Chapter 11

Gaussian Elimination and the *LU* Decomposition

You should be familiar with

- Matrix arithmetic
- Elementary row operations
- Gaussian elimination using an augmented matrix
- Upper- and lower-triangular matrices

In Chapter 2, we presented the process of solving a nonsingular linear system $Ax = b$ using Gaussian elimination. We formed the augmented matrix $A\,|b$ and applied the elementary row operations

1. Multiplying a row by a scalar.
2. Subtracting a multiple of one row from another
3. Exchanging two rows

to reduce A to upper-triangular form. Following this step, back substitution computed the solution. In many applications where linear systems appear, one needs to solve $Ax = b$ for many different vectors b. For instance, suppose a truss must be analyzed under several different loads. The matrix remains the same, but the right-hand side changes with each new load. Most of the work in Gaussian elimination is applying row operations to arrive at the upper-triangular matrix. If we need to solve several different systems with the same A, then we would like to avoid repeating the steps of Gaussian elimination on A for every different b. This can be accomplished by the *LU decomposition*, which in effect records the steps of Gaussian elimination.

Since Gaussian elimination is used so often, the algorithm must be stable. Unfortunately, this is not true, and we must add an operation termed partial pivoting. There are very rare cases when even the enhanced algorithm is still not stable. The solution to a linear system is actually much more difficult and interesting than it appeared to be in earlier chapters.

While the results of Gaussian elimination are normally very good, it is possible that a simple technique termed iterative improvement can help produce even better results.

11.1 *LU* DECOMPOSITION

The main idea of the *LU* decomposition is to record the steps used in Gaussian elimination with A in the places that would normally become zero. Consider the matrix:

$$A = \begin{bmatrix} 1 & -1 & 3 \\ 2 & -3 & 1 \\ 3 & 2 & 1 \end{bmatrix}.$$

The first step of Gaussian elimination is to use $a_{11} = 1$ as the pivot and subtract 2 times the first row from the second and 3 times the first row from the third. Record these actions by placing the multipliers 2 and 3 into the entries they made zero. In order to make it clear that we are recording multipliers and not elements of A, put the entries in parentheses. This leads to:

$$\begin{bmatrix} 1 & -1 & 3 \\ (2) & -1 & -5 \\ (3) & 5 & -8 \end{bmatrix}.$$

Numerical Linear Algebra with Applications. http://dx.doi.org/10.1016/B978-0-12-394435-1.00011-9

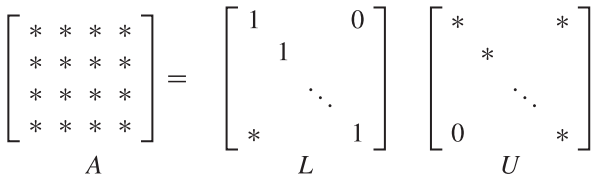

FIGURE 11.1 *LU* decomposition of a matrix.

To zero-out the element in the third row, second column, the pivot is -1, and we need to subtract -5 times the second row from the third row. Record the -5 in the spot made zero.

$$\begin{bmatrix} 1 & -1 & 3 \\ (2) & -1 & -5 \\ (3) & (-5) & -33 \end{bmatrix}.$$

Let U be the upper-triangular matrix produced by Gaussian elimination and L be the lower-triangular matrix with the multipliers and ones on the diagonal, i.e.,

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -5 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & -1 & 3 \\ 0 & -1 & -5 \\ 0 & 0 & -33 \end{bmatrix}.$$

Now form the product of L and U:

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -5 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 3 \\ 0 & -1 & -5 \\ 0 & 0 & -33 \end{bmatrix} = A.$$

Thus, we see that A is the product of the lower triangular L and the upper triangular U. When a matrix can be written as a product of simpler matrices, we call that a *decomposition* and this one we call the *LU decomposition* (Figure 11.1). We will explain why this works in Section 11.4.

Remark 11.1. As the elimination process continues, the pivots are on the diagonal of U.

11.2 USING *LU* TO SOLVE EQUATIONS

Factor A into the product of L and U:

$$\begin{aligned} Ax &= b, \\ (LU)\,x &= b, \\ L\,(Ux) &= b. \end{aligned}$$

First solve $Ly = b$. This finds $y = Ux$. Now solve $Ux = y$ to find x. Each of these solution steps is simple. First, the system Ly is lower triangular.

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \dots & 0 \\ \vdots & \vdots & 1 & 0 \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}.$$

Solve for y using forward substitution.

$$y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right).$$

Next, the system $Ux = y$ is upper triangular.

$$\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1,n-1} & u_{1n} \\ 0 & u_{22} & \dots & u_{2,n-1} & u_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & \dots & 0 & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}.$$

Solve for x using back substitution.

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

Example 11.1. Let A be the matrix $A = \begin{bmatrix} 1 & -1 & 3 \\ 2 & -3 & 1 \\ 3 & 2 & 1 \end{bmatrix}$ of Section 11.1, with $b = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$. We determined that $A = LU$, where

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -5 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & -1 & 3 \\ 0 & -1 & -5 \\ 0 & 0 & -33 \end{bmatrix}.$$

Execute forward substitution to solve $Ly = b$:

$$\begin{aligned} (1)\, y_1 &= 1, & y_1 &= 1, \\ 2\,(1) + (1)\, y_2 &= 3, & y_2 &= 1, \\ (3)\,(1) - 5\,(1) + (1)\, y_3 &= 1, & y_3 &= 3. \end{aligned}$$

Execute back substitution to solve $Ux = y$:

$$\begin{aligned} -33x_3 &= 3, & x_3 &= -1/11, \\ -x_2 - 5\,(-1/11) &= 1, & x_2 &= -6/11, \\ x_1 - (-6/11) + 3\,(-1/11) &= 1, & x_1 &= 8/11. \end{aligned}$$

Solution: $x_1 = 8/11,\ x_2 = -6/11,\ x_3 = -1/11$ ■

Example 11.2. Solve $\begin{bmatrix} 1 & 2 & -1 \\ 2 & 3 & 2 \\ 5 & 1 & 4 \end{bmatrix} x = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$. First factor A into the product LU.

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 3 & 2 \\ 5 & 1 & 4 \end{bmatrix} \xrightarrow[R_3 = R_3 - (5)\,R_1]{R_2 = R_2 - (2)\,R_1} \begin{bmatrix} 1 & 2 & -1 \\ (2) & -1 & 4 \\ (5) & -9 & 9 \end{bmatrix} \xrightarrow{R_3 = R_3 - (9)R_2} \begin{bmatrix} 1 & 2 & -1 \\ (2) & -1 & 4 \\ (5) & (9) & -27 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 5 & 9 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 2 & -1 \\ 0 & -1 & 4 \\ 0 & 0 & -27 \end{bmatrix}$$

Forward substitution:

$$\begin{aligned} y_1 &= 1, \\ y_2 &= -1 - 2\,(1) = -3, \\ y_3 &= 2 - 5\,(1) - 9\,(-3) = 24. \end{aligned}$$

Back substitution:

$$\begin{aligned} x_3 &= -24/27 = -8/9, \\ x_2 &= 3 - 4\,(8/9) = -5/9, \\ x_1 &= 1 - 2\,(-5/9) - 8/9 = 11/9. \end{aligned}$$

■

11.3 ELEMENTARY ROW MATRICES

Sections 11.1 and 11.2 describe the LU decomposition using examples. If a mathematical analysis of why the LU decomposition works is not required, the reader can skip this section and most of Section 11.4. However, it is recommended that Example 11.8 and Sections 11.4.1–11.4.3 be read.

The *elementary row matrices* are an important class of nonsingular matrices. Multiplication by one of these matrices performs an elementary row operation, and these matrices help us understand why the LU decomposition works.

Definition 11.1. To each of the three elementary row operations, there corresponds an *elementary row matrix* E_{ij}, E_i, and $E_{ij}(t)$:

a. E_{ij}, $i \neq j$, is obtained from the identity matrix I by exchanging rows i and j.
b. $E_i(t)$, $t \neq 0$ is obtained by multiplying the ith row of I by t.
c. $E_{ij}(t)$ $i \neq j$, is obtained from I by subtracting t times the jth row of I from the ith row of I.

Example 11.3. $E_{23} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$, $E_2(-1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $E_{23}(-2) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$ ■

The elementary row matrices have the following property, and it is this property that will allow us to explain why the LU decomposition works.

Theorem 11.1. *If an $n \times n$ matrix is premultiplied by an $n \times n$ elementary row matrix, the resulting $n \times n$ matrix is the one obtained by performing the corresponding elementary row-operation on A.*

Proof. We will prove that forming $C = E_{ij}A$ is equivalent to interchanging rows i and j of A. The remaining properties of the elementary row matrices is left to the exercises.

By the definition of matrix multiplication, row i of C has components

$$c_{ip} = \sum_{k=1}^{n} e_{ik} a_{kp}, \quad 1 \leq p \leq n.$$

Among the elements $\{e_{i1}, e_{i2}, \ldots, e_{in}\}$ only $e_{ij} = 1$ is nonzero. Thus, $c_{ip} = e_{ij}a_{jp} = a_{jp}$, $1 \leq p \leq n$, and elements of row i are those of row j. Similarly, the elements of row j are those of row i, and the other rows are unaffected. □

Example 11.4.

$$E_{23}\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} = \begin{bmatrix} a & b \\ e & f \\ c & d \end{bmatrix}.$$ ■

Theorem 11.1 implies that premultiplying a matrix by a sequence of elementary row matrices gives the same result as performing the sequence of elementary row operations to A.

Example 11.5. Let $A = \begin{bmatrix} 1 & 2 & 8 & 2 \\ 3 & 9 & -1 & 2 \\ -1 & 2 & 6 & 3 \\ 1 & 5 & 3 & 2 \end{bmatrix}$. Multiply A first by E_{24} and then by $E_{43}(-3)$.

$$E_{43}(-3)E_{24}A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 2 & 8 & 2 \\ 3 & 9 & -1 & 2 \\ -1 & 2 & 6 & 3 \\ 1 & 5 & 3 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix}\begin{bmatrix} 1 & 2 & 8 & 2 \\ 1 & 5 & 3 & 2 \\ -1 & 2 & 6 & 3 \\ 3 & 9 & -1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 8 & 2 \\ 1 & 5 & 3 & 2 \\ -1 & 2 & 6 & 3 \\ 0 & 15 & 17 & 11 \end{bmatrix}.$$

Verify that if you perform elementary row operations by interchanging rows 2 and 4 and then subtracting -3 times row 3 from row 4 you obtain the same result. ■

Theorem 11.2. *Elementary row matrices are nonsingular; in fact*

a. $E_{ij}^{-1} = E_{ij}$
b. $(E_i(t))^{-1} = E_i\left(t^{-1}\right),\ t \neq 0$
c. $\left(E_{ij}(t)\right)^{-1} = E_{ij}(-t)$

Proof. $E_{ij}E_{ij} = I$. Swapping rows i and j of I and then swapping again gives I.
$E_i(t)\,E_i(1/t) = I$, if $t \neq 0$.
$E_{ij}(-t)\,E_{ij}(t) = I$. Multiply row j by t and subtract from row i, then reverse the operation by multiplying row j by $-t$ and subtracting from row i. □

Example 11.6. Find the 3×3 matrix $A = E_3(5)\,E_{23}(2)\,E_{12}$ and then find A^{-1}.

$$\begin{aligned} A &= E_3(5)\,E_{23}(2)\,E_{12} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -2 \\ 0 & 0 & 5 \end{bmatrix}, \end{aligned}$$

$$\begin{aligned} A^{-1} &= (E_3(5)\,E_{23}(2)\,E_{12})^{-1} = E_{12}^{-1}\left(E_{23}(2)\right)^{-1}\left(E_3(5)\right)^{-1} \\ &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{5} \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 & \frac{2}{5} \\ 1 & 0 & 0 \\ 0 & 0 & \frac{1}{5} \end{bmatrix}. \end{aligned}$$

■

If B is row equivalent to A, it seems reasonable that we can invert row operations and row reduce B to A.

