

# Matrix Operations

Because operations on matrices lie at the heart of scientific computing, efficient algorithms for working with matrices have many practical applications. This chapter focuses on how to multiply matrices and solve sets of simultaneous linear equations. Appendix D reviews the basics of matrices.

Section 28.1 shows how to solve a set of linear equations using LUP decompositions. Then, Section 28.2 explores the close relationship between multiplying and inverting matrices. Finally, Section 28.3 discusses the important class of symmetric positive-definite matrices and shows how we can use them to find a least-squares solution to an overdetermined set of linear equations.

One important issue that arises in practice is *numerical stability*. Due to the limited precision of floating-point representations in actual computers, round-off errors in numerical computations may become amplified over the course of a computation, leading to incorrect results; we call such computations *numerically unstable*. Although we shall briefly consider numerical stability on occasion, we do not focus on it in this chapter. We refer you to the excellent book by Golub and Van Loan [144] for a thorough discussion of stability issues.

---

## 28.1 Solving systems of linear equations

Numerous applications need to solve sets of simultaneous linear equations. We can formulate a linear system as a matrix equation in which each matrix or vector element belongs to a field, typically the real numbers  $\mathbb{R}$ . This section discusses how to solve a system of linear equations using a method called LUP decomposition.

We start with a set of linear equations in  $n$  unknowns  $x_1, x_2, \dots, x_n$ :

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\
 a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\
 &\vdots \\
 a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n.
 \end{aligned} \tag{28.1}$$

A **solution** to the equations (28.1) is a set of values for  $x_1, x_2, \dots, x_n$  that satisfy all of the equations simultaneously. In this section, we treat only the case in which there are exactly  $n$  equations in  $n$  unknowns.

We can conveniently rewrite equations (28.1) as the matrix-vector equation

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

or, equivalently, letting  $A = (a_{ij})$ ,  $x = (x_i)$ , and  $b = (b_i)$ , as

$$Ax = b. \tag{28.2}$$

If  $A$  is nonsingular, it possesses an inverse  $A^{-1}$ , and

$$x = A^{-1}b \tag{28.3}$$

is the solution vector. We can prove that  $x$  is the unique solution to equation (28.2) as follows. If there are two solutions,  $x$  and  $x'$ , then  $Ax = Ax' = b$  and, letting  $I$  denote an identity matrix,

$$\begin{aligned}
 x &= Ix \\
 &= (A^{-1}A)x \\
 &= A^{-1}(Ax) \\
 &= A^{-1}(Ax') \\
 &= (A^{-1}A)x' \\
 &= x'.
 \end{aligned}$$

In this section, we shall be concerned predominantly with the case in which  $A$  is nonsingular or, equivalently (by Theorem D.1), the rank of  $A$  is equal to the number  $n$  of unknowns. There are other possibilities, however, which merit a brief discussion. If the number of equations is less than the number  $n$  of unknowns—or, more generally, if the rank of  $A$  is less than  $n$ —then the system is **underdetermined**. An underdetermined system typically has infinitely many solutions, although it may have no solutions at all if the equations are inconsistent. If the number of equations exceeds the number  $n$  of unknowns, the system is **overdetermined**, and there may not exist any solutions. Section 28.3 addresses the important

problem of finding good approximate solutions to overdetermined systems of linear equations.

Let us return to our problem of solving the system  $Ax = b$  of  $n$  equations in  $n$  unknowns. We could compute  $A^{-1}$  and then, using equation (28.3), multiply  $b$  by  $A^{-1}$ , yielding  $x = A^{-1}b$ . This approach suffers in practice from numerical instability. Fortunately, another approach—LUP decomposition—is numerically stable and has the further advantage of being faster in practice.

## Overview of LUP decomposition

The idea behind LUP decomposition is to find three  $n \times n$  matrices  $L$ ,  $U$ , and  $P$  such that

$$PA = LU, \quad (28.4)$$

where

- $L$  is a unit lower-triangular matrix,
- $U$  is an upper-triangular matrix, and
- $P$  is a permutation matrix.

We call matrices  $L$ ,  $U$ , and  $P$  satisfying equation (28.4) an **LUP decomposition** of the matrix  $A$ . We shall show that every nonsingular matrix  $A$  possesses such a decomposition.

