28 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 to 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***. Because actual computers have limits to how precisely they can represent floating-point numbers, round-off errors in numerical computations may become amplified over the course of a computation, leading to incorrect results. Such computations are called ***numerically unstable***. Although we'll briefly consider numerical stability on occasion, we won't focus on it in this chapter. We refer you to the excellent book by Higham [216] for a thorough discussion of stability issues.

28.1 Solving systems of linear equations

Numerous applications need to solve sets of simultaneous linear equations. A linear system can be cast 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.

The process starts with a set of linear equations in n unknowns $x_1, x_2, \ldots, x_n$:

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

A ***solution*** to the equations (28.1) is a set of values for $x_1, x_2, \ldots, x_n$ 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.

Next, 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,

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

This section focuses on the case in which A is nonsingular or, equivalently (by Theorem D.1 on page 1220), the rank of A equals 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's return to the problem of solving the system $Ax = b$ of n equations in n unknowns. One option is to 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

$$PA = LU, \tag{28.4}$$

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'll show that every nonsingular matrix A possesses such a decomposition.

Computing an LUP decomposition for the matrix A has the advantage that linear systems can be efficiently solved when they are triangular, as is the case for both matrices L and U. If you have an LUP decomposition for A, you can solve equation (28.2), $Ax = b$, by solving only triangular linear systems, as follows. Multiply both sides of $Ax = b$ by P, yielding the equivalent equation $PAx = Pb$. By Exercise D.1-4 on page 1219, multiplying both sides by a permutation matrix amounts to permuting the equations (28.1). By the decomposition (28.4), substituting LU for PA gives

$$LUx = Pb .$$

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

$$Ly = Pb \tag{28.5}$$

for the unknown vector y by a method called "forward substitution." Having solved for y, solve the upper-triangular system

$$Ux = y \tag{28.6}$$

for the unknown x by a method called "back substitution." Why does this process solve $Ax = b$? Because the permutation matrix P is invertible (see Exercise D.2-3 on page 1223), multiplying both sides of equation (28.4) by P^{-1} gives $P^{-1}PA = P^{-1}LU$, so that

$$A = P^{-1}LU. \tag{28.7}$$

Hence, the vector x that satisfies $Ux = y$ is the solution to $Ax = b$:

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

The 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. An array $\pi[1:n]$ provides a more compact format to represent the permutation P than an $n \times n$ matrix that is mostly 0s. For $i = 1, 2, \ldots, n$, 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 ith element. Since L is unit lower-triangular, the matrix equation $Ly = Pb$ is equivalent to the n equations

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

The first equation gives $y_1 = b_{\pi[1]}$ directly. Knowing the value of y_1, you can substitute it into the second equation, yielding

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

Next, you 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, you substitute $y_1, y_2, \ldots, y_{i-1}$ "forward" into the ith equation to solve for y_i:

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

Once you've solved for y, you can solve for x in equation (28.6) using ***back substitution***, which is similar to forward substitution. This time, you solve the nth 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, the matrix equation $Ux = y$ is equivalent to the n equations

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

Thus, you can solve for $x_n, x_{n-1}, \ldots, x_1$ 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 on the next page solves for x by combining forward and back substitution. The permutation matrix P is represented by the array π. The procedure first solves for y using forward substitution in lines 2–3, and then it solves for x using backward substitution in lines 4–5. 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 $Ax = b$, where

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix} ,$$

```
LUP-SOLVE(L, U, π, b, n)
1  let x and y be new vectors of length n
2  for i = 1 to n
3       y_i = b_{π[i]} − Σ_{j=1}^{i−1} l_ij y_j
4  for i = n downto 1
5       x_i = (y_i − Σ_{j=i+1}^{n} u_ij x_j) / u_ii
6  return x
```

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

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.2 & 1 & 0 \\ 0.6 & 0.5 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 5 & 6 & 3 \\ 0 & 0.8 & -0.6 \\ 0 & 0 & 2.5 \end{pmatrix}, \quad \text{and} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

(You might want to verify that $PA = LU$.) Using forward substitution, 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. Then, using back substitution, 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

Given an LUP decomposition for a nonsingular matrix A, you can use forward and back substitution to solve the system $Ax = b$ of linear equations. Now let's see

how to efficiently compute an LUP decomposition for A. We start with the simpler case in which A is an $n \times n$ nonsingular matrix and P is absent (or, equivalently, $P = I_n$, the $n \times n$ identity matrix), so that $A = LU$. We call the two matrices L and U an ***LU decomposition*** of A.

