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.

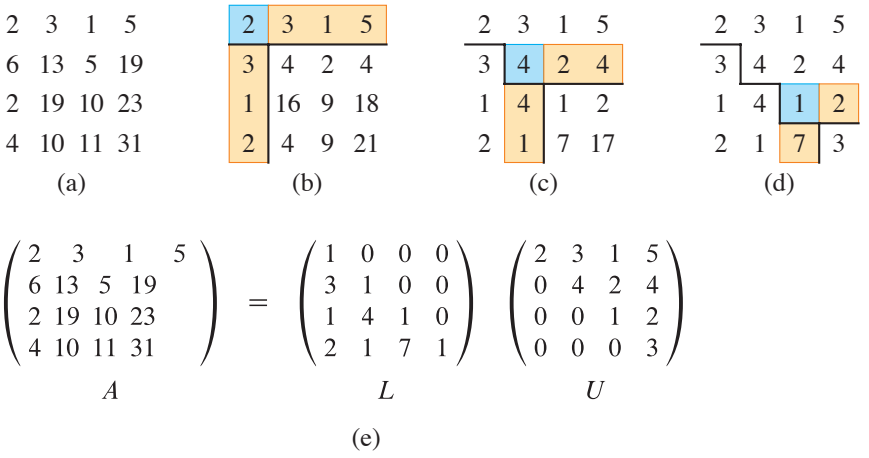

Figure 28.1 The operation of LU-DECOMPOSITION. **(a)** The matrix A. **(b)** The result of the first iteration of the outer **for** loop of lines 4–11. The element $a_{11} = 2$ highlighted in blue is the pivot, the tan column is v/a_{11}, and the tan row is w^{T}. The elements of U computed thus far 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

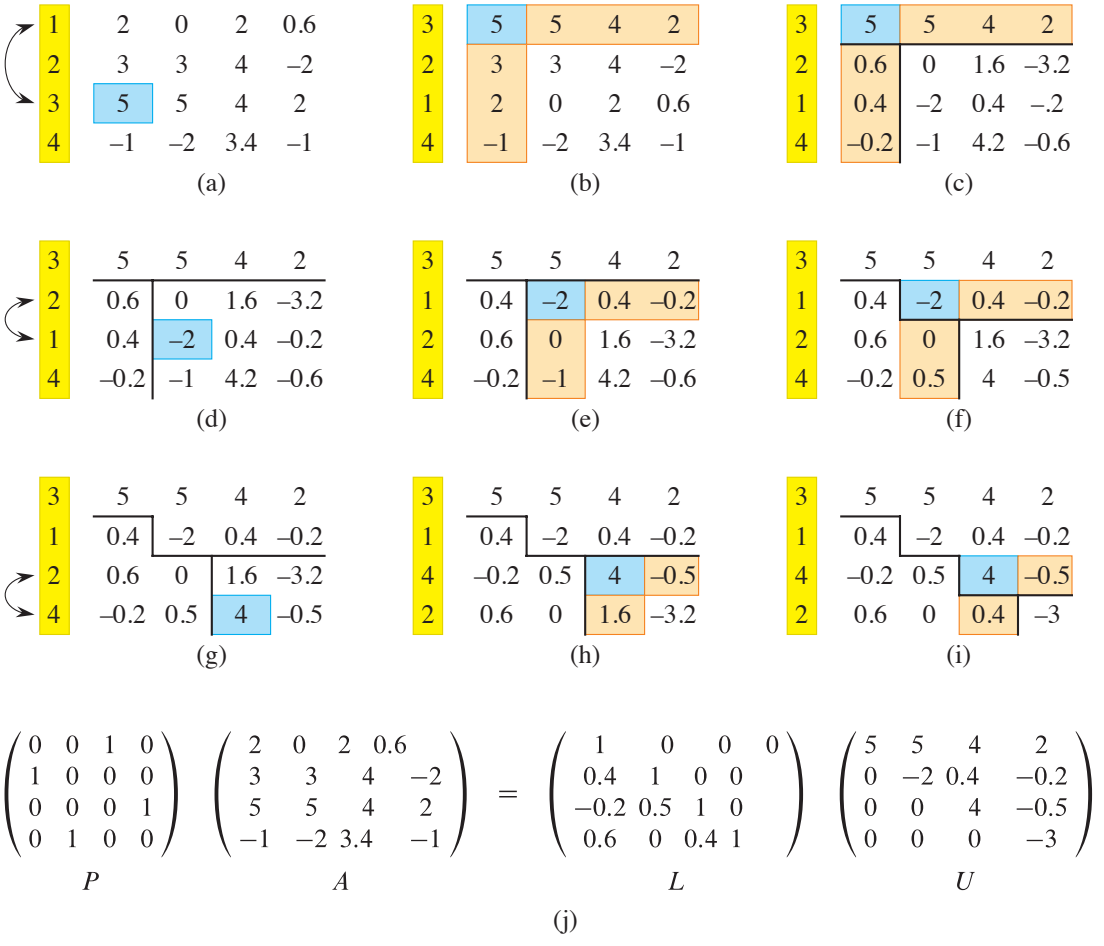

Figure 28.2 The operation of LUP-DECOMPOSITION. **(a)** The input matrix A with the identity permutation of the rows in yellow on the left. The first step of the algorithm determines that the element 5 highlighted in blue in the third row is the pivot for the first column. **(b)** Rows 1 and 3 are swapped 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.

