{1.3}$$
- 2-norm:
$$||\mathbf{x}||_2 = \left(\sum_{i=1}^{n} \left| x_i \right|^2 \right)^{\frac{1}{2}} \tag{1.4}$$
- ∞-norm (or max-norm):
$$||\mathbf{x}||_\infty = \max_{1 \leq i \leq n} \left| x_i \right| \tag{1.5}$$

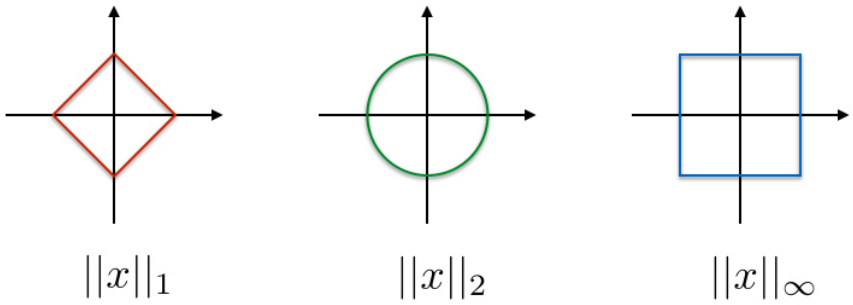


Figure 1. Illustrations of unit circle, $||x|| = 1$, in three different norms: 1-norm, 2-norm and ∞-norm.

Example: For the vector $\mathbf{x} = (-1.6, 1.2)^T$, we get

$$||\mathbf{x}||_1 = 2.8, ||\mathbf{x}||_2 = 2.0, ||\mathbf{x}||_\infty = 1.6. \tag{1.6}$$

□

Definition: The matrix p-norm of $m \times n$ matrix $\mathbf{A}$ can be defined by

$$||\mathbf{A}||_p = \max_{\mathbf{x} \neq \mathbf{0}} \frac{||\mathbf{A}\mathbf{x}||_p}{||\mathbf{x}||_p}. \tag{1.7}$$

Some matrix norms are easier to compute than others, for example,

- 1-norm:
$$||\mathbf{A}||_1 = \max_j \sum_{i=1}^{m} \left|a_{ij}\right| \tag{1.8}$$
- ∞-norm:
$$||\mathbf{A}||_\infty = \max_i \sum_{j=1}^{n} \left|a_{ij}\right| \tag{1.9}$$

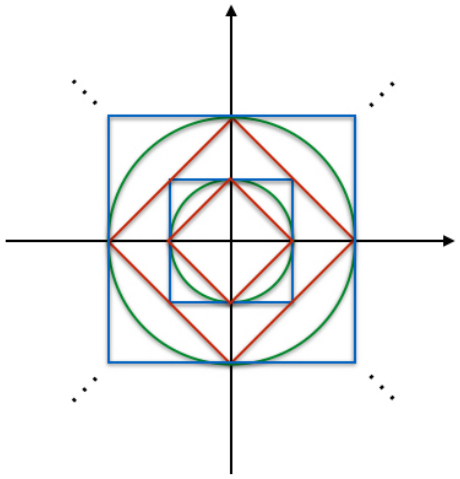
Definition: The condition number of a nonsingular square matrix $\mathbf{A}$ with respect to a given matrix norm is defined to be

$$\text{cond}(\mathbf{A}) = ||\mathbf{A}|| \cdot ||\mathbf{A}^{-1}|| \tag{1.10}$$

By convention, $\text{cond}(\mathbf{A}) = \infty$ if $\mathbf{A}$ is singular.

The following important properties of the condition number are easily derived from the definition and hold for any norm:

1. For any matrix $\mathbf{A}$, $\text{cond}(\mathbf{A}) \geq 1$.
2. For the identity matrix, $\text{cond}(\mathbf{I}) = 1$.



$$||x||_1 \leq C_1||x||_2 \leq C_2||x||_\infty \leq C_3||x||_1 \leq C_4||x||_2 \leq C_5||x||_\infty \leq \cdots$$

Figure 2. The norm equivalence theorem indicates any given norm in finite dimensional vector space can be scaled to be bounded in a different choice of norms.

3. For any matrix $\mathbf{A}$ and nonzero γ, $\text{cond}(\gamma\mathbf{A}) = \text{cond}(\mathbf{A})$.

4. For any diagonal matrix $\mathbf{D} = \text{diag}(d_{ii})$, $\text{cond}(\mathbf{D}) = \frac{\max_i |d_{ii}|}{\min_i |d_{ii}|}$.

Remark: The condition number is a measure of how close a matrix is to being singular: a matrix with a large condition number is nearly singular, whereas a matrix with a condition number close to 1 is far from being singular. □

Remark: Notice that the determinant of a matrix is *not* a good indicator of near singularity. In other words, the magnitude of $\det(\mathbf{A})$ has no information on how close to singular the matrix $\mathbf{A}$ may be. For example, $\det(\alpha\mathbf{I}_n) = \alpha^n$. If $|\alpha| < 1$ the determinant can be very small, yet the matrix $\alpha\mathbf{I}_n$ is perfectly well-conditioned for any nonzero α. □

Remark: The usefulness of the condition number is in accessing the accuracy of solutions to linear system. However, the calculation of the condition number is not trivial as it involves the inverse of the matrix. Therefore, in practice, one often seeks for a good estimated approach to approximate condition numbers. □

2. Direct Methods for Solving Linear Systems

Recall that in this chapter we are interested in solving a well-defined linear system given as

$$\mathbf{Ax} = \mathbf{b}, \tag{1.11}$$

where $\mathbf{A}$ is a $n \times n$ *square* matrix and $\mathbf{x}$ and $\mathbf{b}$ are n-vectors.