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

To implement this strategy, let's start with a recursive formulation. The input is an $n \times n$ nonsingular matrix A. If $n = 1$, then nothing needs to be done: just choose $L = I_1$ and $U = A$. For $n > 1$, break A into four parts:

$$\begin{aligned} A &= \left(\begin{array}{c|ccc} a_{11} & a_{12} & \cdots & a_{1n} \\ \hline a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right) \\ &= \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ v & A' \end{pmatrix}, \end{aligned} \tag{28.8}$$

where $v = (a_{21}, a_{31}, \ldots, a_{n1})$ is a column $(n-1)$-vector, $w^{\mathrm{T}} = (a_{12}, a_{13}, \ldots, a_{1n})^{\mathrm{T}}$ 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), factor A as

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ v & A' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & A' - vw^{\mathrm{T}}/a_{11} \end{pmatrix}. \end{aligned} \tag{28.9}$$

The 0s in the first and second matrices of equation (28.9) are row and column $(n-1)$-vectors, respectively. The term vw^{T}/a_{11} is an $(n-1) \times (n-1)$ matrix formed by taking the outer product of v and w and dividing each element of the result by a_{11}. Thus it conforms in size to the matrix A' from which it is subtracted. The resulting $(n-1) \times (n-1)$ matrix

$$A' - vw^{\mathrm{T}}/a_{11} \tag{28.10}$$

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

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

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 on page 1223 to equation (28.9), 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, it, too, has an LU decomposition, which we can find recursively. Let's say that

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

where L' is unit lower-triangular and U' is upper-triangular. The LU decomposition of A is then $A = LU$, with

$$L = \begin{pmatrix} 1 & 0 \\ v/a_{11} & L' \end{pmatrix} \text{ and } U = \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & U' \end{pmatrix} ,$$

as shown by

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & A' - vw^{\mathrm{T}}/a_{11} \end{pmatrix} \quad \text{(by equation (28.9))} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & L'U' \end{pmatrix} \\ &= \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ v & vw^{\mathrm{T}}/a_{11} + L'U' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & U' \end{pmatrix} \\ &= LU. \end{aligned}$$

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 $A' - vw^{\mathrm{T}}/a_{11}$ is 0, since the next step of the recursion will divide by it. The denominators in each step of LU decomposition are called ***pivots***, and they occupy the diagonal elements of the matrix U. The permutation matrix P included in LUP decomposition provides a way to avoid dividing by 0, as we'll see below. Using permutations to avoid division by 0 (or by small numbers, which can contribute to numerical instability), is called ***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 to avoid dividing by 0 in the recursive strategy outlined above. We will prove this result, as well as several others, in Section 28.3.

The pseudocode in the procedure LU-DECOMPOSITION 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 7-5 on page 202.) The procedure initializes the matrix U with 0s below the diagonal and matrix L with 1s on its diagonal and 0s above the diagonal. Each iteration works on a square submatrix, using its upper leftmost element as the pivot to compute the v and w vectors and the Schur complement, which becomes the square submatrix worked on by the next iteration.

```
LU-DECOMPOSITION(A, n)
 1  let L and U be new n × n matrices
 2  initialize U with 0s below the diagonal
 3  initialize L with 1s on the diagonal and 0s above the diagonal
 4  for k = 1 to n
 5      u_kk = a_kk
 6      for i = k + 1 to n
 7          l_ik = a_ik / a_kk          // a_ik holds v_i
 8          u_ki = a_ki                 // a_ki holds w_i
 9      for i = k + 1 to n              // compute the Schur complement ...
10          for j = k + 1 to n
11              a_ij = a_ij − l_ik u_kj // ... and store it back into A
12  return L and U
```