Computing an LUP decomposition for the matrix  $A$  has the advantage that we can more easily solve linear systems when they are triangular, as is the case for both matrices  $L$  and  $U$ . Once we have found an LUP decomposition for  $A$ , we can solve equation (28.2),  $Ax = b$ , by solving only triangular linear systems, as follows. Multiplying both sides of  $Ax = b$  by  $P$  yields the equivalent equation  $PAx = Pb$ , which, by Exercise D.1-4, amounts to permuting the equations (28.1). Using our decomposition (28.4), we obtain

$$LUx = Pb.$$

We can now solve this equation by solving two triangular linear systems. Let us define  $y = Ux$ , where  $x$  is the desired solution vector. First, we solve the lower-triangular system

$$Ly = Pb \quad (28.5)$$

for the unknown vector  $y$  by a method called “forward substitution.” Having solved for  $y$ , we then solve the upper-triangular system

$$Ux = y \quad (28.6)$$

for the unknown  $x$  by a method called “back substitution.” Because the permutation matrix  $P$  is invertible (Exercise D.2-3), multiplying both sides of equation (28.4) by  $P^{-1}$  gives  $P^{-1}PA = P^{-1}LU$ , so that

$$A = P^{-1}LU . \quad (28.7)$$

Hence, the vector  $x$  is our solution to  $Ax = b$ :

$$\begin{aligned} Ax &= P^{-1}LUx \quad (\text{by equation (28.7)}) \\ &= P^{-1}Ly \quad (\text{by equation (28.6)}) \\ &= P^{-1}Pb \quad (\text{by equation (28.5)}) \\ &= b . \end{aligned}$$

Our next step is to show how forward and back substitution work and then attack the problem of computing the LUP decomposition itself.

### Forward and back substitution

**Forward substitution** can solve the lower-triangular system (28.5) in  $\Theta(n^2)$  time, given  $L$ ,  $P$ , and  $b$ . For convenience, we represent the permutation  $P$  compactly by an array  $\pi[1..n]$ . For  $i = 1, 2, \dots, n$ , the entry  $\pi[i]$  indicates that  $P_{i,\pi[i]} = 1$  and  $P_{ij} = 0$  for  $j \neq \pi[i]$ . Thus,  $PA$  has  $a_{\pi[i],j}$  in row  $i$  and column  $j$ , and  $Pb$  has  $b_{\pi[i]}$  as its  $i$ th element. Since  $L$  is unit lower-triangular, we can rewrite equation (28.5) as

$$\begin{aligned} y_1 &= b_{\pi[1]}, \\ l_{21}y_1 + y_2 &= b_{\pi[2]}, \\ l_{31}y_1 + l_{32}y_2 + y_3 &= b_{\pi[3]}, \\ &\vdots \\ l_{n1}y_1 + l_{n2}y_2 + l_{n3}y_3 + \cdots + y_n &= b_{\pi[n]}. \end{aligned}$$

The first equation tells us that  $y_1 = b_{\pi[1]}$ . Knowing the value of  $y_1$ , we can substitute it into the second equation, yielding

$$y_2 = b_{\pi[2]} - l_{21}y_1 .$$

Now, we can substitute both  $y_1$  and  $y_2$  into the third equation, obtaining

$$y_3 = b_{\pi[3]} - (l_{31}y_1 + l_{32}y_2) .$$

In general, we substitute  $y_1, y_2, \dots, y_{i-1}$  “forward” into the  $i$ th equation to solve for  $y_i$ :

$$y_i = b_{\pi[i]} - \sum_{j=1}^{i-1} l_{ij} y_j .$$

Having solved for  $y$ , we solve for  $x$  in equation (28.6) using **back substitution**, which is similar to forward substitution. Here, we solve the  $n$ th equation first and work backward to the first equation. Like forward substitution, this process runs in  $\Theta(n^2)$  time. Since  $U$  is upper-triangular, we can rewrite the system (28.6) as

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1,n-2}x_{n-2} + u_{1,n-1}x_{n-1} + u_{1n}x_n &= y_1 , \\ u_{22}x_2 + \cdots + u_{2,n-2}x_{n-2} + u_{2,n-1}x_{n-1} + u_{2n}x_n &= y_2 , \\ &\vdots \\ u_{n-2,n-2}x_{n-2} + u_{n-2,n-1}x_{n-1} + u_{n-2,n}x_n &= y_{n-2} , \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1} , \\ u_{nn}x_n &= y_n . \end{aligned}$$

Thus, we can solve for  $x_n, x_{n-1}, \dots, x_1$  successively as follows:

$$\begin{aligned} x_n &= y_n / u_{n,n} , \\ x_{n-1} &= (y_{n-1} - u_{n-1,n}x_n) / u_{n-1,n-1} , \\ x_{n-2} &= (y_{n-2} - (u_{n-2,n-1}x_{n-1} + u_{n-2,n}x_n)) / u_{n-2,n-2} , \\ &\vdots \end{aligned}$$

or, in general,

$$x_i = \left( y_i - \sum_{j=i+1}^n u_{ij} x_j \right) / u_{ii} .$$

Given  $P$ ,  $L$ ,  $U$ , and  $b$ , the procedure LUP-SOLVE solves for  $x$  by combining forward and back substitution. The pseudocode assumes that the dimension  $n$  appears in the attribute  $L.rows$  and that the permutation matrix  $P$  is represented by the array  $\pi$ .