2.1. Invariant Transformations

2.1.1. Permutation To solve a linear system, we wish to transform the given linear system into an easier linear system where the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ remains unchanged. The answer is that we can introduce any nonsingular matrix $\mathbf{M}$ and multiply from the left both sides of the given linear system:

$$\mathbf{MAx} = \mathbf{Mb}. \tag{1.12}$$

We can easily check that the solution remains the same. To see this, let $\mathbf{z}$ be the solution of the linear system in Eqn. 1.12. Then

$$\mathbf{z} = (\mathbf{MA})^{-1}\mathbf{Mb} = \mathbf{A}^{-1}\mathbf{M}^{-1}\mathbf{Mb} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{x}. \tag{1.13}$$

Example: A permutation matrix $\mathbf{P}$, a square matrix having exactly one 1 in each row and column and zeros elsewhere – which is also always a nonsingular – can always be multiplied without affecting the original solution to the system. For instance,

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{1.14}$$

permutes $\mathbf{v}$ as

$$\mathbf{P}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} v_3 \\ v_1 \\ v_2 \end{bmatrix}. \tag{1.15}$$

□

2.1.2. Row scaling Another invariant transformation exists which is called *row scaling*, an outcome of a multiplication by a diagonal matrix $\mathbf{D}$ with nonzero diagonal entries $d_{ii}, i = 1, \ldots n$. In this case, we have

$$\mathbf{DAx} = \mathbf{Db}, \tag{1.16}$$

by which each row of the transformed matrix $\mathbf{DA}$ gets to be scaled by d_{ii} from the original matrix $\mathbf{A}$. Note that the scaling factors are cancelled by the same scaling factors introduced on the right hand side vector, leaving the solution to the original system unchanged.

Note: The column scaling does not preserve the solution in general. □

2.2. LU factorization by Gaussian elimination

Consider the following system of linear equations:

$$\begin{aligned} x_1 + 2x_2 + 2x_3 &= 3, && (1.17)\\ -4x_2 - 6x_3 &= -6, && (1.18)\\ -x_3 &= 1. && (1.19) \end{aligned}$$

We know this is easily solvable since we already know $x_3 = -1$, which gives $x_2 = 3$, therefore recursively arriving a complete set of solution with $x_1 = -1$. When putting these equations into a matrix-vector form, we have

$$\begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix}, \tag{1.20}$$

where the matrix has a form of (upper) triangular.

Therefore, our strategy then is to devise a nonsingular linear transformation that transforms a given general linear system into a triangular linear system. This is a key idea of *LU factorization (or LU decomposition)* or also known as *Gaussian elimination*.

The main idea is to find a matrix $\mathbf{M}_1$ such that the first column of $\mathbf{M}_1\mathbf{A}$ becomes zero below the first row. The right hand side $\mathbf{b}$ is also multiplied by $\mathbf{M}_1$ as well. Again, we repeat this process in the next step so that we find $\mathbf{M}_2$ such that the second column of $\mathbf{M}_2\mathbf{M}_1\mathbf{A}$ becomes zero below the second row, along with applying the equivalent multiplication on the right hand side, $\mathbf{M}_2\mathbf{M}_1\mathbf{b}$. This process is continued for each successive column until all of the subdiagonal entries of the resulting matrix have been annihilated.

If we define the final matrix $\mathbf{M} = \mathbf{M}_{n-1} \cdots \mathbf{M}_1$, the transformed linear system becomes

$$\mathbf{M}_{n-1} \cdots \mathbf{M}_1 \mathbf{A}\mathbf{x} = \mathbf{M}\mathbf{A}\mathbf{x} = \mathbf{M}\mathbf{b} = \mathbf{M}_{n-1} \cdots \mathbf{M}_1 \mathbf{b}. \tag{1.21}$$

Note: As seen in the previous section, we recall that any nonsingular matrix multiplication is an invariant transformation that does not affect the solution to the given linear system.

The resulting transformed linear system $\mathbf{M}\mathbf{A}\mathbf{x} = \mathbf{M}\mathbf{b}$ is upper triangular which is what we want, and can be solved by back-substitution to obtain the solution to the original linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Example: We illustrate Gaussian elimination by considering:

$$\begin{array}{llllll} 2x_1 & +x_2 & +x_3 & & = 3, \\ 4x_1 & +3x_2 & +3x_3 & +x_4 & = 6, \\ 8x_1 & +7x_2 & +9x_3 & +5x_4 & = 10, \\ 6x_1 & +7x_2 & +9x_3 & +8x_4 & = 1. \end{array} \tag{1.22}$$

or in a matrix notation

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 10 \\ 1 \end{bmatrix} = \mathbf{b}. \tag{1.23}$$

The first question is to find a matrix $\mathbf{M}_1$ that annihilates the subdiagonal entries of the first column of $\mathbf{A}$. This can be done if we consider a matrix $\mathbf{M}_1$ that can subtract twice the first row from the second row, four times the first row from the third row, and three times the first row from the fourth row. The matrix $\mathbf{M}_1$ is then identical to the identity matrix $\mathbf{I}_4$, except for those multiplication factors in the first column:

$$\mathbf{M}_1\mathbf{A} = \begin{bmatrix} 1 & & & \\ -2 & 1 & & \\ -4 & & 1 & \\ -3 & & & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & 3 & 5 & 5 \\ & 4 & 6 & 8 \end{bmatrix}, \tag{1.24}$$

where we treat the blank entries to be zero entries. At the same time, we proceed the corresponding multiplication on the right hand side to get:

$$\mathbf{M}_1\mathbf{b} = \begin{bmatrix} 3 \\ 0 \\ -2 \\ -8 \end{bmatrix}. \tag{1.25}$$

The next step would be to annihilate the third and fourth entries from the second column (3 and 4), which will give a next matrix $\mathbf{M}_2$ that has the form:

$$\mathbf{M}_2\mathbf{M}_1\mathbf{A} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & -3 & 1 & \\ & -4 & & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & 3 & 5 & 5 \\ & 4 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 2 & 4 \end{bmatrix}, \tag{1.26}$$

now with the right hand side:

$$\mathbf{M}_2\mathbf{M}_1\mathbf{b} = \begin{bmatrix} 3 \\ 0 \\ -2 \\ -8 \end{bmatrix}. \tag{1.27}$$

The last matrix $\mathbf{M}_3$ will complete the process, resulting an upper triangular matrix $\mathbf{U}$:

$$\mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{A} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 2 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & & 2 \end{bmatrix} = \mathbf{U}, \tag{1.28}$$

together with the right hand side:

$$\mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{b} = \begin{bmatrix} 3 \\ 0 \\ -2 \\ -6 \end{bmatrix} = \mathbf{y}. \tag{1.29}$$

We see that the final transformed linear system $\mathbf{MAx} = \mathbf{Ux} = \mathbf{y}$ is upper triangular which is what we wanted and it can be solved easily by back-substitution, starting from obtaining $x_4 = -3$, followed by x_3, x_2, and x_1 in reverse order to find a complete solution

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ -3 \end{bmatrix}. \tag{1.30}$$

The full LU factorization $\mathbf{A} = \mathbf{LU}$ can be established if we compute

$$\mathbf{L} = (\mathbf{M}_3\mathbf{M}_2\mathbf{M}_1)^{-1} = \mathbf{M}_1^{-1}\mathbf{M}_2^{-1}\mathbf{M}_3^{-1}. \tag{1.31}$$