Each recursive step in the description above takes place in one iteration of the outer **for** loop of lines 4–11. Within this loop, line 5 determines the pivot to be $u_{kk} = a_{kk}$. The **for** loop in lines 6–8 (which does not execute when $k = n$) uses the v and w vectors to update L and U. Line 7 determines the below-diagonal elements of L, storing v_i/a_{kk} in l_{ik}, and line 8 computes the above-diagonal elements of U, storing w_i in u_{ki}. Finally, lines 9–11 compute the elements of the Schur complement and store them back into the matrix A. (There is no need to divide by a_{kk} in line 11 because that already happened when line 7 computed l_{ik}.) Because line 11 is triply nested, LU-DECOMPOSITION runs in $\Theta(n^3)$ time.

Figure 28.1 illustrates the operation of LU-DECOMPOSITION. It shows a standard optimization of the procedure that stores the significant elements of L and U in place in the matrix A. Each element a_{ij} corresponds to either l_{ij} (if $i > j$) or u_{ij} (if $i \leq j$), so that the matrix A holds both L and U when the procedure terminates. To obtain the pseudocode for this optimization from the pseudocode for the LU-DECOMPOSITION procedure, just replace each reference to l or u by a. You can verify that this transformation preserves correctness.

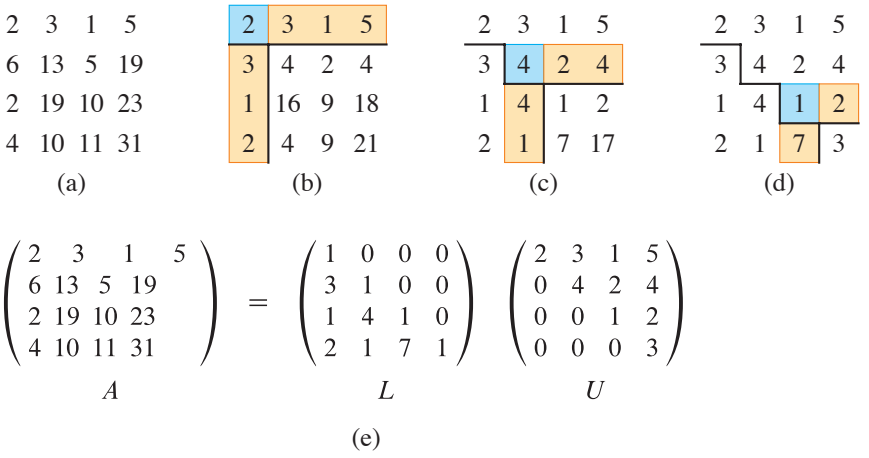

Figure 28.1 The operation of LU-DECOMPOSITION. **(a)** The matrix A. **(b)** The result of the first iteration of the outer **for** loop of lines 4–11. The element $a_{11} = 2$ highlighted in blue is the pivot, the tan column is v/a_{11}, and the tan row is w^{T}. The elements of U computed thus far are above the horizontal line, and the elements of L are to the left of the vertical line. The Schur complement matrix $A' - vw^{\mathrm{T}}/a_{11}$ occupies the lower right. **(c)** The result of the next iteration of the outer **for** loop, on the Schur complement matrix from part (b). The element $a_{22} = 4$ highlighted in blue is the pivot, and the tan 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 iteration, the matrix A is factored. The element 3 in the new Schur complement becomes part of U when the recursion terminates.) **(e)** The factorization $A = LU$.

Computing an LUP decomposition

If the diagonal of the matrix given to LU-DECOMPOSITION contains any 0s, then the procedure will attempt to divide by 0, which would cause disaster. Even if the diagonal contains no 0s, but does have numbers with small absolute values, dividing by such numbers can cause numerical instabilities. Therefore, LUP decomposition pivots on entries with the largest absolute values that it can find.