### LUP-SOLVE( $L, U, \pi, b$ )

- 1  $n = L.rows$
- 2 let  $x$  be a new vector of length  $n$
- 3 **for**  $i = 1$  **to**  $n$
- 4      $y_i = b_{\pi[i]} - \sum_{j=1}^{i-1} l_{ij} y_j$
- 5 **for**  $i = n$  **downto** 1
- 6      $x_i = (y_i - \sum_{j=i+1}^n u_{ij} x_j) / u_{ii}$
- 7 **return**  $x$

Procedure LUP-SOLVE solves for  $y$  using forward substitution in lines 3–4, and then it solves for  $x$  using backward substitution in lines 5–6. Since the summation within each of the **for** loops includes an implicit loop, the running time is  $\Theta(n^2)$ .

As an example of these methods, consider the system of linear equations defined by

$$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}x = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix},$$

where

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix},$$

$$b = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix},$$

and we wish to solve for the unknown  $x$ . The LUP decomposition is

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.2 & 1 & 0 \\ 0.6 & 0.5 & 1 \end{pmatrix},$$

$$U = \begin{pmatrix} 5 & 6 & 3 \\ 0 & 0.8 & -0.6 \\ 0 & 0 & 2.5 \end{pmatrix},$$

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

(You might want to verify that  $PA = LU$ .) Using forward substitution, we solve  $Ly = Pb$  for  $y$ :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0.2 & 1 & 0 \\ 0.6 & 0.5 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 3 \\ 7 \end{pmatrix},$$

obtaining

$$y = \begin{pmatrix} 8 \\ 1.4 \\ 1.5 \end{pmatrix}$$

by computing first  $y_1$ , then  $y_2$ , and finally  $y_3$ . Using back substitution, we solve  $Ux = y$  for  $x$ :

$$\begin{pmatrix} 5 & 6 & 3 \\ 0 & 0.8 & -0.6 \\ 0 & 0 & 2.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 1.4 \\ 1.5 \end{pmatrix},$$

thereby obtaining the desired answer

$$x = \begin{pmatrix} -1.4 \\ 2.2 \\ 0.6 \end{pmatrix}$$

by computing first  $x_3$ , then  $x_2$ , and finally  $x_1$ .

## Computing an LU decomposition

We have now shown that if we can create an LUP decomposition for a nonsingular matrix  $A$ , then forward and back substitution can solve the system  $Ax = b$  of linear equations. Now we show how to efficiently compute an LUP decomposition for  $A$ . We start with the case in which  $A$  is an  $n \times n$  nonsingular matrix and  $P$  is absent (or, equivalently,  $P = I_n$ ). In this case, we factor  $A = LU$ . We call the two matrices  $L$  and  $U$  an ***LU decomposition*** of  $A$ .

We use a process known as ***Gaussian elimination*** to create an LU decomposition. We start by subtracting multiples of the first equation from the other equations in order to remove the first variable from those equations. Then, we subtract multiples of the second equation from the third and subsequent equations so that now the first and second variables are removed from them. We continue this process until the system that remains has an upper-triangular form—in fact, it is the matrix  $U$ . The matrix  $L$  is made up of the row multipliers that cause variables to be eliminated.

Our algorithm to implement this strategy is recursive. We wish to construct an LU decomposition for an  $n \times n$  nonsingular matrix  $A$ . If  $n = 1$ , then we are done, since we can choose  $L = I_1$  and  $U = A$ . For  $n > 1$ , we break  $A$  into four parts:

$$\begin{aligned} A &= \left( \begin{array}{c|ccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array} \right) \\ &= \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix}, \end{aligned}$$

where  $v$  is a column  $(n - 1)$ -vector,  $w^T$  is a row  $(n - 1)$ -vector, and  $A'$  is an  $(n - 1) \times (n - 1)$  matrix. Then, using matrix algebra (verify the equations by

simply multiplying through), we can factor  $A$  as

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}. \end{aligned} \quad (28.8)$$

The 0s in the first and second matrices of equation (28.8) are row and column  $(n - 1)$ -vectors, respectively. The term  $vw^T/a_{11}$ , formed by taking the outer product of  $v$  and  $w$  and dividing each element of the result by  $a_{11}$ , is an  $(n - 1) \times (n - 1)$  matrix, which conforms in size to the matrix  $A'$  from which it is subtracted. The resulting  $(n - 1) \times (n - 1)$  matrix

$$A' - vw^T/a_{11} \quad (28.9)$$

is called the *Schur complement* of  $A$  with respect to  $a_{11}$ .