Theorem 11.3. *If B is row-equivalent to A, then A is row equivalent to B.*

Proof. If $B = E_kE_{k-1}\ldots E_2E_1A$, then

$$A = (E_kE_{k-1}\ldots E_2E_1)^{-1}B.$$

Since

$$E = E_kE_{k-1}\ldots E_2E_1$$

is nonsingular by Theorem 11.2, $A = E^{-1}B$. Now, $E^{-1} = E_1^{-1}E_2^{-1}\ldots E_{k-1}^{-1}E_k^{-1}$ is a product of elementary row matrices, so forming $E^{-1}B$ is equivalent to performing elementary row operations on B to obtain A. □

Theorems 11.4 and 11.5 tell us how elementary row matrices and nonsingular matrices are related.

Theorem 11.4. *Let A be a nonsingular $n \times n$ matrix. Then*

a. *A is row-equivalent to I.*
b. *A is a product of elementary row matrices.*

Proof. A sequence of elementary row operations will reduce A to I; otherwise, the system $Ax = 0$ would have a non-trivial solution.

Assume E_k, E_{k-1}, $\ldots E_2$, E_1 is the sequence of elementary row operations that reduces A to I so that $E_k E_{k-1} \ldots E_2 E_1 A = I$. It follows that $A = E_1^{-1} E_2^{-1} \cdots E_{k-1}^{-1} E_k^{-1}$ is a product of elementary row matrices. □

Theorem 11.5. *Let A be an $n \times n$ matrix and suppose that A is row-equivalent to I. Then A is nonsingular, and A^{-1} can be found by performing the same sequence of elementary row operations on I as were used to convert A to I.*

Proof. Suppose that $E_k E_{k-1} \ldots E_2 E_1 A = I$. Thus $BA = I$, where $B = E_k E_{k-1} \ldots E_2 E_1$ is nonsingular, and $A^{-1} = B = (E_k E_{k-1} \ldots E_2 E_1) I$, which shows that A^{-1} is obtained by performing the same sequence of elementary row operations on I that were used to transform A to I. □

Remark 11.2. Theorems 11.4 and 11.5 together imply that A is nonsingular if and only if it is row equivalent to I. This means that a singular matrix is row-equivalent to a matrix that has a zero row.

Example 11.7. Theorem 11.5 justifies the method we used in Chapter 2 for the computation of A^{-1}. If $A = \begin{bmatrix} 1 & 2 \\ -1 & 3 \end{bmatrix}$, find A^{-1} and express A as a product of elementary row matrices.
Attach I to A as a series of augmented columns, and apply a sequence of elementary row operations to A that reduce it to I. The attached matrix is A^{-1}.

$$\left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ -1 & 3 & 0 & 1 \end{array}\right] \xrightarrow{R_2 = R_2 - (-1) R_1} \left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 0 & 5 & 1 & 1 \end{array}\right] \xrightarrow{R_1 = R_1 - \frac{2}{5} R_2}$$

$$\left[\begin{array}{cc|cc} 1 & 0 & 3/5 & -2/5 \\ 0 & 5 & 1 & 1 \end{array}\right] \xrightarrow{R_2 = \frac{1}{5} R_2} \left[\begin{array}{cc|cc} 1 & 0 & 3/5 & -2/5 \\ 0 & 1 & 1/5 & 1/5 \end{array}\right]$$

A is row-equivalent to I, so A is nonsingular, and

$$A^{-1} = \begin{bmatrix} 3/5 & -2/5 \\ 1/5 & 1/5 \end{bmatrix}.$$

The sequence of elementary row matrices that correspond to the row reduction from A to A^{-1} is

$$E_2\left(\frac{1}{5}\right) E_{12}\left(\frac{2}{5}\right) E_{21}(-1).$$

Thus,

$$A^{-1} = E_2\left(\frac{1}{5}\right) E_{12}\left(\frac{2}{5}\right) E_{21}(-1),$$

so

$$A = E_{21}(1) E_{12}\left(-\frac{2}{5}\right) E_2(5) = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2/5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ -1 & 3 \end{bmatrix}.$$ ■

11.4 DERIVATION OF THE *LU* DECOMPOSITION

We can use the elementary row matrices to explain the LU decomposition. For now we will assume there are no row exchanges, but will add row exchanges later when we discuss Gaussian elimination with partial pivoting. We will also assume no multiplication of a row by a scalar, so all we will use is the elementary row matrix $E_{ij}(t)$ that adds a multiple of row j to row i.

As we perform row operations, matrix elements change, but we will not use a notation for it, such as $\overline{a_{ij}}$ and simply maintain the notation a_{ij}. Let's look at the row reduction to an upper-triangular matrix column by column. First, start with column 1, multiply row 1 by a_{21}/a_{11}, and subtract from row 2. This eliminates the element in row 2, column 1:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ a_{31} & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix} \xrightarrow{R2 = R2 - \left(\frac{a_{21}}{a_{11}}\right) R1} \begin{bmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ 0 & a_{22} & \dots & \dots & a_{2n} \\ a_{31} & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix}.$$

Recall that a_{11} is called the pivot element. Now, multiply row 1 by a_{31}/a_{11} and subtract from row 3, eliminating the element in row 3, column 1:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ 0 & a_{22} & \dots & \dots & a_{2n} \\ a_{31} & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix} \xrightarrow{R3 = R3 - \left(\frac{a_{31}}{a_{11}}\right) R1} \begin{bmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ 0 & a_{22} & \dots & \dots & a_{2n} \\ 0 & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix}.$$

Continue in this way until subtracting the multiple a_{n1}/a_{11} of row 1 from row n. These actions correspond to multiplication on the left by the series of elementary matrices

$$E_{n1}\left(\frac{a_{n1}}{a_{11}}\right) \dots E_{31}\left(\frac{a_{31}}{a_{11}}\right) E_{21}\left(\frac{a_{21}}{a_{11}}\right),$$

resulting in the row reduced matrix

$$E_{n1}\left(\frac{a_{n1}}{a_{11}}\right) \dots E_{31}\left(\frac{a_{31}}{a_{11}}\right) E_{21}\left(\frac{a_{21}}{a_{11}}\right) A = \begin{bmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ 0 & a_{22} & \dots & \dots & a_{2n} \\ 0 & a_{32} & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix}.$$

Continue this same process in column 2 with pivot element a_{22} to obtain

$$\begin{aligned} &\left[E_{n2}\left(\frac{a_{n2}}{a_{22}}\right) \dots E_{42}\left(\frac{a_{42}}{a_{22}}\right) E_{32}\left(\frac{a_{32}}{a_{22}}\right)\right]\left[E_{n1}\left(\frac{a_{n1}}{a_{11}}\right) \dots E_{31}\left(\frac{a_{31}}{a_{11}}\right) E_{21}\left(\frac{a_{21}}{a_{11}}\right)\right] A \\ &= \begin{bmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ 0 & a_{22} & \dots & \dots & a_{2n} \\ 0 & 0 & a_{33} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3} & \dots & a_{nn} \end{bmatrix}. \end{aligned}$$

In general, elimination in column i corresponds to the elementary matrix product

$$E_{ni}\left(\frac{a_{ni}}{a_{ii}}\right) \dots E_{i+2,i}\left(\frac{a_{i+2,i}}{a_{ii}}\right) E_{i+1,i}\left(\frac{a_{i+1,i}}{a_{ii}}\right).$$

Putting this all together, we have

$$\begin{aligned} &E_{n,n-1}\left(\frac{a_{n,n-1}}{a_{n-1,n-1}}\right) \\ &\quad \times \dots \times E_{ni}\left(\frac{a_{ni}}{a_{ii}}\right) \dots E_{i+2,i}\left(\frac{a_{i+2,i}}{a_{ii}}\right) E_{i+1,i}\left(\frac{a_{i+1,i}}{a_{ii}}\right) \\ &\quad \times \dots \times E_{n2}\left(\frac{a_{n2}}{a_{22}}\right) \dots E_{42}\left(\frac{a_{42}}{a_{22}}\right) E_{32}\left(\frac{a_{32}}{a_{22}}\right) \\ &\quad \times E_{n1}\left(\frac{a_{n1}}{a_{11}}\right) \dots E_{31}\left(\frac{a_{31}}{a_{11}}\right) E_{21}\left(\frac{a_{21}}{a_{11}}\right) A = U. \end{aligned}$$

We can simplify this expression by combining factors. Let E_i be the product of the elementary row matrices that perform row elimination in column i:

$$E_i = E_{ni}\left(\frac{a_{ni}}{a_{ii}}\right)\ldots E_{i+2,i}\left(\frac{a_{i+2,i}}{a_{ii}}\right)E_{i+1,i}\left(\frac{a_{i+1,i}}{a_{ii}}\right).$$

If A is a 3×3 matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix},$$

$$\begin{aligned} E_1 &= E_{31}\left(\frac{a_{31}}{a_{11}}\right)E_{21}\left(\frac{a_{21}}{a_{11}}\right) \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\dfrac{a_{31}}{a_{11}} & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ -\dfrac{a_{21}}{a_{11}} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ &= \begin{bmatrix} 1 & 0 & 0 \\ -\dfrac{a_{21}}{a_{11}} & 1 & 0 \\ -\dfrac{a_{31}}{a_{11}} & 0 & 1 \end{bmatrix}. \end{aligned} \tag{11.1}$$

In general,

$$E_i = \begin{bmatrix} 1 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ddots & \vdots & \vdots & \vdots & \cdots & 0 \\ & 0 & & 0 & 0 & \ldots & \ldots & 0 \\ & & & 1 & 0 & \ldots & \ldots & 0 \\ \vdots & \vdots & & -\dfrac{a_{i+1,i}}{a_{ii}} & 1 & \ldots & \ldots & 0 \\ & & & -\dfrac{a_{i+2,i}}{a_{ii}} & 0 & \ddots & \ldots & 0 \\ & & & \vdots & \ldots & \ldots & \ddots & \vdots \\ 0 & 0 & & -\dfrac{a_{ni}}{a_{ii}} & 0 & \ldots & 0 & 1 \end{bmatrix}. \tag{11.2}$$

We now have $E_{n-1}E_{n-2}\ldots E_2E_1A = U$, and

$$A = (E_{n-1}E_{n-2}\ldots E_2E_1)^{-1}U = \left(E_1^{-1}E_2^{-1}E_3^{-1}\ldots E_{n-2}^{-1}E_{n-1}^{-1}\right)U.$$

In the case of a 3×3 matrix, from Equation 11.1 we see that

$$\begin{aligned} E_1^{-1} &= \left[E_{21}\left(\frac{a_{21}}{a_{11}}\right)\right]^{-1}\left[E_{31}\left(\frac{a_{31}}{a_{11}}\right)\right]^{-1} \\ &= E_{21}\left(-\frac{a_{21}}{a_{11}}\right)E_{31}\left(-\frac{a_{31}}{a_{11}}\right) \\ &\quad \begin{bmatrix} 1 & 0 & 0 \\ \dfrac{a_{21}}{a_{11}} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \dfrac{a_{31}}{a_{11}} & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ \dfrac{a_{21}}{a_{11}} & 1 & 0 \\ \dfrac{a_{31}}{a_{11}} & 0 & 1 \end{bmatrix} \end{aligned}$$

In general,

$$E_i^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \ddots & \vdots & \vdots & \vdots & \dots & 0 \\ & 0 & & 0 & 0 & \dots & \dots & 0 \\ & & & 1 & 0 & \dots & \dots & 0 \\ \vdots & \vdots & & \frac{a_{i+1,i}}{a_{ii}} & 1 & \dots & \dots & 0 \\ & & & \frac{a_{i+2,i}}{a_{ii}} & 0 & \ddots & \dots & 0 \\ & & & \vdots & \dots & \dots & \ddots & \vdots \\ 0 & 0 & & \frac{a_{ni}}{a_{ii}} & 0 & \dots & 0 & 1 \end{bmatrix}. \tag{11.3}$$

The product of the lower-triangular matrices $L = E_1^{-1}E_2^{-1}E_3^{-1}\dots E_{n-2}^{-1}E_{n-1}^{-1}$ in Equation 11.3 is a lower-triangular matrix. From Equation 11.3, in the case of a 3×3 matrix,

$$\begin{aligned} L &= E_1^{-1}E_2^{-1} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ \frac{a_{21}}{a_{11}} & 1 & 0 \\ \frac{a_{31}}{a_{11}} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{a_{32}}{a_{22}} & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ \frac{a_{21}}{a_{11}} & 1 & 0 \\ \frac{a_{31}}{a_{11}} & \frac{a_{32}}{a_{22}} & 1 \end{bmatrix} \end{aligned}$$

For the $n \times n$ problem, $L = E_1^{-1}E_2^{-1}E_3^{-1}\dots E_{n-2}^{-1}E_{n-1}^{-1}$ is a lower-diagonal matrix with ones on the main diagonal. Below each 1 are the multipliers used to perform row elimination. This is precisely what we described in Section 11.1, and we have shown that if the pivot is never zero, we can factor A as $A = LU$.