28.1-5
Describe the LUP decomposition of a permutation matrix, 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 you can use equation (28.3) to solve a system of linear equations by computing a matrix inverse, in practice you are better off using more numerically stable techniques, such as LUP decomposition. Sometimes, however, you really do need to compute a matrix inverse. This section shows how to use LUP decomposition to compute a matrix inverse. It also proves that matrix multiplication and computing the inverse of a matrix are equivalently hard problems, in that (subject to technical conditions) an algorithm for one can solve the other in the same asymptotic running time. Thus, you 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 idea that a set of a linear equations could be solved more quickly than by the usual method.

Computing a matrix inverse from an LUP decomposition

Suppose that you 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, you can solve an equation of the form $Ax = b$ in $\Theta(n^2)$ time. Since the LUP decomposition depends on A but not b, you can run LUP-SOLVE on a second set of equations of the form $Ax = b'$ in $\Theta(n^2)$ additional time. In general, once you have the LUP decomposition of A, you can solve, in $\Theta(kn^2)$ time, k versions of the equation $Ax = b$ that differ only in the vector b.

Let's think of the equation

$$AX = I_n \,, \tag{28.11}$$

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 ith column of X, and recall that the

unit vector e_i is the ith column of I_n. You can then solve equation (28.11) for X by using the LUP decomposition for A to solve each equation

$$AX_i = e_i$$

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

Matrix multiplication and matrix inversion

Now let's see how the theoretical speedups obtained for matrix multiplication translate to speedups for matrix inversion. In fact, we'll 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 a nonsingular $n \times n$ matrix can be inverted in $O(M(n))$ time. Moreover, if $I(n)$ denotes the time to invert a nonsingular $n \times n$ matrix, then two $n \times n$ matrices can be multiplied in $O(I(n))$ time. We prove these results as two separate theorems.

Theorem 28.1 (Multiplication is no harder than inversion)
If an $n \times n$ matrix can be inverted in $I(n)$ time, where $I(n) = \Omega(n^2)$ and $I(n)$ satisfies the regularity condition $I(3n) = O(I(n))$, then two $n \times n$ matrices can be multiplied in $O(I(n))$ time.

Proof Let A and B be $n \times n$ matrices. To compute their product $C = AB$, 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 to compute the product AB, just take the upper right $n \times n$ submatrix of D^{-1}.

Constructing matrix D takes $\Theta(n^2)$ time, which is $O(I(n))$ from the assumption that $I(n) = \Omega(n^2)$, and inverting D takes $O(I(3n)) = O(I(n))$ 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 proved in Section 28.3.

Theorem 28.2 (Inversion is no harder than multiplication)
Suppose that two $n \times n$ real matrices can be multiplied in $M(n)$ time, where $M(n) = \Omega(n^2)$ and $M(n)$ satisfies the following two regularity conditions:

1. $M(n+k) = O(M(n))$ for any k in the range $0 \le k < n$, and
2. $M(n/2) \le cM(n)$ for some constant $c < 1/2$.

Then the inverse of any real nonsingular $n \times n$ matrix can be computed in $O(M(n))$ time.

Proof Let A be an $n \times n$ matrix with real-valued entries that is nonsingular. Assume that n is an exact power of 2 (i.e., $n = 2^l$ for some integer l); we'll see at the end of the proof what to do if n is not an exact power of 2.

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

$$A = \begin{pmatrix} B & C^{\mathrm{T}} \\ C & D \end{pmatrix} \text{ and } A^{-1} = \begin{pmatrix} R & T \\ U & V \end{pmatrix}. \tag{28.12}$$

Then, if we let

$$S = D - CB^{-1}C^{\mathrm{T}} \tag{28.13}$$

be the Schur complement of A with respect to B (we'll see more about this form of Schur complement in Section 28.3), we have

$$A^{-1} = \begin{pmatrix} R & T \\ U & V \end{pmatrix} = \begin{pmatrix} B^{-1} + B^{-1}C^{\mathrm{T}}S^{-1}CB^{-1} & -B^{-1}C^{\mathrm{T}}S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix}, \tag{28.14}$$

since $AA^{-1} = I_n$, as you can verify by performing the matrix multiplication. Because A is symmetric and positive-definite, Lemmas 28.4 and 28.5 in Section 28.3 imply that B and S are both symmetric and positive-definite. By Lemma 28.3 in Section 28.3, therefore, the inverses B^{-1} and S^{-1} exist, and by Exercise D.2-6 on page 1223, B^{-1} and S^{-1} are symmetric, so that $(B^{-1})^{\mathrm{T}} = B^{-1}$ and $(S^{-1})^{\mathrm{T}} = S^{-1}$. Therefore, to compute the submatrices

$$\begin{aligned} R &= B^{-1} + B^{-1}C^{\mathrm{T}}S^{-1}CB^{-1}, \\ T &= -B^{-1}C^{\mathrm{T}}S^{-1}, \\ U &= -S^{-1}CB^{-1}, \text{ and} \\ V &= S^{-1} \end{aligned}$$

of A^{-1}, do the following, where all matrices mentioned are $n/2 \times n/2$:

1. Form the submatrices B, C, C^{T}, and D of A.
2. Recursively compute the inverse B^{-1} of B.
3. Compute the matrix product $W = CB^{-1}$, and then compute its transpose W^{T}, which equals $B^{-1}C^{\mathrm{T}}$ (by Exercise D.1-2 on page 1219 and $(B^{-1})^{\mathrm{T}} = B^{-1}$).
4. Compute the matrix product $X = WC^{\mathrm{T}}$, which equals $CB^{-1}C^{\mathrm{T}}$, and then compute the matrix $S = D - X = D - CB^{-1}C^{\mathrm{T}}$.
5. Recursively compute the inverse S^{-1} of S.
6. Compute the matrix product $Y = S^{-1}W$, which equals $S^{-1}CB^{-1}$, and then compute its transpose Y^{T}, which equals $B^{-1}C^{\mathrm{T}}S^{-1}$ (by Exercise D.1-2, $(B^{-1})^{\mathrm{T}} = B^{-1}$, and $(S^{-1})^{\mathrm{T}} = S^{-1}$).
7. Compute the matrix product $Z = W^{\mathrm{T}}Y$, which equals $B^{-1}C^{\mathrm{T}}S^{-1}CB^{-1}$.
8. Set $R = B^{-1} + Z$.
9. Set $T = -Y^{\mathrm{T}}$.
10. Set $U = -Y$.
11. Set $V = S^{-1}$.

Thus, to invert an $n \times n$ symmetric positive-definite matrix, invert two $n/2 \times n/2$ matrices in steps 2 and 5; perform four multiplications of $n/2 \times n/2$ matrices in steps 3, 4, 6, and 7; plus incur an additional cost of $O(n^2)$ for extracting submatrices from A, inserting submatrices into A^{-1}, and performing a constant number of additions, subtractions, and transposes on $n/2 \times n/2$ matrices. The running time is given by the recurrence

$$\begin{aligned} I(n) &\le 2I(n/2) + 4M(n/2) + O(n^2) \\ &= 2I(n/2) + \Theta(M(n)) \\ &= O(M(n)) \,. \end{aligned} \tag{28.15}$$

The second line follows from the assumption that $M(n) = \Omega(n^2)$ and from the second regularity condition in the statement of the theorem, which implies that $4M(n/2) < 2M(n)$. Because $M(n) = \Omega(n^2)$, case 3 of the master theorem (Theorem 4.1) applies to the recurrence (28.15), giving the $O(M(n))$ result.

It remains to prove how to obtain the same asymptotic running time for matrix multiplication as for matrix inversion when A is invertible but not symmetric and positive-definite. The basic idea is that for any nonsingular matrix A, the matrix $A^{\mathrm{T}}A$ is symmetric (by Exercise D.1-2) and positive-definite (by Theorem D.6 on page 1222). The trick, then, is to reduce the problem of inverting A to the problem of inverting $A^{\mathrm{T}}A$.

The reduction is based on the observation that when A is an $n \times n$ nonsingular matrix, we have

$$A^{-1} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}} ,$$

since $((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}})A = (A^{\mathrm{T}}A)^{-1}(A^{\mathrm{T}}A) = I_n$ and a matrix inverse is unique. Therefore, to compute A^{-1}, first multiply A^{T} by A to obtain $A^{\mathrm{T}}A$, then invert the symmetric positive-definite matrix $A^{\mathrm{T}}A$ using the above divide-and-conquer algorithm, and finally multiply the result by A^{T}. Each of these three steps takes $O(M(n))$ time, and thus any nonsingular matrix with real entries can be inverted in $O(M(n))$ time.

The above proof assumed that A is an $n \times n$ matrix, where n is an exact power of 2. If n is not an exact power of 2, then let $k < n$ be such that $n + k$ is an exact power of 2, and define the $(n + k) \times (n + k)$ matrix A' as

$$A' = \begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix} .$$

Then the inverse of A' is

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

Apply the method of the proof to A' to compute the inverse of A', and take the first n rows and n columns of the result as the desired answer A^{-1}. The first regularity condition on $M(n)$ ensures that enlarging the matrix in this way increases the running time by at most a constant factor. ■

The proof of Theorem 28.2 suggests how to solve the equation $Ax = b$ by using LU decomposition without pivoting, so long as A is nonsingular. Let $y = A^{\mathrm{T}}b$. Multiply both sides of the equation $Ax = b$ by A^{T}, yielding $(A^{\mathrm{T}}A)x = A^{\mathrm{T}}b = y$. This transformation doesn't affect the solution x, since A^{T} is invertible. Because $A^{\mathrm{T}}A$ is symmetric positive-definite, it can be factored by computing an LU decomposition. Then, use forward and back substitution to solve for x in the equation $(A^{\mathrm{T}}A)x = y$. Although this method is theoretically correct, in practice the procedure LUP-DECOMPOSITION works much better. LUP decomposition requires fewer arithmetic operations by a constant factor, and it has somewhat better numerical properties.

Exercises

28.2-1

Let $M(n)$ be the time to multiply two $n \times n$ matrices, and let $S(n)$ denote the time required to square an $n \times n$ matrix. Show that multiplying and squaring matrices have essentially the same difficulty: an $M(n)$-time matrix-multiplication algorithm implies an $O(M(n))$-time squaring algorithm, and an $S(n)$-time squaring algorithm implies an $O(S(n))$-time matrix-multiplication algorithm.

28.2-2
Let $M(n)$ be the time to multiply two $n \times n$ matrices. Show that an $M(n)$-time matrix-multiplication algorithm implies an $O(M(n))$-time LUP-decomposition algorithm. (The LUP decomposition your method produces need not be the same as the result produced by the LUP-DECOMPOSITION procedure.)