We claim that if  $A$  is nonsingular, then the Schur complement is nonsingular, too. Why? Suppose that the Schur complement, which is  $(n - 1) \times (n - 1)$ , is singular. Then by Theorem D.1, it has row rank strictly less than  $n - 1$ . Because the bottom  $n - 1$  entries in the first column of the matrix

$$\begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}$$

are all 0, the bottom  $n - 1$  rows of this matrix must have row rank strictly less than  $n - 1$ . The row rank of the entire matrix, therefore, is strictly less than  $n$ . Applying Exercise D.2-8 to equation (28.8),  $A$  has rank strictly less than  $n$ , and from Theorem D.1 we derive the contradiction that  $A$  is singular.

Because the Schur complement is nonsingular, we can now recursively find an LU decomposition for it. Let us say that

$$A' - vw^T/a_{11} = L'U',$$

where  $L'$  is unit lower-triangular and  $U'$  is upper-triangular. Then, using matrix algebra, we have

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & L'U' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & U' \end{pmatrix} \\ &= LU, \end{aligned}$$

thereby providing our LU decomposition. (Note that because  $L'$  is unit lower triangular, so is  $L$ , and because  $U'$  is upper-triangular, so is  $U$ .)

Of course, if  $a_{11} = 0$ , this method doesn't work, because it divides by 0. It also doesn't work if the upper leftmost entry of the Schur complement  $A' - vw^T/a_{11}$  is 0, since we divide by it in the next step of the recursion. The elements by which we divide during LU decomposition are called *pivots*, and they occupy the diagonal elements of the matrix  $U$ . The reason we include a permutation matrix  $P$  during LUP decomposition is that it allows us to avoid dividing by 0. When we use permutations to avoid division by 0 (or by small numbers, which would contribute to numerical instability), we are *pivoting*.

An important class of matrices for which LU decomposition always works correctly is the class of symmetric positive-definite matrices. Such matrices require no pivoting, and thus we can employ the recursive strategy outlined above without fear of dividing by 0. We shall prove this result, as well as several others, in Section 28.3.

Our code for LU decomposition of a matrix  $A$  follows the recursive strategy, except that an iteration loop replaces the recursion. (This transformation is a standard optimization for a “tail-recursive” procedure—one whose last operation is a recursive call to itself. See Problem 7-4.) It assumes that the attribute  $A.rows$  gives the dimension of  $A$ . We initialize the matrix  $U$  with 0s below the diagonal and matrix  $L$  with 1s on its diagonal and 0s above the diagonal.

#### LU-DECOMPOSITION( $A$ )

```

1   $n = A.rows$ 
2  let  $L$  and  $U$  be new  $n \times n$  matrices
3  initialize  $U$  with 0s below the diagonal
4  initialize  $L$  with 1s on the diagonal and 0s above the diagonal
5  for  $k = 1$  to  $n$ 
6     $u_{kk} = a_{kk}$ 
7    for  $i = k + 1$  to  $n$ 
8       $l_{ik} = a_{ik}/u_{kk}$            //  $l_{ik}$  holds  $v_i$ 
9       $u_{ki} = a_{ki}$              //  $u_{ki}$  holds  $w_i^T$ 
10   for  $i = k + 1$  to  $n$ 
11     for  $j = k + 1$  to  $n$ 
12        $a_{ij} = a_{ij} - l_{ik}u_{kj}$ 
13   return  $L$  and  $U$ 
```

The outer **for** loop beginning in line 5 iterates once for each recursive step. Within this loop, line 6 determines the pivot to be  $u_{kk} = a_{kk}$ . The **for** loop in lines 7–9 (which does not execute when  $k = n$ ), uses the  $v$  and  $w^T$  vectors to update  $L$  and  $U$ . Line 8 determines the elements of the  $v$  vector, storing  $v_i$  in  $l_{ik}$ , and line 9 computes the elements of the  $w^T$  vector, storing  $w_i^T$  in  $u_{ki}$ . Finally, lines 10–12 compute the elements of the Schur complement and store them back into the ma-

2	3	1	5
6	13	5	19
2	19	10	23
4	10	11	31

(a)

<b>2</b>	3	1	5
<b>3</b>	4	2	4
<b>1</b>	16	9	18
<b>2</b>	4	9	21

(b)

2	3	1	5
3	4	2	4
1	4	1	2
2	1	7	17

1

2	3	1	5
3	4	2	4
1	4	1	2
2	1	7	3

(d)

$$\left( \begin{array}{cccc} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{array} \right) = \left( \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 2 & 1 & 7 & 1 \end{array} \right) \left( \begin{array}{cccc} 2 & 3 & 1 & 5 \\ 0 & 4 & 2 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 \end{array} \right)$$

(e)