In LUP decomposition, the input is an $n \times n$ nonsingular matrix A, with a goal of finding a permutation matrix P, a unit lower-triangular matrix L, and an upper-triangular matrix U such that $PA = LU$. Before partitioning the matrix A, as LU decomposition does, LUP decomposition moves a nonzero element, say a_{k1}, from somewhere in the first column to the $(1, 1)$ position of the matrix. For the greatest numerical stability, LUP decomposition chooses the element in the first column with the greatest absolute value as a_{k1}. (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 on page 1221.) In order to preserve the set of equations, LUP decomposition exchanges row 1 with row k, which is equivalent to multiplying A by a

permutation matrix Q on the left (Exercise D.1-4 on page 1219). Thus, the analog to equation (28.8) expresses QA as

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

where $v = (a_{21}, a_{31}, \ldots, a_{n1})$, except that a_{11} replaces a_{k1}; $w^{\mathrm{T}} = (a_{k2}, a_{k3}, \ldots, a_{kn})^{\mathrm{T}}$; and A' is an $(n-1) \times (n-1)$ matrix. Since $a_{k1} \neq 0$, the analog to equation (28.9) guarantees no division by 0:

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

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

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

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 on page 1219). This definition of P gives

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

which yields 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, both the column vector v/a_{k1} and the Schur complement $A' - vw^{\mathrm{T}}/a_{k1}$ are multiplied by the permutation matrix P'. The procedure LUP-DECOMPOSITION gives the pseudocode for LUP decomposition.

```
LUP-DECOMPOSITION(A, n)
 1  let π[1:n] be a new array
 2  for i = 1 to n
 3      π[i] = i                          // initialize π to the identity permutation
 4  for k = 1 to n
 5      p = 0
 6      for i = k to n                    // find largest absolute value in column k
 7          if |a_ik| > p
 8              p = |a_ik|
 9              k' = i                    // row number of the largest found so far
10      if p == 0
11          error "singular matrix"
12      exchange π[k] with π[k']
13      for i = 1 to n
14          exchange a_ki with a_k'i      // exchange rows k and k'
15      for i = k + 1 to n
16          a_ik = a_ik/a_kk
17          for j = k + 1 to n
18              a_ij = a_ij − a_ik a_kj   // compute L and U in place in A
```

Like LU-DECOMPOSITION, the LUP-DECOMPOSITION procedure replaces the recursion with an iteration loop. As an improvement over a direct implementation of the recursion, the procedure dynamically maintains the permutation matrix P as an array π, where $\pi[i] = j$ means that the ith row of P contains a 1 in column j. The LUP-DECOMPOSITION procedure also implements the improvement mentioned earlier, computing L and U in place in the matrix A. Thus, when the procedure terminates,

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

Figure 28.2 illustrates how LUP-DECOMPOSITION factors a matrix. Lines 2–3 initialize the array π to represent the identity permutation. The outer **for** loop of lines 4–18 implements the recursion, finding an LUP decomposition of