Remark 11.3. For our decomposition algorithm to work, a_{ii} cannot equal 0. If during elimination $a_{ii} = 0$, requiring a row exchange, the LU decomposition as we have developed fails.

Example 11.8. $A = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 3 \\ 4 & -1 & 1 \end{bmatrix}$. Let's carry out the LU decomposition.

$$\begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 3 \\ 4 & -1 & 1 \end{bmatrix} \xrightarrow[R3 = R3 - (2)R1]{R2 = R2 - (1)R1} \begin{bmatrix} 2 & 1 & 1 \\ (1) & 0 & 2 \\ (2) & -3 & -1 \end{bmatrix}$$

$$\xrightarrow{R2 \leftrightarrow R3} \begin{bmatrix} 2 & 1 & 1 \\ (2) & -3 & -1 \\ (1) & 0 & 2 \end{bmatrix} \xrightarrow{R_3 = R_3 - (0)R_2} \begin{bmatrix} 2 & 1 & 1 \\ (2) & -3 & -1 \\ (1) & (0) & 2 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -3 & -1 \\ 0 & 0 & 2 \end{bmatrix}$$

and

$$LU = \begin{bmatrix} 2 & 1 & 1 \\ 4 & -1 & 1 \\ 2 & 1 & 3 \end{bmatrix}.$$

The result is the original matrix with rows two and three swapped. Somehow, we need to account for the fact that we swapped rows two and three during row reduction. ■

The process we have described results in $A = LU$. Perhaps we could have done the decomposition differently and arrived at another L and U; in other words, how do we know that there are no other LU factorizations of an $n \times n$ matrix A?

Theorem 11.6. *The LU decomposition of a nonsingular matrix A is unique.*

Proof. Assume there are two factorizations $L_1U_1 = L_2U_2 = A$. We need to show that $L_1 = L_2$ and $U_1 = U_2$. Since $L_1U_1 = L_2U_2$,

$$U_1U_2^{-1} = L_1^{-1}L_2. \tag{11.4}$$

In Equation 11.4, $U_1U_2^{-1}$ is an upper-triangular matrix, and $L_1^{-1}L_2$ is a lower-triangular matrix with 1s on its diagonal. Here is what we must have:

$$U_1U_2^{-1} = \begin{bmatrix} \overline{a_{11}} & \overline{a_{12}} & \dots & \overline{a_{1n}} \\ 0 & \overline{a_{22}} & \dots & \overline{a_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \overline{a_{nn}} \end{bmatrix} = L_1^{-1}L_2 = \begin{bmatrix} 1 & 0 & \dots & 0 \\ \overline{b_{21}} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \overline{b_{n1}} & \overline{b_{n2}} & \dots & 1 \end{bmatrix}.$$

The diagonals of both matrices must be equal, so $\overline{a_{ii}} = \overline{b_{ii}} = 1$. Since $U_1U_2^{-1}$ is an upper-triangular matrix it must have 0s below the diagonal, and since $L_1^{-1}L_2$ is a lower-triangular matrix it must have 0s above the diagonal. Thus, both matrices must be the identity, and $U_1U_2^{-1} = I$, so $U_1 = U_2$. Likewise, $L_1 = L_2$. □

With the exception of forming L, the algorithm for the LU decomposition is a straightforward expression of what we did by hand in Chapter 2. To maintain L, after performing a row elimination step, store the multiplier in the entry of A that becomes zero. We can now describe the algorithm simply:

Beginning with row 1:

Multiply row 1 by $\frac{a_{j1}}{a_{11}}$ and subtract from row j, $2 \le j \le n$, in order to eliminate all the elements in column 1 below a_{11}. For each j, store $\frac{a_{1j}}{a_{11}}$ in location $(j, 1)$.

Multiply row 2 by $\frac{a_{j2}}{a_{22}}$ and subtract from row j, $3 \le j \le n$, in order to eliminate all the elements in column 2 below a_{22}. For each j, store $\frac{a_{j2}}{a_{22}}$ in location $(j, 2)$.

• • •

Multiply row n-1 by $\frac{a_{n,n-1}}{a_{n-1,n-1}}$ and subtract from row n, in order to eliminate the element in row n, column n-1. Store $\frac{a_{n,n-1}}{a_{n-1,n-1}}$ in location $(n, n-1)$.

For the sake of clarity and brevity, we will begin using the colon notation in algorithms. It is the same format as the corresponding notation in MATLAB.

11.4.1 Colon Notation

When our pseudocode deals with an individual element in row i, column j, of an $m \times n$ matrix A, we use the notation a_{ij}. We adopt the MATLAB *colon notation* that allows us to access blocks of elements from matrices and perform operations. The notation $A(:, j)$ references column j of A:

$$A(:, j) = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{m-1,j} \\ a_{mj} \end{bmatrix}.$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2k} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} & \cdots & a_{kn} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} & \cdots & a_{nn} \end{bmatrix}$$

FIGURE 11.2 $k \times k$ submatrix.

For row i, the notation is $A(i, :)$:

$$A(i, :) = \begin{bmatrix} a_{i1} & a_{i2} & \ldots & a_{i,n-1} & a_{in} \end{bmatrix}.$$

If A is an $n \times n$ matrix and we want to multiply row i by 3, use the following statement:

$$A(i,:) = 3 * A(i,:).$$

If we need to multiply row i by 7 and subtract it from row j, we can write

$$A(j,:) = A(j,:) - 7 * A(i,:).$$

The colon notation can reference both rows and columns. For instance, if A is an $n \times n$ matrix, $A(1 : k, 1 : k)$ is the $k \times k$ submatrix beginning in the upper left-hand corner of A (Figure 11.2).

The following statement replaces the upper $k \times k$ submatrix by the $k \times k$ matrix B:

$$A(1 : k, 1 : k) = B.$$

The next statement subtracts the elements in row i, columns i through n from the same portion of each row below row i:

$$A(i+1 : n, i : n) = A(i+1 : n, i : n) - \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}^{(n-i)\times 1} A(i, i : n).$$

For instance, if $A = \begin{bmatrix} 1 & 2 & 5 \\ 8 & 3 & 7 \\ 1 & 1 & 4 \end{bmatrix}$,

$$\begin{aligned} A(2 : 3, 1 : 3) &= A(2 : 3, 1 : 3) - \begin{bmatrix} 1 \\ 1 \end{bmatrix} A(1, 1 : 3) \\ &= \begin{bmatrix} 8 & 3 & 7 \\ 1 & 1 & 4 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 5 \end{bmatrix} \\ &= \begin{bmatrix} 8 & 3 & 7 \\ 1 & 1 & 4 \end{bmatrix} - \begin{bmatrix} 1 & 2 & 5 \\ 1 & 2 & 5 \end{bmatrix} \\ &= \begin{bmatrix} 7 & 1 & 2 \\ 0 & -1 & -1 \end{bmatrix}, \end{aligned}$$

and

$$A = \begin{bmatrix} 1 & 2 & 5 \\ 7 & 1 & 2 \\ 0 & -1 & -1 \end{bmatrix}.$$

Remark 11.4. Using the colon notation frees us to think at the vector and matrix level and concentrate on more important computational issues.

11.4.2 The *LU* Decomposition Algorithm

Algorithm 11.1 describes the *LU* factorization, assuming the pivot element is nonzero. The algorithm makes use of the colon notation and includes use of the functions `triu` and `tril`. A call to `triu(A)` returns the upper-triangular portion of A, and `tril(A,-1)` returns the portion of *A* below the main diagonal.

Algorithm 11.1 *LU* Decomposition Without a Zero on the Diagonal

```
function LUGAUSS(A)
   % Input: n × n matrix A.
   % Output: lower-triangular matrix L and upper-triangular matrix U such that A = LU.
   for i = 1:n-1 do
      if a_ii = 0 then
         print 'The algorithm has encountered a zero pivot.'
         exit
      end if
      % Replace the elements in column i, rows i+1 to n by the multipliers a_ji / a_ii
      A(i+1:n, i) = A(i+1:n, i)/a_ii
      % Modify the elements in rows i+1 to n, columns i+1 to n by subtracting
      % the multiplier for the row times the elements in row i.
      A(i+1:n, i+1:n) = A(i+1:n, i+1:n) − A(i+1:n, i)A(i, i+1:n)
   end for
   % Assign U the upper-triangular portion of A.
   U = triu(A)
   % Initialize L as the identity matrix.
   L = I % Add into L the portion of A below the main diagonal.
   L = L + tril(A,-1);
   return [ L, U ]
end function
```

NLALIB: The function `lugauss` implements Algorithm 11.1.

Example 11.9. In this example, we use the function `lugauss` to factor a 4 × 4 matrix.

```
A =
   1  -3   5   2
   1   0   1  -1
   6   1  -9   2
   1   0  -6   3

>> [L, U] = lugauss(A)

L =
            1            0            0            0
            1            1            0            0
            6       6.3333            1            0
            1            1       0.5122            1

U =
            1           -3            5            2
            0            3           -4           -3
            0            0      -13.667            9
            0            0            0     -0.60976

>> L*U

ans =
            1           -3            5            2
            1            0            1           -1
            6            1           -9            2
            1            0           -6            3
```

■

11.4.3 *LU* Decomposition Flop Count

Gaussian elimination is a relatively slow algorithm. Developing a flop count will tell how much work is actually involved in computing L and U. We will count first for $i = 1$, then $i = 2$, and so forth until $i = n - 1$ and form the sum of the counts. The annotated Figure 11.3 will aid in understanding the computation.

Flop Count

$i = 1$:

The $n - 1$ quotients $a_{21}/a_{11}, a_{31}/a_{11}, \ldots, a_{n1}/a_{11}$ are the row multipliers. The elements at indices $(2, 1), (3, 1), \ldots, (n, 1)$ are replaced by the multipliers, so row elimination operations occur in the $(n-1) \times (n-1)$ submatrix $A(2 : n, 2 : n)$. There are $(n-1)^2$ elements that must be modified. To modify an element requires a multiplication and a subtraction, or 2 flops. The total flops required for row elimination is $2(n-1)^2$. For $i = 1$, the flop count is $(n-1) + 2(n-1)^2$.

$i = 2$:

The $n-2$ quotients $a_{32}/a_{22}, a_{42}/a_{22}, \ldots, a_{n2}/a_{22}$ are the row multipliers. There are $(n-2)^2$ elements that must be modified in rows 3 through n, and the total flops required for the modification is $2(n-2)^2$. For $i = 2$, the flop count is $(n-2) + 2(n-2)^2$.

$\ldots$

$i = n - 1$:

Compute 1 quotient $\frac{a_{n,n-1}}{a_{n-1,n-1}}$ and execute 1 multiplication and 1 subtraction, for a total of $1 + 2(1)$ flops.

The total count is

$$\sum_{i=1}^{n-1} (n-i) + \sum_{i=1}^{n-1} 2(n-i)^2 = \sum_{i=1}^{n-1} i + 2\sum_{i=1}^{n-1} i^2. \tag{11.5}$$

To evaluate Equation 11.5, we need two summation formulas:

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$
$$\sum_{i=1}^{k} i^2 = \frac{k(k+1)(2k+1)}{6}$$

By applying these formulas to Equation 11.5, we have

$$\sum_{i=1}^{n-1} i + 2\sum_{i=1}^{n-1} i^2 = \frac{n(n-1)}{2} + 2\frac{(n-1)n(2n-1)}{6},$$

and after combining terms the flop count is

$$\frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n. \tag{11.6}$$

The dominant term in Equation 11.6 is $\frac{2}{3}n^3$, so the flop count is $\frac{2}{3}n^3 + O\left(n^2\right)$, and LU decomposition is a cubic algorithm.