**Figure 28.1** The operation of LU-DECOMPOSITION. (a) The matrix  $A$ . (b) The element  $a_{11} = 2$  in the black circle is the pivot, the shaded column is  $v/a_{11}$ , and the shaded row is  $w^T$ . The elements of  $U$  computed thus far are above the horizontal line, and the elements of  $L$  are to the left of the vertical line. The Schur complement matrix  $A' - vw^T/a_{11}$  occupies the lower right. (c) We now operate on the Schur complement matrix produced from part (b). The element  $a_{22} = 4$  in the black circle is the pivot, and the shaded column and row are  $v/a_{22}$  and  $w^T$  (in the partitioning of the Schur complement), respectively. Lines divide the matrix into the elements of  $U$  computed so far (above), the elements of  $L$  computed so far (left), and the new Schur complement (lower right). (d) After the next step, the matrix  $A$  is factored. (The element 3 in the new Schur complement becomes part of  $U$  when the recursion terminates.) (e) The factorization  $A = L U$

trix  $A$ . (We don't need to divide by  $a_{kk}$  in line 12 because we already did so when we computed  $l_{ik}$  in line 8.) Because line 12 is triply nested, LU-DECOMPOSITION runs in time  $\Theta(n^3)$ .

Figure 28.1 illustrates the operation of LU-DECOMPOSITION. It shows a standard optimization of the procedure in which we store the significant elements of  $L$  and  $U$  in place in the matrix  $A$ . That is, we can set up a correspondence between each element  $a_{ij}$  and either  $l_{ij}$  (if  $i > j$ ) or  $u_{ij}$  (if  $i \leq j$ ) and update the matrix  $A$  so that it holds both  $L$  and  $U$  when the procedure terminates. To obtain the pseudocode for this optimization from the above pseudocode, just replace each reference to  $l$  or  $u$  by  $a$ ; you can easily verify that this transformation preserves correctness.

## Computing an LUP decomposition

Generally, in solving a system of linear equations  $Ax = b$ , we must pivot on off-diagonal elements of  $A$  to avoid dividing by 0. Dividing by 0 would, of course, be disastrous. But we also want to avoid dividing by a small value—even if  $A$  is

nonsingular—because numerical instabilities can result. We therefore try to pivot on a large value.

The mathematics behind LUP decomposition is similar to that of LU decomposition. Recall that we are given an  $n \times n$  nonsingular matrix  $A$ , and we wish to find a permutation matrix  $P$ , a unit lower-triangular matrix  $L$ , and an upper-triangular matrix  $U$  such that  $PA = LU$ . Before we partition the matrix  $A$ , as we did for LU decomposition, we move a nonzero element, say  $a_{k1}$ , from somewhere in the first column to the  $(1, 1)$  position of the matrix. For numerical stability, we choose  $a_{k1}$  as the element in the first column with the greatest absolute value. (The first column cannot contain only 0s, for then  $A$  would be singular, because its determinant would be 0, by Theorems D.4 and D.5.) In order to preserve the set of equations, we exchange row 1 with row  $k$ , which is equivalent to multiplying  $A$  by a permutation matrix  $Q$  on the left (Exercise D.1-4). Thus, we can write  $QA$  as

$$QA = \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix},$$

where  $v = (a_{21}, a_{31}, \dots, a_{n1})^T$ , except that  $a_{11}$  replaces  $a_{k1}$ ;  $w^T = (a_{k2}, a_{k3}, \dots, a_{kn})$ ; and  $A'$  is an  $(n-1) \times (n-1)$  matrix. Since  $a_{k1} \neq 0$ , we can now perform much the same linear algebra as for LU decomposition, but now guaranteeing that we do not divide by 0:

$$\begin{aligned} QA &= \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix}. \end{aligned}$$

As we saw for LU decomposition, if  $A$  is nonsingular, then the Schur complement  $A' - vw^T/a_{k1}$  is nonsingular, too. Therefore, we can recursively find an LUP decomposition for it, with unit lower-triangular matrix  $L'$ , upper-triangular matrix  $U'$ , and permutation matrix  $P'$ , such that

$$P'(A' - vw^T/a_{k1}) = L'U'.$$

Define

$$P = \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} Q,$$

which is a permutation matrix, since it is the product of two permutation matrices (Exercise D.1-4). We now have

$$\begin{aligned}
 PA &= \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} QA \\
 &= \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & P'(A' - vw^T/a_{k1}) \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & L'U' \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & L' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & U' \end{pmatrix} \\
 &= LU,
 \end{aligned}$$

yielding the LUP decomposition. Because  $L'$  is unit lower-triangular, so is  $L$ , and because  $U'$  is upper-triangular, so is  $U$ .