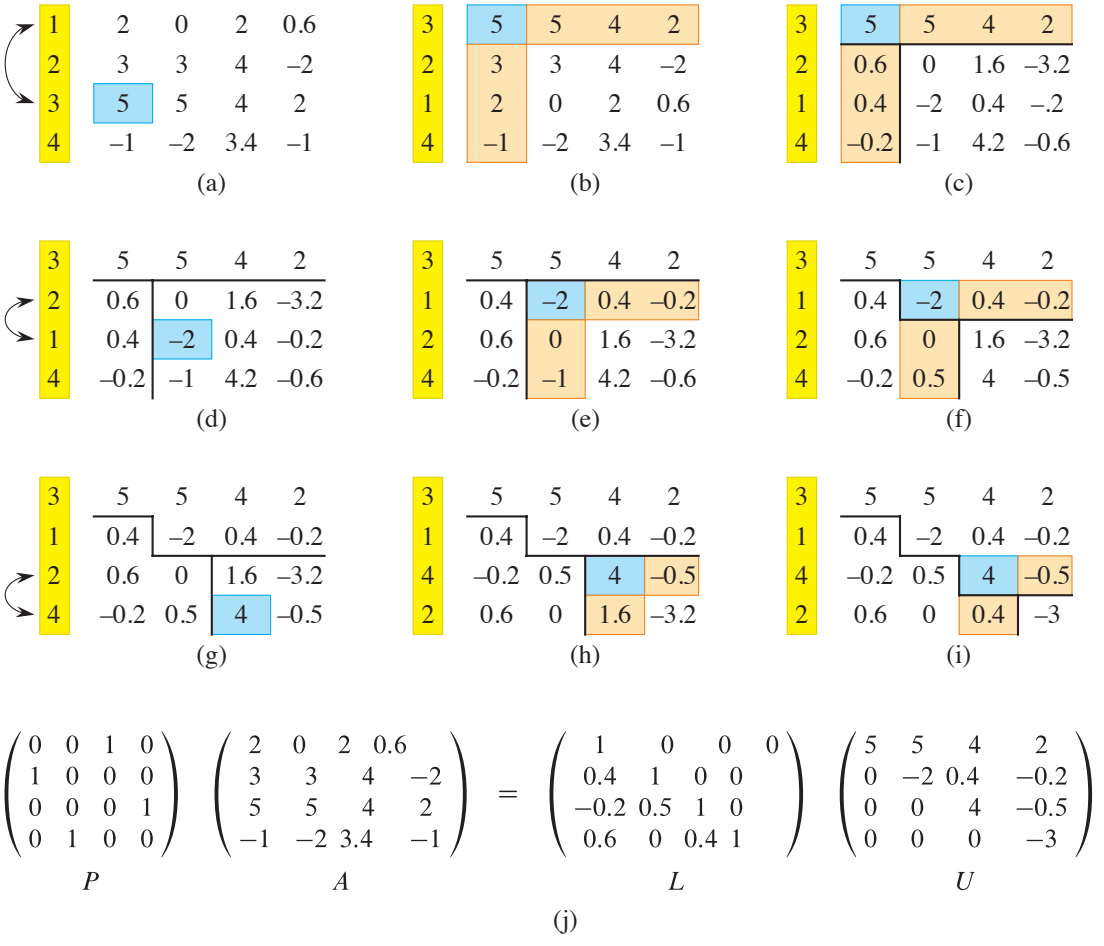

Figure 28.2 The operation of LUP-DECOMPOSITION. **(a)** The input matrix A with the identity permutation of the rows in yellow on the left. The first step of the algorithm determines that the element 5 highlighted in blue in the third row is the pivot for the first column. **(b)** Rows 1 and 3 are swapped and the permutation is updated. The tan column and row represent v and w^{T}. **(c)** The vector v is replaced by $v/5$, and the lower right of the matrix is updated with the Schur complement. Lines 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 $PA = LU$.

the $(n-k+1) \times (n-k+1)$ submatrix whose upper left is in row k and column k. Each time through the outer loop, lines 5–9 determine the element $a_{k'k}$ with the largest absolute value of those in the current first column (column k) of the $(n-k+1) \times (n-k+1)$ submatrix that the procedure is currently working on. If all elements in the current first column are 0, lines 10–11 report that the matrix is singular. To pivot, line 12 exchanges $\pi[k']$ with $\pi[k]$, and lines 13–14 exchange the kth and k'th rows of A, thereby making the pivot element a_{kk}. (The entire rows are swapped because in the derivation of the method above, not only is $A' - vw^{\mathrm{T}}/a_{k1}$ multiplied by P', but so is v/a_{k1}.) Finally, the Schur complement is computed by lines 15–18 in much the same way as it is computed by lines 6–11 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 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

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

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.