At first sight this looks like an expensive process as it involves inverting a series of matrices. Surprisingly, however, this turns out to be a trivial task. The inverse of $\mathbf{M}_i$, $i = 1, 2, 3$ is just itself but with each entry below the diagonal negated. Therefore, we have

$$\begin{aligned}
\mathbf{L} &= \mathbf{M}_1^{-1}\mathbf{M}_2^{-1}\mathbf{M}_3^{-1} \\
&= \begin{bmatrix} 1 & & & \\ -2 & 1 & & \\ -4 & & 1 & \\ -3 & & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & -3 & 1 & \\ & -4 & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1 & 1 \end{bmatrix}^{-1} \\
&= \begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & & 1 & \\ 3 & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & 3 & 1 & \\ & 4 & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & 1 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & 3 & 1 & \\ 3 & 4 & 1 & 1 \end{bmatrix}.
\end{aligned} \tag{1.32}$$

Notice also that the matrix multiplication $\mathbf{M}_1^{-1}\mathbf{M}_2^{-1}\mathbf{M}_3^{-1}$ is also trivial and is just the unit lower triangle matrix with the nonzero subdiagonal entries of $\mathbf{M}_1^{-1}$, $\mathbf{M}_2^{-1}$, and $\mathbf{M}_3^{-1}$ inserted in the appropriate places.

All together, we finally have our decomposition $\mathbf{A} = \mathbf{LU}$:

$$\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & 3 & 1 & \\ 3 & 4 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & & 2 \end{bmatrix}. \tag{1.33}$$

□

Quick summary: Gaussian elimination proceeds in steps until a upper triangular matrix is obtained for back-substitution:

$$\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \xrightarrow{\mathbf{M}_1} \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \xrightarrow{\mathbf{M}_2} \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix} \xrightarrow{\mathbf{M}_3} \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{bmatrix} \tag{1.34}$$

□

Algorithm: LU factorization by Gaussian elimination:

```
for k = 1 to n − 1
    #[loop over column]
    if a_kk = 0 then
        stop
        #[stop if pivot (or divisor) is zero]
    endif
    for i = k + 1 to n
        m_ik = a_ik / a_kk
        #[compute multipliers for each column]
    endfor
    for j = k + 1 to n
        for i = k + 1 to n
            a_ij = a_ij − m_ik a_kj
            #[transformation to remaining submatrix]
        endfor
    endfor
endfor
```

2.3. Pivoting

2.3.1. Need for pivoting We obviously run into trouble when the choice of a divisor – called a *pivot* – is zero, whereby the Gaussian elimination algorithm breaks down. As illustrated in **Algorithm** above, this situation can be easily checked and avoided so that the algorithm stops when one of the diagonal entries become singular.

The solution to this singular pivot issue is almost equally straightforward: if the pivot entry is zero at state k, i.e., $a_{kk} = 0$, then one interchange row k of *both* the matrix and the right hand side vector with some subsequent row whose entry in column k is nonzero and resume the process as usual. Recall that permutation does not alter the solution to the system.

This row interchanging process is called *pivoting*, which is illustrated in the following example.

Example: Pivoting with permutation matrix can be easily explained as below:

$$\begin{bmatrix} * & * & * & * \\ 0 & 0 & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \xrightarrow{\mathbf{P}} \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix} \tag{1.35}$$

where we interchange the second row with the fourth row using a permutation matrix $\mathbf{P}$ given as

$$\mathbf{P} = \begin{bmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{bmatrix}. \tag{1.36}$$

□

Note: The potential need for pivoting has nothing to do with the matrix being singular. For example, the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.37}$$

is nonsingular, yet we can't process LU factorization unless we interchange rows. On the other hand, the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{1.38}$$

can easily allow LU factorization

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \mathbf{LU}, \tag{1.39}$$

while being singular. □

2.3.2. Partial pivoting There is not only zero pivots, but also another situation we must avoid in Gaussian elimination – a case with *small* pivots. The problem is closely related to computer's finite-precision arithmetic which fails to recover any numbers smaller than the machine precision ϵ. Recall that we have $\epsilon \approx 10^{-7}$ for single precision, and $\epsilon \approx 10^{-16}$ for double precision.

Example: Let us now consider a matrix $\mathbf{A}$ defined as

$$\mathbf{A} = \begin{bmatrix} \tilde{\epsilon} & 1 \\ 1 & 1 \end{bmatrix}, \tag{1.40}$$

where $\tilde{\epsilon} < \epsilon \approx 10^{-16}$, say, $\tilde{\epsilon} = 10^{-20}$. If we proceed without any pivoting (i.e., no row interchange) and take $\tilde{\epsilon}$ as the first pivot element, then we obtain the elimination matrix

$$\mathbf{M} = \begin{bmatrix} 1 & 1 \\ -1/\tilde{\epsilon} & 1 \end{bmatrix}, \tag{1.41}$$

and hence the lower triangular matrix

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ 1/\tilde{\epsilon} & 1 \end{bmatrix} \tag{1.42}$$

which is correct. For the upper triangular matrix, however, we see an incorrect floating-point arithmetic operation

$$\mathbf{U} = \begin{bmatrix} \tilde{\epsilon} & 1 \\ 0 & 1 - 1/\tilde{\epsilon} \end{bmatrix} = \begin{bmatrix} \tilde{\epsilon} & 1 \\ 0 & -1/\tilde{\epsilon} \end{bmatrix}, \tag{1.43}$$

since $1/\tilde{\epsilon} >> 1$. But then we simply fail to recover the original matrix $\mathbf{A}$ from the factorization:

$$\mathbf{LU} = \begin{bmatrix} 1 & 0 \\ 1/\tilde{\epsilon} & 1 \end{bmatrix} \begin{bmatrix} \tilde{\epsilon} & 1 \\ 0 & -1/\tilde{\epsilon} \end{bmatrix} = \begin{bmatrix} \tilde{\epsilon} & 1 \\ 1 & 0 \end{bmatrix} \neq \mathbf{A}. \tag{1.44}$$

Using a small pivot, and a correspondingly large multiplier, has caused an unrecoverable loss of information in the transformation.

We can cure the situation by interchanging the two rows first, which gives the first pivot element to be 1 and the resulting multiplier is $-\tilde{\epsilon}$:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 \\ -\tilde{\epsilon} & 1 \end{bmatrix}, \tag{1.45}$$

and hence

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ \tilde{\epsilon} & 1 \end{bmatrix} \text{ and } \mathbf{U} = \begin{bmatrix} 1 & 1 \\ 0 & 1 - \tilde{\epsilon} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{1.46}$$

in floating-point arithmetic. We therefore recover the original relation:

$$\mathbf{LU} = \begin{bmatrix} 1 & 0 \\ \tilde{\epsilon} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \tilde{\epsilon} & 1 \end{bmatrix} = \mathbf{A}, \tag{1.47}$$

which is the correct result after permutation. □

The foregoing example is rather extreme, however, the principle in general holds to find the largest pivot in producing each elimination matrix, by which one obtains a smaller multiplier as an outcome and hence smaller errors in floating-point arithmetic. We see that this process involves repeated use of permutation matrix $\mathbf{P}_k$ that interchanges rows to bring the entry of largest magnitude on or below the diagonal in column k into the diagonal pivot position.

Quick summary: Gaussian elimination with *partial pivoting* proceeds as below. Assume x_{ik} is chosen to be the maximum in magnitude among the entries in k-th column, thereby selected as a k-th pivot:

$$\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & x_{ik} & * & * \\ & * & * & * \end{bmatrix} \xrightarrow{\mathbf{P}_1} \begin{bmatrix} * & * & * & * \\ & x_{ik} & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix} \xrightarrow{\mathbf{M}_1} \begin{bmatrix} * & * & * & * \\ & x_{ik} & * & * \\ & 0 & * & * \\ & 0 & * & * \end{bmatrix} \tag{1.48}$$

In general, $\mathbf{A}$ becomes an upper triangular matrix $\mathbf{U}$ after $n-1$ steps,

$$\mathbf{M}_{n-1}\mathbf{P}_{n-1}\cdots\mathbf{M}_1\mathbf{P}_1\mathbf{A} = \mathbf{U}. \tag{1.49}$$

□

Note: The expression in Eq. 1.49 can be rewritten in a way that separates the elimination and the permutation processes into two different groups

$$\mathbf{P} = \mathbf{P}_{n-1}\cdots\mathbf{P}_2\mathbf{P}_1, \tag{1.50}$$
$$\mathbf{L} = (\mathbf{M}_{n-1}^{'}\cdots\mathbf{M}_2^{'}\mathbf{M}_1^{'})^{-1}, \tag{1.51}$$

so that we write the final transformed matrix as

$$\mathbf{PA} = \mathbf{LU}. \tag{1.52}$$

To do this we first need to find what $\mathbf{M}_i^{'}$ should be. Consider reordering the operations in Eq. 1.49 in the form, for instance with $n-1=3$,

$$\mathbf{M}_3\mathbf{P}_3\mathbf{M}_2\mathbf{P}_2\mathbf{M}_1\mathbf{P}_1 = \mathbf{M}_3^{'}\mathbf{M}_2^{'}\mathbf{M}_1^{'}\mathbf{P}_2\mathbf{P}_2\mathbf{P}_1 (= \mathbf{L}^{-1}\mathbf{P}). \tag{1.53}$$

Rearranging operations,

$$\mathbf{M}_3\mathbf{P}_3\mathbf{M}_2\mathbf{P}_2\mathbf{M}_1\mathbf{P}_1 \tag{1.54}$$
$$= (\mathbf{M}_3)(\mathbf{P}_3\mathbf{M}_2\mathbf{P}_3^{-1})(\mathbf{P}_3\mathbf{P}_2\mathbf{M}_1\mathbf{P}_2^{-1}\mathbf{P}_3^{-1})(\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1) \tag{1.55}$$
$$\equiv (\mathbf{M}_3^{'})(\mathbf{M}_2^{'})(\mathbf{M}_1^{'})\mathbf{P}_2\mathbf{P}_2\mathbf{P}_1, \tag{1.56}$$

whereby we can define $\mathbf{M}_i^{'}, i = 1,2,3$ equals to $\mathbf{M}_i$ but with the subdiagonal entries permuted:

$$\mathbf{M}_3^{'} = \mathbf{M}_3 \tag{1.57}$$
$$\mathbf{M}_2^{'} = \mathbf{P}_3\mathbf{M}_2\mathbf{P}_3^{-1} \tag{1.58}$$
$$\mathbf{M}_1^{'} = \mathbf{P}_3\mathbf{P}_2\mathbf{M}_1\mathbf{P}_2^{-1}\mathbf{P}_3^{-1} \tag{1.59}$$

We can see that the matrix $\mathbf{M}_{n-1}^{'}\cdots\mathbf{M}_2^{'}\mathbf{M}_1^{'}$ is unit lower triangular and hence easily invertible by negating the subdiagonal entries to obtain $\mathbf{L}$. □

Example: To see what is going on, consider

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}. \tag{1.60}$$

With partial pivoting, let's interchange the first and third rows with $\mathbf{P}_1$:

$$\begin{bmatrix} & & 1 & \\ & 1 & & \\ 1 & & & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{bmatrix}. \tag{1.61}$$

The first elimination step now looks like this with left-multiplication by $\mathbf{M}_1$:

$$\begin{bmatrix} 1 & & & \\ -1/2 & 1 & & \\ -1/4 & & 1 & \\ -3/4 & & & 1 \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ & -1/2 & -3/2 & -3/2 \\ & -3/4 & -5/4 & -5/4 \\ & 7/4 & 9/4 & 17/4 \end{bmatrix}. \tag{1.62}$$

Now the second and fourth rows are interchanged with $\mathbf{P}_2$:

$$\begin{bmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ & -1/2 & -3/2 & -3/2 \\ & -3/4 & -5/4 & -5/4 \\ & 7/4 & 9/4 & 17/4 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ & 7/4 & 9/4 & 17/4 \\ & -3/4 & -5/4 & -5/4 \\ & -1/2 & -3/2 & -3/2 \end{bmatrix}. \tag{1.63}$$

With multiplication by $\mathbf{M}_2$ the second elimination step looks like:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & 3/7 & 1 & \\ & 2/7 & & 1 \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ & 7/4 & 9/4 & 17/4 \\ & -3/4 & -5/4 & -5/4 \\ & -1/2 & -3/2 & -3/2 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ & 7/4 & 9/4 & 17/4 \\ & & -2/7 & 4/7 \\ & & -6/7 & -2/7 \end{bmatrix} \tag{1.64}$$

Interchanging the third and fourth rows now with $\mathbf{P}_3$:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ & 7/4 & 9/4 & 17/4 \\ & & -2/7 & 4/7 \\ & & -6/7 & -2/7 \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ & 7/4 & 9/4 & 17/4 \\ & & -6/7 & -2/7 \\ & & -2/7 & 4/7 \end{bmatrix}. \tag{1.65}$$

The final elimination step is obtained with $\mathbf{M}_3$:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1/3 & 1 \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ & 7/4 & 9/4 & 17/4 \\ & & -6/7 & -2/7 \\ & & -2/7 & 4/7 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ & 7/4 & 9/4 & 17/4 \\ & & -6/7 & -2/7 \\ & & & 2/3 \end{bmatrix}. \tag{1.66}$$

□

Remark: The name "partial" pivoting comes from the fact that *only the current column* is searched for a suitable pivot. A more exhausting pivoting strategy is *complete pivoting*, in which the entire remaining unreduced sub matrix is searched for the largest entry, which is then permuted into the diagonal pivot position. □

Algorithm: LU factorization by Gaussian elimination with Partial Pivoting:

```
for k = 1 to n − 1
    #[loop over column]
    Find index p such that
        |a_pk| ≥ |a_ik| for k ≤ i ≤ n
        #[search for pivot in current column]
    if p ≠ k then
        interchange rows k and p
        #[interchange rows if needed]
    endif
    if a_kk = 0 then
        continue with next k
        #[skip current column if zero]
    endif
    for i = k + 1 to n
        m_ik = a_ik / a_kk
        #[compute multipliers for each column]
    endfor
    for j = k + 1 to n
        for i = k + 1 to n
            a_ij = a_ij − m_ik a_kj
            #[transformation to remaining submatrix]
        endfor
    endfor
endfor
```

2.4. Gauss-Jordan elimination for inverse matrix

We have seen in Gaussian elimination that the LU factorization transforms a general matrix into a triangular form which becomes easier to solve than the original linear system. Can we extend this transformation technique bit further so that we can possibly obtain a system that is even easier than the triangular form? The answer is yes.

We notice that a diagonal linear system appears to be the next desirable target. This method is called *Gauss-Jordan elimination* and is a variation of standard Gaussian elimination in which the matrix is reduced to diagonal form rather than merely a triangular form.

Quick summary: Gauss-Jordan elimination can be illustrated as in the following pictorial steps:

$$\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \rightarrow \begin{bmatrix} 1 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & * & * \\ 0 & 1 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix} \tag{1.67}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & * \\ 0 & 1 & 0 & * \\ 0 & 0 & 1 & * \\ 0 & 0 & 0 & * \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.68}$$

□

Definition: Let us introduce a form called *augmented matrix* of the system $\mathbf{Ax} = \mathbf{b}$ which writes the $n \times n$ matrix $\mathbf{A}$ and the n-vector $\mathbf{b}$ together in a new $n \times (n+1)$ matrix form:

$$\Big[\mathbf{A}\Big|\mathbf{b}\Big]. \tag{1.69}$$

The use of augmented matrix allows us to write each transformation step of the linear system (i.e., both $\mathbf{A}$ and $\mathbf{b}$) in a compact way.

Example: Consider the following system using Gauss-Jordan elimination without pivoting:

$$\begin{array}{llll} x_1 & +x_2 & +x_3 & = 4 \\ 2x_1 & +2x_2 & +5x_3 & = 11 \\ 4x_1 & +6x_2 & +8x_3 & = 24 \end{array}, \tag{1.70}$$

which can be put in as an augmented matrix form:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 5 & 11 \\ 4 & 6 & 8 & 24 \end{array}\right]. \tag{1.71}$$

First step is to annihilate the first column:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 5 & 11 \\ 4 & 6 & 8 & 24 \end{array}\right] \xrightarrow{\mathbf{M}_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 0 & 3 & 3 \\ 0 & 2 & 4 & 8 \end{array}\right], \text{ where } \mathbf{M}_1 = \begin{bmatrix} 1 & & \\ -2 & 1 & \\ -4 & & 1 \end{bmatrix}. \tag{1.72}$$

Next we permute:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 0 & 3 & 3 \\ 0 & 2 & 4 & 8 \end{array}\right] \xrightarrow{\mathbf{P}_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 2 & 4 & 8 \\ 0 & 0 & 3 & 3 \end{array}\right], \text{ where } \mathbf{P}_1 = \begin{bmatrix} 1 & & \\ & & 1 \\ & 1 & \end{bmatrix}. \tag{1.73}$$

Next row scaling by multiplying a diagonal matrix $\mathbf{D}_1$:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 2 & 4 & 8 \\ 0 & 0 & 3 & 3 \end{array}\right] \xrightarrow{\mathbf{D}_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 1 & 2 & 4 \\ 0 & 0 & 1 & 1 \end{array}\right], \text{ where } \mathbf{D}_1 = \begin{bmatrix} 1 & & \\ & 1/2 & \\ & & 1/3 \end{bmatrix}. \tag{1.74}$$

Next annihilate the remaining upper diagonal entries in the third column:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 1 & 2 & 4 \\ 0 & 0 & 1 & 1 \end{array}\right] \xrightarrow{\mathbf{M}_2} \left[\begin{array}{ccc|c} 1 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{array}\right], \text{ where } \mathbf{M}_2 = \begin{bmatrix} 1 & & -1 \\ & 1 & -2 \\ & & 1 \end{bmatrix}. \tag{1.75}$$

Finally, annihilate the upper diagonal entry in the second column:

$$\left[\begin{array}{ccc|c} 1 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{array}\right] \xrightarrow{\mathbf{M}_3} \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{array}\right], \text{ where } \mathbf{M}_3 = \begin{bmatrix} 1 & -1 & \\ & 1 & \\ & & 1 \end{bmatrix}. \tag{1.76}$$

□

In this example the right hand side is a single n-vector. What happens if we perform the same procedure using multiple n-vectors? In other words, consider now a new choice of the right hand side in the form of $n \times n$ augmented matrix:

$$\left[\mathbf{b}_1 \middle| \mathbf{b}_2 \middle| \cdots \middle| \mathbf{b}_n\right]. \tag{1.77}$$

We see that the same operation can easily be performed simultaneously on individual $\mathbf{b}_i, 1 \leq i \leq n$.