Notice that in this derivation, unlike the one for LU decomposition, we must multiply both the column vector  $v/a_{k1}$  and the Schur complement  $A' - vw^T/a_{k1}$  by the permutation matrix  $P'$ . Here is the pseudocode for LUP decomposition:

#### LUP-DECOMPOSITION( $A$ )

```

1   $n = A.\text{rows}$ 
2  let  $\pi[1..n]$  be a new array
3  for  $i = 1$  to  $n$ 
4     $\pi[i] = i$ 
5  for  $k = 1$  to  $n$ 
6     $p = 0$ 
7    for  $i = k$  to  $n$ 
8      if  $|a_{ik}| > p$ 
9         $p = |a_{ik}|$ 
10        $k' = i$ 
11     if  $p == 0$ 
12       error "singular matrix"
13     exchange  $\pi[k]$  with  $\pi[k']$ 
14     for  $i = 1$  to  $n$ 
15       exchange  $a_{ki}$  with  $a_{k'i}$ 
16     for  $i = k + 1$  to  $n$ 
17        $a_{ik} = a_{ik}/a_{kk}$ 
18       for  $j = k + 1$  to  $n$ 
19          $a_{ij} = a_{ij} - a_{ik}a_{kj}$ 

```

Like LU-DECOMPOSITION, our LUP-DECOMPOSITION procedure replaces the recursion with an iteration loop. As an improvement over a direct implementation of the recursion, we dynamically maintain the permutation matrix  $P$  as an array  $\pi$ , where  $\pi[i] = j$  means that the  $i$ th row of  $P$  contains a 1 in column  $j$ . We also implement the code to compute  $L$  and  $U$  “in place” in the matrix  $A$ . Thus, when the procedure terminates,

$$a_{ij} = \begin{cases} l_{ij} & \text{if } i > j, \\ u_{ij} & \text{if } i \leq j. \end{cases}$$

Figure 28.2 illustrates how LUP-DECOMPOSITION factors a matrix. Lines 3–4 initialize the array  $\pi$  to represent the identity permutation. The outer **for** loop beginning in line 5 implements the recursion. Each time through the outer loop, lines 6–10 determine the element  $a_{k'k}$  with largest absolute value of those in the current first column (column  $k$ ) of the  $(n - k + 1) \times (n - k + 1)$  matrix whose LUP decomposition we are finding. If all elements in the current first column are zero, lines 11–12 report that the matrix is singular. To pivot, we exchange  $\pi[k']$  with  $\pi[k]$  in line 13 and exchange the  $k$ th and  $k'$ th rows of  $A$  in lines 14–15, thereby making the pivot element  $a_{kk}$ . (The entire rows are swapped because in the derivation of the method above, not only is  $A' - \nu w^T/a_{k1}$  multiplied by  $P'$ , but so is  $\nu/a_{k1}$ .) Finally, the Schur complement is computed by lines 16–19 in much the same way as it is computed by lines 7–12 of LU-DECOMPOSITION, except that here the operation is written to work in place.

Because of its triply nested loop structure, LUP-DECOMPOSITION has a running time of  $\Theta(n^3)$ , which is the same as that of LU-DECOMPOSITION. Thus, pivoting costs us at most a constant factor in time.