A good reason for computing L and U is that the cubic LU decomposition algorithm allows us to repeatedly solve $Ax = b$ for many $b's$ without having to recompute either L or U, as we will show in Section 11.6.

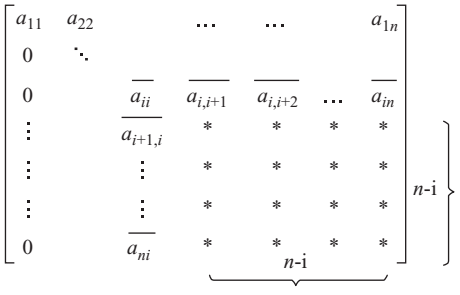

FIGURE 11.3 Gaussian elimination flop count.

11.5 GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING

Gaussian elimination can produce extremely bad results under certain circumstances; in fact, the results can be completely wrong. Consider the matrix

$$A = \begin{bmatrix} 0.00001 & 3 \\ 2 & 1 \end{bmatrix},$$

and use three-digit arithmetic. There is only one step required to produce the LU decomposition. Use the multiplier $\frac{2}{0.00001} = 2 \times 10^5$. In exact arithmetic, the elimination step gives

$$\begin{bmatrix} 0.00001 & 1 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0.00001 & 3 \\ 0 & 1 - 3\left(\frac{2}{0.00001}\right) \end{bmatrix} = \begin{bmatrix} 0.00001 & 3 \\ 0 & -599999 \end{bmatrix},$$

and the LU decomposition is $L = \begin{bmatrix} 1 & 0 \\ 200000 & 1 \end{bmatrix}$, $U = \begin{bmatrix} 0.00001 & 3 \\ 0 & -599999 \end{bmatrix}$.
In our three-digit arithmetic, the 1 is lost, and the result is

$$L = \begin{bmatrix} 1 & 0 \\ 200000 & 1 \end{bmatrix}, \; U = \begin{bmatrix} 0.00001 & 3 \\ 0 & -600000 \end{bmatrix}$$

Now compute LU to get

$$\begin{bmatrix} 0.00001 & 3 \\ 2 & 0 \end{bmatrix},$$

which is disastrously different from A! The problem arose when we divided by a small pivot element and obtained a large multiplier. The large multiplier resulted in the addition of a very large number to a much smaller number. The result was loss of any contribution from the small number. There is a good solution to this problem. When choosing the pivot element on the diagonal at position a_{ii}, locate the element in column i at or below the diagonal that is largest in magnitude, say a_{ji}, $i \leq j \leq n$. If $j \neq i$, interchange row j with row i, and then the multipliers, $\frac{a_{ji}}{a_{ii}}$, satisfy $\left|\frac{a_{ji}}{a_{ii}}\right| \leq 1$, $i + 1 \leq j \leq n$, and we avoid multiplying a row by a large number and losing precision. We call this *Gaussian elimination with partial pivoting* (GEPP). Apply this strategy to our matrix, rounding to three-digit precision.

$$\begin{bmatrix} 0.00001 & 3 \\ 2 & 1 \end{bmatrix} \overrightarrow{R_1 \longleftrightarrow R_2} = \begin{bmatrix} 2 & 1 \\ 0.00001 & 3 \end{bmatrix} \overrightarrow{R_2 = R_2 - \frac{0.00001}{2} R_1} \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$$

We now have
$L = \begin{bmatrix} 1 & 0 \\ \frac{0.00001}{2} & 1 \end{bmatrix}$, $U = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$, and

$$LU = \begin{bmatrix} 2 & 1 \\ 0.00001 & 3 \end{bmatrix}.$$

Of course, this is not the original matrix A, but A with its two rows swapped (permuted). If we use GEPP, then an LU decomposition for A consists of three matrices P, L, and U such that

$$PA = LU.$$

P is a *permutation matrix*, also called the *pivot matrix*. Start with $P = I$, and swap rows i and j of the permutation matrix whenever rows i and j are swapped during GEPP. For instance,

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

would be the permutation matrix if the second and third rows of A are interchanged during pivoting.

Example 11.10. Factor the matrix $A = \begin{bmatrix} 3 & 8 & 1 \\ 5 & 2 & 0 \\ 6 & 1 & 12 \end{bmatrix}$.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 3 & 8 & 1 \\ 5 & 2 & 0 \\ 6 & 1 & 12 \end{bmatrix}.$$

Pivot row = 1. Swap rows 1 and 3, and permute P. Do not interchange rows of L until arriving at the pivot in row 2, column 2 (Remark 11.5).

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 6 & 1 & 12 \\ 5 & 2 & 0 \\ 3 & 8 & 1 \end{bmatrix}.$$

Apply the pivot element, and add multipliers to L.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 5/6 & 1 & 0 \\ 1/2 & 0 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 6 & 1 & 12 \\ 0 & 7/6 & -10 \\ 0 & 15/2 & -5 \end{bmatrix}.$$

Pivot row = 2. Swap rows 2 and 3. Permute P and L.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 5/6 & 0 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 6 & 1 & 12 \\ 0 & 15/2 & -5 \\ 0 & 7/6 & -10 \end{bmatrix}.$$

Apply the pivot element and update L.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 5/6 & 7/45 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 6 & 1 & 12 \\ 0 & 15/2 & -5 \\ 0 & 0 & -83/9 \end{bmatrix}.$$

The final results are

$$\begin{aligned} L &= \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 5/6 & 7/45 & 1 \end{bmatrix}, \\ U &= \begin{bmatrix} 6 & 1 & 12 \\ 0 & 15/2 & -5 \\ 0 & 0 & -83/9 \end{bmatrix}, \\ P &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \end{aligned}$$

You should verify that $PA = LU$. ■

Remark 11.5. Even if a row interchange occurs when dealing with column 1, do not interchange the corresponding rows of L until moving to column 2. Think of it this way. The matrix A after a row swap defines the starting configuration. The elements in the first column of L correspond to the multipliers after the row interchange involving pivot position $(1, 1)$, if any.

Summary

At step i of Gaussian elimination, the pivot element is a_{ii}. To eliminate each element a_{ji}, $i + 1 \leq j \leq n$, multiply row i by a_{ji}/a_{ii} and subtract from row j. These multipliers should not be large or precision can be lost. Find the largest of the elements $\{|a_{ii}|, |a_{i+1,i}|, |a_{i+2,i}|, \ldots, |a_{ni}|\}$. If the row index of the largest absolute value is $j \neq i$, exchange rows i and j. Now all the multipliers a_{ji}/a_{ii} have absolute value less than or equal to 1.

11.5.1 Derivation of *PA=LU*

For a reader primarily interested in applications, this subsection may be skipped; however, it is important to read Section 11.5.2.

For the explanation, we will assume that P_i is the permutation matrix corresponding a possible interchange when dealing with column i. The matrix E_i (Equation 11.2) is the product of the elementary row matrices that perform row elimination

in column i. In the sequence of steps that follow, "exchange rows" means multiply by a permutation matrix or the identity if no row exchanges are required.

$$A = \begin{bmatrix} x & x & x & \dots & x & x & x \\ x & x & x & \dots & x & x & x \\ x & x & x & \dots & x & x & x \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ x & x & x & \dots & x & x & x \\ x & x & x & \dots & x & x & x \\ x & x & x & \dots & x & x & x \end{bmatrix}.$$

Exchange rows, then perform elimination in column 1.

$$i = 1: \quad E_1 P_1 A = \begin{bmatrix} x & x & x & \dots & x & x & x \\ 0 & x & x & \dots & x & x & x \\ 0 & x & x & \dots & x & x & x \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & x & x & \dots & x & x & x \\ 0 & x & x & \dots & x & x & x \\ 0 & x & x & \dots & x & x & x \end{bmatrix}.$$

Beginning with the matrix after step 1, exchange rows, then perform elimination in column 2.

$$i = 2: \quad E_2 P_2 E_1 P_1 A = \begin{bmatrix} x & x & x & \dots & x & x & x \\ 0 & x & x & \dots & x & x & x \\ 0 & 0 & x & \dots & x & x & x \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & x & \dots & x & x & x \\ 0 & 0 & x & \dots & x & x & x \\ 0 & 0 & x & \dots & x & x & x \end{bmatrix}.$$

Beginning with the matrix after step 2, exchange rows, then perform elimination in column 3.

$$i = 3: \quad E_3 P_3 E_2 P_2 E_1 P_1 A = \begin{bmatrix} x & x & x & \dots & x & x & x \\ 0 & x & x & \dots & x & x & x \\ 0 & 0 & x & \dots & x & x & x \\ \vdots & \vdots & 0 & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \vdots & \dots & x & x & x \\ 0 & 0 & 0 & \dots & x & x & x \\ 0 & 0 & 0 & \dots & x & x & x \end{bmatrix}$$

Beginning with the matrix after step $n-2$, exchange rows, then eliminate the element in row n, column $n-1$.

$$i = n-1: \quad E_{n-1} P_{n-1} \dots E_3 P_3 E_2 P_2 E_1 P_1 A = \begin{bmatrix} x & x & x & \dots & x & x & x \\ 0 & x & x & \dots & x & x & x \\ 0 & 0 & x & \dots & x & x & x \\ \vdots & \vdots & 0 & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \vdots & \dots & x & x & x \\ 0 & 0 & 0 & \dots & 0 & x & x \\ 0 & 0 & 0 & \dots & 0 & 0 & x \end{bmatrix} = U$$

The product $E_{n-1} P_{n-1} \dots E_3 P_3 E_2 P_2 E_1 P_1$ must be rewritten in order to isolate the permutation matrices in the order $P_{n-1} P_{n-2} \dots P_2 P_1$. First, note that if P_i is a permutation matrix, then $P_i^2 = I$, since $P_i P_i$ permutes I and then permutes the permutation back to I. Equivalently, $P_i^{-1} = P_i$. Let's look at a 3×3 example:

$$U = E_2 P_2 E_1 P_1 A = E_2 \left(P_2 E_1 P_2^{-1} \right) (P_2 P_1) A.$$

P_2P_1 is the product of permutation matrices, and so is a permutation matrix. Now, E_2 is an elementary matrix obtained by subtracting a multiple of row 2($\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$) from row 3 ($\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$) of I, and so it is a lower-triangular matrix. The factor $P_2E_1P_2^{-1}$ is interesting. P_2E_1 permutes rows of E_1, and E_1 is a lower-triangular matrix. Multiplying on the right by a permutation matrix exchanges columns (Problem 11.5). Thus, $P_2E_1P_2^{-1}$ is a lower-triangular matrix, and is E_1 with its subdiagonal elements permuted. The product of lower-triangular matrices is lower triangular, and $\overline{L}PA = U$, where $\overline{L} = E_2\left(P_2E_1P_2^{-1}\right)$ is a lower-triangular matrix. Since it is built from permutations of elementary matrices, it is invertible, and $PA = \left(\overline{L}\right)^{-1}U = LU$, where $L = \left(\overline{L}\right)^{-1}$. The inverse of a lower-triangular matrix is lower triangular, so L is lower triangular. As was the case without pivoting, L has ones on its diagonal.

Now consider a 4×4 example:

$$U = E_3P_3E_2P_2E_1P_1 = E_3\left(P_3E_2P_3^{-1}\right)\left(P_3P_2E_1P_2^{-1}P_3^{-1}\right)(P_3P_2P_1)A.$$

Each of $\{E_1, E_2, E_3\}$ is a lower-triangular matrix, and the permutation matrices are arranged such that each factor remains a lower-triangular matrix.

This manipulation can be carried out for any $n \times n$ matrix. Each factor before the final $(P_{n-1}P_{n-2}\ldots P_2P_1)$ is an invertible lower-triangular matrix, say $\hat{T}_i$, so we have

$$U = \hat{T}_{n-1}\hat{T}_{n-2}\ldots\hat{T}_2\hat{T}_1\,(P_{n-1}P_{n-2}\ldots P_2P_1)\,A,$$

and

$$(P_{n-1}P_{n-2}\ldots P_2P_1)\,A = \left(\hat{T}_{n-1}\hat{T}_{n-2}\ldots\hat{T}_2\hat{T}_1\right)^{-1}U,$$

or $PA = LU$.

Example 11.11. Let $A = \begin{bmatrix} -1 & -1 & 0 & 1 \\ -1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 \end{bmatrix}$. This example illustrates the process just described that derives GEPP.

Start: $L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, $P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$i = 1$: Exchange rows 1 and 4. Eliminate entries at indices $(2, 1) - (4, 1)$:

$$E_1 = E_{41}\left(-\tfrac{1}{2}\right)E_{31}\left(\tfrac{1}{2}\right)E_{21}\left(-\tfrac{1}{2}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ -0.5 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix}$$

$$P_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$E_1P_1A = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 1.5 & 0 \\ 0 & 1 & 0.5 & 1 \\ 0 & -1 & 0.5 & 1 \end{bmatrix}$$

$i = 2$: A row exchange is not necessary. Eliminate the entries at indices $(3, 2)$ and $(4, 2)$.

$$E_2 = E_{42}(-1)\,E_{32}(1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$E_2P_2E_1P_1A = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 1.5 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 2 & 1 \end{bmatrix}$$

$i = 3$: Interchange rows 3 and 4. Eliminate the entry at $(4,\ 3)$.

$$E_3 = E_{43}\left(-\tfrac{1}{2}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 1 \end{bmatrix}$$

$$P_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$E_3P_3E_2P_2E_1P_1A = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 1.5 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1.5 \end{bmatrix} = U$$

Form

$$E_3\,(P_3E_2P_3)\,(P_3P_2E_1P_2P_3)\,(P_3P_2P_1)\,A$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ .5 & 1 & 0 & 0 \\ .5 & 0 & 1 & 0 \\ -.5 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & 0 & 1 \\ -1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 \end{bmatrix} = U$$

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},\ L = \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0 & 1 & 0 \\ -0.5 & 0 & 0 & 1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.5 & 1 & 0 & 0 \\ -0.5 & -1 & 1 & 0 \\ 0.5 & 1 & -0.5 & 1 \end{bmatrix}$$

Now,

$$PA = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & 0 & 1 \\ -1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 & 0 \\ -1 & 1 & 1 & 0 \\ -1 & -1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.5 & 1 & 0 & 0 \\ -0.5 & -1 & 1 & 0 \\ 0.5 & 1 & -0.5 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 1.5 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1.5 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 & 0 \\ -1 & 1 & 1 & 0 \\ -1 & -1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

■

11.5.2 Algorithm for Gaussian Elimination with Partial Pivoting

Algorithm 11.2 specifies GEPP, making use of submatrix operations.

Algorithm 11.2 Gaussian Elimination with Partial Pivoting

```
function LUDECOMP(A)
  % LU decomposition using Gaussian elimination with partial pivoting.
  % [P U P interchanges] = ludecomp(A) factors a square
  % matrix so that PA = LU. U is an upper-triangular matrix,
  % L is a lower-triangular matrix, and P is a permutation
  % matrix that reflects the row exchanges required by
  % partial pivoting used to reduce round-off error.
  % In the event that is useful, interchanges is the number
  % of row interchanges required.
  L = I
  P = I
  for i = 1:n-1 do
    k = index of largest matrix entry in column i, rows i through n
    pivotindex = i + k - 1
    if pivotindex ≠ i then
      % Exchange rows i and k, ignoring columns 1 through i-1 in each row.
      tmp = A(i,i:n)
      A(i,i:n) = A(pivotindex,i:n)
      A(pivotindex,i:n) = tmp
      % Swap whole rows in P.
      tmp = P(i,1:n)
      P(i,1:n) = A(pivotindex,1:n)
      P(pivotindex,1:n) = tmp
      % Swap rows of L also, but only in columns 1 through i-1.
      tmp = L(i,1:i-1)
      L(i,1:i-1) = A(pivotindex,1:i-1)
      P(pivotindex,1:i-1) = tmp
    end if
    % Compute the multipliers.
    multipliers = A(i+1:n,i)/A(i,i)
    % Use submatrix calculations instead of a loop to perform
    % the row operations on the submatrix A(i+1:n, i+1:n)
    A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - multipliers*A(i,i+1:n);
    % Set entries in column i, rows i+1:n to 0.
    A(i + 1 : n, i) = [ 0 0 ... 0 0 ]^T
    L(i+1:n,i) = multipliers
  end for
  U = A
  return [ L, U, P ]
end function
```

NLALIB: The function `ludecomp` implements Algorithm 11.2.

Example 11.12.
Factor the 4×4 matrix of Example 11.11 using `ludecomp`.

```
>> [L, U, P] = ludecomp(A)

L =
    1.0000         0         0         0
   -0.5000    1.0000         0         0
   -0.5000   -1.0000    1.0000         0
    0.5000    1.0000   -0.5000    1.0000

U =
    2.0000         0    1.0000         0
         0    1.0000    1.5000         0
         0         0    2.0000    1.0000
         0         0         0    1.5000

P =
     0     0     0     1
     0     1     0     0
     1     0     0     0
     0     0     1     0
```

During the execution of `ludecomp`, if entries a_{ki}, $i \leq k \leq n$ are all less than or equal to `eps`, the algorithm simply moves to the next pivot location $(i+1, i+1)$. As a result, GEPP applies to any matrix, even one that is singular. Factor the singular matrix

$$A = \begin{bmatrix} 2 & 1 & 3 & 5 \\ 1 & 6 & -1 & 2 \\ 3 & 7 & 2 & 7 \\ 5 & 19 & 0 & 11 \end{bmatrix}.$$

```
>> [L, U, P] = ludecomp(A)

L =
    1.0000         0         0         0
    0.4000    1.0000         0         0
    0.6000    0.6667    1.0000         0
    0.2000   -0.3333         0    1.0000

U =
    5.0000   19.0000         0   11.0000
         0   -6.6000    3.0000    0.6000
         0         0         0    0.0000
         0         0   -0.0000   -0.0000

P =
     0     0     0     1
     1     0     0     0
     0     0     1     0
     0     1     0     0

>> P*A

ans =
     5    19     0    11
     2     1     3     5
     3     7     2     7
     1     6    -1     2
```

```
>> L*U

ans =
     5    19     0    11
     2     1     3     5
     3     7     2     7
     1     6    -1     2

>> P'*L*U

ans =
     2     1     3     5
     1     6    -1     2
     3     7     2     7
     5    19     0    11
```

■

11.6 USING THE *LU* DECOMPOSITION TO SOLVE $Ax_i = b_i$, $1 \leq i \leq k$

To use this decomposition to solve a single system $Ax = b$, first multiply both sides by the permutation matrix:

$$PAx = Pb.$$

Let $\overline{p} = Pb$, and substitute LU for PA:

$$LUx = \overline{b}.$$

Execute forward and back substitution:

$$Ly = \overline{b}$$

and

$$Ux = y.$$

Remark 11.6. The MATLAB function `lu` also has a calling sequence `[L U P] = lu(A)`. The *LU* decomposition functions `ludecomp` and `lu` produce the same results, but `lu` is faster because it is implemented in machine code.

Example 11.13. Let $A = \begin{bmatrix} 1 & 4 & 9 \\ -1 & 5 & 1 \\ 3 & 1 & 5 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 6 \\ 2 \end{bmatrix}$. We use the functions `forsolve(L,b)` and `backsolve(U,b)` from the book software that perform forward and back substitution, respectively.

```
>> [L,U,P] = ludecomp(A);y = forsolve(L,P*b);x = backsolve(U,y);
>> norm(b-A*x)
ans = 1.2561e-15
```

■

Solving systems $Ax = b$ for many $b's$ only requires that A be factored once, at a cost of $O\left(n^3\right)$ flops. The combined steps of forward and back substitution require $O\left(n^2\right)$ flops. If there are k right-hand sides, the flop count is $O\left(n^3\right) + kO\left(n^2\right)$. It would be extremely inefficient to perform Gaussian elimination for each right-hand side, since that would require $kO\left(n^3\right)$ flops. Algorithm lusolve takes the decomposition of A, a matrix of right-hand sides, B, and computes a matrix of solutions X.

Algorithm 11.3 Solve $Ax = b$ for Multiple Right-Hand Sides

```
function LUSOLVE(L,U,P,B)
   % Solve multiple equations Ax = b using
   % the result of the LU factorization.
   % X = lusolve(L,U,P,B), where B is an n × k matrix containing
   % k right-hand sides for which a solution to the linear system
   % Ax = b is required. L, U, P are the result of the LU factorization
   % P*A = L*U. The solutions are in the k columns of X.
   pb = P*B
   for i = 1:k do
      y_i = forsolve(L, pb(:, i))
      x_i = backsolve(U, y_i)
      X(:, i) = x_i
   end for
   return X
end function
```

NLALIB: The function `lusolve` implements Algorithm 11.3.

We now use `ludecomp` to factor a 5×5 randomly generated matrix A, and follow this by using `lusolve` to solve $Ax = b$ for three randomly generated right-hand sides. For each solution, the code outputs the 2-norm of the residual.

Example 11.14.

```
>> A = rand(5,5);
>> [L, U, P] = ludecomp(A);
>> B = [rand(5,1) rand(5,1) rand(5,1)];
X = zeros(5,3);
X = lusolve(L, U, P, B);
% check the results
for i=1:3
    norm(A*X(:,i)-B(:,i))
end

ans =
  3.5975e-016
ans =
  1.5701e-016
ans =
  2.0015e-016
```

■

Remark 11.7. There is an algorithm termed *Gaussian elimination with complete pivoting* (*GECP*), in which the pivoting strategy exchanges both rows and columns. Note that column exchanges require renumbering the unknowns. Complete pivoting is more complex and is not often used in practice, since GEPP gives good results with less computational effort.

11.7 FINDING A^{-1}

As stated earlier, it is considerably expensive to compute the inverse of a general nonsingular matrix A, and its computation should be avoided whenever possible. We have previously mentioned that solving $Ax = b$ using $b = A^{-1}b$ is a poor choice.

In Chapter 18, will present an algorithm, called the inverse power method, for finding the smallest eigenvalue and a corresponding eigenvector of a real nonsingular matrix A. Under the correct circumstances, the iteration $x_{i+1} = A^{-1}x_i$ converges to the largest eigenvector of A^{-1}. To avoid computing A^{-1}, just repeatedly solve $Ax_{i+1} = x_i$ after factoring A.

If A^{-1}must be computed, Section 2.5 and Theorem 11.5 say that to find it we solve the systems $Ax_i = e_i$, $1 \leq i \leq n$, where e_i is the vector with a 1 in component i and zeros in all other entries. We are prepared for this computation. Use Algorithm 11.3 with $B = I$.

Example 11.15. Find the inverse of the matrix $A = \begin{bmatrix} -9 & 1 & 3 \\ 1 & 5 & 2 \\ -6 & 12 & 3 \end{bmatrix}$.

```
>> A = [-9 1 3;1 5 2;-6 12 3]
>> [L U P] = ludecomp(A);
>> A_inverse = lusolve(L, U, P, eye(3))
A_inverse =
   -0.0469    0.1719   -0.0677
   -0.0781   -0.0469    0.1094
    0.2187    0.5313   -0.2396
>> norm(A_inverse - inv(A))
ans =
  2.9394e-017
```

■

Remark 11.8. The flop count for finding the inverse is found by forming the flop count for the *LU* factorization plus n instances of forward and backward substitution, so we have $\frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n + n\left[2\left(n^2 + n\right)\right] = \frac{8}{3}n^3 + \frac{3}{2}n^2 - \frac{1}{6}n \approx \frac{8}{3}n^3$. Suppose we simply solve n equations $Ax_i = e_i$ without using the *LU* decomposition. Each solution will cost $O\left(n^3\right)$ flops, so our efforts will take $O\left(n^4\right)$ flops. Of course, nobody would compute the inverse this way.

11.8 STABILITY AND EFFICIENCY OF GAUSSIAN ELIMINATION

Gaussian elimination without partial pivoting is not stable in general, as we showed by using the matrix $A = \begin{bmatrix} 0.00001 & 3 \\ 2 & 1 \end{bmatrix}$. It is theoretically possible for Gaussian elimination with partial pivoting to be explosively unstable [31] on certain "cooked-up" matrices; however, if we consider performance in practice, it is stable. The unusual matrices that produce poor results have never been encountered in applications.

Theorem 11.7 provides a criteria for stability, and its proof can be found in Ref. [26, pp. 163-165]. Prior to stating the theorem, we need to define the *growth factor* of a matrix. During the *LU* factorization, the norm of the matrix L has an upper bound we can compute for the norms we use (Problem 11.8). However, the elements of U can grow very large relative to those of A. If this happens, we expect Gaussian elimination to produce poor results.

Definition 11.2. Apply the *LU* factorization to matrix A. During the elimination steps, we have matrices $A = A^{(0)}, A^{(1)}, A^{(2)}, A^{(k)}, \ldots, A^{(n-1)} = U$. The growth factor is the ratio

$$\rho = \frac{\max_{i,j,k} \left|a_{ij}^{(k)}\right|}{\max_{ij} \left|a_{ij}\right|};$$

in other words, find the ratio of the largest element in magnitude during Gaussian elimination to the largest element of A in magnitude.

Example 11.16. Compute the growth factor for

$$A = \begin{bmatrix} 0.0001 & 3 \\ -1 & 1 \end{bmatrix}.$$

Without partial pivoting:

$$\begin{bmatrix} 0.0001 & 3 \\ -1 & 1 \end{bmatrix} \xrightarrow{R2 = R2 - \left(\frac{-1}{0.0001}\right) R1} \begin{bmatrix} 0.0001 & 3 \\ 0 & 30001 \end{bmatrix},$$

and the growth factor is $\rho = \frac{30{,}001}{3} = 10000.33$.

With partial pivoting:

$$\begin{bmatrix} 0.0001 & 3 \\ -1 & 1 \end{bmatrix} \xrightarrow{R1 \Leftrightarrow R2} \begin{bmatrix} -1 & 1 \\ 0.0001 & 3 \end{bmatrix} \xrightarrow{R2 = R2 - (-0.0001)\,R1} \begin{bmatrix} -1 & 1 \\ 0 & 3.0001 \end{bmatrix},$$

and $\rho = \frac{3.0001}{3} = 1.0000$. Partial pivoting, as expected, improves the growth factor. In this example, the improvement was huge. ■

Theorem 11.7. *Let the factorization $PA = LU$ of a matrix A be computed using Gaussian elimination with partial pivoting on a computer that satisfies Equations 8.3 and 8.5. Then the computed matrices $\tilde{L}$, $\tilde{U}$, and $\tilde{P}$ satisfy*

$$\tilde{L}\tilde{U} = \tilde{P}A + \delta A, \qquad \frac{\|\delta A\|_2}{\|A\|} = O(\rho\ eps)$$

for some $n \times n$ matrix δA, where ρ is the growth factor for A. If $|l_{ij}| < 1$ for each $i > j$, implying there are no ties in the selection of pivots in exact arithmetic, then $\tilde{P} = P$ for all sufficiently small eps.

Theorem 11.7 [26, pp. 163–165] says that GEPP is backward stable if the size of ρ does not vary with n for all $n \times n$ matrices. Using O notation, this means that ρ is $O(1)$ for all such matrices. Unfortunately, there are matrices for which ρ gets large (Problem 11.36).

We showed in Section 11.4.3 that Gaussian elimination without pivoting requires $O(n^2)$ comparisons. At step i, partial pivoting requires a search of elements $A(i:n, i)$ to locate the pivot element, and this requires $(n-1)+(n-2)+\cdots+2+1 = n(n-1)/2$ comparisons, so Gaussian elimination with pivoting also requires $\frac{2}{3}n^3 + O(n^2)$ flops. When performing complete pivoting, step i requires a search of the submatrix $A(i:n, i:n)$ and may perform column interchanges. It requires $\frac{2}{3}n^3 + O(n^2)$ flops, but requires $O(n^3)$ comparisons, so is a slower algorithm. Incidentally, complete pivoting creates a factorization of the form $PAQ = LU$, where both P and Q are permutation matrices. See Refs. [2, pp. 131-133] and [19, pp. 104-109] for an explanation of complete pivoting.

11.9 ITERATIVE REFINEMENT

If a solution to $Ax = b$ is not accurate enough, it is possible to improve the solution using *iterative refinement*. Let $\bar{x}$ be the computed solution of the system $Ax = b$. If $x = \bar{x} + \delta x$ is the exact solution, then $Ax = A(\bar{x} + \delta x) = A\bar{x} + A(\delta x) = b$, and $A(\delta x) = b - A\bar{x} = r$, the residual. If we solve the system $A(\delta x) = r$ for δx, then $Ax = A\bar{x} + A(\delta x) = A\bar{x} + r = A\bar{x} + b - A\bar{x} = b$. It is unlikely that we will obtain an exact solution to $A(\delta x) = r$; however, $\bar{x} + \delta x$ might be better approximation to the true solution than $\bar{x}$. For this to be true, it is necessary to compute the residual r using twice the precision of the original computations; for instance, if the computation of $\bar{x}$ was done using 32-bit floating point precision, then the residual should be computed using 64-bit precision. This process provides a basis for an iteration that continues until we reach a desired relative accuracy or fail to do so. Unless the matrix is very poorly conditioned, the computed solution x is already close to the true solution, so only a few iterations are required. If the matrix has a large condition number, it is not reasonable to expect huge improvements. Algorithm 11.4 describes the iterative improvement algorithm. Note that the algorithm returns the number of iterations performed in an attempt to reach the tolerance or -1 if it is not attained.

Algorithm 11.4 Iterative Improvement

```
function ITERIMP(A,L,U,P,b,x_1,tol,numiter)
   % Perform iterative improvement of a solution x_1
   % to Ax = b, where L, U, P is the LU factorization of A.
   % tol is the error tolerance, and numiter the maximum number
   % of iterations to perform.
   % Returns the improved solution and the number of iterations
   % required, or -1 if the tolerance is not obtained.
   for k = 1:numiter do
      iter = k % Compute the residual.
      r = b − Ax_k
      % Compute the correction.
      δx = lusolve(L, U, P, r)
      % Add the correction to form a new approximate solution.
      x_{k+1} = x_k + δx
      if ||x_{k+1} − x_k|| / ||x_k|| < tol then
         x = x_{k+1}
         return [ x iter ]
      end if
   end for
   % Tolerance not obtained.
   x = x_{k+1}
   iter = -1
end function
```

NLALIB: The function `iterimp` implements Algorithm 11.4.

Example 11.17.

$$A = \begin{bmatrix} 1 & -6 & 3 \\ 2 & 4 & 1 \\ 3 & -9 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}.$$

The solution accurate to four decimal places is $x = \begin{bmatrix} 1.3636 \\ -0.1010 \\ -0.3232 \end{bmatrix}$. Assume we have computed a solution $\bar{x} = \begin{bmatrix} 1 \\ 0 \\ -0.5 \end{bmatrix}$, so let $x_1 = \begin{bmatrix} 1 \\ 0 \\ -0.5 \end{bmatrix}$.

$k = 1$**:**

$$r_1 = b - Ax_1 = \begin{bmatrix} 1.5000 \\ 0.5000 \\ 2.000 \end{bmatrix}.$$

The solution of $A(\delta x) = r_1$ is

$$\delta x_1 = \begin{bmatrix} 0.3636 \\ -0.1010 \\ 0.1768 \end{bmatrix}$$

and then

$$x_2 = x_1 + \delta x_1 = \begin{bmatrix} 1 \\ 0 \\ -0.5 \end{bmatrix} + \begin{bmatrix} 0.3636 \\ -0.1010 \\ 0.1768 \end{bmatrix} = \begin{bmatrix} 1.3636 \\ -0.1010 \\ -0.3232 \end{bmatrix}.$$

We obtained the answer correct to four places in one iteration. Note that $\kappa(A) = 4.6357$, so A is well-conditioned. ■

The flop count for Algorithm 11.4 is $O(n^2)$, since each solution of the equation $A\delta x = r_k$ using `lusolve` has flop count $O(n^2)$ and there a bounded number of iterations.

Example 11.18. Solve the 15×15 pentadiagonal system $\begin{bmatrix} 6 & -4 & 1 & & \\ -4 & 6 & -4 & \ddots & \\ 1 & -4 & \ddots & \ddots & 1 \\ & \ddots & \ddots & 6 & -4 \\ & & 1 & -4 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{14} \\ x_{15} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$ and perform iterative improvement. Assume the solution obtained using the LU factorization is

$$x_1 = [\ 20.00 \quad 52.49 \quad 91.00 \quad 130.00 \quad 165.00 \quad 192.51 \quad 210.00 \quad 216.00 \quad 210.00 \quad 192.51 \quad 165.00 \quad 130.00 \quad 91.00 \quad 52.49 \quad 20.00\ ]^{\mathrm{T}},$$

whereas the true solution correct to two decimal places is

$$x = [\ 20.00 \quad 52.50 \quad 91.00 \quad 130.00 \quad 165.00 \quad 192.50 \quad 210.00 \quad 216.00 \quad 210.00 \quad 192.50 \quad 165.00 \quad 130.00 \quad 91.00 \quad 52.50 \quad 20.00\ ]^{\mathrm{T}}.$$

The true residual is $\|b - Ax\|_2 = 0$, and residual for the approximate solution is $\|b - Ax_1\|_2 = 0.166$. Apply iterative improvement using x_1, $tol = 1 \times 10^{-5}$ and $numiter = 2$.

```
>> [xnew, iter] = iterimp(A,L,U,P,b,x1,1.0e-5,5);
>> norm(b - A*xnew)

ans =
    3.2132e-13
```

The matrix has a condition number of approximately 2611, so with an ill-conditioned matrix we obtained improvement. ■

11.10 CHAPTER SUMMARY

The *LU* Decomposition

Without doing row exchanges, the actions involved in factoring a square matrix A into a product of a lower-triangular matrix, L, and an upper-triangular matrix, U, is simple. Assign L to be the identity matrix. Perform Gaussian elimination on A in order to reduce it to upper-triangular form. Assume we are ready to eliminate elements below the pivot element a_{ii}, $1 \le i \le n-1$. The multipliers used are

$$\frac{a_{i+1,i}}{a_{ii}}, \frac{a_{i+2,i}}{a_{ii}}, \ldots, \frac{a_{ni}}{a_{ii}}.$$

Place these multipliers in L at locations $(i+1, i), (i+2, i), \ldots, (n, i)$. When the row reduction is complete, A is matrix U, and $A = LU$.

Using the *LU* to Solve Equations

After performing the decomposition $A = LU$, consider solving the system $Ax = b$. Substitute LU for A to obtain

$$\begin{aligned} LUx &= b, \\ L(Ux) &= b. \end{aligned}$$

Consider $y = Ux$ to be the unknown and solve

$$Ly = b$$

using forward substitution. Now solve

$$Ux = y$$

using back substitution.

Elementary Row Matrices

Let A be an $n \times n$ matrix. An elementary row matrix, E, is an alteration of the identity matrix such that EA performs one of the three elementary row operations. For instance, if

$$E_{31}(2) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix}$$

then $E_{31}A$ subtracts (2) times row 1 from row 3. The primary purpose of these matrices is to show why the LU decomposition works.

Derivation of the *LU* Decomposition

Use products of elementary row matrices to row reduce A to upper-triangular form to arrive at a product

$$E_k E_{k-1} \dots E_2 E_1 A = U,$$

and so

$$A = (E_k E_{k-1} \dots E_2)^{-1} U.$$

$(E_k E_{k-1} \dots E_2)^{-1}$ is precisely the matrix L.

An analysis shows that the flop count for the LU decomposition is $\approx \frac{2}{3}n^3$, so it is an expensive process.

Gaussian Elimination with Partial Pivoting

We use the pivot to eliminate elements $a_{i+1,i}, a_{i+2,i}, \dots, a_{ni}$. If the pivot, a_{ii}, is small the multipliers $a_{k,i}/a_{ii}, i+1 \leq k \leq n$, will likely be large. As we saw in Chapter 8, adding or subtracting large numbers from smaller ones can cause loss of any contribution from the smaller numbers. For this reason, begin find the maximum element in absolute value from the set $a_{ii}, a_{i+1,i}, a_{i+2,i}, \dots, a_{ni}$ and swap rows so the largest magnitude element is at position (i, i). Proceed with elimination in column i. The end result is a decomposition of the form $PA = LU$, where P is a permutation matrix that accounts for any row exchanges that occurred. This can be justified by an analysis using elementary row matrices.

Computing the *LU* Decomposition Once and Solving $Ax_i = b_i$, $1 \leq i \leq k$

A great advantage of performing the LU decomposition is that if the system must be solved for multiple right-hand sides, the $O(n^3)$ LU decomposition need only be performed once, as follows:

$$\begin{aligned} Ax &= b_i, \quad 1 \leq i \leq k, \\ PAx &= Pb_i, \\ LUx &= Pb_i, \\ L(Ux) &= Pb_i. \end{aligned}$$

Now solve $L(Ux_i) = Pb_i, 1 \leq i \leq k$ using forward and back substitution. The cost of the decomposition is $O(n^3)$, and the cost of the solutions using forward and back substitution is $O(kn^2)$. If we solved each system using Gaussian elimination, the cost would be $O(kn^3)$.

Computing A^{-1}

Conceptually, computing A^{-1} is simple. Apply the LU decomposition to obtain $PA = LU$, and use it to solve systems having as right-hand sides the standard basis vectors

$$e_1 = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \end{bmatrix}^{\mathrm{T}}, e_2 = \begin{bmatrix} 0 & 1 & \dots & 0 & 0 \end{bmatrix}^{\mathrm{T}}, \dots, \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix}^{\mathrm{T}}.$$

The solutions form the columns of A^{-1}. It should be emphasized that computing A^{-1} is expensive and roundoff error builds up.

Stability and Efficiency of Gaussian Elimination

There are instances where GEPP fails (see Problem 11.36), but these examples are pathological. None of these situations has occurred in 50 years of computation using GEPP. There is a method known as complete pivoting that involves exchanging both rows and columns. It is more expensive than GEPP and is not used often.

Iterative Refinement

If a solution to $Ax = b$ is not accurate enough, it is possible to improve the solution using iterative refinement. Let $\bar{x}$ be the computed solution of the system $Ax = b$. If $x = \bar{x} + \delta x$ is the exact solution, then $Ax = A\bar{x} + A(\delta x) = b$, and $A(\delta x) = b - A\bar{x} = r$, the residual. If we solve the system $A(\delta x) = r$ for δx, then $Ax = A\bar{x} + A(\delta x) = A\bar{x} + r = A\bar{x} + b - A\bar{x} = b$. It is unlikely that we will obtain an exact solution to $A(\delta x) = r$; however, $\bar{x} + \delta x$ might be better approximation to the true solution than $\bar{x}$. For this to be true, it is necessary to compute the residual r using twice the precision of the original computations; for instance, if the computation of $\bar{x}$ was done using 32-bit floating point precision, then the residual should be computed using 64-bit precision. This process provides a basis for an iteration that continues until we reach a desired relative accuracy or fail to do so. Unless the matrix is very poorly conditioned, the computed solution x is already close to the true solution, so only a few iterations are required. If the matrix has a large condition number, it is not reasonable to expect huge improvement

11.11 PROBLEMS

Note: The term "step-by-step" can involve computational assistance. For instance, suppose you want to subtract a multiple, t, of row i from row j of matrix A. The following statement will do this for you.

```
>> A(j,:) = A(j,:) - t*A(i,:)
```

This type of computation is essentially "by pencil and paper," except you do not have to perform the row elimination by hand or with a calculator.

11.1 Given $A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 3 & 5 & 4 \end{bmatrix}$, find the LU factorization of A step-by-step without pivoting.

11.2 Show that $A = \begin{bmatrix} 6 & 1 \\ -6 & 2 \end{bmatrix}$ is nonsingular, find A^{-1} and express A as a product of elementary row matrices.

11.3 What two elementary row matrices $E_{21}(t_1)$ and $E_{32}(t_2)$ put $A = \begin{bmatrix} 2 & 1 & 0 \\ 6 & 4 & 2 \\ 0 & 3 & 5 \end{bmatrix}$ into upper-triangular form $E_{21}(t_2) E_{32}(t_1) A = U$? Multiply by $E_{21}^{-1}(t_2) E_{32}^{-1}(t_1) = L$ to factor A into $LU = E_{21}^{-1}(t_1) E_{32}^{-1}(t_2) U$.

11.4 Find the 3×3 matrix $A = E_2(5) E_{31}(2) E_{13}$? Also find A^{-1}.

11.5 Prove that multiplying a matrix A on the right by E_{ij} exchanges columns i and j of A.

11.6 This problem is taken from [8, p. 106]. Using matrix A, suppose you eliminate upwards (almost unheard of). Use the last row to produce zeros in the last column (the pivot is 1). Then use the second row to produce zero above the second pivot. Find the factors in the unusual order $A = UL$.

$$A = \begin{bmatrix} 5 & 3 & 1 \\ 3 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

11.7 If A is singular, and $PA = LU$ is the LU factorization, prove there must be at least one zero on the diagonal of U.

11.8 In the LU factorization of a square matrix, show that

$$\begin{aligned} \|L\|_F &\leq \frac{n(n+1)}{2}, \quad \|L\|_\infty \leq n, \\ \|L\|_1 &\leq n \quad \text{and} \quad \|L\|_2 \leq n. \end{aligned}$$

11.9 Prove Theorem 11.1 for the elementary row matrices $E_{ij}(t)$ and $E_i(t)$.

11.10 Show that during GECP, search for the new pivot element requires $O\left(n^3\right)$ comparisons. After the new pivot element is selected, explain why it may be necessary to exchange both rows and columns.

11.11 Solve the system

$$\begin{bmatrix} 0.00005 & 1 & 1 \\ 2 & -1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}$$

step-by-step with and without partial pivoting using 4 decimal place accuracy. Compare the results.

An alternative algorithm for *LU* factorization is *Crout's Method*. There is a variation of the algorithm with pivoting, but we will not include it here. The elements of L and U are determined using formulas that are easily programmed. In Crout's method, U is the matrix with ones on the diagonal, as indicated in Equation 11.7. Problems 11.12–11.15 develop the method.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix} \begin{bmatrix} 1 & U_{12} & U_{13} & U_{14} \\ 0 & 1 & U_{23} & U_{24} \\ 0 & 0 & 1 & U_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{11.7}$$

11.12 Form the product of the two matrices on the right-hand side of Equation 11.7.

11.13 The entries of the matrices L and U can be determined by equating the two sides of the result from Problem 11.12 and computing the L_{ij} and U_{ij} entries row by row; for instance, $L_{11} = a_{11}$. Once L_{11} is known, the values of U_{12}, U_{13}, and U_{14} are calculated by

$$U_{12} = \frac{a_{12}}{L_{11}}, \quad U_{13} = \frac{a_{13}}{L_{11}}, \quad U_{14} = \frac{a_{14}}{L_{11}}.$$

Continue and as far as necessary until you can state a general formula for the elements of L and U for an $n \times n$ matrix A.

11.14 Using Crout's Method, factor the following 3×3 matrix step-by-step.

a. $A = \begin{bmatrix} 1 & -2 & -1 \\ 2 & 0 & 3 \\ 1 & 5 & 0 \end{bmatrix}$

b. Use the factorization from part to solve the system $Ax = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$.

11.15 In this problem, you will use the results of Problem 11.13 to develop an algorithm for the Crout Method. The algorithm involves four steps. Fill-in the missing elements of each step.

```
function CROUT(A)
   Calculate the first column of L.
   Place 1s on the diagonal of U.
   for i = 1:n do
```

```
   end for
   Calculate the elements in the first row of U. U11 already calculated.
   for j = 2:n do
```

```
   end for
   Calculate the rest of the elements row after row.
   The entries of L are calculated first because they are used for
   calculating the elements of U.
   for i = 2:n do
      for j = 2:i do
```

```
      end for
      for j = i+1:n do
```

```
    end for
  end for
  return [ L  U ]
end function
```

11.16 It is possible to determine that a matrix is singular during GEPP. Specify a condition that occurs while executing the algorithm that will indicate the matrix is singular.

11.17 There are situations where the LU decomposition must be done using pivoting.

a. For the matrix $\begin{bmatrix} 1 & 7 & 1 \\ 9 & -1 & 2 \\ 3 & 5 & 1 \end{bmatrix}$, compute the determinant of the submatrices $\begin{bmatrix} 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 7 \\ 9 & -1 \end{bmatrix}$, and the whole matrix. Can you do an LU factorization without pivoting?

b. Perform part (a) with the matrix $\begin{bmatrix} 1 & 3 & 1 \\ -1 & -3 & 5 \\ 6 & 3 & 9 \end{bmatrix}$.

c. For the matrix $\begin{bmatrix} 1 & 2 & -1 & 1 \\ 3 & 2 & -7 & 2 \\ 1 & 8 & 5 & 3 \\ 1 & -9 & 2 & 5 \end{bmatrix}$, calculate all determinants of submatrices with upper left-hand corner at a_{11}. Can you do an LU decomposition without pivoting?

d. Propose a theorem that specifies when an LU decomposition without pivoting is possible. You do not have to prove it.

11.18 Prove that a permutation matrix is orthogonal.

11.19 If A is an $n \times n$ matrix whose LU decomposition is $PA = LU$, prove that

$$\det(A) = (-1)^r u_{11} u_{22} \dots u_{nn},$$

where r is the number of row interchanges performed during row reduction, and u_{ii} are the diagonal elements of U.

11.20 A is *strictly column diagonally dominant* if $|a_{ii}| > \sum_{k=1, k\neq i}^{n} |a_{ki}|$. This problem investigates the LU decomposition of such a matrix.

a. Determine which matrices are strictly column diagonally dominant:

i. $\begin{bmatrix} 3 & 2 & 2 \\ -2 & 6 & 2 \\ 1 & -1 & 5 \end{bmatrix}$

ii. $\begin{bmatrix} -1 & 1 & 7 & -0.05 \\ 0.3 & 2.5 & -1 & -0.7 \\ 0.2 & 0.75 & -9 & 0.04 \\ 0.4 & 0.70 & 0.9 & 0.8 \end{bmatrix}$

iii. $\begin{bmatrix} 2 & 0.5 & & & \\ 0.5 & 2 & 0.5 & & \\ & & \ddots & & \\ & & 0.5 & 2 & 0.5 \\ & & & 0.5 & 2 \end{bmatrix}$

b. The matrix $A = \begin{bmatrix} 3 & 3 & 1 \\ 1 & 5 & 2 \\ 1 & 1 & -4 \end{bmatrix}$ is strictly column diagonally dominant. Using pencil and paper, form the LU decomposition with partial pivoting to show that no row interchanges are necessary.

c. Prove that if A is strictly column diagonally dominant, the LU decomposition with partial pivoting requires no row interchanges. Hint: Argue that row elimination in column 1 requires no row exchanges. Now consider the matrix $A^{(1)}$ that results from action with column 1. Show it is strictly column diagonally dominant by using the diagonal dominance of A. The result follows by induction.

11.21 The LDU decomposition is a variation of the LU decomposition. L is a lower diagonal matrix all of whose diagonal entries are 1, D is a diagonal matrix whose elements are the pivots, and U is a unit upper-triangular matrix. Without

pivoting, $A = LDU$, and with pivoting $PA = LDU$. In this problem, assume that the LU decomposition uses no pivoting.

a. Show that the LDU decomposition exists when A is nonsingular.

b. Show that if A is symmetric and nonsingular, then $A = LDU$, where $L = U^{\mathrm{T}}$ and $U = L^{\mathrm{T}}$.

c. Is (b) true for the standard LU decomposition?

11.22 There are some matrices whose inverses can be computed easily.

a. Prove that the inverse of a diagonal matrix, D, with nonzero diagonal entries $d_1, d_2, \ldots, d_n$ is the diagonal matrix whose diagonal elements are $1/d_1, 1/d_2, \ldots, 1/d_n$.

b. We will have occasion to use Householder reflections later in the text when we discuss efficient algorithms for computing the QR decomposition. A Householder reflection is formed from a nonzero vector u as follows:

$$H_u = I - \frac{2uu^{\mathrm{T}}}{u^{\mathrm{T}}u}, \quad u \neq 0.$$

Show that $H_u^2 = I$

c. An atomic lower-triangular matrix is a special form of a unit lower-triangular matrix, also called a *Gauss transformation matrix*. All of the off-diagonal entries are zero, except for the entries in a single column. It has the form

$$L = \begin{bmatrix} 1 & & & & & & & \\ 0 & \ddots & & & & & & \\ 0 & & 1 & & & & & \\ 0 & & & 1 & & & & \\ & & & l_{i+1,i} & 1 & & & \\ \vdots & & & l_{i+2,i} & & \ddots & & \\ & & & \vdots & & & 1 & \\ 0 & \ldots & 0 & l_{ni} & 0 & \ldots & 0 & 1 \end{bmatrix}.$$

Show that

$$L^{-1} = \begin{bmatrix} 1 & & & & & & & \\ 0 & \ddots & & & & & & \\ 0 & & 1 & & & & & \\ 0 & & & 1 & & & & \\ & & & -l_{i+1,i} & 1 & & & \\ \vdots & & & -l_{i+2,i} & & \ddots & & \\ & & & \vdots & & & 1 & \\ 0 & \ldots & 0 & -l_{ni} & 0 & \ldots & 0 & 1 \end{bmatrix}.$$

11.23 Consider the system

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 1 & 2 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 4 \\ 2 \end{bmatrix}.$$

Assuming an initial approximation $\bar{x} = \begin{bmatrix} 2 \\ 0 \\ -3 \end{bmatrix}$, perform one iterative improvement iteration using step-by-step, retaining four significant digits. Compare your result to the actual solution.

11.24 Let $A = \begin{bmatrix} 1 & 7 & -1 \\ 2 & 4 & 5 \\ 1 & 2 & -1 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$. Compute the solution x using MATLAB. Let $\bar{x} = \begin{bmatrix} 3 \\ -1 \\ -1 \end{bmatrix}$. Perform one step of iterative improvement with four significant digits, obtain approximate solution $\bar{x}$, and compare it to x.

11.25 If in the LU decomposition, $|l_{ij}| < 1$ for each $i > j$, show there are no ties in the selection of pivots in exact arithmetic.

11.26 Compute the growth factor for $A = \begin{bmatrix} 0.0006 & -8 \\ 1 & 1 \end{bmatrix}$ with and without partial pivoting.

11.27 When using GEPP, show that $\rho \leq 2^{n-1}$, where ρ is the growth factor.

11.11.1 MATLAB Problems

11.28 Let $A = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \end{bmatrix}$.

a. Find the decomposition $PA = LU$ for the matrix.

b. Solve the system $Ax = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^{\mathrm{T}}$ using the results of part (a).

c. Find the decomposition $PA = LU$ for the matrix $B = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$.

d. Solve the system $Bx = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^{\mathrm{T}}$ using the results of part (c).

11.29 Using GEPP, factor each matrix as $PA = LU$.

a. $A = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 3 \\ 5 & 8 & 2 \end{bmatrix}$

b. $A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 3 & 4 & 5 \end{bmatrix}$

11.30 Use `ludecomp` and `lusolve` to find the solution to the following three systems. Compute $\|B - A * X\|_2$, where B is the matrix of right-hand sides, and the solutions are the columns of X.

$$A = \begin{bmatrix} 1 & 6 & 2 & 1 \\ 2 & 2 & 8 & 9 \\ 12 & 5 & 1 & 9 \\ -1 & -7 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = b_i, \text{ where } b_1 = \begin{bmatrix} 1 \\ 8 \\ 0 \\ -1 \end{bmatrix}, b_2 = \begin{bmatrix} 5 \\ 12 \\ 1 \\ -12 \end{bmatrix}, \text{ and } b_3 = \begin{bmatrix} 5 \\ 88 \\ 15 \\ 3 \end{bmatrix}$$

11.31 When a matrix T is tridiagonal, its L and U factors have only two nonzero diagonals. Factor each matrix into $A = LU$ without pivoting.

a. $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$

b. $\begin{bmatrix} 1 & -1 & 0 \\ 2 & 3 & 1 \\ 0 & 4 & 5 \end{bmatrix}$

c. $\begin{bmatrix} 1 & 2 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 3 & 4 \end{bmatrix}$

11.32 Write MATLAB functions to create the elementary row matrices, E_{ij}, $E_i(t)$, and $E_{ij}(t)$. Use the functions to verify your results for Problem 11.4.

11.33 **a.** Write a MATLAB function

```
function [L U] = crout(A)
```

that implements Crout's method developed in Problem 11.15.

b. Factor the matrix in Example 11.9. Apply your function to random 3×3, 5×5, and 25×25 matrices. Test the result in each case by computing $\|A - LU\|_2$.

11.34 **a.** Write

```
function x = gauss(A,b)
```

that uses Gaussian elimination without pivoting to solve the $n \times n$ system $Ax = b$. Note that the function does not perform the LU decomposition but just applies straightforward row operations as explained in Chapter 2. If a pivot element is zero, print an error message and exit.

b. Use your function to solve three random systems of size 3×3, 4×4, and 10×10.

11.35 Make a copy of the function ludecomp, name it `matgr`, and make modifications so it computes the growth factor of a square matrix using partial pivoting. Test it with the matrix in Problem 11.26 and the matrix

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}.$$

11.36 We have stated that Gaussian elimination with partial pivoting is stable in practice, but that contrived examples can cause it to become unstable. This problem uses a matrix pattern devised by *Wilkinson* in his definitive work [9].

a. Write a MATLAB function `function W = wilkpivot(n)` that returns an $n \times n$ matrix of the form

$$\begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & -1 & \dots & 1 \\ -1 & -1 & -1 & \dots & -1 & 1 \end{bmatrix}$$

Your function requires no more than four MATLAB commands.

b. Will any row interchanges occur when solving a system with this coefficient matrix?

c. Using your function `wilkpivot`, show that a Wilkinson 5×5 matrix has the *LU* decomposition

$$\begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ -1 & -1 & 1 & & \\ -1 & -1 & -1 & 1 & \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{bmatrix}.$$

d. What is the value of $U(50, 50)$ in the *LU* decomposition of a 50×50 Wilkinson matrix, W50?

e. Using the matrix from part (d), form the vector `b = [1:25 -1:-1:-25]'` and compute `x = A\b`. Perturb `b` slightly using the statements

```
b(25) = b(25) - 0.00001;
b(30) = b(30) + 0.00001
```

and compute `xbar = A\b`.

f. Evaluate $\|x - \bar{x}\|_\infty / \|x\|_\infty$

g. What is the matrix growth factor? Explain the result of part (f) by considering the matrix growth factor.

11.37 Using row elimination with partial pivoting, write and test a MATLAB function `determinant(A)` that computes the determinant of matrix A and test it with matrices up to size 20×20.

11.38 Problem 11.20(c) required a proof that a strictly column diagonally dominant matrix requires no row interchanges during the *LU* decomposition. Use the MATLAB function `trid` in the software distribution to build the matrix

$$A = \begin{bmatrix} 1 & -0.25 & & & \\ -0.25 & 1 & -0.25 & & \\ & -0.25 & \ddots & \ddots & \\ & & \ddots & 1 & -0.25 \\ & & & -0.25 & 1 \end{bmatrix}$$

for $n = 5, 10, 25, 100$. For each n, demonstrate that no row interchanges are necessary when forming the LU decomposition. The function `ludecomp` returns the number of interchanges required as well as L and U:

```
[L, U, P, interchanges] = ludecomp(A)
```

11.39 Write and test a MATLAB function `ldu(A)` that computes the LDU decomposition of A as described in Problem 11.21.

11.40 Write and test the function `inverse(A)` that computes A^{-1} using the LU decomposition with partial pivoting. Include hilb(10), the 10×10 Hilbert matrix, in your tests.

11.41 A *band matrix* is a sparse matrix whose nonzero entries are confined to a diagonal band, consisting of the main diagonal and zero or more diagonals on either side. A tri- or bidiagonal matrix is a band matrix. Band matrices occur in applications, particularly when finding approximate solutions to partial differential equations. In many such applications, the banded system to be solved is very large. As we have emphasized earlier, it pays to take advantage of matrix structure, particularly when dealing with very large matrices. We will see in Chapter 13 that the LU decomposition of a tridiagonal matrix, T, requires $O(n)$ flops, where L is unit lower bidiagonal, and U is upper bidiagonal with its superdiagonal the same as that of T. Since $T^{-1} = U^{-1}L^{-1}$, and there are efficient algorithms for the inverse of a bidiagonal matrix (see Refs. [32, pp. 151-152] and [39, pp.248-249]), it would seem to be more efficient to compute $U^{-1}L^{-1}$ rather than executing a straightforward evaluation of A^{-1}. Although we will not discuss it here, there are formulas for the direct calculation of T^{-1} [39, 40].

The book software contains a function `tridiagLU` you call as follows:

```
% a, b, c are the subdiagonal, diagonal, and superdigonal of a tridiagonal matrix.

% L is the lower diagonal of the unit bidiagonal matrix, and U is the diagonal of the
upper bidiagonal matrix.

[L,U] = tridiagLU(a,b,c);
```

a. Let a, b, c be the subdiagonal, diagonal, and superdiagonal of the matrix in Problem 11.38 with $n = 5$. Use `tridiagLU` to compute `L` and `U`. Verify that `tridiagLU` returned the proper diagonals.

b. The book software distribution contains functions `upbinv` and `lobinv` that compute the inverse of an upper bidiagonal and a unit lower bidiagonal matrix, respectively. Scan through the source code and determine how to call the functions. Write a function, `trinv`, with arguments a, b, c that computes the inverse of a tridiagonal matrix using `upbinv` and `lbinv`. Test it thoroughly.

c. Build vectors

```
>> a = randn(499,1);
>> b = randn(500,1);
>> c = randn(499,1);
```

and execute the statements

```
>> A = trid(a,b,c);
>> tic;P1 = inv(A);toc;
>> tic;P2 = trinv(a,b,c);toc;
```

Why is `inv` faster than `trinv`?

d. Using the function inverse developed in Problem 11.40, execute the statements

```
tic;P1 = inverse(A);toc;
tic;P2 = trinv(a,b,c);toc;
```

Explain the results.

11.42 The MATLAB function `single(A)` converts each element of the array A from 64-bit to 32-bit precision. Create the following 20×20 pentadiagonal system

$$T = \begin{bmatrix} 6 & -4 & 1 & & & 0 \\ -4 & 6 & -4 & 1 & \ddots & \\ 1 & -4 & 6 & -4 & 1 & \ddots \\ \vdots & \ddots & & & & \ddots \\ \dots & 1 & -4 & 6 & -4 & 1 \\ \vdots & & & & & \ddots \\ 0 & & \dots & & 1 & -4 \quad 6 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{19} \\ x_{20} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

using the function `pentd` in the book software distribution. Run the following MATLAB program. Explain what the program does and the results.

```
[L,U,P] = lu(T);
b1 = ones(20,1);
x = lusolve(L,U,P,b1);

C = single(T);
b2 = single(b1);
[L1,U1,P] = lu(C);
x1 = lusolve(L1,U1,P,b2);
fprintf('norm(x - single precision solution)/norm(x) = %g\n',...
         norm(x-x1)/norm(x));

[x2,iter] = iterimp(T,double(L1),double(U1),P,b1,...
                    double(x1),1.0e-12,10);
fprintf('norm(x - refined solution)/norm(x) = %g, requiring %d iterations\n',...
         norm(x-x2)/norm(x),iter);
```

11.43

a. Write a function, `makebidiag`, that takes a vector a with n elements and a vector b with $n - 1$ elements and builds an upper bidiagonal matrix.

b. Build a random 5×5 bidiagonal matrix with no zeros on either diagonal. Compute its singular values in vector `S` using the MATLAB function `svd`. Show that the singular values are unique using the MATLAB command "`numunique = length(unique(S));`." Repeat this experiment for random 10×10, 25×25, and 50×50 bidiagonal matrices. Formally state a theorem that describes the behavior.

c. It is true that a symmetric tridiagonal matrix for which $a_{i,i+1} = a_{i,i-1} \neq 0$ has n distinct eigenvalues. Using this fact, prove the theorem you stated in part (b).