Especially, if we choose $\mathbf{b}_i = [0, \cdots, 1, \cdots, 0]^T$ with unity at ith entry, then the collection of vectors $\left[\mathbf{b}_1 \middle| \mathbf{b}_2 \middle| \cdots \middle| \mathbf{b}_n\right]$ actually becomes the identity matrix $\mathbf{I}$. In this case we see that Gauss-Jordan elimination yields the inverse of $\mathbf{A}$:

$$\left[\mathbf{A} \middle| \left[\mathbf{b}_1 \middle| \mathbf{b}_2 \middle| \cdots \middle| \mathbf{b}_n\right]\right] = \left[\mathbf{A} \middle| \mathbf{I}\right] \rightarrow \cdots \rightarrow \left[\mathbf{I} \middle| \mathbf{A}^{-1}\right]. \tag{1.78}$$

Remark: Although the resulting diagonal system in GJ provides an easier way to obtain the final solution than the back-substitution in triangular form, the elimination process of GJ is much more expensive requiring about $n^3/2$ multiplications and a similar number of additions, which is 50 percent more expensive than standard Gaussian elimination. Therefore, in practice, GJ is not a preferred way to compute linear systems. □

Remark: Then why do we learn GJ at all? Because it is straightforward, understandable, solid as a rock, and an exceptionally good "psychological" backup for those times that something is going wrong and you think it might be your linear-equation solver. □

2.5. Cholesky factorization for symmetric positive definite systems

Thus far we have assumed that the linear system has a general matrix and is *dense*, meaning that majority of the matrix entries are nonzero. On the other hand, there are some special cases we can seek for improved efficiency in both working and storing data when operating on some special matrices.

Some examples of special properties of real matrix $\mathbf{A}$ that can be exploited include the following:

- *Symmetric:* $\mathbf{A} = \mathbf{A}^T$, i.e., $a_{ij} = a_{ji}$ for all i, j.
- *Positive definite:* $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for all $\mathbf{x} \neq \mathbf{0}$.
- *Banded:* $a_{ij} = 0$ for all $|i - j| > \beta$, where β is the *bandwidth* of $\mathbf{A}$. An important special case is a *tridiagonal matrix*, for which $\beta = 1$.
- *Sparse:* most entries of $\mathbf{A}$ are zero.

Remark: The properties defined above for *real* matrices have analogues for *complex* matrices. In the complex case the usual matrix transpose (denoted by T) is replaced by the conjugate transpose (denoted by H). For instance, the conjugate transpose of a complex matrix $\mathbf{A}$ is denoted as

$$\mathbf{A}^H = \bar{a}_{ji}, \tag{1.79}$$

where $\bar{a}_{ji}$ represents complex conjugate for each matrix entry. □

Remark: For a real matrix $\mathbf{A}$, $\mathbf{A}^H = \mathbf{A}^T$. □

Definition: An analog to the real symmetric matrix in complex matrix is called *Hermitian* matrix if

$$\mathbf{A} = \mathbf{A}^H. \tag{1.80}$$

Definition: Similarly, a complex matrix $\mathbf{A}$ is called positive definite if

$$\mathbf{x}^H\mathbf{A}\mathbf{x} > 0, \text{ for all complex vector } \mathbf{x} \neq \mathbf{0}. \tag{1.81}$$

Definition: If the matrix $\mathbf{A}$ symmetric and positive definite (SPD), then an LU decomposition of $\mathbf{A}$ indicates that

$$\mathbf{U}^T\mathbf{L}^T = (\mathbf{L}\mathbf{U})^T = \mathbf{A}^T = \mathbf{A} = \mathbf{L}\mathbf{U}, \tag{1.82}$$

so that $\mathbf{U} = \mathbf{L}^T$, that is, $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is lower triangular and has positive diagonal entries (but not in general, a unit diagonal). This is known as the *Cholesky factorization* of $\mathbf{A}$.

Remark: Since $\mathbf{U} = \mathbf{L}^T$, Cholesky factorization is twice faster than the standard Gaussian elimination. □

Example: Let us begin with considering a simple 2×2 case of a symmetric positive definite matrix decomposition:

$$\begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{bmatrix}. \tag{1.83}$$

This implies we have

$$l_{11} = \sqrt{a_{11}}, \tag{1.84}$$
$$l_{21} = a_{21}/l_{11}, \tag{1.85}$$
$$l_{22} = \sqrt{a_{22} - l_{21}^2}. \tag{1.86}$$

□

The algorithm of CF can be generalized as follow:

Algorithm: Cholesky factorization (decomposition):

```
for k = 1 to n
    #[loop over column]
    a_kk = √a_kk
    for i = k + 1 to n
        a_ik = a_ik / a_kk
        #[scale current column]
    endfor
    for j = k + 1 to n
        for i = k + 1 to n
            a_ij = a_ij − a_ik a_kj
            #[from each remaining column,
            # subtract multiple of current column]
        endfor
    endfor
endfor
```

Note: There is a number of facts about the CF algorithm that make it very attractive and popular for symmetric positive definite matrices:

- In the above **Algorithm** we see that the Cholesky factor $\mathbf{L}$ overwrites the original matrix $\mathbf{A}$, without requiring a separate storage space for $\mathbf{L}$.

- The n square roots required are all of positive numbers, therefore CF is well-defined.
- No pivoting is required for numerical stability.
- Only the lower triangle of $\mathbf{A}$ (e.g., a_{11}, a_{21}, a_{22}) is accessed and hence the strict upper triangular potion (e.g., a_{12}) need not be stored.
- Only about $n^3/6$ multiplications and a similar number of additions are required.
- In all, CF requires only about half as much work and half as much storage as are required for LU factorization of a general matrix by Gaussian elimination.

□

Example: To illustrate the algorithm, we compute the CF of the symmetric positive definite (SPD) matrix

$$\mathbf{A} = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix} \tag{1.87}$$

Step1: Dividing the first column by $\sqrt{3} \approx 1.7321$ from the first for-loop:

$$\begin{bmatrix} 1.7321 & & \\ -0.5744 & 3 & \\ -0.5744 & -1 & 3 \end{bmatrix} \tag{1.88}$$

Step2: Second column update from the second for-loop:

$$\begin{bmatrix} 1.7321 & & \\ -0.5744 & 3-(-0.5744)^2 & \\ -0.5744 & -1-(-0.5744)(-0.5744) & 3-(-0.5744)(-0.5744) \end{bmatrix}$$
$$= \begin{bmatrix} 1.7321 & & \\ -0.5744 & 2.6667 & \\ -0.5744 & -1.3333 & 2.6667 \end{bmatrix} \tag{1.89}$$

We are now done with the first iteration of the outer most for-loop (i.e., $k = 1$), and move on to the next one, $k = 2$.

Step4: Second column scaling from the first for-loop:

$$\begin{bmatrix} 1.7321 & & \\ -0.5744 & 2.6667/\sqrt{2.6667} & \\ -0.5744 & -1.3333/\sqrt{2.6667} & 2.6667 \end{bmatrix}$$
$$= \begin{bmatrix} 1.7321 & & \\ -0.5744 & 1.6330 & \\ -0.5744 & -0.8165 & 2.6667 \end{bmatrix} \tag{1.90}$$

Step5: Third column update from the second for-loop:

$$\begin{bmatrix} 1.7321 & & \\ -0.5744 & 1.6330 & \\ -0.5744 & -0.8165 & 2.6667-(-0.8165)^2 \end{bmatrix}$$
$$= \begin{bmatrix} 1.7321 & & \\ -0.5744 & 1.6330 & \\ -0.5744 & -0.8164 & 2.0 \end{bmatrix} \tag{1.91}$$

We are now done with the second iteration of the outer most for-loop (i.e., $k = 2$), and move on to the next one, $k = 3$. In this case, there is nothing to be done in the second for-loop as $j = k + 1 = 4$ which is beyond the size of the matrix.

Step6: Third column scaling from the first for-loop:

$$\mathbf{L} = \begin{bmatrix} 1.7321 & & \\ -0.5744 & 1.6330 & \\ -0.5744 & -0.8164 & 2.0/\sqrt{2.0} \end{bmatrix}$$
$$= \begin{bmatrix} 1.7321 & & \\ -0.5744 & 1.6330 & \\ -0.5744 & -0.8164 & 1.4142 \end{bmatrix} \tag{1.92}$$

□

2.6. Short Discussion on Operation Counts of Gaussian elimination and Gauss-Jordan

In general, the overall operation count of seeking for the solution $\mathbf{X}$ to the linear system $\mathbf{AX} = \mathbf{B}$, where $\mathbf{A}$ is an $n \times m$ matrix, $\mathbf{X}$ is an $m \times m$ matrix, and

$$\mathbf{B} = \Big[\mathbf{b}_1\Big|\mathbf{b}_2\Big|\cdots\Big|\mathbf{b}_m\Big], \text{ an } n \times m \text{ matrix}, \tag{1.93}$$

can be found out to be (see more details in one of our references *Numerical Recipes*):

- Gaussian elimination: $\mathcal{O}(\frac{n^3}{3} + \frac{n^2 m}{2} + \frac{n^2 m}{2})$
- Gauss-Jordan: $\mathcal{O}(n^3 + n^2 m)$

Note: The above quick estimation tells us that GE is about a factor 3 advantage over GJ for small number of $m << n$, e.g., $m = 1$. □

Note: One also can see that (again, see *Numerical Recipes* for more logical discussion) for matrix inversion, the two methods turn out to be identical in performance. □

2.7. Crout's Method for LU decomposition

As yet another variant of Gaussian elimination method we consider a more compact transformation method, called *Crout's method*, that allows to convert **A** to **L** and **U** directly, including an efficient storage algorithm of entries of them.

Recall that the previous LU decomposition method using Gaussian elimination, with or without pivoting, does not provide both **L** and **U** simultaneously during the calculation steps. Rather we first compute the upper triangular matrix **U**, and we separately compute the lower triangular matrix by multiplying elimination matrices $\mathbf{M}_i$. The solution **x** is then evaluated by the back-substitution. One can say that the Cholesky factorization algorithm does provide both **L** and **U** simultaneously but it CF is only limited to a special case for symmetric positive definite systems.

The Crout's algorithm can be applied for any general (dense) $n \times n$ matrix **A** and directly decomposes **A** into **L** and **U**, $\mathbf{A} = \mathbf{LU}$, and hence

$$\mathbf{b} = \mathbf{Ax} = (\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{Ux}). \tag{1.94}$$

This can be further broken into two successive linear systems:

$$\mathbf{Ly} = \mathbf{b}, \tag{1.95}$$
$$\mathbf{Ux} = \mathbf{y}. \tag{1.96}$$

Notice that Eq. 1.95 can be solved by *forward-substitution*, which is analogous to the back-substitution in Eq. 1.96.

The idea in Crout's method is to consider an efficient method to decompose $\mathbf{A} = \mathbf{LU}$. Putting this in a 4×4 component form for instance,

$$\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ l_{31} & l_{32} & l_{33} & \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & u_{22} & u_{23} & u_{24} \\ & & u_{33} & u_{34} \\ & & & u_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{1.97}$$

We first note that there are 16 equations and $\frac{4\times 5}{2} = 10$ unknowns for each l_{ij} and u_{ij} (20 total unknowns), hence the system can't be solved. However, we can overcome this difficulty by imposing

$$l_{ii} = 1, \tag{1.98}$$

as can be observed experimentally in Eq. 1.32. This then removes 4 unknowns from **L**, whereby we can easily solve for **L** and **U**, given **A**:

$$\begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ l_{31} & l_{32} & 1 & \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & u_{22} & u_{23} & u_{24} \\ & & u_{33} & u_{34} \\ & & & u_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{1.99}$$

2.7.1. First try without pivoting We see that we can write the relation in Eq. 1.99 as ($n = 4$ in our example)

$$a_{ij} = \sum_{k=1}^{n} l_{ik} u_{kj} = \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj}, \tag{1.100}$$

since

$$\begin{cases} l_{ik} = 0 & \text{if } k > i \\ u_{kj} = 0 & \text{if } k > j \end{cases} \tag{1.101}$$

Evaluating first few steps, we get:

- Step 1:
  For $i = 1$,

$$a_{1j} = \sum_{k=1}^{1} l_{1k} u_{kj} = u_{1j}, \forall j. \tag{1.102}$$

  This completes evaluations denoted by u_{1j} which can be simply stored in the position of corresponding a_{1j}.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{1.103}$$

  Now for $i \geq 2, j = 1$,

$$a_{i1} = \sum_{k=1}^{1} l_{ik} u_{k1} = l_{i1} u_{11}, \forall i \geq 2, \forall j, \tag{1.104}$$

  hence,

$$l_{i1} = \frac{a_{i1}}{u_{11}}, \forall i \geq 2. \tag{1.105}$$

  This completes evaluations denoted by l_{i1} that can stored in the place of corresponding a_{i1}:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & a_{22} & a_{23} & a_{24} \\ l_{31} & a_{32} & a_{33} & a_{34} \\ l_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{1.106}$$

- Step 2:
  Now for $i = 2$,

$$a_{2j} = \sum_{k=1}^{2} l_{2k} u_{kj} = l_{21} u_{1j} + 1 \cdot u_{2j}, \forall j, \tag{1.107}$$

giving

$$u_{2j} = a_{2j} - l_{21}u_{1j}, \forall j. \tag{1.108}$$

This completes evaluations denoted by u_{2j} which can be simply stored in the position of corresponding a_{2j}:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & a_{32} & a_{33} & a_{34} \\ l_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{1.109}$$

Similarly, for $i \geq 3$, $j = 2$, we get:

$$a_{i2} = \sum_{k=1}^{2} l_{ik}u_{k2} = l_{i1}u_{12} + l_{i2}u_{22}, \forall i \geq 3, \tag{1.110}$$

giving

$$l_{i2} = \frac{a_{i2} - l_{i1}u_{12}}{u_{22}}, \forall i \geq 3. \tag{1.111}$$

This completes evaluations denoted by l_{i2} which can be simply stored in the position of corresponding a_{i2}:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & a_{33} & a_{34} \\ l_{41} & l_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{1.112}$$

- Step 3: We repeat the same process until the end.

We can generalize this and write Crout's algorithm without pivot as

---

Algorithm: Crout's algorithm without pivot:

```
for k = 1 to n
    #[sweep though Step1, Step2, etc.]
    for j = k to n
        u_kj = a_kj − Σ_{m=1}^{k−1} l_km u_mj
        #[fill out each row]
        #[this can be stored at a_kj]
    endfor
    for i = k + 1 to n
        l_ik = (1/u_kk)(a_ik − Σ_{m=1}^{k−1} l_im u_mk)
        #[fill out subdiagonal entries of each column]
        #[this can be stored at a_ik]
    endfor
endfor
```

2.7.2. Second try with pivoting The previous attempt of designing the Crout's algorithm does not facilitate to provide any pivoting, which is essential to stability and accuracy, as the order of processes (i.e., Steps) alternates rows and columns, which is not suitable for pivoting. Also, such an implementation will significantly slows down array handling in both Fortran and C because the use of indices i, j are adjacent at all.

We can rectify the situation by re-ordering the operations to column-wise operation only by postponing the row-wise evaluation (i.e., no alternating column fill and row fill, but just column fill) so that the algorithm allows a partial pivoting.

The modified algorithm can be written as:

Algorithm: Crout's algorithm with column-wise only:

```
for j = 1 to n
    #[loop over column]
    for i = 1 to j
        u_ij = a_ij − Σ_{m=1}^{i−1} l_im u_mj
        #[fill out each row]
        #[this can be stored at a_ij]
    endfor
    for i = j + 1 to n
        l_ij = (1/u_jj) ( a_ij − Σ_{m=1}^{j−1} l_im u_mj )
        #[fill out subdiagonal entries of each column]
        #[this can be stored at a_ij]
    endfor
endfor
```

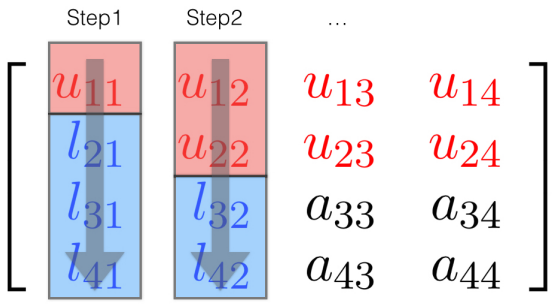


Figure 3. Column-wise operation of Crout's algorithm, delaying the row fill operation until later.

The Crout's algorithm with partial pivoting can be written as:

Algorithm: Crout's algorithm with implicit pivoting:

```
for i = 1 to n
    scale(i) = max_{1≤j≤n} |a_ij|
    #[loop over rows to get max]
endfor

for j = 1 to n
    #[loop over column]
    for i = 1 to j − 1
        u_kj = a_kj − Σ_{m=1}^{k−1} l_km u_mj
        #[compute all u_ij except for u_jj
        # which must be selected by pivoting]
    endfor

    for i = j to n
        l_ij = a_ij − Σ_{m=1}^{j−1} l_im u_mk
        #[partial evaluation of l_ij omitting division by u_jj.
        # The largest of these will be the pivot u_jj.
        # Note that this formula is correct for u_jj]
        pivot = max_{j≤i≤n} |l_ij / scale(i)|
        set i_pivot = i for row index i that contains pivot
    endfor

    if j ≠ i_pivot then
        interchange row j with row i_pivot
        u_jj = pivot
        #[switch rows if max pivot is found]
        record the switch for RHS
    endif
    for i = j + 1 to n
        l_ij = l_ij / u_jj
        #[divide by u_jj to complete the l_ij calculation]
    endif
```

Remark: This algorithm uses *implicit* pivoting where each equation is first scaled by its largest entry, then the Crout's algorithm performed.

The Crout's algorithm with partial pivoting can be written as:

Algorithm: Crout's algorithm with partial pivoting:

```
for j = 1 to n
    #[loop over column]
    for i = 1 to j
        a_ij = a_ij − Σ_{m=1}^{i−1} l_im u_mj
        i_pivot = i
        pivot = |a_j,j|
        #[compute all u_ij except for u_jj
        # which must be selected by pivoting]
        #[The summation is zero if i − 1 < 1]
    endfor

    for i = j + 1 to n
        a_ij = a_ij − Σ_{m=1}^{j−1} l_im u_mj
        #[partial evaluation of l_ij omitting division by u_jj.
        # The largest of these will be the pivot u_jj.
        # Note that this formula is correct for u_jj]
        #[The summation is zero if j − 1 < 1]
        if pivot < |a_ij| then
            pivot = |a_i,j|
            i_pivot = i
            #[Record a new max]
        endif
    endfor

    if j ≠ i_pivot then
        interchange row j with row i_pivot
        #[switch rows if max pivot is found]
        record the switch for RHS
    endif

    for i = j + 1 to n
        a_ij = a_ij / a_jj
        #[divide by u_jj to complete the l_ij calculation]
    endif
```

Remark: This algorithm overwrites entries of **L** and **U** to **A** as shown in Fig. 3.

Remark: Notice that this type of pivoting is what we used in the Gaussian elimination. The resulting matrix **A** holds the elements of both the lower and the upper triangular matrices, arranged as in Fig. 3. Note also that the product of the resulting lower and upper matrices **LU** looks slightly different from the original matrix **A** because of the row swappings from the partial pivoting.

However, **LU** is equivalent to the original **A** except for the corresponding swaps, or permutations, and the successive forward and backward substitutions should give the correct solution **x**.