## Exercises

### 28.1-1

Solve the equation

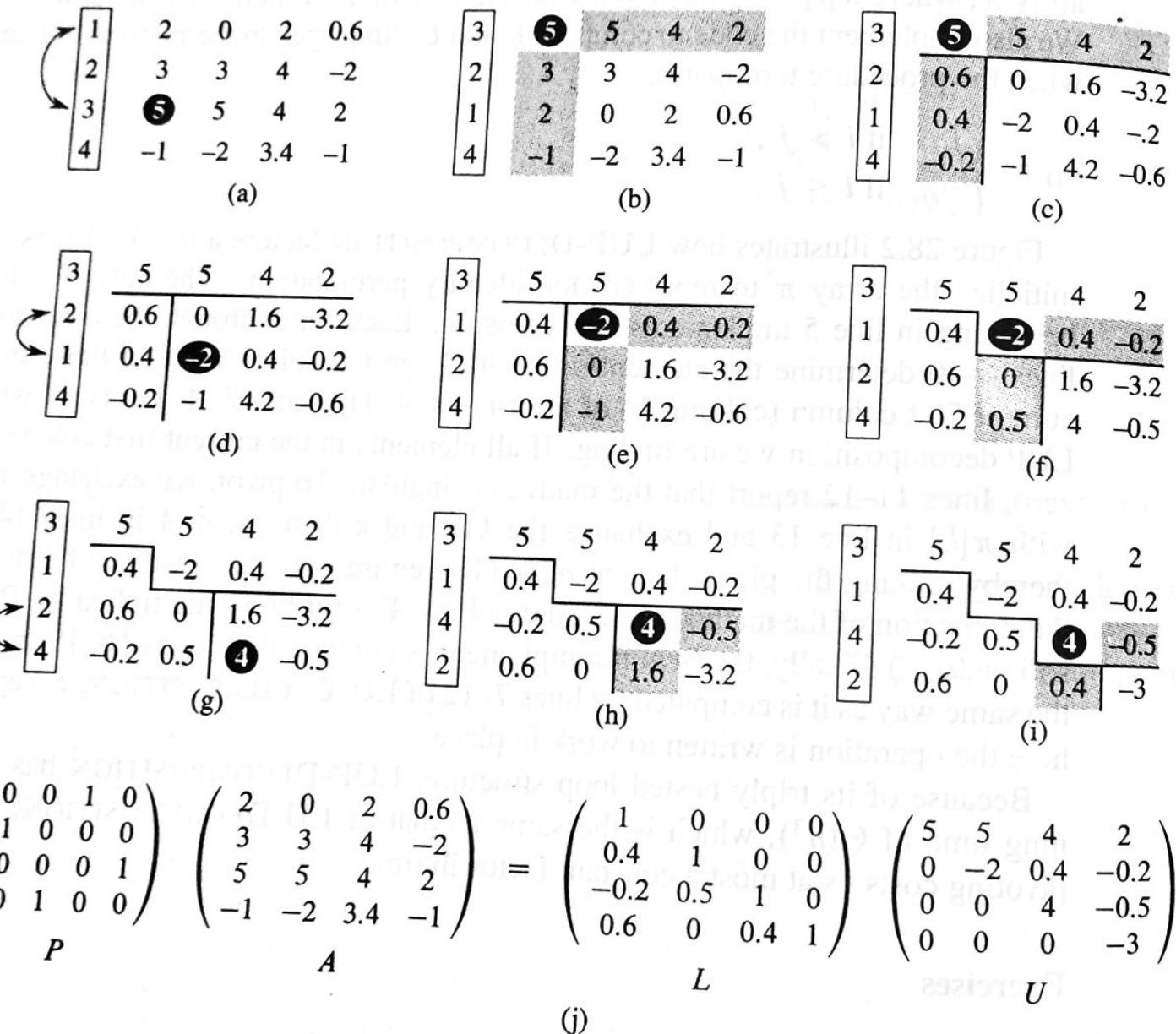
$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ -6 & 5 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 14 \\ -7 \end{pmatrix}$$

by using forward substitution.

### 28.1-2

Find an LU decomposition of the matrix

$$\begin{pmatrix} 4 & -5 & 6 \\ 8 & -6 & 7 \\ 12 & -7 & 12 \end{pmatrix}.$$



**Figure 28.2** The operation of LUP-DECOMPOSITION. (a) The input matrix  $A$  with the identity permutation of the rows on the left. The first step of the algorithm determines that the element 5 in the black circle in the third row is the pivot for the first column. (b) Rows 1 and 3 are swapped. The shaded column and row represent  $v$  and  $w^T$ . (c) The vector  $v$  divide the matrix into three regions: elements of  $U$  (above), elements of  $L$  (left), and elements of the Schur complement (lower right). (d)–(f) The second step. (g)–(i) The third step. No further changes occur on the fourth (final) step. (j) The LUP decomposition  $PA = LU$ .

**28.1-3**

Solve the equation

$$\begin{pmatrix} 1 & 5 & 4 \\ 2 & 0 & 3 \\ 5 & 8 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 9 \\ 5 \end{pmatrix}$$

by using an LUP decomposition.

**28.1-4**

Describe the LUP decomposition of a diagonal matrix.

**28.1-5**

Describe the LUP decomposition of a permutation matrix  $A$ , and prove that it is unique.

**28.1-6**

Show that for all  $n \geq 1$ , there exists a singular  $n \times n$  matrix that has an LU decomposition.

**28.1-7**

In LU-DECOMPOSITION, is it necessary to perform the outermost **for** loop iteration when  $k = n$ ? How about in LUP-DECOMPOSITION?

## 28.2 Inverting matrices

Although in practice we do not generally use matrix inverses to solve systems of linear equations, preferring instead to use more numerically stable techniques such as LUP decomposition, sometimes we need to compute a matrix inverse. In this section, we show how to use LUP decomposition to compute a matrix inverse. We also prove that matrix multiplication and computing the inverse of a matrix are equivalently hard problems, in that (subject to technical conditions) we can use an algorithm for one to solve the other in the same asymptotic running time. Thus, we can use Strassen's algorithm (see Section 4.2) for matrix multiplication to invert a matrix. Indeed, Strassen's original paper was motivated by the problem of showing that a set of linear equations could be solved more quickly than by the usual method.