28.2-3
Let $M(n)$ be the time to multiply two $n \times n$ boolean matrices, and let $T(n)$ be the time to find the transitive closure of an $n \times n$ boolean matrix. (See Section 23.2.) Show that an $M(n)$-time boolean matrix-multiplication algorithm implies an $O(M(n) \lg n)$-time transitive-closure algorithm, and a $T(n)$-time transitive-closure algorithm implies an $O(T(n))$-time boolean matrix-multiplication algorithm.

28.2-4
Does the matrix-inversion algorithm based on Theorem 28.2 work when matrix elements are drawn from the field of integers modulo 2? Explain.

★ ***28.2-5***
Generalize the matrix-inversion algorithm of Theorem 28.2 to handle matrices of complex numbers, and prove that your generalization works correctly. (*Hint:* Instead of the transpose of A, use the ***conjugate transpose*** A^*, which you obtain from the transpose of A by replacing every entry with its complex conjugate. Instead of symmetric matrices, consider ***Hermitian*** matrices, which are matrices A such that $A = A^*$.)

28.3 Symmetric positive-definite matrices and least-squares approximation

Symmetric positive-definite matrices have many interesting and desirable properties. An $n \times n$ matrix A is ***symmetric positive-definite*** if $A = A^{\mathrm{T}}$ (A is symmetric) and $x^{\mathrm{T}}Ax > 0$ for all n-vectors $x \neq 0$ (A is positive-definite). Symmetric positive-definite matrices are nonsingular, and an LU decomposition on them will not divide by 0. This section proves these and several other important properties of symmetric positive-definite matrices. We'll also see an interesting application to curve fitting by a least-squares approximation.

The first property we prove is perhaps the most basic.

Lemma 28.3
Any positive-definite matrix is nonsingular.

Proof Suppose that a matrix A is singular. Then by Corollary D.3 on page 1221, there exists a nonzero vector x such that $Ax = 0$. Hence, $x^{\mathrm{T}}Ax = 0$, and A cannot be positive-definite. ■

The proof that an LU decomposition on a symmetric positive-definite matrix A won't divide by 0 is more involved. We begin by proving properties about certain submatrices of A. Define the kth ***leading submatrix*** of A to be the matrix A_k consisting of the intersection of the first k rows and first k columns of A.

Lemma 28.4
If A is a symmetric positive-definite matrix, then every leading submatrix of A is symmetric and positive-definite.

Proof Since A is symmetric, each leading submatrix A_k is also symmetric. We'll prove that A_k is positive-definite by contradiction. If A_k is not positive-definite, then there exists a k-vector $x_k \neq 0$ such that $x_k^{\mathrm{T}} A_k x_k \leq 0$. Let A be $n \times n$, and

$$A = \begin{pmatrix} A_k & B^{\mathrm{T}} \\ B & C \end{pmatrix} \tag{28.16}$$

for submatrices B (which is $(n-k)\times k$) and C (which is $(n-k)\times(n-k)$). Define the n-vector $x = (\, x_k^{\mathrm{T}} \quad 0\,)^{\mathrm{T}}$, where $n-k$ 0s follow x_k. Then we have

$$\begin{aligned} x^{\mathrm{T}}Ax &= (\, x_k^{\mathrm{T}} \quad 0\,)\begin{pmatrix} A_k & B^{\mathrm{T}} \\ B & C \end{pmatrix}\begin{pmatrix} x_k \\ 0 \end{pmatrix} \\ &= (\, x_k^{\mathrm{T}} \quad 0\,)\begin{pmatrix} A_k x_k \\ B x_k \end{pmatrix} \\ &= x_k^{\mathrm{T}} A_k x_k \\ &\leq 0\,, \end{aligned}$$

which contradicts A being positive-definite. ■

We now turn to some essential properties of the Schur complement. Let A be a symmetric positive-definite matrix, and let A_k be a leading $k \times k$ submatrix of A. Partition A once again according to equation (28.16). Equation (28.10) generalizes to define the ***Schur complement*** S of A with respect to A_k as

$$S = C - BA_k^{-1}B^{\mathrm{T}}\,. \tag{28.17}$$

(By Lemma 28.4, A_k is symmetric and positive-definite, and therefore, A_k^{-1} exists by Lemma 28.3, and S is well defined.) The earlier definition (28.10) of the Schur complement is consistent with equation (28.17) by letting $k = 1$.

The next lemma shows that the Schur-complement matrices of symmetric positive-definite matrices are themselves symmetric and positive-definite. We used this

result in Theorem 28.2, and its corollary will help prove that LU decomposition works for symmetric positive-definite matrices.

Lemma 28.5 (Schur complement lemma)
If A is a symmetric positive-definite matrix and A_k is a leading $k \times k$ submatrix of A, then the Schur complement S of A with respect to A_k is symmetric and positive-definite.

Proof Because A is symmetric, so is the submatrix C. By Exercise D.2-6 on page 1223, the product $BA_k^{-1}B^{\mathrm{T}}$ is symmetric. Since C and $BA_k^{-1}B^{\mathrm{T}}$ are symmetric, then by Exercise D.1-1 on page 1219, so is S.

It remains to show that S is positive-definite. Consider the partition of A given in equation (28.16). For any nonzero vector x, we have $x^{\mathrm{T}}Ax > 0$ by the assumption that A is positive-definite. Let the subvectors y and z consist of the first k and last $n - k$ elements in x, respectively, and thus they are compatible with A_k and C, respectively. Because A_k^{-1} exists, we have

$$\begin{aligned}
x^{\mathrm{T}}Ax &= \begin{pmatrix} y^{\mathrm{T}} & z^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} A_k & B^{\mathrm{T}} \\ B & C \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} \\
&= \begin{pmatrix} y^{\mathrm{T}} & z^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} A_k y + B^{\mathrm{T}} z \\ By + Cz \end{pmatrix} \\
&= y^{\mathrm{T}} A_k y + y^{\mathrm{T}} B^{\mathrm{T}} z + z^{\mathrm{T}} By + z^{\mathrm{T}} Cz \\
&= (y + A_k^{-1}B^{\mathrm{T}}z)^{\mathrm{T}} A_k (y + A_k^{-1}B^{\mathrm{T}}z) + z^{\mathrm{T}}(C - BA_k^{-1}B^{\mathrm{T}})z \,, \qquad (28.18)
\end{aligned}$$

This last equation, which you can verify by multiplying through, amounts to "completing the square" of the quadratic form. (See Exercise 28.3-2.)

Since $x^{\mathrm{T}}Ax > 0$ holds for any nonzero x, pick any nonzero z and then choose $y = -A_k^{-1}B^{\mathrm{T}}z$, which causes the first term in equation (28.18) to vanish, leaving

$$z^{\mathrm{T}}(C - BA_k^{-1}B^{\mathrm{T}})z = z^{\mathrm{T}}Sz$$

as the value of the expression. For any $z \neq 0$, we therefore have $z^{\mathrm{T}}Sz = x^{\mathrm{T}}Ax > 0$, and thus S is positive-definite. ∎

Corollary 28.6
LU decomposition of a symmetric positive-definite matrix never causes a division by 0.

Proof Let A be an $n \times n$ symmetric positive-definite matrix. In fact, we'll prove a stronger result than the statement of the corollary: every pivot is strictly positive. The first pivot is a_{11}. Let e_1 be the length-n unit vector $(1\ 0\ 0\ \cdots\ 0)^{\mathrm{T}}$, so that $a_{11} = e_1^{\mathrm{T}}Ae_1$, which is positive because e_1 is nonzero and A is positive

definite. Since the first step of LU decomposition produces the Schur complement of A with respect to $A_1 = (a_{11})$, Lemma 28.5 implies by induction that all pivots are positive. ■

Least-squares approximation

One important application of symmetric positive-definite matrices arises in fitting curves to given sets of data points. You are given a set of m data points

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m) ,$$

where you know that the y_i are subject to measurement errors. You wish to determine a function $F(x)$ such that the approximation errors

$$\eta_i = F(x_i) - y_i \tag{28.19}$$

are small for $i = 1, 2, \ldots, m$. The form of the function F depends on the problem at hand. Let's assume that it has the form of a linearly weighted sum

$$F(x) = \sum_{j=1}^{n} c_j f_j(x) ,$$

where the number n of summands and the specific ***basis functions*** f_j are chosen based on knowledge of the problem at hand. A common choice is $f_j(x) = x^{j-1}$, which means that

$$F(x) = c_1 + c_2 x + c_3 x^2 + \cdots + c_n x^{n-1}$$

is a polynomial of degree $n - 1$ in x. Thus, if you are given m data points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, you need to calculate n coefficients $c_1, c_2, \ldots, c_n$ that minimize the approximation errors $\eta_1, \eta_2, \ldots, \eta_m$.

By choosing $n = m$, you can calculate each y_i *exactly* in equation (28.19). Such a high-degree polynomial F "fits the noise" as well as the data, however, and generally gives poor results when used to predict y for previously unseen values of x. It is usually better to choose n significantly smaller than m and hope that by choosing the coefficients c_j well, you can obtain a function F that finds the significant patterns in the data points without paying undue attention to the noise. Some theoretical principles exist for choosing n, but they are beyond the scope of this text. In any case, once you choose a value of n that is less than m, you end up with an overdetermined set of equations whose solution you wish to approximate. Let's see how to do so.

Let

$$A = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \dots & f_n(x_m) \end{pmatrix}$$

denote the matrix of values of the basis functions at the given points, that is, $a_{ij} = f_j(x_i)$. Let $c = (c_k)$ denote the desired n-vector of coefficients. Then,

$$\begin{aligned} Ac &= \begin{pmatrix} f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \dots & f_n(x_m) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \\ &= \begin{pmatrix} F(x_1) \\ F(x_2) \\ \vdots \\ F(x_m) \end{pmatrix} \end{aligned}$$

is the m-vector of "predicted values" for y. Thus,

$$\eta = Ac - y$$

is the m-vector of ***approximation errors***.

To minimize approximation errors, let's minimize the norm of the error vector η, which gives a ***least-squares solution***, since

$$\|\eta\| = \left(\sum_{i=1}^{m} \eta_i^2 \right)^{1/2} .$$

Because

$$\|\eta\|^2 = \|Ac - y\|^2 = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} c_j - y_i \right)^2 ,$$

to minimize $\|\eta\|$, differentiate $\|\eta\|^2$ with respect to each c_k and then set the result to 0:

$$\frac{d \|\eta\|^2}{dc_k} = \sum_{i=1}^{m} 2 \left(\sum_{j=1}^{n} a_{ij} c_j - y_i \right) a_{ik} = 0 . \tag{28.20}$$

The n equations (28.20) for $k = 1, 2, \dots, n$ are equivalent to the single matrix equation

$$(Ac - y)^{\mathrm{T}} A = 0$$

or, equivalently (using Exercise D.1-2 on page 1219), to

$$A^{\mathrm{T}}(Ac - y) = 0 \,,$$

which implies

$$A^{\mathrm{T}}Ac = A^{\mathrm{T}}y \,. \tag{28.21}$$

In statistics, equation (28.21) is called the ***normal equation***. The matrix $A^{\mathrm{T}}A$ is symmetric by Exercise D.1-2, and if A has full column rank, then by Theorem D.6 on page 1222, $A^{\mathrm{T}}A$ is positive-definite as well. Hence, $(A^{\mathrm{T}}A)^{-1}$ exists, and the solution to equation (28.21) is

$$\begin{aligned} c &= \left((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}\right) y \\ &= A^{+}y \,, \end{aligned} \tag{28.22}$$

where the matrix $A^{+} = ((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}})$ is the ***pseudoinverse*** of the matrix A. The pseudoinverse naturally generalizes the notion of a matrix inverse to the case in which A is not square. (Compare equation (28.22) as the approximate solution to $Ac = y$ with the solution $A^{-1}b$ as the exact solution to $Ax = b$.)

As an example of producing a least-squares fit, suppose that you have five data points

$$\begin{aligned} (x_1, y_1) &= (-1, 2) \,, \\ (x_2, y_2) &= (1, 1) \,, \\ (x_3, y_3) &= (2, 1) \,, \\ (x_4, y_4) &= (3, 0) \,, \\ (x_5, y_5) &= (5, 3) \,, \end{aligned}$$

shown as orange dots in Figure 28.3, and you want to fit these points with a quadratic polynomial

$$F(x) = c_1 + c_2 x + c_3 x^2 \,.$$

Start with the matrix of basis-function values

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 5 & 25 \end{pmatrix} ,$$

whose pseudoinverse is

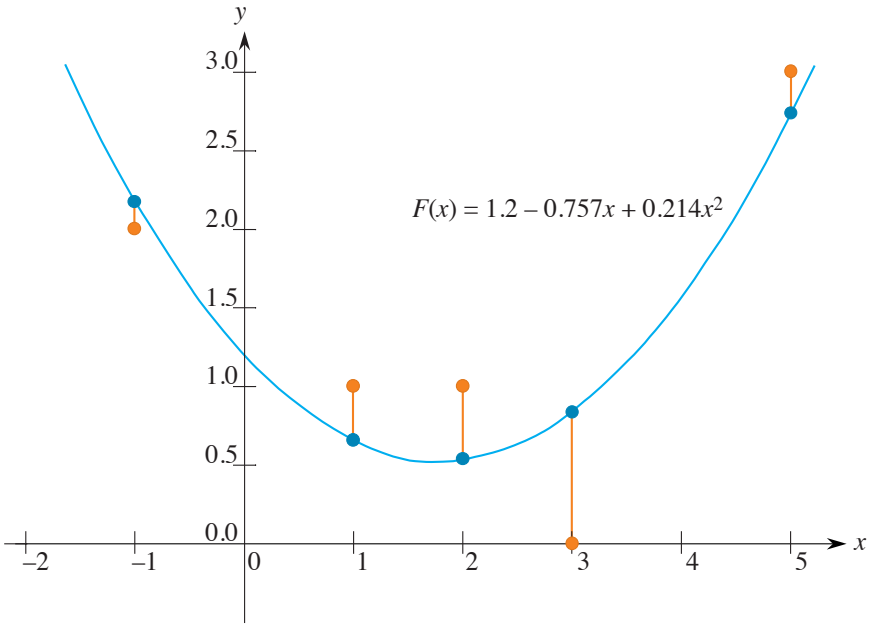

Figure 28.3 The least-squares fit of a quadratic polynomial to the set of five data points $\{(-1, 2), (1, 1), (2, 1), (3, 0), (5, 3)\}$. The orange dots are the data points, and the blue dots are their estimated values predicted by the polynomial $F(x) = 1.2 - 0.757x + 0.214x^2$, the quadratic polynomial that minimizes the sum of the squared errors, plotted in blue. Each orange line shows the error for one data point.

$$A^+ = \begin{pmatrix} 0.500 & 0.300 & 0.200 & 0.100 & -0.100 \\ -0.388 & 0.093 & 0.190 & 0.193 & -0.088 \\ 0.060 & -0.036 & -0.048 & -0.036 & 0.060 \end{pmatrix} .$$

Multiplying y by A^+ gives the coefficient vector

$$c = \begin{pmatrix} 1.200 \\ -0.757 \\ 0.214 \end{pmatrix} ,$$

which corresponds to the quadratic polynomial

$$F(x) = 1.200 - 0.757x + 0.214x^2$$

as the closest-fitting quadratic to the given data, in a least-squares sense.

As a practical matter, you would typically solve the normal equation (28.21) by multiplying y by A^{T} and then finding an LU decomposition of $A^{\mathrm{T}}A$. If A has full rank, the matrix $A^{\mathrm{T}}A$ is guaranteed to be nonsingular, because it is symmetric and positive-definite. (See Exercise D.1-2 and Theorem D.6.)

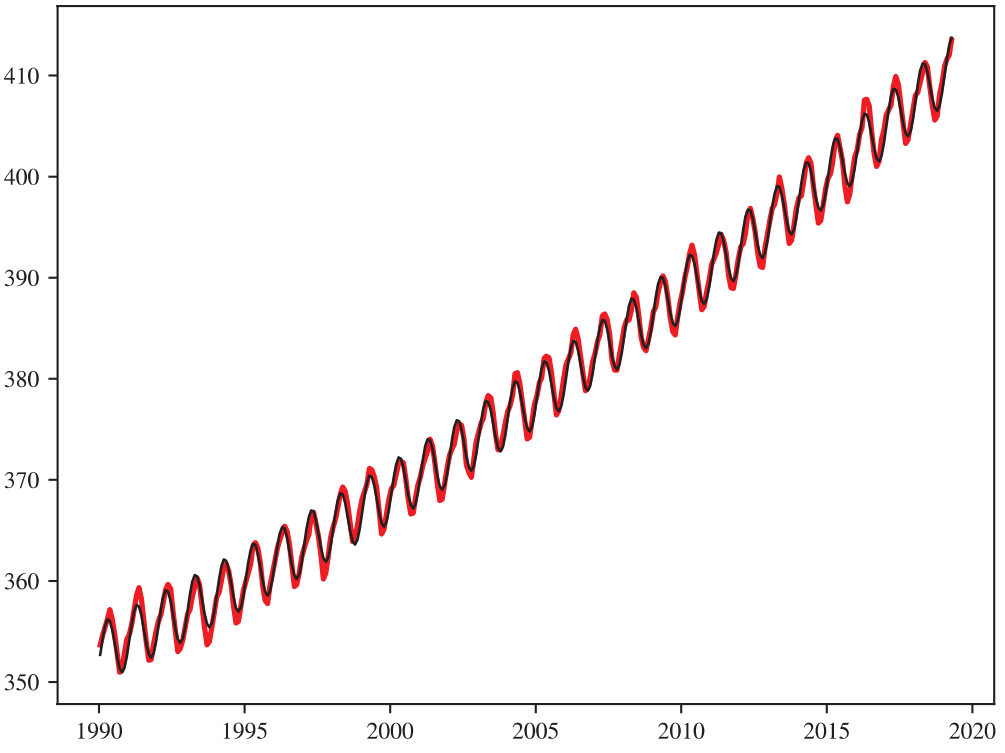

Figure 28.4 A least-squares fit of a curve of the form

$$c_1 + c_2 x + c_3 x^2 + c_4 \sin(2\pi x) + c_5 \cos(2\pi x)$$

for the carbon-dioxide concentrations measured in Mauna Loa, Hawaii from 1990[1] to 2019, where x is the number of years elapsed since 1990. This curve is the famous "Keeling curve," illustrating curve-fitting to nonpolynomial formulas. The sine and cosine terms allow modeling of seasonal variations in CO_2 concentrations. The red curve shows the measured CO_2 concentrations. The best fit, shown in black, has the form

$$352.83 + 1.39x + 0.02x^2 + 2.83 \sin(2\pi x) - 0.94 \cos(2\pi x) \; .$$

We close this section with an example in Figure 28.4, illustrating that a curve can also fit a nonpolynomial function. The curve confirms one aspect of climate change: that carbon dioxide (CO_2) concentrations have steadily increased over a period of 29 years. Linear and quadratic terms model the annual increase, and sine and cosine terms model seasonal variations.

[1] The year in which *Introduction to Algorithms* was first published.

Exercises

28.3-1
Prove that every diagonal element of a symmetric positive-definite matrix is positive.

28.3-2
Let $A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$ be a 2×2 symmetric positive-definite matrix. Prove that its determinant $ac - b^2$ is positive by "completing the square" in a manner similar to that used in the proof of Lemma 28.5.

28.3-3
Prove that the maximum element in a symmetric positive-definite matrix lies on the diagonal.

28.3-4
Prove that the determinant of each leading submatrix of a symmetric positive-definite matrix is positive.

28.3-5
Let A_k denote the kth leading submatrix of a symmetric positive-definite matrix A. Prove that $\det(A_k)/\det(A_{k-1})$ is the kth pivot during LU decomposition, where, by convention, $\det(A_0) = 1$.

28.3-6
Find the function of the form

$$F(x) = c_1 + c_2 x \lg x + c_3 e^x$$

that is the best least-squares fit to the data points

$$(1, 1), (2, 1), (3, 3), (4, 8) .$$

28.3-7
Show that the pseudoinverse A^+ satisfies the following four equations:

$$\begin{aligned} AA^+A &= A , \\ A^+AA^+ &= A^+ , \\ (AA^+)^{\mathrm{T}} &= AA^+ , \\ (A^+A)^{\mathrm{T}} &= A^+A . \end{aligned}$$

Problems

28-1 Tridiagonal systems of linear equations

Consider the tridiagonal matrix

$$A = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}.$$

a. Find an LU decomposition of A.

b. Solve the equation $Ax = (1\ 1\ 1\ 1\ 1)^{\mathrm{T}}$ by using forward and back substitution.

c. Find the inverse of A.

d. Show how to solve the equation $Ax = b$ for any $n \times n$ symmetric positive-definite, tridiagonal matrix A and any n-vector b in $O(n)$ time by performing an LU decomposition. Argue that any method based on forming A^{-1} is asymptotically more expensive in the worst case.

e. Show how to solve the equation $Ax = b$ for any $n \times n$ nonsingular, tridiagonal matrix A and any n-vector b in $O(n)$ time by performing an LUP decomposition.

28-2 Splines

A practical method for interpolating a set of points with a curve is to use ***cubic splines***. You are given a set $\{(x_i, y_i) : i = 0, 1, \ldots, n\}$ of $n + 1$ point-value pairs, where $x_0 < x_1 < \cdots < x_n$. Your goal is to fit a piecewise-cubic curve (spline) $f(x)$ to the points. That is, the curve $f(x)$ is made up of n cubic polynomials $f_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$ for $i = 0, 1, \ldots, n-1$, where if x falls in the range $x_i \le x \le x_{i+1}$, then the value of the curve is given by $f(x) = f_i(x - x_i)$. The points x_i at which the cubic polynomials are "pasted" together are called ***knots***. For simplicity, assume that $x_i = i$ for $i = 0, 1, \ldots, n$.

To ensure continuity of $f(x)$, require that

$$f(x_i) = f_i(0) = y_i ,$$
$$f(x_{i+1}) = f_i(1) = y_{i+1}$$

for $i = 0, 1, \ldots, n-1$. To ensure that $f(x)$ is sufficiently smooth, also require the first derivative to be continuous at each knot:

$$f'(x_{i+1}) = f_i'(1) = f_{i+1}'(0)$$

for $i = 0, 1, \ldots, n-2$.

a. Suppose that for $i = 0, 1, \ldots, n$, in addition to the point-value pairs $\{(x_i, y_i)\}$, you are also given the first derivative $D_i = f'(x_i)$ at each knot. Express each coefficient a_i, b_i, c_i, and d_i in terms of the values y_i, y_{i+1}, D_i, and D_{i+1}. (Remember that $x_i = i$.) How quickly can you compute the $4n$ coefficients from the point-value pairs and first derivatives?

The question remains of how to choose the first derivatives of $f(x)$ at the knots. One method is to require the second derivatives to be continuous at the knots:

$$f''(x_{i+1}) = f_i''(1) = f_{i+1}''(0)$$

for $i = 0, 1, \ldots, n-2$. At the first and last knots, assume that $f''(x_0) = f_0''(0) = 0$ and $f''(x_n) = f_{n-1}''(1) = 0$. These assumptions make $f(x)$ a ***natural*** cubic spline.

b. Use the continuity constraints on the second derivative to show that for $i = 1, 2, \ldots, n-1$,

$$D_{i-1} + 4D_i + D_{i+1} = 3(y_{i+1} - y_{i-1}) \,. \tag{28.23}$$

c. Show that

$$2D_0 + D_1 = 3(y_1 - y_0) \,, \tag{28.24}$$

$$D_{n-1} + 2D_n = 3(y_n - y_{n-1}) \,. \tag{28.25}$$

d. Rewrite equations (28.23)–(28.25) as a matrix equation involving the vector $D = (\, D_0 \;\; D_1 \;\; D_2 \;\; \cdots \;\; D_n \,)^{\mathrm{T}}$ of unknowns. What attributes does the matrix in your equation have?

e. Argue that a natural cubic spline can interpolate a set of $n + 1$ point-value pairs in $O(n)$ time (see Problem 28-1).

f. Show how to determine a natural cubic spline that interpolates a set of $n + 1$ points (x_i, y_i) satisfying $x_0 < x_1 < \cdots < x_n$, even when x_i is not necessarily equal to i. What matrix equation must your method solve, and how quickly does your algorithm run?

Chapter notes

Many excellent texts describe numerical and scientific computation in much greater detail than we have room for here. The following are especially readable: George

and Liu [180], Golub and Van Loan [192], Press, Teukolsky, Vetterling, and Flannery [365, 366], and Strang [422, 423].

Golub and Van Loan [192] discuss numerical stability. They show why $\det(A)$ is not necessarily a good indicator of the stability of a matrix A, proposing instead to use $\|A\|_\infty \|A^{-1}\|_\infty$, where $\|A\|_\infty = \max\left\{\sum_{j=1}^{n} |a_{ij}| : 1 \le i \le n\right\}$. They also address the question of how to compute this value without actually computing A^{-1}.

Gaussian elimination, upon which the LU and LUP decompositions are based, was the first systematic method for solving linear systems of equations. It was also one of the earliest numerical algorithms. Although it was known earlier, its discovery is commonly attributed to C. F. Gauss (1777–1855). In his famous paper [424], Strassen showed that an $n \times n$ matrix can be inverted in $O(n^{\lg 7})$ time. Winograd [460] originally proved that matrix multiplication is no harder than matrix inversion, and the converse is due to Aho, Hopcroft, and Ullman [5].

Another important matrix decomposition is the ***singular value decomposition***, or ***SVD***. The SVD factors an $m \times n$ matrix A into $A = Q_1 \Sigma Q_2^{\mathrm{T}}$, where Σ is an $m \times n$ matrix with nonzero values only on the diagonal, Q_1 is $m \times m$ with mutually orthonormal columns, and Q_2 is $n \times n$, also with mutually orthonormal columns. Two vectors are ***orthonormal*** if their inner product is 0 and each vector has a norm of 1. The books by Strang [422, 423] and Golub and Van Loan [192] contain good treatments of the SVD.

Strang [423] has an excellent presentation of symmetric positive-definite matrices and of linear algebra in general.