### Computing a matrix inverse from an LUP decomposition

Suppose that we have an LUP decomposition of a matrix  $A$  in the form of three matrices  $L$ ,  $U$ , and  $P$  such that  $PA = LU$ . Using LUP-SOLVE, we can solve an equation of the form  $Ax = b$  in time  $\Theta(n^2)$ . Since the LUP decomposition depends on  $A$  but not  $b$ , we can run LUP-SOLVE on a second set of equations of the form  $Ax = b'$  in additional time  $\Theta(n^2)$ . In general, once we have the LUP decomposition of  $A$ , we can solve, in time  $\Theta(kn^2)$ ,  $k$  versions of the equation  $Ax = b$  that differ only in  $b$ .

We can think of the equation

$$AX = I_n, \quad (28.10)$$

which defines the matrix  $X$ , the inverse of  $A$ , as a set of  $n$  distinct equations of the form  $Ax = b$ . To be precise, let  $X_i$  denote the  $i$ th column of  $X$ , and recall that the unit vector  $e_i$  is the  $i$ th column of  $I_n$ . We can then solve equation (28.10) for  $X$  by using the LUP decomposition for  $A$  to solve each equation

$$AX_i = e_i$$

separately for  $X_i$ . Once we have the LUP decomposition, we can compute each of the  $n$  columns  $X_i$  in time  $\Theta(n^2)$ , and so we can compute  $X$  from the LUP decomposition of  $A$  in time  $\Theta(n^3)$ . Since we can determine the LUP decomposition of  $A$  in time  $\Theta(n^3)$ , we can compute the inverse  $A^{-1}$  of a matrix  $A$  in time  $\Theta(n^3)$ .

### Matrix multiplication and matrix inversion

We now show that the theoretical speedups obtained for matrix multiplication translate to speedups for matrix inversion. In fact, we prove something stronger: matrix inversion is equivalent to matrix multiplication, in the following sense. If  $M(n)$  denotes the time to multiply two  $n \times n$  matrices, then we can invert a nonsingular  $n \times n$  matrix in time  $O(M(n))$ . Moreover, if  $I(n)$  denotes the time to invert a nonsingular  $n \times n$  matrix, then we can multiply two  $n \times n$  matrices in time  $O(I(n))$ . We prove these results as two separate theorems.

#### **Theorem 28.1 (Multiplication is no harder than inversion)**

If we can invert an  $n \times n$  matrix in time  $I(n)$ , where  $I(n) = \Omega(n^2)$  and  $I(n)$  satisfies the regularity condition  $I(3n) = O(I(n))$ , then we can multiply two  $n \times n$  matrices in time  $O(I(n))$ .

**Proof** Let  $A$  and  $B$  be  $n \times n$  matrices whose matrix product  $C$  we wish to compute. We define the  $3n \times 3n$  matrix  $D$  by

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}.$$

The inverse of  $D$  is

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix},$$

and thus we can compute the product  $AB$  by taking the upper right  $n \times n$  submatrix of  $D^{-1}$ .

We can construct matrix  $D$  in  $\Theta(n^2)$  time, which is  $O(I(n))$  because we assume that  $I(n) = \Omega(n^2)$ , and we can invert  $D$  in  $O(I(3n)) = O(I(n))$  time, by the regularity condition on  $I(n)$ . We thus have  $M(n) = O(I(n))$ . ■

Note that  $I(n)$  satisfies the regularity condition whenever  $I(n) = \Theta(n^c \lg^d n)$  for any constants  $c > 0$  and  $d \geq 0$ .

The proof that matrix inversion is no harder than matrix multiplication relies on some properties of symmetric positive-definite matrices that we will prove in Section 28.3.

### **Theorem 28.2 (Inversion is no harder than multiplication)**

Suppose we can multiply two  $n \times n$  real matrices in time  $M(n)$ , where  $M(n) = \Omega(n^2)$  and  $M(n)$  satisfies the two regularity conditions  $M(n+k) = O(M(n))$  for any  $k$  in the range  $0 \leq k \leq n$  and  $M(n/2) \leq cM(n)$  for some constant  $c < 1/2$ . Then we can compute the inverse of any real nonsingular  $n \times n$  matrix in time  $O(M(n))$ .

**Proof** We prove the theorem here for real matrices. Exercise 28.2-6 asks you to generalize the proof for matrices whose entries are complex numbers.

We can assume that  $n$  is an exact power of 2, since we have

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

for any  $k > 0$ . Thus, by choosing  $k$  such that  $n+k$  is a power of 2, we enlarge the matrix to a size that is the next power of 2 and obtain the desired answer  $A^{-1}$  from the answer to the enlarged problem. The first regularity condition on  $M(n)$  ensures that this enlargement does not cause the running time to increase by more than a constant factor.

For the moment, let us assume that the  $n \times n$  matrix  $A$  is symmetric and positive-definite. We partition each of  $A$  and its inverse  $A^{-1}$  into four  $n/2 \times n/2$  submatrices: