applications involving large matrices (or small computers), storage requirements can sometimes present problems. In Problem 1 if A, B, and C are all saved, then $3n^2$ locations are needed. Suppose that A is no longer needed once the product C has been formed. How might the program in Problem 1 be modified so that only $2n^2 + n$ locations are needed to form C? [Hint: Once the first row of C is formed, the first row of A is no longer needed.]

3. How many multiplications must be performed to form the product of two $(n \times n)$ matrices?

4. How many multiplications must be performed to form the product $L\mathbf{x}$ when L is an $(n \times n)$ lower triangular matrix and $\mathbf{x}$ is an $(n \times 1)$ column vector? (Do not count multiplications by zero.)

*5. Write a computer program to evaluate the determinant of an arbitrary (5×5) matrix, using a cofactor expansion along the first row. (This is a fairly challenging programming problem.)

6. How many multiplications must be performed to evaluate the determinant of an $(n \times n)$ matrix according to the procedure of Problem 5 when $n = 3$, when $n = 4$, when $n = 5$, and for arbitrary n? (For $n = 10$, more than 3,000,000 multiplications are required. If a person were able to multiply two numbers and record the result at a rate of one per second, it would require 126 eight-hour days to find the determinant of a (10×10) matrix using a cofactor expansion.)

7. Let $L = (\ell_{ij})$ be a lower triangular (5×5) matrix such that $\ell_{11}\ell_{22}\ell_{33}\ell_{44}\ell_{55} \neq 0$. By considering the equation $L\mathbf{x} = \theta$, show that L is nonsingular. [Hint: Testing $L\mathbf{x} = \theta$ is criterion (1) for nonsingularity.] Extend your proof to an arbitrary $(n \times n)$ lower triangular matrix L such that $\ell_{11}\ell_{22} \cdots \ell_{nn} \neq 0$.

8. Let L be as in Problem 7, but this time suppose that $\ell_{44} = 0$ and $\ell_{55} \neq 0$. Show there is a nonzero vector $\mathbf{x}$ such that $L\mathbf{x} = \theta$.

*9. Let $L = (\ell_{ij})$ be an $(n \times n)$ lower triangular matrix. Show that L is singular if and only if $\ell_{11}\ell_{22} \cdots \ell_{nn} = 0$.

10. Let T and S be $(n \times n)$ upper triangular matrices. Use the definition of matrix multiplication to show that the product ST is also upper triangular.

*11. Let A be a lower (upper) triangular nonsingular matrix. Show that A^{-1} is also lower (upper) triangular. [Hint: Consider the equation $AA^{-1} = I$, entry by entry.]

12. If A and B are $(n \times n)$ nonsingular matrices, show that AB and BA are also nonsingular. Furthermore, show that $(AB)^{-1} = B^{-1}A^{-1}$.

*13. a) If A is $(r \times s)$ and B is $(r \times s)$, show that $(A + B)^{\mathrm{T}} = A^{\mathrm{T}} + B^{\mathrm{T}}$.

b) If A is $(r \times s)$ and B is $(s \times p)$, show that $(AB)^{\mathrm{T}} = B^{\mathrm{T}}A^{\mathrm{T}}$.

c) Show $(A^{\mathrm{T}})^{\mathrm{T}} = A$.

2.2 DIRECT METHODS

The term *direct method* refers to a numerical procedure that can be executed in a finite number of steps. Direct methods are in contrast to *iterative methods*, which generate an infinite sequence of approximations that (it is hoped) converge to the solution. We will discuss iterative methods in Section 2.4. The two

most common types of direct methods for solving the system (2.1) are elimination methods and factorization methods.

2.2.1. Gauss Elimination

Gauss elimination is the familiar variable elimination technique whereby the variables are eliminated one at a time to reduce the original system to an equivalent triangular system. The first step in this procedure is to replace the ith equation by the equation that is the result of multiplying the first equation by $(-a_{i1}/a_{11})$ and adding it to the original ith equation. Proceeding thus for $i = 2, 3, \ldots, n$, we obtain a system of equations equivalent to (2.1):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)} \\ &\vdots \\ a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n &= b_n^{(1)}. \end{aligned} \tag{2.5}$$

(We assume here that $a_{11} \neq 0$. If $a_{11} = 0$, we find an $a_{i1} \neq 0$, interchange the first and ith rows, and proceed in the same fashion.) We continue in this manner until the system is in an equivalent triangular form:

$$\begin{aligned} a'_{11}x_1 + a'_{12}x_2 + a'_{13}x_3 + \cdots + a'_{1,\,n-1}x_{n-1} + a'_{1n}x_n &= b'_1 \\ a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2,\,n-1}x_{n-1} + a'_{2n}x_n &= b'_2 \\ a'_{33}x_3 + \cdots + a'_{3,\,n-1}x_{n-1} + a'_{3n}x_n &= b'_3 \\ &\vdots \\ a'_{n-1,\,n-1}x_{n-1} + a'_{n-1,\,n}x_n &= b'_{n-1} \\ a'_{n,\,n}x_n &= b'_n. \end{aligned} \tag{2.6}$$

The solution to this triangular system is now easily found by *backsolving*; that is, by solving the last equation for x_n, then the $(n-1)$st equation for x_{n-1}, and continuing until each x_k where $k = n, n-1, n-2, \ldots 1$, is determined.

EXAMPLE 2.6. As a specific case to demonstrate Gauss elimination, consider the linear system

$$\begin{aligned} 5x_1 + 7x_2 + 6x_3 + 5x_4 &= 23 \\ 7x_1 + 10x_2 + 8x_3 + 7x_4 &= 32 \\ 6x_1 + 8x_2 + 10x_3 + 9x_4 &= 33 \\ 5x_1 + 7x_2 + 9x_3 + 10x_4 &= 31. \end{aligned}$$

Proceeding as indicated by (2.5), multiply the first equation by $-7/5$ and add the result to the second equation to eliminate x_1 in the new second equation. Proceeding similarly to eliminate x_1 from the third and fourth equations, we obtain an equivalent reduced linear system

$$\begin{aligned} 5x_1 + 7x_2 + 6x_3 + 5x_4 &= 23 \\ 0.2x_2 - 0.4x_3 &= -0.2 \\ -0.4x_2 + 2.8x_3 + 3x_4 &= 5.4 \\ 3x_3 + 5x_4 &= 8 \end{aligned}$$

that corresponds to (2.5). Eliminating x_2 from the third equation gives the equivalent linear system

$$\begin{aligned} 5x_1 + 7x_2 + 6x_3 + 5x_4 &= 23 \\ 0.2x_2 - 0.4x_3 &= -0.2 \\ 2x_3 + 3x_4 &= 5 \\ 3x_3 + 5x_4 &= 8. \end{aligned}$$

Finally by eliminating x_3 from the fourth equation, we obtain a linear system in the triangular form of (2.6):

$$\begin{aligned} 5x_1 + 7x_2 + 6x_3 + 5x_4 &= 23 \\ 0.2x_2 - 0.4x_3 &= -0.2 \\ 2x_3 + 3x_4 &= 5 \\ 0.5x_4 &= 0.5. \end{aligned}$$

Backsolving the triangular linear system, we find from the fourth equation that $x_4 = 1$. Because we have x_4, the third equation yields $x_3 = 1$; with x_3 and x_4 known, the second equation says $x_2 = 1$; and finally from the first equation, we obtain $x_1 = 1$. The coefficient matrix for the original system is

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}$$

This matrix is due to T.S. Wilson (see Todd, 1962). The matrix has several interesting properties and will serve as one example when we discuss topics such as error analysis and condition numbers.

Continuing this example of solution by variable elimination, consider the (3×4) linear system

$$\begin{aligned} 5x_1 + 7x_2 + 6x_3 + 5x_4 &= 23 \\ 7x_1 + 10x_2 + 8x_3 + 7x_4 &= 32 \\ 6x_1 + 8x_2 + 10x_3 + 9x_4 &= 33. \end{aligned}$$

Proceeding exactly as above, we arrive at the equivalent system

$$\begin{aligned} 5x_1 + 7x_2 + 6x_3 + 5x_4 &= 23 \\ 0.2x_2 - 0.4x_3 &= -0.2 \\ 2x_3 + 3x_4 &= 5. \end{aligned}$$

Here we see that either x_3 or x_4 can be chosen completely arbitrarily. For example, let x_4 be arbitrary; then $x_3 = (1/2)(5 - 3x_4)$. Continuing, we obtain $x_2 = 4 - 3x_4$ and $x_1 = 5x_4 - 4$, and we may substitute any value for x_4 and have a corresponding solution of the system. In particular if we let $x_4 = 1$, we obtain $x_3 = x_2 = x_1 = 1$ as above. This example of a (3×4) system shows how Gauss elimination can be used for rectangular systems as well as for square systems.

In the simple (4×4) example above, no row interchanges were necessary. When Gauss elimination is programmed for use on a digital computer, we will clearly have to include a statement to check whether we might be dividing by zero. Thus, given (2.1) to solve, the first step would be to test a_{11} to determine whether or not $a_{11} = 0$, and then to perform a row interchange if $a_{11} = 0$. Having

obtained the first reduced system (2.5), we would again test $a_{22}^{(1)}$ to see if $a_{22}^{(1)} = 0$ and perform a row interchange if necessary. Example 2.7 below illustrates the sort of numerical results that a simple Gauss elimination program will produce.

EXAMPLE 2.7. The basic steps of Gauss elimination that are outlined above were programmed to solve the system in Example 2.6. The program was executed in single precision; and as can be seen, the errors in the computed solution range from 0.000583 to 0.00009. For purposes of comparison, the final triangular form of the matrix was printed out as shown and can be compared with the exact triangular form displayed in Example 2.6.

SOLUTION VECTOR
0.999417E 00
0.100035E 01
0.100015E 01
0.999910E 00

THE TRIANGULARIZED MATRIX

0.500000E 01	0.700000E 01	0.600000E 01	0.500000E 01
0.000000E 00	0.200003E 00	−0.399998E 00	0.190735E-05
0.000000E 00	0.000000E 00	0.200002E 01	0.300000E 01
0.000000E 00	0.000000E 00	0.000000E 00	0.500038E 00

This computer example illustrates the effects of round-off error, and we note that this computer solution would probably not be satisfactory in most practical problems. As we shall see later, the coefficient matrix of Example 2.6 is moderately "ill-conditioned" (see Section 2.3.3). Thus even a few small computer-round-off errors will induce relatively large errors in the computed solution.

This program was executed again, in double precision, and gave the solution vector correct to as many places as are listed:

SOLUTION VECTOR
0.10000000D 01
0.10000000D 01
0.10000000D 01
0.10000000D 01

THE TRIANGULARIZED MATRIX

0.500000D 01	0.700000D 01	0.600000D 01	0.500000D 01
0.000000D 00	0.200000D 00	−0.400000D 00	0.444089D-15
0.000000D 00	0.000000D 00	0.200000D 01	0.300000D 01
0.000000D 00	0.000000D 00	0.000000D 00	0.500000D 00

Since linear systems of any appreciable size are normally solved on the computer, round-off errors can cause problems and the technique of Gauss elimination must be examined more carefully. In addition to the question of rounding errors, other aspects must be considered for practical and efficient implementation of numerical procedures on the machine. Three aspects that can influence the choice of an algorithm are storage requirements, round-off errors, and execution time.

2.2.2. Operations Counts

For the system (2.1), storage requirements are clearly linked to the size of the system, n. In order to give a feeling for the execution time and rounding errors of direct methods, an *operations count* is customary. An operations count is a calculation of the number of arithmetic operations necessary to carry out the direct method. One rationale for an operations count is that execution time and the error caused by round-off in the computed solution are both related to the total number of arithmetic operations. For Gauss elimination, multiplication and addition are (essentially) the only arithmetic operations performed. (We consider "division" as a multiplication and "subtraction" as an addition.) To illustrate how an operations count is made, we will count the multiplications necessary to solve the system (2.1) by Gauss elimination. The reader can quickly verify that the necessary number of additions is about the same as the number of multiplications.

To transform (2.1) to the equivalent system (2.5) requires $(n-1)(n+1)$ multiplications. This result is seen by noting that to eliminate the variable x_1 in the ith equation of (2.1), we multiply the first equation $a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$ by the scalar $-a_{i1}/a_{11}$ and add the resulting multiple of the first equation to the ith equation. It takes one multiplication to form the scalar $-a_{i1}/a_{11}$ (we don't count multiplications by -1) and an additional n multiplications to form each of the products.

$$\frac{-a_{i1}}{a_{11}}a_{12}, \frac{-a_{i1}}{a_{11}}a_{13}, \ldots, \frac{-a_{i1}}{a_{11}}a_{1n}, \qquad \frac{-a_{i1}}{a_{11}}b_1.$$

Note that there is no need to form the product $(-a_{i1}/a_{11})a_{11}$ since we *already* know we are going to have 0 as the scalar multiplying x_1 in the new ith equation. Thus multiplying the first equation by $-a_{i1}/a_{11}$ and adding the result to the ith equation to produce the new ith equation

$$a_{i2}^{(1)}x_2 + a_{i3}^{(1)}x_3 + \cdots + a_{in}^{(1)}x_n = b_i^{(1)} \tag{2.7}$$

takes $(n+1)$ multiplications (and n additions). We have to perform this variable elimination in rows $2, 3, \ldots, n$ and hence require a total of $(n-1)(n+1)$ multiplications to produce the equivalent reduced system (2.5).

Given the system (2.5), the same analysis shows that n multiplications are needed to eliminate x_2 in equations $3, 4, \ldots, n$ of (2.5). Thus the next step of Gauss elimination that results in

$$\begin{aligned}
a_{11}^{(2)}x_1 + a_{12}^{(2)}x_2 + a_{13}^{(2)}x_3 + \cdots + a_{1n}^{(2)}x_n &= b_1^{(2)} \\
a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \cdots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\
a_{33}^{(2)}x_3 + \cdots + a_{3n}^{(2)}x_n &= b_3^{(2)} \\
\vdots \qquad\qquad & \vdots \qquad \vdots \\
a_{n3}^{(2)}x_3 + \cdots + a_{nn}^{(2)}x_n &= b_n^{(2)}
\end{aligned} \tag{2.8}$$

takes $(n-2)n$ multiplications. Continuing until the triangular system (2.6) is obtained, we have performed a total of $(n-1)(n+1)+(n-2)n+\cdots+1\cdot 3$ multiplications. To evaluate this sum, we write it symbolically as

$$\sum_{j=1}^{n-1} j(j+2)=\sum_{j=1}^{n-1} j^2+2\sum_{j=1}^{n-1} j. \tag{2.9}$$

In this form, using the well-known formulas for the sum of the first $(n-1)$ integers and the sum of the first $(n-1)$ integers squared, we find that a total of

$$\frac{n(n-1)(2n-1)}{6}+n(n-1)=\frac{n(n-1)(2n+5)}{6} \tag{2.10}$$

multiplications are needed to produce an equivalent triangular system from a square $(n\times n)$ system.

That essentially $n^3/3$ multiplications are needed to triangularize (2.1) turns out to be quite an important observation. For the moment, however, we have not completed the operations count of Gauss elimination. We have yet to count the multiplications necessary to backsolve the triangular system (2.6). We leave it to the reader to show that $n(n+1)/2$ multiplications are required to solve (2.6). Thus the bulk of the computation involved in Gauss elimination is in triangularizing the coefficient matrix.

2.2.3. Pivoting and Scaling in Gauss Elimination

One modification of Gauss elimination is relatively easy to program and should in most cases reduce the effects of round-off error. To illustrate, we first consider the following example from Forsythe (1967); for simplicity, we consider a three-digit floating-decimal machine. [Recall, we assumed in Chapter 1 that on such a machine the arithmetic operations are done in a six-digit mode and then rounded to three digits. For example, this machine will record $(1-10000)$ as -10000.]

EXAMPLE 2.8.*

$$\begin{aligned} 0.0001x+1.00y&=1.00\\ 1.00x+1.00y&=2.00 \end{aligned}$$

The true solution rounded to five decimals is $x=1.00010$ and $y=0.99990$. If we proceed as above, without rearranging the equations, we obtain $-10000y=-10000$ and so we get $y=1.00$ and $x=0.00$, quite different from the true solution. However if we rewrite

*Reprinted with permission from G. E. Forsythe, "Today's computational methods of linear algebra," SIAM Review, Vol. 9, 1967. Copyright © 1967 by Society for Industrial and Applied Mathematics.

the system as

$$1.00x + 1.00y = 2.00$$
$$0.0001x + 1.00y = 1.00,$$

then we obtain the answer $x = 1.00$ and $y = 1.00$, a remarkable improvement

This example demonstrates one source of round-off error; namely, rounding errors can become an important factor in adding two numbers if one is much larger than the other, and the significant digits of the smaller one are essentially lost. A technique to compensate for this loss is called *pivoting* and is explained below. In ordinary Gauss elimination if $|a_{11}|$ is relatively small in comparison with some $|a_{i1}|$, then the multiplicative factor $(-a_{i1}/a_{11})$ is a relatively larger number in magnitude. Let $\lambda = -a_{i1}/a_{11}$; then the result of multiplying the first equation by λ and adding to the ith equation is the new ith equation

$$(a_{i2} + \lambda a_{12})x_2 + \cdots + (a_{in} + \lambda a_{1n})x_n = b_i + \lambda b_1.$$

If λ is large, then significant rounding errors can occur and can be quite harmful. In particular if λ is large, then adding a multiple of λ times the first equation to the ith equation may destroy most of the information in the ith equation (if λa_{1j} is large relative to a_{ij} for $2 \leq j \leq n$, the effect is almost the same as replacing the ith equation by a multiple of the first equation). If λa_{1j} is small relative to a_{ij}, rounding can still occur. However in this case the effect will be minimal in that the information in the ith equation will be left nearly intact.

With these ideas in mind, we modify the steps of Gauss elimination as follows. Find a row index I such that

$$|a_{I1}| = \max_{1 \leq i \leq n} |a_{i1}|,$$

and rewrite the system so that the Ith equation becomes the first equation. (Note that this altered system still has the same solution as the original system.) Now the multiplicative constant to eliminate the coefficients of x_1 is $(-a_{i1}/a_{I1})$, which is the relatively smallest factor we could obtain in this fashion. Thus the effect of the round-off error is reduced in the remaining equations. This process is repeated for each successive variable elimination until the triangularization is completed. This technique is known as *partial pivoting*.

Further reduction of round-off error can be accomplished in the following manner. As the first step, find $|a_{IJ}| = \max_{1 \leq i,\, j \leq n} |a_{ij}|$. Retain the Ith equation and eliminate the coefficient of x_J in the ith equation, $1 \leq i \leq n$, $i \neq I$, by replacing the ith equation by $(-a_{iJ}/a_{IJ})$ times the Ith equation plus the original ith equation. Repeat this process until the final system can be backsolved in a manner similar to that for a triangular system. This process, called *total pivoting*, reduces the effect of round-off error over partial pivoting but is more complicated to program since the order in which the variables are eliminated must be recorded.

In Fig. 2.1 we list a FORTRAN subroutine that uses Gauss elimination with partial pivoting to solve a square system $A\mathbf{x} = \mathbf{b}$. For convenience of computa-

```
      SUBROUTINE GAUSS(A,B,X,N,MAINDM,IERROR,RNORM)
      DIMENSION A(MAINDM,MAINDM),B(MAINDM),X(MAINDM)
      DIMENSION AUG(50,51)
      NM1=N-1
      NP1=N+1
C
C  SET UP THE AUGMENTED MATRIX FOR AX=B.
C
        DO 2 I=1,N
          DO 1 J=1,N
          AUG(I,J)=A(I,J)
    1     CONTINUE
        AUG(I,NP1)=B(I)
    2   CONTINUE
C
C  THE OUTER LOOP USES ELEMENTARY ROW OPERATIONS TO TRANSFORM
C  THE AUGMENTED MATRIX TO ECHELON FORM.
C
        DO 8 I=1,NM1
C
C  SEARCH FOR THE LARGEST ENTRY IN COLUMN I, ROWS I THROUGH N.
C  IPIVOT IS THE ROW INDEX OF THE LARGEST ENTRY.
C
        PIVOT=0.
          DO 3 J=I,N
          TEMP=ABS(AUG(J,I))
          IF(PIVOT.GE.TEMP)  GO TO 3
          PIVOT=TEMP
          IPIVOT=J
    3     CONTINUE
        IF(PIVOT.EQ.0.)  GO TO 13
        IF(IPIVOT.EQ.I)  GO TO  5
C
C  INTERCHANGE ROW I AND ROW IPIVOT.
C
          DO 4 K=I,NP1
          TEMP=AUG(I,K)
          AUG(I,K)=AUG(IPIVOT,K)
          AUG(IPIVOT,K)=TEMP
    4     CONTINUE
C
C  ZERO ENTRIES (I+1,I), (I+2,I),...,(N,I) IN THE AUGMENTED MATRIX.
C
    5   IP1=I+1
          DO 7 K=IP1,N
          Q=-AUG(K,I)/AUG(I,I)
          AUG(K,I)=0.
            DO 6 J=IP1,NP1
            AUG(K,J)=Q*AUG(I,J)+AUG(K,J)
    6       CONTINUE
    7     CONTINUE
    8   CONTINUE
      IF(AUG(N,N).EQ.0.)  GO TO 13
C
C  BACKSOLVE TO OBTAIN A SOLUTION TO AX=B.
C
      X(N)=AUG(N,NP1)/AUG(N,N)
        DO 10 K=1,NM1
        Q=0.
          DO 9 J=1,K
          Q=Q+AUG(N-K,NP1-J)*X(NP1-J)
    9     CONTINUE
        X(N-K)=(AUG(N-K,NP1)-Q)/AUG(N-K,N-K)
   10   CONTINUE
```

Figure 2.1 Subroutine GAUSS.

```
C
C  CALCULATE THE NORM OF THE RESIDUAL VECTOR, B-AX.
C  SET IERROR=1 AND RETURN.
C
      RSQ=0.
        DO 12 I=1,N
        Q=0.
          DO 11 J=1,N
          Q=Q+A(I,J)*X(J)
   11     CONTINUE
        RESI=B(I)-Q
        RMAG=ABS(RESI)
        RSQ=RSQ+RMAG**2
   12   CONTINUE
      RNORM=SQRT(RSQ)
      IERROR=1
      RETURN
C
C  ABNORMAL RETURN --- REDUCTION TO ECHELON FORM PRODUCES A ZERO
C  ENTRY ON THE DIAGONAL.  THE MATRIX A MAY BE SINGULAR.
C
   13 IERROR=2
      RETURN
      END
```

Figure 2.1 (continued)

tion we perform Gauss elimination on the *augmented* matrix $[A\vdots\mathbf{b}]$. [Recall that if A is $(n \times n)$, then the augmented matrix for the system $A\mathbf{x} = \mathbf{b}$ is the $(n \times (n + 1))$ matrix obtained by attaching the column $\mathbf{b}$ to the $(n \times n)$ array A so that $\mathbf{b}$ is the $(n + 1)$st column of $[A\vdots\mathbf{b}]$. Using Gauss elimination to transform A to triangular form means, in the language of linear algebra, that we are trying to reduce the augmented matrix $[A\vdots\mathbf{b}]$ to "echelon" form.] Like others listed later, the program in Fig. 2.1 was written with an emphasis on simplicity and readability, with a minimum of special features and options. Also, in the interests of clarity, we include only enough comments to highlight the main computational segments of the program.

The inputs required to use Subroutine GAUSS to solve an $(n \times n)$ system $A\mathbf{x} = \mathbf{b}$ are these: A, an $(N \times N)$ matrix; B, an $(N \times 1)$ vector; N, the size of the system $A\mathbf{x} = \mathbf{b}$; MAINDM, the declared dimension of the array A in the calling program. Subroutine GAUSS will return these: X, an $(N \times 1)$ array containing the machine solution to $A\mathbf{x} = \mathbf{b}$ if Gauss elimination is successful; IERROR, an error flag set to 1 if the elimination is successful and set to 2 if elimination cannot proceed because of a zero pivot; RNORM, a value that measures the size of the residual vector (we will discuss this in Section 2.3). Fig. 2.2 lists a simple program that reads in a system $A\mathbf{x} = \mathbf{b}$ and calls GAUSS to solve the system.

This section concludes with a brief discussion of a concept known as scaling. From the discussion above on pivoting one might expect that if the magnitudes of the elements of the coefficient matrix vary greatly in size, then the solution of $A\mathbf{x} = \mathbf{b}$ is more susceptible to rounding error. In practice this

```
      DIMENSION A(20,20),B(20),X(20)
      MAINDM=20
    1 READ 100,N
      IF(N.LE.1)  STOP
        DO 2 I=1,N
        READ 101,(A(I,J),J=1,N),B(I)
    2   CONTINUE
      CALL GAUSS(A,B,X,N,MAINDM,IERROR,RNORM)
      PRINT 102,IERROR
      IF(IERROR.EQ.2)  GO TO 1
      PRINT 103,RNORM
      PRINT 104,(X(I),I=1,N)
  100 FORMAT(I2)
  101 FORMAT(21F4.0)
  102 FORMAT(8H IERROR=,I3)
  103 FORMAT(7H RNORM=,E20.6)
  104 FORMAT(1H ,6E16.6)
      GO TO 1
      END
```

Figure 2.2

expectation is usually correct. This variation in magnitudes can be countered by multiplying the rows and columns of A by scaling constants to produce a matrix of the form $B = D_r A D_c$ where D_r and D_c are diagonal matrices for the row scaling and the column scaling, respectively, with the scaling constants as their diagonal elements. The system $A\mathbf{x} = \mathbf{b}$ is then equivalent to the system $(D_r A D_c)(D_c^{-1}\mathbf{x}) = D_r\mathbf{b}$, which is solved by successively solving $B\mathbf{z} = D_r\mathbf{b}$ and $\mathbf{x} = D_c\mathbf{z}$. The problem of constructing a practical computational scheme for scaling a matrix A is not well understood at this time, and an extensive discussion can be found in Young and Gregory (1973) and in the LINPACK User's Guide (1979). (LINPACK is an excellent library of mathematical software for numerical linear algebra.)

One frequently used approach to scaling is to divide each entry in each row by the largest (in magnitude) entry in that row. This approach uses row scaling only; so in the context of the discussion above, $D_c = I$. In particular, for each i, $1 \le i \le n$, define

$$d_i = \max_{1 \le j \le n} |a_{ij}|,$$

and let $1/d_i$ be the diagonal elements of D_r. This method produces $B = (b_{ij})$ where $B = D_r A$ and where $\max_{1 \le j \le n}|b_{ij}| = 1$ for $1 \le i \le n$. Unfortunately this technique can introduce rounding error in every element of B. An alternative, called *machine-base scaling*, is to replace $1/d_i$ by $1/b^{k_i}$ where b is the base of the machine arithmetic and k_i is an integer such that $d_i \simeq b^{k_i}$. The resulting divisions are performed exactly using an exponent shift so that no rounding

results (for example, on a hexadecimal machine such as an IBM 370, row i is scaled by a power of 16).

Another sort of scaling that is frequently employed is called *implicit pivoting*; here, the term "implicit" is used because the scaling multiplications are not actually performed, but rather the Gauss elimination algorithm selects the pivot element at each stage of the elimination as if scaling had taken place. In particular, let $A^{(k)} = (a_{ij}^{(k)})$ be the coefficient matrix at the kth step of Gauss elimination. For $A^{(k)}$, compute values $d_i^{(k)}$ as above, find the row index I that satisfies

$$\frac{|a_{Ik}^{(k)}|}{d_I^{(k)}} = \max_{k \le i \le n} \frac{|a_{ik}^{(k)}|}{d_i^{(k)}}.$$

We use row I as the pivot row; so we interchange row k with row I, b_k with b_I, and then proceed with the elimination as usual.

2.2.4. Implementation of Gauss Elimination and *LU*-decomposition

Besides pivoting, another important consideration is the efficient programming of Gauss elimination and we wish to mention one aspect of this general organizational problem. Frequently in practice, one is confronted with solving a succession of linear systems that have the same coefficient matrix. That is, solve

$$A\mathbf{x} = \mathbf{b}_i, \qquad i = 1, 2, \ldots, m. \tag{2.11}$$

(For example, such was the case when we were computing A^{-1} in the first section.) If we employ a Gauss elimination routine to solve, say, $A\mathbf{x} = \mathbf{b}_1$, then we first obtain an equivalent triangular system [such as (2.6)]

$$A'\mathbf{x} = \mathbf{b}'_1; \tag{2.12}$$

and this system is backsolved. If we next employ the same routine to solve $A\mathbf{x} = \mathbf{b}_2$, we obtain, as above,

$$A'\mathbf{x} = \mathbf{b}'_2; \tag{2.13}$$

and (2.13) is then backsolved.

A moment's reflection shows that instead of calling the same routine repeatedly to solve the m systems in (2.11), it would be better if we could store A' and just have a procedure to generate $\mathbf{b}'_i$ from $\mathbf{b}_i$. If such could be done, the approximately $n^3/3$ operations necessary to triangularize A would have to be performed only once. This programming problem can be solved by considering how the triangular system (2.6) is obtained from the original system (2.1). In particular as the system $A\mathbf{x} = \mathbf{b}$ is transformed to $A'\mathbf{x} = \mathbf{b}'$, the vector $\mathbf{b}$ is transformed to the vector $\mathbf{b}'$ by the same sequence of scalar multiples that

transforms A to the triangular matrix A'. Thus what is needed is some way to store the sequence of the scalar multiples of the triangularization. Rather than trying to elaborate abstractly on this method, we will give an example that makes the procedure clear.

EXAMPLE 2.9. Consider the linear system

$$\begin{aligned} x_1 + 2x_2 + x_3 &= b_1 \\ 3x_1 + 4x_2 &= b_2. \\ 2x_1 + 10x_2 + 4x_3 &= b_3 \end{aligned} \tag{2.14}$$

We are seeking the scalar multiples to transform $\mathbf{b}$ into $\mathbf{b}'$; so we first determine how A is transformed to A'. That is, we focus our attention upon how to triangularize the matrix

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 4 & 0 \\ 2 & 10 & 4 \end{bmatrix} \tag{2.15}$$

by adding multiples of one row to the next. Multiplying the first row by -3 and adding the result to the second row, then multiplying the first row by -2 and adding the result to the third row, we obtain

$$A^{(1)} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & -2 & -3 \\ 0 & 6 & 2 \end{bmatrix}, \qquad M_{21} = -3, \qquad M_{31} = -2 \tag{2.16}$$

where M_{21} and M_{31} are respective multiples needed to introduce zeros in the first column below the main diagonal. Continuing this simple example, if we multiply the second row of $A^{(1)}$ by 3 and add to the third row, we obtain A' where

$$A' = \begin{bmatrix} 1 & 2 & 1 \\ 0 & -2 & -3 \\ 0 & 0 & -7 \end{bmatrix}, \qquad M_{32} = 3. \tag{2.17}$$

If we had performed Gauss elimination on the system (2.14), the coefficient matrix of the equivalent triangular system would clearly be A' in (2.17).

Moreover, the multiples M_{21}, M_{31}, and M_{32} tell us how to transform $\mathbf{b}$. To be specific, we first obtain

$$\mathbf{b}^{(1)} = \begin{bmatrix} b_1 \\ b_2 + M_{21}b_1 \\ b_3 + M_{31}b_1 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}, \tag{2.18}$$

and finally

$$\mathbf{b}' = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} + M_{32}b_2^{(1)} \end{bmatrix} = \begin{bmatrix} b_1' \\ b_2' \\ b_3' \end{bmatrix}. \tag{2.19}$$

Clearly, in terms of efficient storage, we can overlay the scalar multiples M_{ij} in the zero entries of A' that they introduce during the triangularization.

Thus we proceed as indicated by the arrows:

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 4 & 0 \\ 2 & 10 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ -3 & -2 & -3 \\ -2 & 6 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ -3 & -2 & -3 \\ -2 & 3 & -7 \end{bmatrix} = A''. \tag{2.20}$$

(Note: no additional locations are needed to store M_{21}, M_{31}, and M_{32}.) Moreover, the positions of the multiples in A'' describe how they are to operate on vector $\mathbf{b}$ to transform $\mathbf{b}$ to $\mathbf{b}'$. As an illustration, consider

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 3 \\ 3x_1 + 4x_2 &= 3 \\ 2x_1 + 10x_2 + 4x_3 &= 10, \end{aligned} \tag{2.21}$$

which we know from above is equivalent to

$$\begin{aligned} x_1 + 2x_2 + x_3 &= b_1' \\ -2x_2 - 3x_3 &= b_2' \\ -7x_3 &= b_3' \end{aligned} \tag{2.22}$$

where we get the coefficients of (2.22) from the upper triangle portion of A''. To find b_1', b_2', b_3', we change $\mathbf{b}$ to $\mathbf{b}'$ by the sequence

$$\mathbf{b} = \begin{bmatrix} 3 \\ 3 \\ 10 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ 3 + (-3)\cdot 3 \\ 10 + (-2)\cdot 3 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ -6 \\ 4 + 3\cdot(-6) \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ -14 \end{bmatrix} \tag{2.23}$$

where the numbers below the main diagonal in A'' define the sequence. Thus from A'', we can read that (2.21) is equivalent to

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 3 \\ -2x_2 - 3x_3 &= -6 \\ -7x_3 &= -14; \end{aligned} \tag{2.24}$$

and backsolving, we obtain $x_3 = 2$, $x_2 = 0$, and $x_1 = 1$.

The basic ideas in Example 2.9 are fairly easy to program. One sees in (2.23) how solving several systems of the form $A\mathbf{x} = \mathbf{b}_k$, $1 \le k \le m$, can be accommodated by simple transformations of the vector on the right-hand side. In this simple example no row interchanges were necessary, but accounting for interchanges is also an easy bookkeeping task, one which we will be discussing shortly.

One might suppose if m in (2.11) is very large, that the most efficient way of solving the m linear systems, $A\mathbf{x} = \mathbf{b}_k$, $1 \le k \le m$, is first to calculate A^{-1} and then generate the kth solution vector, $\mathbf{y}_k = A^{-1}\mathbf{b}_k$, for each k. However since A^{-1} is an $(n \times n)$ matrix, it follows that n^2 multiplications are required to compute $A^{-1}\mathbf{b}_k$. By contrast, Problem 9 shows that a total of $n(n-1)/2$ multiplications are necessary to generate $\mathbf{b}_k'$ from $\mathbf{b}_k$ and a total of $n(n+1)/2$ multiplications are needed to solve $A'\mathbf{x} = \mathbf{b}_k'$. Thus if we know the triangular matrix A' and the $n(n-1)/2$ scalar multiples needed to generate $\mathbf{b}_k'$, we can solve $A\mathbf{x} = \mathbf{b}_k$ with precisely as many multiplications as it takes to form $A^{-1}\mathbf{b}_k$. Clearly then, the computation of A^{-1} for solving $A\mathbf{x} = \mathbf{b}_i$, $i = 1, 2, \ldots, m$ is not

justified since in order to find A^{-1}, we must first solve n systems of equations of the form $A\mathbf{x} = \mathbf{e}_i$, $i = 1, 2, \ldots, n$ (see Section 2.1).

At this point, we would like to offer as a guideline the following: in a practical problem it is seldom necessary to compute either the inverse of a matrix, the determinant of a matrix, or the power of a matrix. In the formulation of a problem it is often convenient to use inverses, powers, and determinants in an abstract mathematical sense (e.g., the solution of $A\mathbf{x} = \mathbf{b}$ is given by $\mathbf{x} = A^{-1}\mathbf{b}$). However, it is rare to encounter a problem in which the *solution* requires the computation of inverses, powers, or determinants. Thus for example, the solution to $A\mathbf{x} = \mathbf{b}$ is found efficiently by using Gauss elimination and is only *represented mathematically* by $A^{-1}\mathbf{b}$. Exceptions to the guidelines above do occur. For example, in certain problems in statistical analysis, the inverse matrix is of significance and may have to be calculated.

Although the basic ideas in Example 2.9 are fairly simple to program, they require examination in more detail and expression in matrix terms for some later analyses. Briefly, we will see that Gauss elimination with partial or implicit pivoting applied to $A\mathbf{x} = \mathbf{b}$ amounts (in effect) to constructing a nonsingular matrix S and then forming an equivalent system $SA\mathbf{x} = S\mathbf{b}$ where $SA = U$ is an upper-triangular matrix. There are primarily two reasons for wanting to view Gauss elimination in matrix terms. First of all, such a view provides a logical and conceptually convenient framework in which to organize an efficient Gauss elimination package. Perhaps more important, this framework will serve to show how to obtain estimates to the "condition number" of A (see Section 2.3.3). Such estimates are quite important because they give us a way to assess the accuracy of the machine solution to a problem $A\mathbf{x} = \mathbf{b}$.

As we stated above, using Gauss elimination to solve $A\mathbf{x} = \mathbf{b}$ amounts *formally* to two steps (the "factor" step and the "solve" step).

1. Find a nonsingular matrix S such that $SA = U$ where U is upper triangular.
2. Solve the equivalent triangular system $U\mathbf{x} = S\mathbf{b}$.

To see this two-step procedure in matrix terms, let A and L_1 be the matrices

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & & & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \qquad L_1 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & 0 \\ m_{31} & 0 & 1 & \cdots & 0 \\ \vdots & & & & \vdots \\ m_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

where $m_{i1} = -a_{i1}/a_{11}$, $2 \le i \le n$. Forming the product $A_1 = L_1A$ will produce the matrix

$$A_1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ 0 & a'_{32} & a'_{33} & \cdots & a'_{3n} \\ \vdots & & & & \vdots \\ 0 & a'_{n2} & a'_{n3} & \cdots & a'_{nn} \end{bmatrix};$$

and in fact, forming $A_1 = L_1A$ is the same as adding a multiple of m_{i1} times the first row of A to the ith row of A, $2 \le i \le n$. In other words, forming $A_1 = L_1A$ is equivalent to carrying out the first stage of Gauss elimination on A if no row interchange is made. Similarly if L_2 is the matrix

$$L_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & m_{32} & 1 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & m_{n2} & 0 & \cdots & 1 \end{bmatrix}$$

where $m_{i2} = -a'_{i2}/a'_{22}$, $3 \le i \le n$, then the matrix $A_2 = L_2A_1$ is what would be produced by the second stage of Gauss elimination, again if no row interchange is made. If no row interchanges are made at any stage, then Gauss elimination amounts (in matrix terms) to constructing an upper-triangular matrix U where U is given by

$$U = L_{n-1}L_{n-2} \cdots L_2L_1A.$$

Setting $S = L_{n-1}L_{n-2} \cdots L_2L_1$, we have $U = SA$, which completes the factor step. (Note that S is lower triangular; hence $S^{-1} = L$ is also lower triangular. Thus since $U = SA$, we have in effect factored A into a product of the form $A = LU$ where L is lower triangular and U is upper triangular. Such a factorization is called an LU-decomposition; we return to this topic later.) Having $U = SA$, we can now solve a problem $A\mathbf{x} = \mathbf{b}$ by solving the equivalent triangular system $U\mathbf{x} = S\mathbf{b}$.

In the discussion above we did not account for the possibility of row interchanges; and as we know, Gauss elimination with partial or implicit pivoting might require a row interchange at each stage of the elimination. We can express row interchanges in matrix terms by using permutation matrices. In particular if an $(n \times n)$ matrix P is derived from the identity by interchanging rows i and j of I, then P is called a permutation matrix. Next, it is easy to verify that forming the product PA gives the same result as interchanging the ith and jth rows of A. Thus permutation matrices can be used to incorporate partial or implicit pivoting into a matrix-theoretic description of Gauss elimination. Suppose P_k is the permutation matrix that describes the row interchange at the kth stage of Gauss elimination where we set $P_k = I$ if no interchange is made. Then in matrix terms the first step of Gauss elimination amounts to forming an upper-triangular matrix U where

$$U = (L_{n-1}P_{n-1})(L_{n-2}P_{n-2}) \cdots (L_2P_2)(L_1P_1)A. \tag{2.25a}$$

As before, we set $S = L_{n-1}P_{n-1}L_{n-2}P_{n-2} \cdots L_2P_2L_1P_1$ so that $U = SA$; and this completes the factor step.

The solve step now amounts to calculating $S\mathbf{b}$ from

$$S\mathbf{b} = L_{n-1}P_{n-1}L_{n-2}P_{n-2} \cdots L_2P_2L_1P_1\mathbf{b} \tag{2.25b}$$

and then solving the triangular system $U\mathbf{x} = S\mathbf{b}$. Clearly if we have several

systems $A\mathbf{x} = \mathbf{b}_i$, $i = 1, 2, \ldots, m$, then we perform the factor step once to obtain $U = SA$. Next, for each i, we form $S\mathbf{b}_i$ and solve $U\mathbf{x} = S\mathbf{b}_i$, $1 \le i \le m$. The factor step takes on the order of $n^3/3$ operations, and each application of the solve step requires n^2 operations.

Giving Gauss elimination in matrix terms is conceptually useful and also useful for purposes of error analyses. However, a Gauss elimination program would not actually calculate the matrix S in the factor step nor form the product $S\mathbf{b}$ in the solve step; instead the program would be structured along the lines of Example 2.9. In particular, an elementary Gauss elimination program should consist of at least two subroutines, FACTOR and SOLVE; and we ask the reader to modify the Gauss elimination program in Fig. 2.1 to accomplish this task (see Problem 11). As illustrated in Example 2.9, the FACTOR routine accepts a matrix A and generates a triangular array of multipliers $m_{21}, m_{31}, \ldots m_{n1}, m_{32}, m_{42}, \ldots$ and also generates the upper-triangular matrix U. The array of multipliers and the matrix U can be written over A, or stored in a separate $(n \times n)$ array if we do not wish to destroy A in FACTOR. In addition, FACTOR must keep a record of row interchanges; and this record can be stored in a vector IPIVOT of length $n - 1$ where IPIVOT $(K) = J$ means that row k and row j were interchanged at the kth stage of the elimination.

Given a system $A\mathbf{x} = \mathbf{b}$, subroutine SOLVE is passed the vector $\mathbf{b}$, along with the array of multipliers, the triangular matrix U, and the vector IPIVOT. The calculation of $S\mathbf{b}$ in SOLVE is done in $n - 1$ stages as was illustrated in Example 2.9. In the first stage if IPIVOT(1) $= J$, then the entries b_1 and b_j are interchanged in $\mathbf{b}$, and the interchange produces a new vector, $\tilde{\mathbf{b}}_1 = [\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_n]^T$. Then the entries of $\tilde{\mathbf{b}}_1$ are modified to produce $\mathbf{b}_1$ where the ith entry of $\tilde{\mathbf{b}}_1$ is changed to $\tilde{b}_i + m_{i1}\tilde{b}_1$, $2 \le i \le n$. In terms of (2.25b),

$$\tilde{\mathbf{b}}_1 = P_1\mathbf{b} \qquad \text{and} \qquad \mathbf{b}_1 = L_1P_1\mathbf{b}.$$

At the second stage, if IPIVOT(2) $= K$, then the second and kth entries of $\mathbf{b}_1$ are switched to produce $\tilde{\mathbf{b}}_2$. The entries of $\tilde{\mathbf{b}}_2$ are then modified to produce $\mathbf{b}_2$, by adding a multiple of m_{i2} times the second entry of $\tilde{\mathbf{b}}_2$ to the ith entry of $\tilde{\mathbf{b}}_2$, for $3 \le i \le n$. In terms of (2.25b),

$$\tilde{\mathbf{b}}_2 = P_2\mathbf{b}_1 \qquad \text{and} \qquad \mathbf{b}_2 = L_2P_2\mathbf{b}_1.$$

The remaining stages follow analogously, each stage consists of an interchange followed by the multiplier operations. The end result is the vector $S\mathbf{b}$ in (2.25b), and SOLVE can complete its assigned task by backsolving $U\mathbf{x} = S\mathbf{b}$.

EXAMPLE 2.10. To illustrate the factor and solve steps, we consider a simple (3×3) system $A\mathbf{x} = \mathbf{b}$. For brevity we let v denote the two-dimensional pivot vector; so $v_1 =$ IPIVOT(1) and $v_2 =$ IPIVOT(2). We also write the multipliers and the matrix U over the entries of A although it is not necessarily desirable to destroy A in the factor step.

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

Factor step:

$$A \xrightarrow[R_1 \longleftrightarrow R_2]{(v_1 = 2)} \begin{bmatrix} 2 & -1 & 1 \\ 1 & -1 & 0 \\ 2 & -2 & -1 \end{bmatrix} \xrightarrow{m_{21} = -\frac{1}{2},\ m_{31} = -1} \left[\begin{array}{ccc} 2 & -1 & 1 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ -1 & -1 & -2 \end{array}\right]$$

$$\xrightarrow[R_2 \longleftrightarrow R_3]{(v_2 = 3)} \left[\begin{array}{ccc} 2 & -1 & 1 \\ -\frac{1}{2} & -1 & -2 \\ -1 & -\frac{1}{2} & -\frac{1}{2} \end{array}\right] \xrightarrow{m_{32} = -\frac{1}{2}} \left[\begin{array}{ccc} 2 & -1 & 1 \\ -\frac{1}{2} & -1 & -2 \\ -1 & -\frac{1}{2} & \frac{1}{2} \end{array}\right]$$

$$\mathbf{v} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad \text{(pivot vector)}.$$

Solve step:

$$\mathbf{b} \xrightarrow[R_1 \longleftrightarrow R_2]{(v_1 = 2)} \begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix} \xrightarrow{m_{21} = -\frac{1}{2},\ m_{31} = -1} \begin{bmatrix} 4 \\ 0 \\ -1 \end{bmatrix} \xrightarrow[R_2 \longleftrightarrow R_3]{(v_2 = 3)} \begin{bmatrix} 4 \\ -1 \\ 0 \end{bmatrix} \xrightarrow{m_{32} = -\frac{1}{2}} \begin{bmatrix} 4 \\ -1 \\ \frac{1}{2} \end{bmatrix} = S\mathbf{b}.$$

Backsolving $U\mathbf{x} = S\mathbf{b}$ yields $x_3 = 1$, $x_2 = -1$, $x_1 = 1$.

As we mentioned earlier, finding a nonsingular matrix S such that $SA = U$ is equivalent to factoring A as $A = S^{-1}U$. A related idea is that of an *LU-decomposition* for A. Specifically, suppose we can factor the $(n \times n)$ matrix A into a product of two matrices L and U so that

$$A = LU$$

where L is an $(n \times n)$ lower-triangular matrix and U is an $(n \times n)$ upper-triangular matrix.

If A has a factorization $A = LU$, then solutions of the system $A\mathbf{x} = \mathbf{b}$ are found by solving $LU\mathbf{x} = \mathbf{b}$. To solve $LU\mathbf{x} = \mathbf{b}$, we proceed in two stages.

1. Solve $L\mathbf{y} = \mathbf{b}$
2. Solve $U\mathbf{x} = \mathbf{y}$.

In this fashion if $U\mathbf{x} = \mathbf{y}$, then $LU\mathbf{x} = L\mathbf{y} = \mathbf{b}$; moreover, both systems $L\mathbf{y} = \mathbf{b}$ and $U\mathbf{x} = \mathbf{y}$ are triangular and hence easy to solve. If Gauss elimination proceeds with no row interchanges, then as we noted before, the matrix S in (2.25a) is lower triangular and therefore $S^{-1} = L$ is also lower triangular. Thus Gauss elimination without pivoting can sometimes be used to produce an LU-decomposition for A. Another procedure is to consider the matrix equation $A = LU$:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \vdots & & & & \vdots \\ \ell_{n1} & \ell_{n2} & & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}. \tag{2.26}$$

Writing (2.26) out at length and equating both sides coordinatewise gives n^2 equations, which can then be (possibly) solved to determine L and U. We do not intend to do an extensive analysis of the general case represented by (2.26), but we do want to observe that as (2.26) is written, L and U both contain $n(n+1)/2$ undetermined entries, a total of $n^2 + n$ undetermined entries. Since we have just n^2 equations, it seems feasible that n quantities may be chosen as we wish. In practice, these free parameters are normally specified by either

$$\ell_{11} = \ell_{22} = \cdots = \ell_{nn} = 1$$

or

$$\ell_{11} = u_{11}, \qquad \ell_{22} = u_{22}, \ldots, \ell_{nn} = u_{nn}.$$

The following example should serve to illustrate LU-decomposition.

Given

$$A = \begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 3 \\ -3 & 2 & -3 \end{bmatrix},$$

find an LU-decomposition of the form

$$\begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 3 \\ -3 & 2 & -3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}. \tag{2.27}$$

The most natural way to set up the equations that must be solved is in a row-by-row fashion:

$$\begin{array}{llll} u_{11} = 3 & u_{12} = 1 & u_{13} = 2 & \\ \ell_{21}u_{11} = 6 & \ell_{21}u_{12} + u_{22} = 3 & \ell_{21}u_{13} + u_{23} = 3 & (2.28) \\ \ell_{31}u_{11} = -3 & \ell_{31}u_{12} + \ell_{32}u_{22} = 2 & \ell_{31}u_{13} + \ell_{32}u_{23} + u_{33} = -3. & \end{array}$$

Solving these equations successively, we obtain

$$\begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 3 \\ -3 & 2 & -3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 2 \end{bmatrix}. \tag{2.29}$$

Not all matrices have an LU-decomposition. As a simple example, the nonsingular matrix

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.30}$$

can be shown (Problem 19) to have no LU-decomposition; the singular matrix $I + A$ does have an LU-decomposition. It *is* known that if a matrix A is nonsingular, then there is some rearrangement of the rows of A such that the rearranged matrix has an LU-decomposition. See Forsythe and Moler (1967), for a proof.

To investigate the relations between various LU-decompositions of A, consider the two factorizations $A = L_1U_1$ and $A = L_2U_2$. If A is nonsingular, then the factors are nonsingular; and $L_1U_1 = L_2U_2$ implies that $L_2^{-1}L_1 = U_2U_1^{-1}$. Since the products and inverses of lower- (upper-) triangular matrices remain lower (upper) triangular, then $L_2^{-1}L_1 = U_2U_1^{-1}$ means that $L_2^{-1}L_1$ is a diagonal matrix. Suppose $D = L_2^{-1}L_1$ where $D = (d_{ij})$, $L_1 = (b_{ij})$, and $L_2 = (c_{ij})$; then $L_1 = L_2D$ implies $b_{ii} = c_{ii}d_{ii}$, $1 \le i \le n$. Thus if the diagonal elements of L_1 and L_2 are the same, then $D = I$ and $L_1 = L_2$. Therefore, setting $\ell_{ii} = 1$, $1 \le i \le n$, in a factorization yields the same factorization as Gauss elimination without pivoting (Problem 16). The solution of $A\mathbf{x} = \mathbf{b}$ by this factorization is called the Doolittle method. Analogously, setting $u_{ii} = 1$, $1 \le i \le n$, is known as the Crout method. An advantage of such factorizations (called "compact" methods) of A into LU is that the elements ℓ_{ij} and u_{ij} are computed in terms of inner products. One can form these inner products in double precision and increase the accuracy of L and U. Any similar use of limited double precision in the usual form of Gauss elimination is not effective [see Atkinson (1978) for a more detailed discussion].

A similar compact method is often used when A is symmetric and positive definite; that is, $A^T = A$ and $\mathbf{x}^TA\mathbf{x} > 0$ for all nonzero vectors $\mathbf{x}$. If we set $\ell_{ii} = u_{ii}$, $1 \le i \le n$, then the resulting factorization yields $U = L^T$ and $A = LL^T$. Recursively the elements of L are given by

$$\ell_{ij} = (a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk})/\ell_{jj}, \qquad j = 1, 2, \ldots, i-1$$

$$\ell_{ii} = (a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2)^{\frac{1}{2}}.$$

Since A is positive definite, the arguments of the square roots for ℓ_{ii} are positive. Since A is symmetric and $U = L^T$, fewer operations are necessary to compute the factorization. Again inner products may be performed in double precision. This technique, known as Cholesky's method, can be shown to be effective without pivoting or scaling.

There are special sorts of problems for which factorization methods are very natural to use. One such problem that occurs in many different applied settings is that of solving a *tridiagonal* linear system:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\
a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\
a_{43}x_3 + a_{44}x_4 + a_{45}x_5 &= b_4 \\
\ddots \quad \ddots \quad \ddots &\ \vdots \\
a_{n,n-1}x_{n-1} + a_{nn}x_n &= b_n
\end{aligned} \tag{2.31}$$

Such systems are called tridiagonal because the coefficient matrix A for the

system (2.31) has three *diagonals;* or more precisely, $a_{ij} = 0$ if $j < i - 1$ or if $j > i + 1$. These systems commonly arise, for example, when numerical methods are used to solve two-point boundary value problems or when cubic spline approximations are used to fit data (see Chapters 7 and 5). In most practical situations, a factorization of the form (2.32) below is possible:

$$LU = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & 0 & \cdots & 0 \\ 0 & \ell_{32} & 1 & 0 & \cdots & 0 \\ 0 & 0 & \ell_{43} & 1 & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ 0 & u_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & 0 & u_{33} & a_{34} & \cdots & 0 \\ 0 & 0 & 0 & u_{44} & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}. \tag{2.32}$$

If A is the coefficient matrix of (2.31), we can express A as $A = LU$ by solving recursively:

$$\begin{aligned} u_{11} &= a_{11} \\ \ell_{i,\,i-1} &= a_{i,\,i-1}/u_{i-1,\,i-1} \qquad i = 2, 3, \ldots, n \\ u_{ii} &= a_{ii} - \ell_{i,\,i-1}a_{i-1,\,i} \end{aligned} \tag{2.33a}$$

Having the factorization above, we can solve (2.31) by first solving $L\mathbf{y} = \mathbf{b}$ and then $U\mathbf{x} = \mathbf{y}$. The explicit equations are

$$\begin{aligned} y_1 &= b_1 \\ y_i &= b_i - \ell_{i,\,i-1}y_{i-1}; \qquad i = 2, 3, \ldots, n \\ x_n &= y_n/u_{nn} \\ x_{n-i} &= (y_{n-i} - a_{n-i,\,n+1-i}x_{n+1-i})/u_{n-i,\,n-i}; \qquad i = 1, 2, \ldots, n-1. \end{aligned} \tag{2.33b}$$

PROBLEMS, SECTION 2.2.4

1. Use Subroutine GAUSS to solve the (3 × 3) system of Example 2.2, Section 2.1.
2. Repeat Problem 1 for the (3 × 3) system of Example 2.14, Section 2.3.3.
3. Write a program using Subroutine GAUSS to find the inverse of a nonsingular ($n \times n$) matrix and to find the solution of $A\mathbf{x} = \mathbf{b}$ by forming $\mathbf{x} = A^{-1}\mathbf{b}$. Use this program on the system in Problem 2 and compare answers of both problems to the exact solution: $x_1 = 1$, $x_2 = 3$, $x_3 = 2$.
4. In Example 2.8 replace .0001 by 10^{-n}. Use three-digit floating-decimal calculations for solving the system without pivoting, and determine the positive integer values of n for which the computed solution is "significantly" different from the true solution.
5. Repeat Problem 4 for the system
$$\begin{aligned} 10^{-n}x_1 + x_2 &= 3 \\ x_1 - x_2 &= -2. \end{aligned}$$
6. Use the program in Problem 3 to find A^{-1} for the (4 × 4) matrix A in Example 2.6 (the exact inverse is given in Problem 7 at the end of Section 2.3.4). In theory, AA^{-1}

$= A^{-1}A$; but because of round-off errors, we cannot expect this equality in practice. To illustrate the problem, calculate and print the two products AA^{-1} and $A^{-1}A$, using the machine version of A^{-1} found by the program in Problem 3.

7. Small changes in the right-hand side of $A\mathbf{x} = \mathbf{b}$ may lead to relatively large changes in the solution. As an illustration, use GAUSS to solve $A\mathbf{x} = \mathbf{b}$ where A is the matrix in Problem 6 and $\mathbf{b}$ is either of the vectors

a) $\mathbf{b} = \begin{bmatrix} 23.01 \\ 31.99 \\ 32.99 \\ 31.01 \end{bmatrix}$ b) $\mathbf{b} = \begin{bmatrix} 23.1 \\ 31.9 \\ 32.9 \\ 31.1 \end{bmatrix}$.

8. Let A_1 and A_2 be matrices obtained from the (4×4) matrix in Problem 6 by replacing the (1, 1) entry of A by 5.01 and 4.99, respectively. As in Problem 6, calculate A_1^{-1} and A_2^{-1}. Compare your results with A^{-1}.

9. Given the system of equations (2.6), show that $n(n + 1)/2$ multiplications are required to solve for $x_n, x_{n-1}, \ldots, x_2, x_1$.

10. Construct a (3×3) coefficient matrix where the implicit-scaling row interchanges are different from the row interchanges of partial pivoting.

11. Write FACTOR and SOLVE subroutines along the lines described in Section 2.2.4. If you write these subroutines in FORTRAN and use execution-time dimensioning, they should be of a form similar to

```
SUBROUTINE FACTOR(A, SA, IPIVOT, N, MAINDM, IERROR)
SUBROUTINE SOLVE(SA, B, X, IPIVOT, N, MAINDM)
```

In this version of FACTOR, the array SA contains U in the upper-triangular portion and the multipliers in the lower-triangular portion; the array A is not destroyed in FACTOR. To test your program (if you use partial pivoting), put some temporary print statements in FACTOR and SOLVE and then verify that you duplicate each stage of the problem worked in Example 2.10. Subroutine GAUSS can be used as a rough model for designing these two routines.

12. Let the $(n \times n)$ matrix P be derived from the identity matrix by interchanging the ith and jth rows. If $\mathbf{x}$ is any $(n \times 1)$ vector, argue that $P\mathbf{x}$ is the same as $\mathbf{x}$ with the ith and jth components interchanged. Use this result to describe PA where A is any $(n \times n)$ matrix. Show that $P = P^{\mathrm{T}} = P^{-1}$.

13. If L_1 and P_1 are given as in (2.25), show that L_1P_1A is the resulting matrix of the first step of Gauss elimination with partial pivoting on A.

14. Argue that the matrix $S = L_{n-1}P_{n-1} \ldots L_2P_2L_1P_1$ in (2.25) is nonsingular. (Hint: First show that AB is nonsingular if and only if both A and B are nonsingular.)

15. The entries of the $(n \times n)$ Hilbert matrix $H = (h_{ij})$ are given by $h_{ij} = 1/(i + j - 1)$. Hilbert matrices are considered to be badly behaved or *ill-conditioned* (see Section 2.3.3), and are often used to test solution routines. Write a program using the subroutines of Problem 11 to find the inverse of the (4×4) Hilbert matrix. Test your answer by forming the products HH^{-1} and $H^{-1}H$. Do the same with the program in Problem 3 and compare results. (Compute the entries h_{ij} rather than reading them as data.)

16. Noting that Gauss elimination in Example 2.6 proceeds without row interchanges, use only the computations given in this example to give an LU-decomposition for the (4×4) matrix A. Use this decomposition to solve the system $A\mathbf{x} = \mathbf{b}$ where $\mathbf{b} = [-1, -1, -8, -11]^T$.

17. If L is a lower-triangular matrix equal to I except that its jth column is $[0, \ldots, 0, 1, M_{j+1,j}, \ldots, M_{nj}]^T$, show that L^{-1} is of the same form with jth column $[0, \ldots, 0, 1, -M_{j+1,j}, \ldots, -M_{nj}]^T$.

18. The Hilbert matrices of Problem 15 are known to be symmetric and positive definite. Program Cholesky's method to yield an LU-decomposition of the (4×4) Hilbert matrix. Use this decomposition to compute H^{-1} and again compare results with Problem 15.

19. Show that the nonsingular matrix

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

has no LU-decomposition, but the singular matrix $A + I$ does have. Give the permutation matrix P such that PA has an LU-decomposition.

20. Apply the Doolittle factorization to the (3×3) coefficient matrix in Problem 1 and use the factorization to solve the system. Use the Cholesky method on the system in Problem 16.

2.3 ERROR ANALYSIS AND NORMS

There are many different types of linear systems of equations each having their own special characteristics. Thus we can hardly expect any particular direct method, like Gauss elimination, to be the best possible method to use in all circumstances. Moreover if we do use a direct method, our computed solution will almost certainly be incorrect because of round-off error. Therefore we need some way to determine the size of the error of any computed solution and also some way to improve this computed solution. In this section, we will briefly develop a little of the theoretical background that is needed both for the analysis of errors and for the analysis of the "iterative" algorithms of Section 2.4 that generate a sequence of approximate solutions to $A\mathbf{x} = \mathbf{b}$.

The subject of error analysis must be approached somewhat carefully since a particular computed solution (say $\mathbf{x}_c$) to $A\mathbf{x} = \mathbf{b}$ may be considered badly in error or quite acceptable depending on how we intend to use the vector $\mathbf{x}_c$. For example, let $\mathbf{x}_t$ denote the "true" solution of $A\mathbf{x} = \mathbf{b}$ and let $\mathbf{r} = A\mathbf{x}_c - \mathbf{b}$ be the *residual vector*. Then $\mathbf{r} = A\mathbf{x}_c - \mathbf{b} = A\mathbf{x}_c - A\mathbf{x}_t$ or

$$\mathbf{x}_c - \mathbf{x}_t = A^{-1}\mathbf{r}. \tag{2.34}$$

If A^{-1} has some very large entries, then Eq. (2.34) demonstrates that the residual vector, $\mathbf{r}$, might be small, and yet $\mathbf{x}_c$ might be quite far from $\mathbf{x}_t$. Depending on the context of the problem that gave rise to the equation $A\mathbf{x} = \mathbf{b}$, we might be happy with having $A\mathbf{x}_c - \mathbf{b}$ small or we might need $\mathbf{x}_c$ to be near $\mathbf{x}_t$.

2.3.1. Vector Norms

In the discussion above, we were forced in a natural way to use words like "large" and "small" to describe the size of a vector and words like "near" and "far" to describe the proximity of two vectors. Also, in subsequent material we will be developing methods that generate *sequences* of vectors, $\{\mathbf{x}^{(k)}\}_{k=1}^{\infty}$, which one hopes *converge* to some vector $\mathbf{x}$. In these methods we will need to have some idea of how "close" each $\mathbf{x}^{(k)}$ is to $\mathbf{x}$ in order that we may know how large k must be so that $\mathbf{x}^{(k)}$ is an acceptable approximation to $\mathbf{x}$. Thus, we must have some meaningful way to measure the size of a vector or the distance between two vectors. To do this, we extend the concept of *absolute value* or *magnitude* from the real numbers to vectors. For a vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.35}$$

we already know the Euclidean length $|\mathbf{x}| = \sqrt{(x_1)^2 + (x_2)^2 + \cdots + (x_n)^2}$ as a measure of size. It turns out, as we shall see, that other measures of size for a vector are also *practical* to use in many computational problems. This realization leads us to the definition of a *norm*.

To make the setting precise, let R^n denote the set of all n-dimensional vectors with real components

$$R^n = \left\{ \mathbf{x} \middle| \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad x_1, x_2, \ldots, x_n \text{ real} \right\}. \tag{2.36}$$

A norm on R^n is a real-valued function $\|\cdot\|$ defined on R^n and satisfying the three conditions of (2.37) below (where, as before, θ denotes the zero vector in R^n):

$$\|\mathbf{x}\| \geq 0, \text{ and } \|\mathbf{x}\| = 0 \text{ if and only if } \mathbf{x} = \theta; \tag{2.37a}$$

$$\|\alpha\mathbf{x}\| = |\alpha| \, \|\mathbf{x}\|, \text{ for all scalars } \alpha \text{ and vectors } \mathbf{x}; \tag{2.37b}$$

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \text{ for all vectors } \mathbf{x} \text{ and } \mathbf{y}. \tag{2.37c}$$

As noted above, the quantity $\|\mathbf{x}\|$ is thought of as being a measure of the size of the vector $\mathbf{x}$, and double bars are used to emphasize the distinction between the norm of a vector and the absolute value of a scalar. Three useful examples of norms are the so-called ℓ_p norms, $\|\cdot\|_p$, for R^n, $p = 1, 2, \infty$:

$$\begin{aligned} \|\mathbf{x}\|_1 &= |x_1| + |x_2| + \cdots + |x_n| \\ \|\mathbf{x}\|_2 &= \sqrt{(x_1)^2 + (x_2)^2 + \cdots + (x_n)^2} \\ \|\mathbf{x}\|_\infty &= \max\{|x_1|, |x_2|, \ldots, |x_n|\}. \end{aligned} \tag{2.38}$$

EXAMPLE 2.11. By way of illustration for R^3,

$$\mathbf{x} = \begin{bmatrix} 1 \\ -2 \\ -2 \end{bmatrix}, \qquad \|\mathbf{x}\|_1 = 5, \qquad \|\mathbf{x}\|_2 = 3, \qquad \|\mathbf{x}\|_\infty = 2. \tag{2.39}$$

To emphasize further the properties of norms, we now show that the definition of the function $\|\mathbf{x}\|_1$ in (2.38) satisfies the three conditions of (2.37). Clearly for each $\mathbf{x} \in R^n$, $\|\mathbf{x}\|_1 \geq 0$. The rest of (2.37a) is also trivial, for if $\mathbf{x} = \theta$, then

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \text{and} \qquad \|\mathbf{x}\|_1 = 0.$$

Conversely if

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \text{and} \qquad \|\mathbf{x}\|_1 = 0,$$

then $|x_1| + |x_2| + \cdots + |x_n| = 0$ and hence $x_1 = x_2 = \cdots = x_n = 0$, or $\mathbf{x} = \theta$. For part (2.37b),

$$\alpha\mathbf{x} = \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix};$$

so $\|\alpha\mathbf{x}\|_1 = |\alpha x_1| + |\alpha x_2| + \cdots + |\alpha x_n| = |\alpha| \, \|\mathbf{x}\|_1$. If

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \qquad \text{then} \qquad \mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix};$$

and therefore

$$\begin{aligned} \|\mathbf{x} + \mathbf{y}\|_1 &= |x_1 + y_1| + |x_2 + y_2| + \cdots + |x_n + y_n| \\ &\leq (|x_1| + |x_2| + \cdots + |x_n|) + (|y_1| + |y_2| + \cdots + |y_n|) \\ &= \|\mathbf{x}\|_1 + \|\mathbf{y}\|_1. \end{aligned}$$

Similarly (Problem 5) it is easy to show that $\|\cdot\|_\infty$ is a norm for R^n. The remaining ℓ_p norm, $\|\cdot\|_2$, is handled not quite so easily. In this case, the triangle inequality in condition (2.37c) does not follow immediately from the triangle inequality for absolute values, and (2.37c) is usually demonstrated with an application of the Cauchy-Schwarz inequality (see Section 2.6).

Having the concept of a norm, we can now make precise quantitative statements about size and distance, and can say that $A\mathbf{x}_c - \mathbf{b}$ is small if $\|A\mathbf{x}_c - \mathbf{b}\|$ is small and that $\mathbf{x}_c$ is near $\mathbf{x}_t$ if $\|\mathbf{x}_c - \mathbf{x}_t\|$ is small. The definition of a norm also has some flexibility. For example, we might have reason to insist that the first coordinate of the residual vector $\mathbf{r} = A\mathbf{x}_c - \mathbf{b}$ is quite critical and must be small, even at the expense of growth in the other coordinates. In this case we might select a norm like the one below where $\mathbf{x}$ is as in (2.35)

$$\|\mathbf{x}\| = \max\{10|x_1|, |x_2|, |x_3|, \ldots, |x_n|\}.$$

A norm such as this emphasizes the first coordinate. For example, saying that $\|\mathbf{x}\| \leq 10^{-5}$ implies that $|x_i| \leq 10^{-5}$ for $i = 2, 3, \ldots, n$ and $|x_1| \leq 10^{-6}$. Weightings of this sort are quite common in problems that involve fitting curves to data and we shall see some examples when we discuss topics such as least-squares fits.

An idea related to norms that weight different coordinates differently is the concept of *relative error*. This is the simple and practical notion that when the size of the error $\mathbf{x}_c - \mathbf{x}_t$ is measured, we should take into account the size of the components of $\mathbf{x}_t$. That is, if $\mathbf{x}_t$ has components of the order of 10^4, then an error of 0.01 is probably acceptable; if $\mathbf{x}_t$ has components that are generally of the order of 10^{-4}, then an error like 0.01 is disastrous. Thus if $\|\cdot\|$ is a norm for R^n, we define $\|\mathbf{x}_c - \mathbf{x}_t\|$ to be the *absolute error* and define the quantity $\|\mathbf{x}_c - \mathbf{x}_t\|/\|\mathbf{x}_t\|$ to be the *relative error*. We will usually be more interested in the relative error than in the absolute error.

EXAMPLE 2.12. Consider the two vectors

$$\mathbf{x}_t = \begin{bmatrix} 0.000397 \\ 0.000214 \\ 0.000309 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_c = \begin{bmatrix} 0.000504 \\ 0.000186 \\ 0.000342 \end{bmatrix}.$$

Then the vector $\mathbf{x}_c - \mathbf{x}_t$ is given by

$$\mathbf{x}_c - \mathbf{x}_t = \begin{bmatrix} 0.000107 \\ -0.000028 \\ 0.000033 \end{bmatrix}.$$

One measure of the absolute error is $\|\mathbf{x}_c - \mathbf{x}_t\|_1 = 0.000168$; so $\mathbf{x}_c$ seems a reasonably good approximation to $\mathbf{x}_t$. However, checking the relative error, we see that

$$\frac{\|\mathbf{x}_c - \mathbf{x}_t\|_1}{\|\mathbf{x}_t\|_1} = \frac{0.000168}{0.000920} \approx 0.183,$$

which more nearly reflects the true state of affairs, i.e., that our approximation $\mathbf{x}_c$ is in error by nearly 20 percent. Relative errors become particularly important in computer programs in which a criterion is needed for terminating an iteration.

2.3.2. Matrix Norms

In Eq. (2.34) we see that in order to measure the size of $\mathbf{x}_c - \mathbf{x}_t$, we must also be able to measure the size of the *vector* $(A^{-1}\mathbf{r})$ since $\mathbf{x}_t$ is not known. If we knew A^{-1}, then we could simply multiply A^{-1} times $\mathbf{r}$ and measure the size of $(A^{-1}\mathbf{r})$ by one of the vector norms of the previous section. However since A^{-1} is not known, we must use a different approach since it would be quite inefficient to compute A^{-1} accurately just to check the accuracy of $\mathbf{x}_c$. We are thus led to consider a way of measuring the "size" or "norms" of matrices as well as of vectors. We shall do this measurement in such a way that we are able to estimate the "size" of A^{-1} by knowing the "size" of A and then to estimate the norm of $(A^{-1}\mathbf{r})$. The concept of matrix norms is not limited to this particular problem, i.e., estimating $\|\mathbf{x}_c - \mathbf{x}_t\|$, but is practically indispensable in deriving computational procedures and error estimates for many other problems as we shall see presently.

By way of notation, let M_n denote the set of all $(n \times n)$ matrices and let $\mathcal{O}$ denote the $(n \times n)$ zero matrix. Then a *matrix norm* for M_n is a real-valued function $\|\cdot\|$ which is defined on M_n and will satisfy for all $(n \times n)$ matrices and A and B

$$\|A\| \geq 0 \text{ and } \|A\| = 0 \text{ if and only if } A = \mathcal{O} \tag{2.40a}$$

$$\|\alpha A\| = |\alpha| \, \|A\| \text{ for any scalar } \alpha \tag{2.40b}$$

$$\|A + B\| \leq \|A\| + \|B\| \tag{2.40c}$$

$$\|AB\| \leq \|A\| \, \|B\|. \tag{2.40d}$$

The addition of condition (2.40d) should be noted. Thus matrix norms have a triangle inequality for both addition and multiplication.

Just as there are numerous ways of defining specific vector norms, there are also numerous ways of defining specific matrix norms. We will concentrate on three norms that are easily computable and are intrinsically related to the three basic vector norms discussed in the previous section. Specifically if $A = (a_{ij}) \in M_n$, we define

$$\|A\|_1 \equiv \max_{1 \leq j \leq n} \left[\sum_{i=1}^{n} |a_{ij}| \right] \text{— (Maximum absolute column sum)}$$

$$\|A\|_\infty \equiv \max_{1 \leq i \leq n} \left[\sum_{j=1}^{n} |a_{ij}| \right] \text{— (Maximum absolute row sum)} \tag{2.41}$$

$$\|A\|_E \equiv \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij})^2}.$$

EXAMPLE 2.13. Let

$$A = \begin{bmatrix} 0 & 0 & 10 & 0 \\ 1 & 1 & 5 & 1 \\ 0 & 1 & 5 & 1 \\ 0 & 0 & 5 & 1 \end{bmatrix};$$

then $\|A\|_1 = \text{Max}\{1, 2, 25, 3\} = 25$, $\|A\|_\infty = \text{Max}\{10, 8, 7, 6\} = 10$, and $\|A\|_E = \sqrt{181} \approx 13.454$.

These three norms are stressed here because they are *compatible* with the ℓ_1, ℓ_∞, and ℓ_2 vector norms, respectively. Thus given any matrix $A \in M_n$, then for *all* vectors $\mathbf{x} \in R^n$ it is true that

$$\|A\mathbf{x}\|_1 \le \|A\|_1\|\mathbf{x}\|_1, \quad \|A\mathbf{x}\|_\infty \le \|A\|_\infty\|\mathbf{x}\|_\infty, \quad \text{and} \quad \|A\mathbf{x}\|_2 \le \|A\|_E\|\mathbf{x}\|_2. \tag{2.42}$$

(The reader should be careful to distinguish between vector norms and matrix norms since they bear the same subscript notation in two of the three cases. This distinction is clear from the usage. For example, $A\mathbf{x}$ is a vector; so $\|A\mathbf{x}\|_1$ denotes the use of the $\|\cdot\|_1$ vector norm, whereas $\|A\|_1$ denotes use of the matrix norm. The reader should also note that the pairs of matrix and vector norms are compatible only in the orders given by (2.42) and cannot be mixed. For example, let A be given as in the example above and let $\mathbf{x} = (0, 0, 1, 0)^T$. Then $\|\mathbf{x}\|_1 = 1$, but $\|A\mathbf{x}\|_1 = 25$ and $\|A\|_\infty\|\mathbf{x}\|_1 = 10$. That is, we cannot expect to have the inequality $\|A\mathbf{x}\|_1 \le \|A\|_\infty\|\mathbf{x}\|_\infty$ or $\|A\mathbf{x}\|_1 \le \|A\|_\infty\|\mathbf{x}\|_1$.)

Compatibility is a property that connects vector norms and matrix norms. For example, in (2.34) we had $(\mathbf{x}_c - \mathbf{x}_t) = A^{-1}\mathbf{r}$. Using the idea of compatible vector and matrix norms, we can estimate $\|\mathbf{x}_c - \mathbf{x}_t\|_1$ in terms of $\|A^{-1}\|_1$ and $\|\mathbf{r}\|_1$:

$$\|\mathbf{x}_c - \mathbf{x}_t\|_1 = \|A^{-1}\mathbf{r}\|_1 \le \|A^{-1}\|_1\|\mathbf{r}\|_1.$$

As we shall see in Section 2.4, compatibility is crucial also to a clear understanding of iterative methods.

Thus far we have not shown that the three matrix norms satisfy the compatibility properties, (2.42), or even that their definitions given by (2.41) satisfy the necessary norm properties given by (2.40). For the sake of brevity we shall supply only the necessary proofs for the $\|\cdot\|_1$ norm and leave the $\|\cdot\|_\infty$ and $\|\cdot\|_E$ norms to the reader.

Let $A = (a_{ij}) \in M_n$; and write A in terms of its column vectors, $A = [\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n]$. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ be any vector in R^n, and recall from Section 2.1 that $A\mathbf{x}$ may be written as $A\mathbf{x} = x_1\mathbf{A}_1 + x_2\mathbf{A}_2 + \cdots + x_n\mathbf{A}_n$. Since $A\mathbf{x}$ is an $(n \times 1)$ vector, we use (2.37) to get

$$\begin{aligned} \|A\mathbf{x}\|_1 &= \|x_1\mathbf{A}_1 + x_2\mathbf{A}_2 + \cdots + x_n\mathbf{A}_n\|_1 \\ &\le \|x_1\mathbf{A}_1\|_1 + \|x_2\mathbf{A}_2\|_1 + \cdots + \|x_n\mathbf{A}_n\|_1 \\ &= |x_1|\,\|\mathbf{A}_1\|_1 + |x_2|\,\|\mathbf{A}_2\|_1 + \cdots + |x_n|\,\|\mathbf{A}_n\|_1 \\ &\le (|x_1| + |x_2| + \cdots + |x_n|)\left(\max_{1 \le j \le n} \|\mathbf{A}_j\|_1\right) \equiv \|A\|_1\|\mathbf{x}\|_1. \end{aligned} \tag{2.43}$$

Thus we have shown compatibility for the $\|\cdot\|_1$ norm.

It is trivial to see that parts (2.40a) and (b) hold for $\|\cdot\|_1$. Since the ith column of $A + B$ is precisely the ith column of A plus the ith column of B, part (2.40c) is also easily seen to be true. For part (2.40d), we recall from Section 2.1 that the ith column of AB equals $A\mathbf{B}_i$ where $\mathbf{B}_i$ is the ith column of B. By compatibility, $\|A\mathbf{B}_i\|_1 \leq \|A\|_1\|\mathbf{B}_i\|_1$. Now choose i such that $\|\mathbf{B}_i\|_1 \geq \|\mathbf{B}_j\|_1$ for $1 \leq j \leq n$. Then $\|B\|_1 = \|\mathbf{B}_i\|_1$, and

$$\|AB\|_1 = \max_{1 \leq j \leq n} \|A\mathbf{B}_j\|_1 \leq \max_{1 \leq j \leq n} \|A\|_1\|\mathbf{B}_j\|_1 = \|A\|_1\|\mathbf{B}_i\|_1 = \|A\|_1\|B\|_1;$$

and thus part (2.40d) is satisfied. We have therefore shown that $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}|$ is a matrix norm and that it is compatible with the $\|\cdot\|_1$ vector norm.

We conclude this material with an observation on matrix norms, and consider the three numbers K_p, $p = 1, 2, \infty$, where if $A \in M_n$ is given, then

$$K_p = \inf\{K \in R^1: \|A\mathbf{x}\|_p \leq K\|\mathbf{x}\|_p, \text{ for all } \mathbf{x} \in R^n\} \tag{2.44}$$

(where "inf" denotes *infimum* or *greatest lower bound*). It can be shown that $K_1 = \|A\|_1$ and $K_\infty = \|A\|_\infty$ although the demonstration goes beyond our purposes. We introduce these numbers, however, since $K_2 \neq \|A\|_E$; and this explains why we use the subscript "E" instead of "2." Thus although we have $\|A\mathbf{x}\|_2 \leq \|A\|_E\|\mathbf{x}\|_2$ (compatibility), there is a matrix norm $K_2 \equiv \|A\|_2$, smaller for most matrices than $\|A\|_E$, and such that $\|A\mathbf{x}\|_2 \leq \|A\|_2\|\mathbf{x}\|_2$ for all $\mathbf{x} \in R^n$. $\|A\|_2$ is rather unwieldly in computations and involves some deeper theory to derive, and so we are satisfied to use the easily computable and compatible $\|A\|_E$ in place of $\|A\|_2$ (see Theorem 3.7).

2.3.3. Condition Numbers and Error Estimates

In this section, we use the ideas of matrix and vector norms to provide some more information that is useful in helping to determine how good a computed (approximate) solution to the system $A\mathbf{x} = \mathbf{b}$ is. The norms used below can be any pair of compatible matrix and vector norms; for convenience we have omitted the subscripts. The reader can tell from the context whether a particular norm is a matrix norm or a vector norm. The following theorem provides valuable information with respect to the relative error.

Theorem 2.1.

Suppose $A \in M_n$ is nonsingular and $\mathbf{x}_c$ is an approximation to $\mathbf{x}_t$, the exact solution of $A\mathbf{x} = \mathbf{b}$ where $\mathbf{b} \neq \theta$. Then for any compatible matrix and vector norms

$$\frac{1}{\|A\|\|A^{-1}\|} \frac{\|A\mathbf{x}_c - \mathbf{b}\|}{\|\mathbf{b}\|} \leq \frac{\|\mathbf{x}_c - \mathbf{x}_t\|}{\|\mathbf{x}_t\|} \leq \|A\|\,\|A^{-1}\| \frac{\|A\mathbf{x}_c - \mathbf{b}\|}{\|\mathbf{b}\|}. \tag{2.45}$$

Proof. Again by (2.34), $\mathbf{x}_c - \mathbf{x}_t = A^{-1}\mathbf{r}$ where $\mathbf{r} = A\mathbf{x}_c - \mathbf{b}$. Thus by the compatability conditions, (2.42), $\|\mathbf{x}_c - \mathbf{x}_t\| \leq \|A^{-1}\| \|\mathbf{r}\| = \|A^{-1}\| \|A\mathbf{x}_c - \mathbf{b}\|$. Now $A\mathbf{x}_t = \mathbf{b}$; so $\|A\| \|\mathbf{x}_t\| \geq \|\mathbf{b}\|$, and $\|A\|/\|\mathbf{b}\| \geq 1/\|\mathbf{x}_t\|$. Thus,

$$\frac{\|\mathbf{x}_c - \mathbf{x}_t\|}{\|\mathbf{x}_t\|} \leq \|A^{-1}\| \|A\mathbf{x}_c - \mathbf{b}\| \frac{\|A\|}{\|\mathbf{b}\|},$$

establishing the right-hand side of (2.45). Now

$$\|A\mathbf{x}_c - \mathbf{b}\| = \|\mathbf{r}\| = \|A\mathbf{x}_c - A\mathbf{x}_t\| \leq \|A\| \|\mathbf{x}_c - \mathbf{x}_t\|;$$

and since $\|A\| > 0$,

$$\|\mathbf{x}_c - \mathbf{x}_t\| \geq \|A\mathbf{x}_c - \mathbf{b}\|/\|A\|.$$

Also $\mathbf{x}_t = A^{-1}\mathbf{b}$; so $\|\mathbf{x}_t\| \leq \|A^{-1}\| \|\mathbf{b}\|$, or $1/\|\mathbf{x}_t\| \geq 1/\|A^{-1}\| \|\mathbf{b}\|$. Combining these last two inequalities establishes the left-hand side of 2.45). ∎

Note the appearance of the term $\|A\| \|A^{-1}\|$ in both the upper and the lower bounds for the relative error. This term is called the *condition number* and is denoted by $\kappa(A)$. If one lets $\epsilon = \|A\mathbf{x}_c - \mathbf{b}\|/\|\mathbf{b}\|$, Eq. (2.45) becomes

$$\frac{\epsilon}{\kappa(A)} \leq \frac{\|\mathbf{x}_c - \mathbf{x}_t\|}{\|\mathbf{x}_t\|} \leq \epsilon\kappa(A).$$

It is easy to show that $\kappa(A) \geq 1$ (Problem 6), and that the closer $\kappa(A)$ is to 1, the more accurate ϵ becomes as a measurement of the relative error. If $\kappa(A) >> 1$, we are alerted to the possibility that the relative error may not be small even if ϵ is small. This situation occurs when the system $A\mathbf{x} = \mathbf{b}$ is ill-conditioned; that is, when small changes in input data can cause large variations in the solution $\mathbf{x}$. The ultimate in ill-conditioning occurs when A is singular. In this case there are infinitely many vectors $\mathbf{u} \neq \theta$ such that $A\mathbf{u} = \theta$, and hence $\mathbf{x}_t$ and $\mathbf{x}_t + \mathbf{u}$ are both solutions of $A\mathbf{x} = \mathbf{b}$ even when A and $\mathbf{b}$ undergo no change at all. To show the connection between the size of $\kappa(A)$ and how close A is to being singular, it can be proved that if A is nonsingular, then [see Conte and deBoor (1980)]

$$\frac{1}{\kappa(A)} = \min\left\{\frac{\|A - B\|}{\|A\|} : B \text{ is singular}\right\}. \tag{2.46}$$

Hence A can be well approximated (in a relative sense) by a singular matrix if and only if $\kappa(A)$ is large. To elaborate, we observe that if $\mathbf{x}_c$ is the computed solution to $A\mathbf{x} = \mathbf{b}$ and if $\mathbf{x}_c \neq \mathbf{x}_t$, then we have in effect solved a perturbed problem of the form $(A + E)\mathbf{x} = \mathbf{b}$ where E is a matrix that accounts for the errors in the computation. From (2.46) we see that if $\kappa(A)$ is large, then $A + E$ could be singular or almost singular even if E is small. With $A + E$ almost singular we need not expect $\mathbf{x}_c$ to be close to $\mathbf{x}_t$. Although we cannot justify it here, a basic rule of thumb is that if $\kappa(A) = 10^k$ and we are working in d-decimal arithmetic, then we should not expect more than $(d - k)$ accurate figures in $\mathbf{x}_c$ if A is properly scaled. [See the *LINPACK User's Guide* (1979) for a more com-

plete discussion. If $d \leq k$ we say that A is "singular to working precision."]

From the discussion above one sees that a Gauss elimination package should not only provide a machine estimate $\mathbf{x}_c$ to the solution of $A\mathbf{x} = \mathbf{b}$, but also give an estimate for $\kappa(A)$ to test the reliability of $\mathbf{x}_c$. Since $\|A\|$ can be calculated immediately, the crux of the problem is to estimate $\|A^{-1}\|$. Obviously it would be extremely inefficient to try to compute A^{-1} accurately just for this purpose. An alternative is provided by the following. For any nonzero vector $\mathbf{x}$, $\mathbf{x} = A^{-1}A\mathbf{x}$, $\|\mathbf{x}\| \leq \|A^{-1}\| \, \|A\mathbf{x}\|$; and so

$$\|A^{-1}\| \geq \frac{\|\mathbf{x}\|}{\|A\mathbf{x}\|}.$$

Hence we can estimate $\|A^{-1}\|$ by trying to make the quotient $\|\mathbf{x}\|/\|A\mathbf{x}\|$ as large as possible. It can be shown that there exist infinitely many vectors $\tilde{\mathbf{x}}$ such that $\|A^{-1}\| = \|\tilde{\mathbf{x}}\|/\|A\tilde{\mathbf{x}}\|$. In practice we do not actually try to compute such an $\tilde{\mathbf{x}}$ precisely; we merely look for an efficient way of making the quotient $\|\mathbf{x}\|/\|A\mathbf{x}\|$ reasonably large. One reason is that the upper bound for the relative error in (2.45) is usually very conservative; that is, the bound is usually too large, and so a smaller estimate for $\|A^{-1}\|$ is probably acceptable in analyzing the relative error.

Although a detailed theoretical development goes beyond the scope of this text, the *LINPACK* authors recommend the following way of obtaining an estimate for $\|A^{-1}\|$. Let $\mathbf{c}$ be a vector whose components are each ± 1, solve $A^{\mathrm{T}}\mathbf{y} = \mathbf{c}$, and then solve $A\mathbf{x} = \mathbf{y}$. Use $\|\mathbf{x}\|/\|\mathbf{y}\|$ as the estimate for $\|A^{-1}\|$; since $\mathbf{y} = A\mathbf{x}$, it follows that $\|A^{-1}\| \geq \|\mathbf{x}\|/\|\mathbf{y}\|$. We discuss the choice of ± 1 in $\mathbf{c}$ momentarily, but for now we consider the solution process for $A^{\mathrm{T}}\mathbf{y} = \mathbf{c}$ with respect to the computations already performed in Gauss elimination with partial pivoting in solving $A\mathbf{x} = \mathbf{b}$. In particular, the factor step produces matrices S and U such that $SA = U$ where

$$S = L_{n-1}P_{n-1} \cdots L_2P_2L_1P_1.$$

Thus since $A = S^{-1}U$, $A^{\mathrm{T}}\mathbf{y} = \mathbf{c}$ becomes $U^{\mathrm{T}}(S^{-1})^{\mathrm{T}}\mathbf{y} = \mathbf{c}$. Setting $\mathbf{z} = (S^{-1})^{\mathrm{T}}\mathbf{y}$, we can solve $A^{\mathrm{T}}\mathbf{y} = \mathbf{c}$ by solving $U^{\mathrm{T}}\mathbf{z} = \mathbf{c}$ and then setting $\mathbf{y} = S^{\mathrm{T}}\mathbf{z}$. Since U^{T} is lower triangular, $U^{\mathrm{T}}\mathbf{z} = \mathbf{c}$ is simply solved by successive variable substitution. The matrix S is characterized by the elimination multiples M_{ij} and by the vector IPIVOT that records the necessary row interchanges of the pivoting. Without actually forming S or S^{T}, we can compute $S^{\mathrm{T}}\mathbf{z}$ as follows. The permutation matrices P_k are symmetric, $P_k^{\mathrm{T}} = P_k$, and so $S^{\mathrm{T}} = P_1L_1^{\mathrm{T}}P_2L_2^{\mathrm{T}} \cdots P_{n-1}L_{n-1}^{\mathrm{T}}$. Each L_k^{T} is the same as the identity matrix I except the kth row is

$$[0, \ldots, 0, 1, M_{k+1,\,k}, M_{k+2,\,k}, \ldots, M_{nk}].$$

Hence for $\mathbf{w} = [w_1, w_2, \ldots, w_n]^{\mathrm{T}}$, $L_k^{\mathrm{T}}\mathbf{w}$ equals $\mathbf{w}$ except in the kth component, which is given by

$$(L_k^{\mathrm{T}}\mathbf{w})_k = w_k + M_{k+1,\,k}w_{k+1} + M_{k+2,\,k}w_{k+2} + \cdots + M_{nk}w_n. \qquad (2.47)$$

Thus to form $S^T\mathbf{z}$, we successively multiply by L_k as indicated above, and then perform the interchange specified by P_k.

As a simple numerical example, consider the solution of $A^T\mathbf{y} = \mathbf{c}$ where A is as in Example 2.10; and arbitrarily select $\mathbf{c} = [1, -1, -1]^T$. The matrix U is displayed in Example 2.10, and it is routine to solve $U^T\mathbf{z} = \mathbf{c}$ to obtain

$$\mathbf{z} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ -1 \end{bmatrix}.$$

The multipliers and the pivot vector $\mathbf{v} = [2, 3]^T$ are also given in Example 2.10; so $\mathbf{y} = S^T\mathbf{z}$ is calculated [see (2.47)] by

$$\mathbf{z} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ -1 \end{bmatrix} \xrightarrow{m_{32} = -\frac{1}{2}} \begin{bmatrix} \frac{1}{2} \\ 1 \\ -1 \end{bmatrix} \xrightarrow[R_2 \longleftrightarrow R_3]{v_2 = 3}$$

$$\begin{bmatrix} \frac{1}{2} \\ -1 \\ 1 \end{bmatrix} \xrightarrow[m_{31} = -1]{m_{21} = -\frac{1}{2}} \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \xrightarrow[R_1 \longleftrightarrow R_2]{v_1 = 2} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \mathbf{y}.$$

In the example above, the vector $\mathbf{c}$ was selected at random to illustrate how the solution of $A^T\mathbf{y} = \mathbf{c}$ is organized. What is really wanted is an algorithm that forms a vector $\mathbf{c}$ (whose entries are ± 1) such that the quantity $\|\mathbf{x}\|/\|\mathbf{y}\|$ is a good lower estimate to $\|A^{-1}\|$ and where, as before, $A^T\mathbf{y} = \mathbf{c}$ and $A\mathbf{x} = \mathbf{y}$. The reason for choosing $\mathbf{c}$ to have components ± 1 is beyond the scope of this text and is developed in Cline, Moler, Stewart, and Wilkinson (1979). However, the system $A^T\mathbf{y} = \mathbf{c}$ is solved as above where the specific choices for components of $\mathbf{c}$ are predicated on making $\mathbf{z}$ as large as possible, where $U^T\mathbf{z} = \mathbf{c}$ and $\mathbf{y} = S^T\mathbf{z}$. Part of the explanation for this choice of $\mathbf{c}$ lies in the fact that pivoting usually implies that any ill-conditioning in A is reflected in a corresponding ill-conditioning in U and that S is usually not ill-conditioned. (The bulk of the explanation lies in the "singular-value decomposition of A," the eigenvalues of A^TA, and their relation to $\|A\|_2$. See the references above.)

Denoting $U = (u_{ij})$, $\mathbf{z} = (z_i)$, and $\mathbf{c} = (c_i)$ where $c_i = \pm 1$, the kth step of the solution of $U^T\mathbf{z} = \mathbf{c}$ is solving the equation

$$u_{kk}z_k = c_k - (u_{1k}z_1 + \cdots + u_{k-1,k}z_{k-1}).$$

A first strategy at this point is to choose the sign of c_k to be opposite to the sign of $q_k \equiv (u_{1k}z_1 + \cdots + u_{k-1,k}z_{k-1})$ in order to maximize the size of z_k. Setting $c_1 = 1$ and doing this for $2 \leq k \leq n$ is one way to determine the vector $\mathbf{c}$. The authors mentioned above advocate a second, more sophisticated strategy, which uses some information from the last $n - k$ equations as well as the kth equation. Again we consider the kth stage (choosing c_k) where we assume that

$z_1, \ldots, z_{k-1}$ and $c_1, \ldots, c_{k-1}$ have been already determined. For $k \le i \le n$ we have that the ith equation of $U^T\mathbf{z} = \mathbf{c}$ is

$$u_{ii}z_i = -p_i - (u_{ki}z_k + \cdots + u_{i-1,i}z_{i-1}) + c_i \tag{2.48a}$$

where $p_i = (u_{1i}z_1 + u_{2i}z_2 + \cdots + u_{k-1,i}z_{k-1})$. Note that p_i is independent of the choice of c_k (p_i is determined by the choices $c_1, c_2, \ldots, c_{k-1}$ that have already been made) and note also that the kth equation is $u_{kk}z_k = -p_k + c_k$ in this notation. The choice of $c_k = 1$ or $c_k = -1$ will determine z_k, but it will also influence the last components $z_{k+1}, z_{k+2}, \ldots, z_n$ of $\mathbf{z}$. To make a choice for c_k, let z_k^+ and z_k^- denote the solution of (2.48a) for $i = k$ and for $c_k = 1$ and $c_k = -1$, respectively. Given the two possible choices for c_k, we can rewrite (2.48a) for $k + 1 \le i \le n$ as either

$$u_{ii}z_i = -p_i^+ - (u_{k+1,i}z_{k+1} + \cdots + u_{i-1,i}z_{i-1}) + c_i$$

or (2.48b)

$$u_{ii}z_i = -p_i^- - (u_{k+1,i}z_{k+1} + \cdots + u_{i-1,i}z_{i-1}) + c_i$$

where $p_i^+ = p_i + u_{ki}z_k^+$ and $p_i^- = p_i + u_{ki}z_k^-$. Since we do not know c_j or z_j for $j \ge k + 1$, we temporarily set these equal to zero on the right-hand side of (2.48b); select $c_k = +1$ if

$$|-p_k + 1| + \sum_{i=k+1}^{n} |p_i^+| \ge |-p_k - 1| + \sum_{i=k+1}^{n} |p_i^-|,$$

and choose $c_k = -1$ otherwise. Essentially this procedure is trying to maximize $\|\mathbf{z}\|$ by forcing the absolute sum of the terms on the right-hand side of (2.48a) to be as large as possible when we neglect z_j and c_j for $k + 1 \le j \le n$. For $k = 1$, we can choose $c_k = +1$ to start the selection algorithm.

Using this strategy on the matrix A in Example 2.10,

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 2 & -1 & 1 \\ 2 & -2 & -1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 2 & -1 & 1 \\ 0 & -1 & -2 \\ 0 & 0 & \frac{1}{2} \end{bmatrix},$$

we obtain $\mathbf{c} = [1, 1, -1]^T$, $\mathbf{z} = [1/2, -3/2, -9]^T$, and $\mathbf{y} = [-9, 2, 3]^T$ where $U^T\mathbf{z} = \mathbf{c}$, $\mathbf{y} = S^T\mathbf{z}$, and $A^T\mathbf{y} = \mathbf{c}$. Then $A\mathbf{x} = \mathbf{y}$ where $\mathbf{x} = [32, 41, -21]^T$, and so $\|A^{-1}\|_1 \ge \|\mathbf{x}\|_1/\|\mathbf{y}\|_1 = 94/14 \simeq 6.7$. Actual calculation of A^{-1} yields $\|A^{-1}\|_1 = 9$. The matrix above is not ill-conditioned. The strategy was also applied to the (3×3) Hilbert matrix H_3 (Problem 15, Section 2.2.4) with the result that $\|H_3^{-1}\|_1 \ge 371.35$ where $\|H_3^{-1}\|_1 = 408$. (We leave it to the reader to verify this result.)

EXAMPLE 2.14. Consider the linear system

$$\begin{aligned} 6x_1 + 6x_2 + 3.00001x_3 &= 30.00002 \\ 10x_1 + 8x_2 + 4.00003x_3 &= 42.00006 \\ 6x_1 + 4x_2 + 2.00002x_3 &= 22.00004, \end{aligned}$$

which has the unique solution $x_1 = 1$, $x_2 = 3$, $x_3 = 2$. This example illustrates the notion of an ill-conditioned system as well as how matrix norms can be used to analyze the results of a computational solution. (Recall that a system of linear equations is called ill-conditioned if "small" changes in the coefficients produce "large" changes in the solution.) Before solving this system, note that the perturbed system below

$$\begin{aligned} 6x_1 + 6x_2 + 3.00001x_3 &= 30 \\ 10x_1 + 8x_2 + 4.00003x_3 &= 42 \\ 6x_1 + 4x_2 + 2.00002x_3 &= 22 \end{aligned}$$

has a unique solution $x_1 = 1$, $x_2 = 4$, $x_3 = 0$, which is substantially different from the solution of the first system, even though the coefficient matrices are the same and the constants on the right-hand side are the same through six significant figures. Therefore we call these two systems ill-conditioned.

The first system demonstrates also that we should be cautious about how we determine whether an estimate to a solution is a good estimate or not. If we try $x_1 = 1$, $x_2 = 4$, and $x_3 = 0$ in the first system, we obtain the residual vector

$$\mathbf{r} = \begin{bmatrix} -2. \times 10^{-5} \\ -6. \times 10^{-5} \\ -4. \times 10^{-5} \end{bmatrix}$$

whereas the actual error in using this estimate is on the order of 10^5 times as large as the residual vector would indicate. By (2.34) it follows that the inverse of this coefficient matrix has large entries.

To show what happens when Gauss elimination is used to solve the first system of equations, a single-precision Gauss elimination routine was employed and found

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.9999898 \\ 1.907699 \\ 4.184615 \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} 0.000019 \\ 0.000076 \\ 0.000049 \end{bmatrix}, \quad \mathbf{x}_t - \mathbf{x}_c = \begin{bmatrix} 0.000010 \\ 1.092301 \\ -2.184615 \end{bmatrix}.$$

These results are typical if this system is solved on any digital device that has six- to eight-place accuracy. A double-precision computation on a computer would give almost exactly the correct answer, but going to double precision is obviously not a cure-all, for there are simple examples like the one above for which double-precision arithmetic is not sufficient.

Although this example is somewhat contrived (so that the essence of the problem is not hidden in a mass of cumbersome calculations), many real-life problems, particularly in areas such as statistical analysis and least-squares fits, are ill-conditioned. Thus it is appropriate that a person who must deal with numerical solutions of linear equations have at hand as many tools as possible in order to test computed results for accuracy. Often a very reliable test is simply a feeling for what the computed results are supposed to represent physically; i.e., whether the answers fit the physical problem. In the absence of a physical intuition for what the answer should be, or in terms of a tool for a mathematical analysis of the significance of the computed results, the estimates of Theorem 2.1 provide a beginning.

We continue now with Example 2.14 to illustrate how the $\|\cdot\|_1$ and $\|\cdot\|_\infty$ norms can be readily used to analyze errors in computed solutions. Thus $A\mathbf{x} = \mathbf{b}$ has $\mathbf{x}_t$ as its solution vector where

$$A = \begin{bmatrix} 6 & 6 & 3.00001 \\ 10 & 8 & 4.00003 \\ 6 & 4 & 2.00002 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 30.00002 \\ 42.00006 \\ 22.00004 \end{bmatrix}, \quad \mathbf{x}_t = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}. \tag{2.49}$$

In order not to obscure the point of this example, let us suppose that a computed estimate to the solution, $\mathbf{x}_c$ and hence the residual $\mathbf{r} = A\mathbf{x}_c - \mathbf{b}$ are given by

$$\mathbf{x}_c = \begin{bmatrix} 1 \\ 4 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{r} = \begin{bmatrix} -0.00002 \\ -0.00006 \\ -0.00004 \end{bmatrix}. \tag{2.50}$$

For an analysis of $\mathbf{x}_c - \mathbf{x}_t$, let us use the ℓ_∞ norm and carry only three significant figures. Thus

$$\frac{\|A\mathbf{x}_c - \mathbf{b}\|_\infty}{\|\mathbf{b}\|_\infty} \approx \frac{6 \times 10^{-5}}{42} \approx 1.43 \times 10^{-6}. \tag{2.51}$$

Clearly $\|A\|_\infty = 22.00003 \approx 22$, and therefore by Theorem 2.1

$$\frac{\|\mathbf{x}_c - \mathbf{x}_t\|_\infty}{\|\mathbf{x}_t\|_\infty} \leq (22.)(1.43 \times 10^{-6})\|A^{-1}\|_\infty \leq (3.15 \times 10^{-5})\|A^{-1}\|_\infty. \tag{2.52}$$

Hence an estimate for $\|A^{-1}\|_\infty$ is in order. With the following vector $\mathbf{x}'$ we obtain

$$\mathbf{x}' = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix}, \quad A\mathbf{x}' = \begin{bmatrix} 0.00002 \\ 0.00006 \\ 0.00004 \end{bmatrix}. \tag{2.53}$$

Thus a lower bound for $\|A^{-1}\|_\infty$ is provided by

$$\frac{\|\mathbf{x}'\|_\infty}{\|A\mathbf{x}'\|_\infty} = \frac{2}{6. \times 10^{-5}} \approx 3.33 \times 10^4 \leq \|A^{-1}\|_\infty. \tag{2.54}$$

This estimate for $\|A^{-1}\|_\infty$ makes the estimate of (2.52) more meaningful, for now the upper bound for the relative error in (2.52) is at least as large as $(3.15 \times 10^{-5})(3.33 \times 10^4) \approx 1.05$; and in fact,

$$\frac{\|\mathbf{x}_c - \mathbf{x}_t\|_\infty}{\|\mathbf{x}_t\|_\infty} = \frac{2}{3}.$$

Subsequent sections will make further use of matrix norms in such topics as iterative procedures to solve $A\mathbf{x} = \mathbf{b}$, methods for finding eigenvalues of matrices, methods for solving nonlinear systems, and methods of optimization.

2.3.4. Iterative Improvement

If an error analysis indicates that the computed solution is unacceptable, either we can start again with a different method (such as an indirect method or even Gauss elimination with higher precision), or we can try to improve the answer

we have. Iterative improvement (sometimes known as the "method of residual correction") is a procedure that uses a limited amount of double precision to try to refine a computed solution $\mathbf{x}_c$ of the system $A\mathbf{x} = \mathbf{b}$. The procedure is relatively simple to implement and frequently will yield improved estimates of the solution. In order to describe this method, we again let $\mathbf{x}_t$ denote the true solution of $A\mathbf{x} = \mathbf{b}$ and use $\mathbf{r} = A\mathbf{x}_c - \mathbf{b}$ to denote the residual vector [where $\mathbf{x}_c$ is a computed (approximate) solution to $A\mathbf{x} = \mathbf{b}$]. If we let $\mathbf{e} = \mathbf{x}_c - \mathbf{x}_t$, then $A\mathbf{e} = A\mathbf{x}_c - A\mathbf{x}_t = A\mathbf{x}_c - \mathbf{b} = \mathbf{r}$. Thus if we could solve $A\mathbf{e} = \mathbf{r}$, we would be able to find $\mathbf{x}_t$ from the equation $\mathbf{x}_t = \mathbf{x}_c - \mathbf{e}$. The method of iterative improvement is precisely the implementation of this idea. We immediately see the difficulty inherent in such a procedure since if we cannot solve $A\mathbf{x} = \mathbf{b}$ exactly, we cannot expect to solve $A\mathbf{e} = \mathbf{r}$ exactly.

Iterative improvement, when implemented properly on the computer, attempts to overcome the objection noted above by calculating the residual, $\mathbf{r}$, in double precision. To be precise, the steps in iterative improvement are these.

1. Calculate $\mathbf{r} = A\mathbf{x}_c - \mathbf{b}$ in double precision.
2. Solve $A\mathbf{e} = \mathbf{r}$ and let $\mathbf{e}_c$ be the computed solution to this system.
3. Let $\mathbf{x}_c' = \mathbf{x}_c - \mathbf{e}_c$.

Solving $A\mathbf{e} = \mathbf{r}$ is greatly facilitated by retaining the factored form of A from the previous computation since then one need only form $S\mathbf{r}$ and backsolve the equivalent system $U\mathbf{e} = S\mathbf{r}$. We hope that $\mathbf{x}_c'$ is a better approximation to $\mathbf{x}_t$ than was $\mathbf{x}_c$. In order to analyze the method a bit more carefully, let $\mathbf{e}_t$ denote the true solution to $A\mathbf{e} = \mathbf{r}$. First we can obtain some qualitative information from the steps outlined above by noting that

$$\frac{\|\mathbf{e}_t\|}{\|\mathbf{x}_t\|} = \frac{\|\mathbf{x}_c - \mathbf{x}_t\|}{\|\mathbf{x}_t\|}.$$

Thus we can hope that given the numbers we have, namely $\|\mathbf{e}_c\|$ and $\|\mathbf{x}_c\|$, the ratio

$$\frac{\|\mathbf{e}_c\|}{\|\mathbf{x}_c\|}$$

might give an indication of the relative error.

We next note that the relative error of solving $A\mathbf{e} = \mathbf{r}$ is given by

$$\frac{\|\mathbf{e}_c - \mathbf{e}_t\|}{\|\mathbf{e}_t\|}.$$

From Step 3 we have that $\mathbf{e}_c = \mathbf{x}_c - \mathbf{x}_c'$; so since $\mathbf{e}_t = \mathbf{x}_c - \mathbf{x}_t$, it follows that

$$\frac{\|\mathbf{e}_c - \mathbf{e}_t\|}{\|\mathbf{e}_t\|} = \frac{\|\mathbf{x}_t - \mathbf{x}_c'\|}{\|\mathbf{x}_t - \mathbf{x}_c\|}.$$

Therefore if we can solve $Ae = r$ with a relative error less than 1, then x_c' is a better approximation to x_t than is x_c.

Finally we recall from (2.45) that an error bound for the relative error is of the form

$$\frac{\|x_c - x_t\|}{\|x_t\|} \leq M\|Ax_c - b\| \tag{2.55a}$$

where $M = \|A\|\,\|A^{-1}\|/\|b\|$. We can find a similar bound for $\|x_c' - x_t\|/\|x_t\|$ as follows. We let $r_c = Ae_c - r$ so that $A^{-1}r_c = e_c - e_t$, and hence $\|A^{-1}\|\,\|r_c\| \geq \|e_c - e_t\|$. But, as above, $\|e_c - e_t\| = \|x_t - x_c'\| = \|x_c' - x_t\|$, so $\|x_c' - x_t\| \leq \|A^{-1}\|\,\|Ae_c - r\|$. Just as in Theorem 2.1, $\|A\|\,\|x_t\| \geq \|b\|$; so we get $1/\|x_t\| < \|A\|/\|b\|$. Using these two inequalities, we find

$$\frac{\|x_c' - x_t\|}{\|x_t\|} \leq \frac{\|A\|\,\|A^{-1}\|}{\|b\|}\|Ae_c - r\|,$$

which is an inequality of the form

$$\frac{\|x_c' - x_t\|}{\|x_t\|} \leq M\|Ae_c - r\| \tag{2.55b}$$

where M is the same constant that appears in (2.55a). We can calculate both the numbers $\|Ax_c - b\|$ and $\|Ae_c - r\|$; and if $\|Ae_c - r\| < \|Ax_c - b\|$, we expect that x_c' might be a relatively better approximation to x_t than is x_c. Notice that since x_c' is an approximation to x_t just as was x_c, we can apply iterative improvement again and hope to improve x_c' by solving $Ae = Ax_c' - b$. If the corrections, $e_c^{(i)}$, that we obtain in this fashion do not decrease in size, then we have a strong indication that the matrix A is ill-conditioned.

To illustrate iterative improvement, the method was used on the system in Example 2.7. The approximate solution, x_c, was calculated using a single precision Gauss elimination routine. Displayed below are the results of two applications of iterative improvement. That is, x_c' is the result of solving $Ae = r'$ where $r' = Ax_c - b$; and x_c'' is the result of solving $Ae = r''$ where $r'' = Ax_c' - b$.

$$x_c = \begin{bmatrix} 0.999417\text{E }00 \\ 0.100035\text{E }01 \\ 0.100015\text{E }01 \\ 0.999910\text{E }00 \end{bmatrix} \qquad x_c' = \begin{bmatrix} 0.100004\text{E }01 \\ 0.999977\text{E }00 \\ 0.999991\text{E }00 \\ 0.100000\text{E }01 \end{bmatrix}$$

$$x_c'' = \begin{bmatrix} 0.100000\text{E }01 \\ 0.100000\text{E }01 \\ 0.100000\text{E }01 \\ 0.100000\text{E }01 \end{bmatrix}$$

Thus two steps of the iterative improvement algorithm give the answer correct to as many places as are printed.

PROBLEMS, SECTION 2.3

1. For the matrix A of Problem 7, find a (4×1) vector $\mathbf{x}$, $\mathbf{x} \neq \theta$, such that $\|A\mathbf{x}\|_\infty = \|A\|_\infty \|\mathbf{x}\|_\infty$. [Hint: Consider only vectors $\mathbf{x}$ such that $\|\mathbf{x}\|_\infty = 1$.] Find a (4×1) vector $\mathbf{y}$, $\mathbf{y} \neq \theta$, such that $\|A\mathbf{y}\|_1 = \|A\|_1 \|\mathbf{y}\|_1$. Use A^{-1} and repeat this problem.
2. Let I be the $(n \times n)$ identity matrix. What is $\|I\|_E$? Is there any $(n \times 1)$ vector $\mathbf{x}$, $\mathbf{x} \neq \theta$, such that $\|I\mathbf{x}\|_2 = \|I\|_E \|\mathbf{x}\|_2$?
3. Let $\mathbf{x}$ be any vector in R^n as given by (2.36), and let $\{\mathbf{x}^{(j)}\}_{j=1}^{\infty}$ be any sequence of vectors in R^n.
   a) Show that $\|\mathbf{x}\|_1 \geq \|\mathbf{x}\|_2 \geq \|\mathbf{x}\|_\infty \geq (1/n)\|\mathbf{x}\|_1$ for all $\mathbf{x} \in R^n$.
   b) From the definition of convergence of a sequence of real numbers, use Part (a) to show that if $\lim_{j\to\infty} \|\mathbf{x}^{(j)}\|_p = 0$ for $p = 1, 2$, or ∞, then $\lim_{j\to\infty} \|\mathbf{x}^{(j)}\|_q = 0$ for $q = 1, 2$, or ∞, $q \neq p$.
   c) For p given as 1, 2, or ∞, show that $\lim_{j\to\infty} \|\mathbf{x}^{(j)} - \mathbf{x}\|_p = 0$ implies that
   $$\lim_{j\to\infty} \|A(\mathbf{x}^{(j)} - \mathbf{x})\|_p \equiv \lim_{j\to\infty} \|A\mathbf{x}^{(j)} - A\mathbf{x}\|_p = 0.$$
4. For each norm, $\|\cdot\|_p$, $p = 1, 2, \infty$, graph the set of points $\mathbf{x} = [x_1, x_2]^T$ in R^2 such that $\|\mathbf{x}\|_p = 1$. Compare this result with the results of Problem 3a.
5. If $\|\cdot\|_\infty$ is given by (2.38), show that it satisfies all three properties of (2.37) and thus is a norm for R^n.
6. Let $\|\cdot\|_m$ be any matrix norm on M_n. Show that $\|A\|_m \|A^{-1}\|_m \geq 1$. [Hint: Consider $\|I^2\|_m = \|I\|_m$ and use the norm properties (2.40).]
7. Let A be the (4×4) coefficient matrix of the system in Example 2.6. Then A^{-1} is given by
   $$A^{-1} = \begin{bmatrix} 68 & -41 & -17 & 10 \\ -41 & 25 & 10 & -6 \\ -17 & 10 & 5 & -3 \\ 10 & -6 & -3 & 2 \end{bmatrix}.$$
   Find the condition number $\|A\|_\infty \|A^{-1}\|_\infty$ of this matrix. Use the inequality (2.45) of Theorem 2.1 and the computer results of Example 2.7 to estimate the relative error $\|\mathbf{x}_c - \mathbf{x}_t\|_\infty / \|\mathbf{x}_t\|_\infty$. What is the true relative error?
8. Let A_n be the (2×2) matrix given by
   $$A_n = \begin{bmatrix} 1 & 2 \\ 2 & 4 + 1/n^2 \end{bmatrix}.$$
   Find A_n^{-1} and the condition number $\|A_n\|_\infty \|A_n^{-1}\|_\infty$. Let $n = 100$ so that
   $$A_n = \begin{bmatrix} 1 & 2 \\ 2 & 4.0001 \end{bmatrix},$$
   and let
   $$\mathbf{b} = \begin{bmatrix} 1 \\ 2 - 1/n^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.9999 \end{bmatrix}.$$
   Solve $A_{100}\mathbf{x} = \mathbf{b}$ mathematically and call the answer $\mathbf{x}_t$. Let $\mathbf{x}_c$ be given by
   $$\mathbf{x}_c = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Find $\mathbf{r} = A_{100}\mathbf{x}_c - \mathbf{b}$ and check the error bound $\|\mathbf{x}_c - \mathbf{x}_t\|_\infty \leq \|A^{-1}\|_\infty \|\mathbf{r}\|_\infty$. One might expect from this problem and from Example 2.14 that ill-conditioned matrices must have small determinants. Find the determinant of the matrix A in Problem 7 to see that this suspicion is not always valid.

9. For the matrix A_n in Problem 8, find a (2×2) singular matrix B as in (2.46) such that $(\|A_n - B\|/\|A_n\| - 1/\kappa(A_n)) < 1/n^2$.

10. Let D_n be the $(n \times n)$ diagonal matrix with diagonal entries all equal to 0.1. Compute $\det(D_n)$ and $\|D_n\|_\infty \|D_n^{-1}\|_\infty$. Among all singular $(n \times n)$ matrices B what is about the smallest that the number $\|B - D_n\|_\infty$ can be? For large n, $\det(D_n) \approx 0$; does this mean that D_n is "almost singular"? [See (2.46).] Is D_n ill-conditioned?

11. In the lines of the computations following (2.47), use the previously computed information on the factorization of A from Example 2.10 to solve the (3×3) system $A^T\mathbf{y} = \mathbf{b}$ where $\mathbf{b}$ is given by

(a) $[-5, 3, -1]^T$ (b) $[8, -8, -3]^T$ (c) $[1, -2, -2]^T$

12. Let

$$A'' = \begin{bmatrix} 1 & -1 & 4 \\ -\frac{1}{2} & -1 & 4 \\ \frac{1}{2} & \frac{1}{4} & -3 \end{bmatrix}, \qquad \mathbf{v} = \begin{bmatrix} 3 \\ 3 \end{bmatrix},$$

$$\text{and } \mathbf{b} = \begin{bmatrix} -2 \\ 4 \\ 2 \end{bmatrix}.$$

Suppose that A'' contains the upper-triangular part of a (3×3) matrix A after Gauss elimination, and the lower triangular part of A'' contains the elimination multiples; and suppose that $\mathbf{v}$ has recorded the row interchanges. Follow the lines of Section 2.3.3.

a) Solve $A\mathbf{x} = \mathbf{b}$. b) Solve $A^T\mathbf{y} = \mathbf{b}$. c) Compute an estimate for $\|A^{-1}\|_1$, as in Problem 13.

d) Reconstruct the original matrix A and use it to check parts (a) and (b).

13. Using the first strategy [described prior to (2.48)] for the choice of the vector $\mathbf{c}$, calculate an estimate for $\|H_3^{-1}\|_1$ where H_3 is the (3×3) Hilbert matrix.

14. Repeat Problem 13 and use the second strategy [described following (2.48)] for the choice of the vector $\mathbf{c}$.

15. In describing the technique for solving $A^T\mathbf{y} = \mathbf{b}$, we used the fact that $(S^{-1})^T = (S^T)^{-1}$; prove this.

16. Following the technique described in Section 2.3 and making use of the subroutines FACTOR and SOLVE, write a subroutine to estimate the condition number of an $(n \times n)$ matrix A. (Use either the first or the second strategy for choosing the vector $\mathbf{c}$ and either the 1 or ∞ norms.)

17. Use the subroutine in Problem 16 to estimate $\|H_3^{-1}\|_1$ and $\|A^{-1}\|_1$ where H_3 is the (3×3) Hilbert matrix and A is the (4×4) matrix in Example 2.6.

18. If an $(n \times n)$ matrix A has an LU-decomposition where L and U are known, what are the two necessary steps for solving the system $A^T\mathbf{y} = \mathbf{b}$? Find an LU-decomposition of the (3×3) coefficient matrix in Example 2.2, Section 2.1, and use the decomposition in this manner to solve $A^T\mathbf{y} = \mathbf{e}_1$.

2.4 ITERATIVE METHODS

There are instances in which direct methods such as Gauss elimination and LU-decomposition may not be the best methods to use for solving the system of linear equations $A\mathbf{x} = \mathbf{b}$. There is an alternative class of methods that can be used for this problem; namely, the *iterative* methods. We will first develop the concept of an iterative method and give some particular examples. Later we will discuss the relative merits of iterative and direct methods.

The methods presented here are called iterative because each method is designed to generate a sequence of vectors (*iterates*), $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$, which converge to the true solution, $\mathbf{x}_t$, of $A\mathbf{x} = \mathbf{b}$. The basic idea of iterative methods can be described as follows.

1. The matrix A is written as the difference of two matrices N and P so that $A = N - P$. This decomposition of A is called a *splitting*.
2. An initial guess $\mathbf{x}^{(0)}$ is made for the solution vector $\mathbf{x}_t$.
3. A sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \ldots$, of estimates to $\mathbf{x}_t$ is generated by the formula

$$N\mathbf{x}^{(k+1)} = P\mathbf{x}^{(k)} + \mathbf{b}, \qquad k = 0, 1, 2, \ldots. \tag{2.56}$$

Since the idea of iteration is probably not too familiar, we present an example below before discussing iterative methods further.

EXAMPLE 2.15. Let

$$A = \begin{bmatrix} 4 & 1 & 0 \\ 2 & 5 & 1 \\ -1 & 2 & 4 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}, \text{ and}$$

let $A\mathbf{x} = \mathbf{b}$ be the linear system to be solved. Let the splitting be given by

$$N = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 4 \end{bmatrix} \quad \text{and} \quad P = \begin{bmatrix} 0 & -1 & 0 \\ -2 & 0 & -1 \\ 1 & -2 & 0 \end{bmatrix}$$

so that $A = N - P$. If we denote the vector $\mathbf{x}^{(k)}$ by $\mathbf{x}^{(k)} = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix}$,

then writing out formula (2.56) yields the equations

$$\begin{aligned} 4x_{k+1} &= -y_k + 1 \\ 5y_{k+1} &= -2x_k - z_k \\ 4z_{k+1} &= x_k - 2y_k + 3, \end{aligned}$$

which define $\mathbf{x}^{(k+1)}$ for $k = 0, 1, 2, \ldots$. As an initial guess, let

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Then the first few iterates are

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{x}^{(1)} = \begin{bmatrix} 0 \\ -\frac{3}{5} \\ \frac{1}{2} \end{bmatrix}, \quad \mathbf{x}^{(2)} = \begin{bmatrix} \frac{2}{5} \\ -\frac{1}{10} \\ \frac{21}{20} \end{bmatrix}.$$

[The true solution in this example is $\mathbf{x}_t = (1/3, -1/3, 1)^T$.]

Let us return now to the general procedure outlined in steps (1) to (3). The first thing to observe is that if $A\mathbf{x}_t = \mathbf{b}$ then $N\mathbf{x}_t = P\mathbf{x}_t + \mathbf{b}$, and vice versa (that is, solving $A\mathbf{x} = \mathbf{b}$ is equivalent to solving $N\mathbf{x} = P\mathbf{x} + \mathbf{b}$). In order to get some idea of what might constitute a good choice for N and P, consider formula (2.56). This formula says that if we have $\mathbf{x}^{(k)}$, then we can get the next iterate $\mathbf{x}^{(k+1)}$ provided we can solve the linear system $N\mathbf{x}^{(k+1)} = \mathbf{h}^{(k)}$ where the vector $\mathbf{h}^{(k)}$ is given by $\mathbf{h}^{(k)} = P\mathbf{x}^{(k)} + \mathbf{b}$. Thus it is clear that we must require N to be nonsingular in order to be assured that we can implement the iteration. Furthermore, for an iterative procedure to be efficient, N should be chosen so that $N\mathbf{x}^{(k+1)} = \mathbf{h}^{(k)}$ is quite easy to solve. This is the case if, for instance, N is chosen to be a triangular matrix (or a diagonal matrix as in Example 2.15 above).

Last, the question of convergence to $\mathbf{x}_t$ must be considered. It is fairly easy to make a start at answering this question. Let $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}_t$ denote the error vector at the kth step. As noted above, $N\mathbf{x}_t = P\mathbf{x}_t + \mathbf{b}$; so from this and (2.56) it follows that $N(\mathbf{x}^{(k+1)} - \mathbf{x}_t) = P(\mathbf{x}^{(k)} - \mathbf{x}_t)$ or $N\mathbf{e}^{(k+1)} = P\mathbf{e}^{(k)}$. Since we have required that N be nonsingular, we can multiply by N^{-1} to obtain $\mathbf{e}^{(k+1)} = N^{-1}P\mathbf{e}^{(k)}$ for $k = 0, 1, 2, \ldots$. If we set $M \equiv N^{-1}P$, then a fundamental relationship among the error vectors has been established:

$$\mathbf{e}^{(k+1)} = M\mathbf{e}^{(k)}, \qquad k = 0, 1, 2, \ldots. \tag{2.57}$$

If M is in some sense a "small" matrix, then (2.57) would indicate that the errors are diminishing, or that $\{\mathbf{x}^{(k)}\} \to \mathbf{x}_t$. This statement is made more precise in Theorem 2.2 below. In this theorem, we use the ℓ_∞ vector and matrix norms. It is evident from the proof, however, that any compatible vector and matrix norms could have been used.

Theorem 2.2

Suppose $A = N - P$ and suppose $\|N^{-1}P\|_\infty \leq \lambda < 1$. Then

1. A is nonsingular;
2. if $\mathbf{x}_t$ is the solution of $A\mathbf{x} = \mathbf{b}$ and if $\{\mathbf{x}^{(j)}\}$ is given by (2.56), then $\lim_{j\to\infty} \mathbf{x}^{(j)} = \mathbf{x}_t$; and
3. $\|\mathbf{x}^{(j)} - \mathbf{x}_t\|_\infty \leq \lambda^j \|\mathbf{x}^{(0)} - \mathbf{x}_t\|_\infty$ (or $\|\mathbf{e}^{(j)}\|_\infty \leq \lambda^j \|\mathbf{e}^{(0)}\|_\infty$).

Proof. Before establishing Theorem 2.2, it is worth observing that a theorem of this type has a lot of practical computational value when it is possible to

verify the hypotheses. The theorem

1. guarantees a solution to the problem,
2. shows that the numerical method will converge to the solution for any initial guess $\mathbf{x}^{(0)}$, and
3. indicates to the user of the numerical method how many steps should be taken to attain a desired accuracy.

Part (1) is easiest to prove by contradiction. If A is singular, then there must be a nonzero vector $\mathbf{y}$ such that $A\mathbf{y} = \theta$. Therefore $(N - P)\mathbf{y} = \theta$ or $\mathbf{y} = N^{-1}P\mathbf{y}$. From the compatibility properties of the ℓ_∞ vector and matrix norms, it follows that

$$\|\mathbf{y}\|_\infty = \|N^{-1}P\mathbf{y}\|_\infty \leq \|N^{-1}P\|_\infty \|\mathbf{y}\|_\infty \leq \lambda \|\mathbf{y}\|_\infty. \tag{2.58}$$

Since $\mathbf{y} \neq \theta$, then $\|\mathbf{y}\|_\infty > 0$; and hence the inequality (2.58) means that $1 \leq \lambda$. This statement is a contradiction of the hypothesis; so it must be that there is no nonzero vector $\mathbf{y}$ such that $A\mathbf{y} = \theta$. Hence A is nonsingular.

We next establish part (3). If we set $M = N^{-1}P$ and $\mathbf{e}^{(j)} = \mathbf{x}^{(j)} - \mathbf{x}_t$, then a repeated application of (2.57) gives

$$\mathbf{e}^{(j)} = M\mathbf{e}^{(j-1)} = M(M\mathbf{e}^{(j-2)}) = M^2\mathbf{e}^{(j-2)} = \cdots = M^j\mathbf{e}^{(0)}.$$

Since $\|\mathbf{e}^{(j)}\|_\infty \leq \|M^j\|_\infty \|\mathbf{e}^{(0)}\|_\infty \leq \|M\|_\infty^j \|\mathbf{e}^{(0)}\|_\infty \leq \lambda^j \|\mathbf{e}^{(0)}\|_\infty$, part (3) of the theorem is proved. Since $\lim_{j\to\infty} \|\mathbf{x}^{(j)} - \mathbf{x}_t\|_\infty = 0$ and since $\|\mathbf{x}^{(j)} - \mathbf{x}_t\|_\infty$ is the absolute value of the largest component of the vector $\mathbf{x}^{(j)} - \mathbf{x}_t$, it follows that $\lim_{j\to\infty} \mathbf{x}^{(j)} = \mathbf{x}_t$. This proves part (2). (We note from Problem 3, Section 2.3, that $\|\mathbf{x}^{(j)} - \mathbf{x}_t\|_2 \to 0$ and $\|\mathbf{x}^{(j)} - \mathbf{x}_t\|_1 \to 0$ as well). ■

Theorem 2.2 together with the observations made previously allows us to summarize the properties that a splitting should have in order to define a useful iterative method. For $A = N - P$, the properties are these.

1. N should be nonsingular.
2. The equation $N\mathbf{x} = \mathbf{h}$ should be easy to solve.
3. $\|N^{-1}P\|$ should be less than 1 for some matrix norm.

In this context, condition (1) assures us that the sequence $\{\mathbf{x}^{(j)}\}$ given by formula (2.56) can be generated. Condition (2) assures us that the sequence $\{\mathbf{x}^{(j)}\}$ can be generated efficiently (after all, we do not want to work as hard to perform one step of an iterative method as we would to solve $A\mathbf{x} = \mathbf{b}$ by a direct method). Last, condition (3) assures us that the sequence we generate will in fact converge to the solution $\mathbf{x}_t$.

2.4.1. Basic Iterative Methods

The first and simplest iterative method described in this section is the Jacobi method. For this method N is taken to be a diagonal matrix with its main

diagonal entries equal to a_{ii}. The matrix P is then determined by $P = N - A$. We note that P can be visualized as the sum of a lower-triangular matrix and an upper-triangular matrix. For the purposes of analyzing the Jacobi method and the ensuing Gauss-Seidel method, it is quite convenient to think of P in this way. To be specific, let $A = (a_{ij})$ be an $(n \times n)$ matrix. Define L, D, and U to be the lower-triangular, diagonal, and upper-triangular parts of A:

$$L = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ a_{21} & 0 & 0 & \cdots & 0 \\ a_{31} & a_{32} & 0 & \cdots & 0 \\ \vdots & & & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & 0 \end{bmatrix}, \quad D = \begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ 0 & 0 & 0 & \cdots & a_{3n} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

Thus $A = L + D + U$, and so the Jacobi splitting is given by $N = D$ and $P = -(L + U)$.

The Jacobi method for solving $A\mathbf{x} = \mathbf{b}$ (that is, the splitting defined above) is this:

$$D\mathbf{x}^{(k+1)} = -(L + U)\mathbf{x}^{(k)} + \mathbf{b}. \tag{2.59}$$

The matrix $M_J = -D^{-1}(L + U)$ is called the *Jacobi matrix*. In actual computation, Eq. (2.59) would have to be written out elementwise. Suppose the vector $\mathbf{x}^{(k)}$ is given by

$$\mathbf{x}^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}, \qquad k = 0, 1, 2, \ldots . \tag{2.60}$$

Then Eq. (2.59) leads to the following iteration for the ith component of $\mathbf{x}^{(k+1)}$:

$$x_i^{(k+1)} = \frac{-1}{a_{ii}}\left[\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}x_j^{(k)} - b_i\right], \qquad i = 1, 2, \ldots, n. \tag{2.61}$$

This formula shows that the Jacobi iteration is quite easy to program. The only real problem is to determine an efficient test for terminating the iteration. Also, it is obvious from (2.61) or from (2.59) that in order for the Jacobi method to be used, the diagonal elements of A must all be nonzero. In practice, this requirement causes no real difficulty. If A is the coefficient matrix of the system $A\mathbf{x} = \mathbf{b}$ and if $a_{ii} = 0$, then the ith equation can be interchanged with another equation that will give a coefficient matrix with a nonzero diagonal entry in the

ith row. Thus the situation with the Jacobi method is similar to that of Gauss elimination in which the possibility of zero pivot elements must be guarded against.

Finally, we note that when D^{-1} exists, it is relatively easy (in comparison to a direct method) to carry out each step of the iteration. Thus in those cases in which $\|-D^{-1}(L + U)\| < 1$, the Jacobi method provides an alternative to direct methods.

Examination of (2.61) reveals that each component of the vector $\mathbf{x}^{(k+1)}$ is computed entirely from the vector $\mathbf{x}^{(k)}$. If $x_j^{(k+1)}$ is assumed to be closer to the true answer than $x_j^{(k)}$, the estimate for $x_i^{(k+1)}$ should be improved by replacing $x_j^{(k)}$ by $x_j^{(k+1)}$ whenever $j < i$. That is, we should use our most recent information as soon as it becomes available. The implementation of this idea leads to the procedure known as the Gauss-Seidel method.

If we use the new information as soon as it is available in (2.61), we obtain (after multiplication by a_{ii}) this equation:

$$a_{ii}x_i^{(k+1)} = -\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + b_i, \qquad i = 1, \ldots, n, \quad (2.62)$$

(in which we interpret the first sum as zero when $i = 1$). We can write this equation in matrix form, using $A = L + D + U$ as in the Jacobi method, and obtain

$$D\mathbf{x}^{(k+1)} = -L\mathbf{x}^{(k+1)} - U\mathbf{x}^{(k)} + \mathbf{b}. \quad (2.63)$$

Putting this in the standard form Eq. (2.56) for an iterative method, we have

$$(D + L)\mathbf{x}^{(k+1)} = -U\mathbf{x}^{(k)} + \mathbf{b}. \quad (2.64)$$

The matrix $M_G = -(D + L)^{-1}U$ is called the *Gauss-Seidel matrix*. Since the Gauss-Seidel method is refinement of the Jacobi method, the former usually (but not always) converges faster. For deeper results on convergence and comparison of rates of convergence, see the Ostrowski-Reich and Stein-Rosenberg Theorems in Varga (1962). Note that the choice of the starting vector $\mathbf{x}^{(0)}$ is not particularly critical, and one natural choice is $\mathbf{x}^{(0)} = \theta$. We will have more to say of this choice in Section 3.4.

EXAMPLE 2.16. As an example of the sorts of computational results that the Jacobi and Gauss-Seidel methods give, consider the linear system

$$\begin{aligned} 3x_1 + x_2 + x_3 &= 5 \\ 2x_1 + 6x_2 + x_3 &= 9 \\ x_1 + x_2 + 4x_3 &= 6 \end{aligned} \qquad \text{with solution vector} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

With $\mathbf{x}^{(0)} = \theta$, we obtain Tables 2.1 and 2.2. The coefficient matrix of the system is *diagonally dominant*, a condition that is sufficient to guarantee convergence of the Jacobi and Gauss-Seidel iterations (see Theorem 2.3).

TABLE 2.1 Jacobi iteration.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	0.166667E 01	0.150000E 01	0.150000E 01
2	0.666667E 00	0.694445E 00	0.708333E 00
3	0.119907E 01	0.115972E 01	0.115972E 01
4	0.893518E 00	0.907022E 00	0.910301E 00
5	0.106089E 01	0.105044E 01	0.104986E 01
6	0.966564E 00	0.971392E 00	0.972166E 00
7	0.101881E 01	0.101578E 01	0.101551E 01
8	0.989568E 00	0.991144E 00	0.991350E 00
9	0.100584E 01	0.100492E 01	0.100482E 01
10	0.996753E 00	0.997251E 00	0.997312E 00
11	0.100181E 01	0.100153E 01	0.100150E 01
12	0.998991E 00	0.999146E 00	0.999165E 00
13	0.100056E 01	0.100047E 01	0.100047E 01
14	0.999687E 00	0.999735E 00	0.999741E 00
15	0.100017E 01	0.100015E 01	0.100014E 01
16	0.999903E 00	0.999918E 00	0.999919E 00
17	0.100005E 01	0.100005E 01	0.100004E 01
18	0.999970E 00	0.999974E 00	0.999975E 00
19	0.100002E 01	0.100001E 01	0.100001E 01
20	0.999991E 00	0.999992E 00	0.999992E 00

TABLE 2.2 Gauss-Seidel iteration.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	0.166667E 01	0.944445E 00	0.847222E 00
2	0.106944E 01	0.100231E 01	0.982060E 00
3	0.100521E 01	0.100125E 01	0.998385E 00
4	0.100012E 01	0.100023E 01	0.999913E 00
5	0.999953E 00	0.100003E 01	0.100000E 01
6	0.999989E 00	0.100000E 01	0.100000E 01
7	0.999998E 00	0.100000E 01	0.100000E 01
8	0.100000E 01	0.100000E 01	0.100000E 01

As an example in which iteration is not so successful, consider the (4×4) linear system of Example 2.6 (solved by Gauss elimination in Example 2.7). This coefficient matrix is *positive-definite* and hence the Gauss-Seidel iteration will converge (see Theorem 2.4); but as can be seen, convergence is exceedingly slow. (See Table 2.3.) The question of how fast an iterative procedure will converge is considered in Section 3.4. Through the theory of the above-mentioned section it can be shown that the Jacobi method will not converge for the (4×4) system above.

TABLE 2.3

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$x_4^{(k)}$
1	0.460000E 01	−0.199982E−01	0.556000E 00	0.313602E 00
2	0.364720E 01	−0.173599E−01	0.843327E 00	0.529559E 00
3	0.308276E 01	−0.327911E−02	0.976370E 00	0.682185E 00
4	0.275076E 01	0.158432E−01	0.102290E 01	0.792917E 00
5	0.255742E 01	0.364410E−01	0.102277E 01	0.875290E 00
6	0.244637E 01	0.566254E−01	0.999118E 00	0.937971E 00
7	0.238381E 01	0.754578E−01	0.965172E 00	0.986620E 00
8	0.234954E 01	0.925567E−01	0.928278E 00	0.102499E 01
9	0.233149E 01	0.107840E 00	0.892337E 00	0.105566E 01
10	0.232256E 01	0.121379E 00	0.859268E 00	0.108042E 01
⋮				
50	0.220322E 01	0.276739E 00	0.694661E 00	0.117948E 01
51	0.219950E 01	0.278992E 00	0.695580E 00	0.117894E 01
52	0.219578E 01	0.281235E 00	0.696501E 00	0.117840E 01
⋮				
100	0.203333E 01	0.378932E 00	0.737633E 00	0.115422E 01
101	0.203012E 01	0.380858E 00	0.738446E 00	0.115374E 01
102	0.202693E 01	0.382777E 00	0.739256E 00	0.115326E 01
⋮				
196	0.176445E 01	0.540536E 00	0.805902E 00	0.111409E 01
197	0.176208E 01	0.541962E 00	0.806505E 00	0.111373E 01
198	0.175972E 01	0.543384E 00	0.807103E 00	0.111338E 01
199	0.175736E 01	0.544801E 00	0.807701E 00	0.111303E 01
200	0.175501E 01	0.546213E 00	0.808298E 00	0.111268E 01

The question of convergence for iterative methods is sometimes hard to analyze, for convergence depends on $M = N^{-1}P$ being small in some sense as is shown in (2.57). For the Jacobi method, it is easy to display the form of the Jacobi matrix M_J. From (2.59) it follows quickly that

$$M_J = -\begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \cdots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \frac{a_{23}}{a_{22}} & \cdots & \frac{a_{2n}}{a_{22}} \\ \vdots & & & & \vdots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \frac{a_{n3}}{a_{nn}} & \cdots & 0 \end{bmatrix}. \tag{2.65}$$

An explicit representation for M_G for a general matrix A is obviously more complicated and it is of little use to find M_G.

While Theorem 2.2 provides a general criterion for convergence, it would be helpful to have some convergence theorems that related specifically to the Jacobi and Gauss-Seidel methods. In particular, the ideal theorem would be

one that guaranteed convergence for all matrices in some particular class or for all matrices that satisfy some particular property. Below, we illustrate two theorems of this type, theorems that guarantee convergence for *diagonally dominant* matrices and for *symmetric positive-definite* matrices. Diagonally dominant matrices and symmetric positive-definite matrices, two important types of matrices that occur frequently in practical problems, are defined below.

We say a matrix $A = (a_{ij})$ is diagonally dominant if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \text{ for } i = 1, 2, \ldots, n. \tag{2.66}$$

That is, each main diagonal entry is larger in absolute value than the sum of the absolute values of all the off-diagonal entries in that row. An $(n \times n)$ matrix $A = (a_{ij})$ is called *symmetric* if

$$A^T = A. \tag{2.67}$$

Clearly, an equivalent way of stating that A is symmetric is to say that $a_{ij} = a_{ji}$ for $1 \leq i \leq n$, $1 \leq j \leq n$. Finally, an $(n \times n)$ matrix $A = (a_{ij})$ is said to be *symmetric* and *positive-definite* if

1. A is symmetric and
2. $\mathbf{x}^T A\mathbf{x} > 0$ for all $\mathbf{x} \in R^n$, $\mathbf{x} \neq \theta$. (2.68)

[Note that since A is $(n \times n)$ and $\mathbf{x}$ is $(n \times 1)$, then $A\mathbf{x}$ is $(n \times 1)$. Thus since $\mathbf{x}$ is $(n \times 1)$, $\mathbf{x}^T$ is $(1 \times n)$ and so we have that $\mathbf{x}^T A\mathbf{x}$ is a scalar quantity. Most readers will recognize $\mathbf{x}^T A\mathbf{x}$ as the familiar "dot product" of the vector $\mathbf{x}$ and the vector $A\mathbf{x}$.]

EXAMPLE 2.17. Consider the simple (2×2) matrix A, $A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$. Clearly A is diagonally dominant and also symmetric. To determine whether A is positive-definite, we consider

$$\mathbf{x}^T A\mathbf{x} = [x_1, x_2] \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [x_1, x_2] \begin{bmatrix} 2x_1 + x_2 \\ x_1 + 3x_2 \end{bmatrix}.$$

Therefore $\mathbf{x}^T A\mathbf{x} = 2x_1^2 + x_1x_2 + x_2x_1 + 3x_2^2 = (x_1 + x_2)^2 + x_1^2 + 2x_2^2$. Hence, we see that A is positive-definite since $\mathbf{x}^T A\mathbf{x} > 0$ whenever $\mathbf{x} \neq \theta$. If we modify A and consider $B = \begin{bmatrix} 2 & -\frac{1}{3} \\ 1 & -3 \end{bmatrix}$, then we still have that B is symmetric and diagonally dominant. However, B is not positive-definite since for $\mathbf{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, $\mathbf{x}^T B\mathbf{x} = (0, 1)\begin{pmatrix} -\frac{1}{3} \\ -3 \end{pmatrix} = -3 < 0$.

Having the concepts of diagonally dominant and positive-definite, we can now state two convergence theorems.

Theorem 2.3.

If A is diagonally dominant, then A is nonsingular and the sequence $\{\mathbf{x}^{(k)}\}$ defined by the Jacobi method (2.59) converges for any initial guess $\mathbf{x}^{(0)}$.

Proof. Dividing both sides of inequality (2.66) by $|a_{ii}|$ for $i = 1, 2, \ldots, n$, we obtain

$$1 > \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} \quad \text{for } i = 1, 2, \ldots, n.$$

Referring to (2.65), we see that this inequality means that $\|M_J\|_\infty < 1$. Consequently, by Theorem 2.2, A is nonsingular and the Jacobi method will converge. ∎

The fact that a diagonally dominant matrix is nonsingular will be important also in our study of cubic splines in Chapter 5. The proof of Theorem 2.4 is more difficult and we defer it to Section 3.4.

Theorem 2.4.

If A is symmetric and positive-definite, then the sequence $\{\mathbf{x}^{(k)}\}$ defined by the Gauss-Seidel method (2.64) converges for any initial guess $\mathbf{x}^{(0)}$.

Both of these theorems are quite useful in various computational settings. For example, some numerical procedures result in large linear systems $A\mathbf{x} = \mathbf{b}$, which must be solved, where A is diagonally dominant. Theorem 2.3 guarantees that the Jacobi method will work on this system. In Chapter 8 we consider some finite-difference approximations to partial differential equations which give rise to systems with symmetric, positive-definite coefficient matrices.

2.4.2. Implementation of Iterative Methods

There are many other important iterative methods besides Jacobi and Gauss-Seidel. Two other important, but rather more complicated methods are successive overrelaxation (SOR) and alternating-direction implicit methods (ADI) (see Chapter 8). A good advanced reference is Varga (1962). Iterative methods are normally used when it can easily be shown in advance that they will converge and that they will require less computation than a direct method. The numerical solution of partial differential equations provides the most common area of application for iterative methods. When finite difference or finite element schemes are used, the end result is to replace the partial differential equation by a large matrix equation $A\mathbf{x} = \mathbf{b}$. The matrix equation is an approximation to the partial differential equation, and the solution of $A\mathbf{x} = \mathbf{b}$ gives approximate values for the function that satisfies the partial differential equation. These matrices have a structure that is known in advance; so it is normally easy to see whether or not an iterative procedure will converge. Moreover, these matrices are often quite large [say (1000×1000) perhaps] and *sparse* (that

is, there are relatively few nonzero entries in the matrix). In the following, we indicate how this "sparseness" can be exploited by an iterative method.

Note that an iterative method written in the form

$$\mathbf{x}^{(k+1)} = M\mathbf{x}^{(k)} + \mathbf{g} \tag{2.69}$$

where M is $(n \times n)$ requires just n^2 multiplications per iteration. This observation forms a part of the reason why iterative methods are valuable since a direct method like Gauss elimination takes about $(n^3/3)$ multiplications. Thus if we are satisfied with the approximate solution $\mathbf{x}^{(k)}$ where $k < n/3$, we have saved machine time. In the context of partial differential equations, since the equation $A\mathbf{x} = \mathbf{b}$ is itself an approximation, it is not necessary to ask for an extremely good estimate of the true solution $\mathbf{x}_t$. Another point to be made is that if the matrix A is sparse [a typical example is when A is (1000×1000) with five nonzero entries per row], then iterative methods can take advantage of the fact that multiplication by zero is not necessary. For example in (2.61) or (2.62), if only four of the coefficients a_{ij}/a_{ii} are nonzero for each i, then only 5 multiplications are required to form $x_i^{(k+1)}$. Thus forming $\mathbf{x}^{(k+1)}$ in the Jacobi method (or the Gauss-Seidel method) would take $5n$ multiplications. The total number of multiplications needed to form $\mathbf{x}^{(j)}$ from $\mathbf{x}^{(0)}$ is then $5nj$. So for $j \leq n^2/15$, less effort is needed to compute $\mathbf{x}^{(j)}$ than is needed to solve the system by Gauss elimination. A final consideration is that of storage since a matrix with only five nonzero entries per row needs only $5n$ storage locations. If, however, Gauss elimination is used on the same sparse system, (as a whole) the system may become less sparse as the elimination proceeds. Thus Gauss elimination might require many more than $5n$ storage locations. We should also remark that sparse matrices arising from practical problems may have such a regular pattern that programming this information is easy.

EXAMPLE 2.18. Consider a system of linear equations with the coefficient matrix

$$\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}. \tag{2.70}$$

This provides a graphic example of a sparse matrix that becomes "dense" as Gauss elimination is used. To illustrate our point, this example is admittedly contrived in that we could interchange the first and last rows, and solution by Gauss elimination becomes almost immediate. The point of the example, however, is well taken and remains valid in less contrived situations.

PROBLEMS, SECTION 2.4

1. Write a computer program to implement the Jacobi method.
2. Write a computer program to implement the Gauss-Seidel method.
3. Test the Gauss-Seidel and Jacobi routines of Problem 1 and Problem 2 on the (2×2) system $A^TAx^* = A^Tb$ in Example 2.19 of Section 2.5.
4. Let $A = (a_{ij})$ be the (4×4) diagonally dominant matrix given by $a_{ii} = 10$, $i = 1, 2, 3, 4$, and $a_{ij} = 2$, for $i \neq j$. Use the Gauss-Seidel and Jacobi methods to solve $Ax = b$ where $b^T = (16, 24, 16, 8)$. [The exact solution is $x^T = (1, 2, 1, 0)$.]
5. Let $A = N - P$ where

$$N = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix}, \qquad P = \begin{bmatrix} -1 & -1 & 1 \\ -2 & -1 & 2 \\ -1 & -1 & -1 \end{bmatrix}.$$

   a) Show that the iterative method for this splitting converges to the solution of $Ax = b$.

   b) Use (2.57) to argue why you expect the Jacobi method to converge faster than this method.

6. For $k = 0, 1, \ldots$, let $x^{(k)} = [x_k, y_k, z_k]^T$ be generated by the equations

$$\begin{aligned} 4x_{k+1} &= -x_k - y_k + z_k + 2 \\ 6y_{k+1} &= 2x_k + y_k - z_k - 1 \\ -4z_{k+1} &= -x_k + y_k - z_k + 4. \end{aligned}$$

   Show that the vector sequence $\{x^{(k)}\}_{k=1}^{\infty}$ is convergent for any initial vector $x^{(0)}$, and determine the vector x to which it converges. [Hint: Find A, N, P, and b.]

7. How many multiplications and divisions are required for one iteration of the Gauss-Seidel method? How many iterations may be performed before this number exceeds the operations count of Gauss elimination?
8. If the $(n \times n)$ matrix A is nonsingular, then there is at least one nonzero entry in the first column. (Why?) Suppose now that no matter what row interchange is made to create a nonzero (1, 1) entry, the resulting (2, 2), . . . , $(n, 2)$ entries are zero. Examine the first two columns and explain why this situation contradicts the nonsingularity of A.
9. Let A be an $(m \times n)$ matrix. Using Problem 13, Section 2.1, show that A^TA is an $(n \times n)$ symmetric matrix.
10. If y is any $(n \times 1)$ vector, $y \neq \theta$, show that $y^Ty > 0$.
11. Suppose A is an $(m \times n)$ matrix such that $Ay \neq \theta$ for any $(n \times 1)$ vector y where $y \neq \theta$. Using Problems 9 and 10, show that A^TA is a symmetric, positive-definite matrix.
12. Using Problem 11, verify for the (4×2) matrix A in Example 2.19 that A^TA is symmetric and positive-definite. (From Theorem 2.4, the results of this problem have a bearing on Problem 3.)
13. For the matrix A^TA of Example 2.19, verify that the Jacobi matrix $M_J = -D^{-1}(L + U)$ is given by

$$M_J = -\begin{bmatrix} 0 & \frac{3}{7} \\ \frac{3}{2} & 0 \end{bmatrix} \quad \text{and} \quad M_J^2 = \begin{bmatrix} \frac{9}{14} & 0 \\ 0 & \frac{9}{14} \end{bmatrix}.$$

Using $M_J^4 = M_J^2 M_J^2$, $M_J^6 = M_J^2 M_J^2 M_J^2$, etc., show that the entries of $M_J^{2^n}$ all tend to zero as $n \to \infty$. From this information, give a bound for $\|M_J^j\|_\infty$; and using (2.57), show that the Jacobi method must converge for any initial guess $\mathbf{x}^{(0)}$. Note that A^TA is not diagonally dominant so that Theorem 2.3 does not apply. Additionally, $\|M_J\| > 1$ for the three matrix norms of (2.41) so that Theorem 2.2 does not apply either. However, the analysis of this problem shows why the Jacobi method is convergent in Problem 3.

14. As another example of a slowly converging iterative method, use the Gauss-Seidel iteration to solve the (2×2) system $A_{100}\mathbf{x} = \mathbf{b}$ of Problem 8, Section 2.3.

15. Assume that an original system of equations has been altered to an equivalent system $A\mathbf{x} = \mathbf{b}$ where each of the diagonal entries of A equals one ($a_{ii} = 1$). From (2.64) the splitting for the Gauss-Seidel method is $(I + L)\mathbf{x} = -U\mathbf{x} + \mathbf{b}$. Multiplying this equation by a scalar ω and adding $\mathbf{x}$ to both sides lead to the iterative method

$$(I + \omega L)\mathbf{x}^{(k+1)} = [(1 - \omega)I - \omega U]\mathbf{x}^{(k)} + \omega\mathbf{b}.$$

(Verify these steps.) This method is called successive overrelaxation (SOR) and ω is called the relaxation factor. How would one use (2.57) to choose a value of ω to make this iteration faster than Gauss-Seidel? Program this method and try to find an ω that improves the convergence on the system of Table 2.3. (SOR methods are frequently used on systems arising from partial differential equations. The choice of ω is a difficult and delicate matter, and the value of ω is sometimes changed with each step of the iteration.)

2.5 LEAST-SQUARES SOLUTION OF OVERDETERMINED LINEAR SYSTEMS

We now return our attention to the linear system (2.1) [or Eq. (2.2) in matrix form] in which we have m linear equations in n unknowns; and we consider the case in which $m > n$. Since we have more equations than unknowns, we do not normally expect a solution, $\mathbf{x}$, of $A\mathbf{x} = \mathbf{b}$ to exist. (As pointed out previously, if such an $\mathbf{x}$ does exist, Gauss elimination can be implemented to find it.) However, in this section we shall assume that $A\mathbf{x} = \mathbf{b}$ does *not* have a solution. The reader may thus feel that this is necessarily an unimportant problem. Quite the opposite is true, however, for we can reformulate the problem to ask for the vector $\mathbf{x}^*$ that somehow "minimizes" the vector expression $(A\mathbf{x}^* - \mathbf{b})$. That is, find $\mathbf{x}^*$ such that $\|A\mathbf{x}^* - \mathbf{b}\|$ is minimized for some vector norm $\|\cdot\|$. In the case in which we use the Euclidean norm, $\|\cdot\|_2$, this problem becomes precisely the important statistical problem of finding the best *least-squares solution* of $A\mathbf{x} = \mathbf{b}$. We shall see later that this problem is a special case of the famous Fourier series approximation problem, but for now we shall be content to solve it in this one particular context.

Let A be any given $(m \times n)$ matrix, and define Y to be the set of vectors $\mathbf{y}$ in R^m such that

$$Y = \{\mathbf{y} \in R^m : \mathbf{y} = A\mathbf{x}, \text{ for some } \mathbf{x} \in R^n\}.$$

Thus Y is in reality the "range of A" and is a subset (actually a "subspace") of R^m. Since we have assumed that there is no $\mathbf{x} \in R^n$ such that $A\mathbf{x} = \mathbf{b}$, then $\mathbf{b}$ does not belong to Y in this case. (However, the following procedures would be valid even if $\mathbf{b} \in Y$.) Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ be arbitrary in R^n, and write $A = [\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n]$ where each $\mathbf{A}_k$ is the kth $(m \times 1)$ column vector of A, $1 \leq k \leq n$. Recall from Section 2.1 that $A\mathbf{x}$ can be written as

$$A\mathbf{x} = x_1\mathbf{A}_1 + x_2\mathbf{A}_2 + \cdots + x_n\mathbf{A}_n.$$

Thus we see that Y can be equivalently written as

$$Y = \{\mathbf{y} \in R^m: \mathbf{y} = x_1\mathbf{A}_1 + x_2\mathbf{A}_2 + \cdots + x_n\mathbf{A}_n\}.$$

Therefore we can see that our problem of minimizing $\|A\mathbf{x} - \mathbf{b}\|_2$ can be restated thus: find $\mathbf{y}^* \in Y$ such that $\|\mathbf{y}^* - \mathbf{b}\|_2 \leq \|\mathbf{y} - \mathbf{b}\|_2$ for all $\mathbf{y} \in Y$, and then find $\mathbf{x}^* \in R^n$ such that $A\mathbf{x}^* = \mathbf{y}^*$. This problem can be formally solved easily by use of the following theorem.

Theorem 2.5

Assume that there exists a vector $\mathbf{y}^* \in Y$ such that $(\mathbf{b} - \mathbf{y}^*)^T\mathbf{y} = 0$ for all $\mathbf{y} \in Y$. Then $\|\mathbf{b} - \mathbf{y}^*\|_2 \leq \|\mathbf{b} - \mathbf{y}\|_2$ for all $\mathbf{y} \in Y$.

Proof. We observe that if $\mathbf{z}$ is any vector in R^m, then $\mathbf{z}^T\mathbf{z} = \|\mathbf{z}\|_2^2$ and $\mathbf{x}^T\mathbf{z} = \mathbf{z}^T\mathbf{x}$. Let $\mathbf{y}$ be any vector in Y and remember in the following that $(\mathbf{b} - \mathbf{y}^*)^T\mathbf{y} = 0$.

$$\begin{aligned} 0 \leq \|\mathbf{b} - \mathbf{y}\|_2^2 &= (\mathbf{b} - \mathbf{y})^T(\mathbf{b} - \mathbf{y}) = \mathbf{b}^T\mathbf{b} - 2\mathbf{b}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} \\ &= \mathbf{b}^T\mathbf{b} + 2(\mathbf{b} - \mathbf{y}^*)^T\mathbf{y} - 2\mathbf{b}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} \\ &= \mathbf{b}^T\mathbf{b} + 2\mathbf{b}^T\mathbf{y} - 2\mathbf{y}^{*T}\mathbf{y} - 2\mathbf{b}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} \\ &= \mathbf{b}^T\mathbf{b} - \mathbf{y}^{*T}\mathbf{y}^* + (\mathbf{y}^{*T}\mathbf{y}^* - 2\mathbf{y}^{*T}\mathbf{y} + \mathbf{y}^T\mathbf{y}) \\ &= \|\mathbf{b}\|_2^2 - \|\mathbf{y}^*\|_2^2 + (\mathbf{y}^* - \mathbf{y})^T(\mathbf{y}^* - \mathbf{y}). \end{aligned}$$

Thus $\|\mathbf{b} - \mathbf{y}\|_2^2 = \|\mathbf{b}\|_2^2 - \|\mathbf{y}^*\|_2^2 + \|\mathbf{y}^* - \mathbf{y}\|_2^2$, and obviously this last expression is minimized if and only if $\mathbf{y} = \mathbf{y}^*$. ∎

The geometric rationale underlying this theorem is as follows: suppose that Y is a plane in R^3 (in our particular case it is either a plane or a line through the origin when $m = 3$). If $\mathbf{b}$ is not in this plane, then the closest point on the plane to $\mathbf{b}$ is $\mathbf{y}^*$ where $\mathbf{y}^*$ is such that the projection vector, $(\mathbf{b} - \mathbf{y}^*)$, is perpendicular to every $\mathbf{y}$ in the plane. Since $(\mathbf{b} - \mathbf{y}^*)^T\mathbf{y}$ is the usual "dot product" of $(\mathbf{b} - \mathbf{y}^*)$ and $\mathbf{y}$, $(\mathbf{b} - \mathbf{y}^*)^T\mathbf{y} = 0$ means that $(\mathbf{b} - \mathbf{y}^*)$ and $\mathbf{y}$ are perpendicular. So this theorem is a natural analytical extension of this geometric concept.

Later, when we investigate this problem in a more general setting, we will have the necessary mathematical tools to prove that $\mathbf{y}^*$, as given in the above theorem, always exists and is unique. For now we will merely assume that this statement is always true. Under this assumption, there is always at least one set of scalars, $\{x_1^*, x_2^*, \ldots, x_n^*\}$, such that $\mathbf{y}^* = x_1^*\mathbf{A}_1 + x_2^*\mathbf{A}_2 + \cdots + x_n^*\mathbf{A}_n \equiv$

$A\mathbf{x}^*$ where $\mathbf{x}^* \equiv (x_1^*, x_2^*, \ldots, x_n^*)^T$. Since every $\mathbf{y} \in Y$ can be written as $\mathbf{y} = x_1\mathbf{A}_1 + x_2\mathbf{A}_2 + \cdots + x_n\mathbf{A}_n$, then $(\mathbf{b} - \mathbf{y}^*)^T\mathbf{y} = 0$ for all $\mathbf{y} \in Y$ is equivalent to saying that $(\mathbf{b} - \mathbf{y}^*)^T\mathbf{A}_i = \mathbf{A}_i^T(\mathbf{b} - \mathbf{y}^*) = 0$ for $1 \le i \le n$, or $\mathbf{A}_i^T\mathbf{y}^* = \mathbf{A}_i^T\mathbf{b}$, $1 \le i \le n$. This statement gives a set of n simultaneous equations, which written in vector form becomes

$$\begin{bmatrix} \mathbf{A}_1^T(A\mathbf{x}^*) \\ \mathbf{A}_2^T(A\mathbf{x}^*) \\ \vdots \\ \mathbf{A}_n^T(A\mathbf{x}^*) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1^T\mathbf{b} \\ \mathbf{A}_2^T\mathbf{b} \\ \vdots \\ \mathbf{A}_n^T\mathbf{b} \end{bmatrix}. \tag{2.71}$$

since $\mathbf{y}^* = A\mathbf{x}^*$. It is easily verified by the basic rules of matrix multiplication that if $\mathbf{z}$ is any vector in R^m, then since the rows of A^T are the columns of A,

$$A^T\mathbf{z} = \begin{bmatrix} \mathbf{A}_1^T\mathbf{z} \\ \mathbf{A}_2^T\mathbf{z} \\ \vdots \\ \mathbf{A}_n^T\mathbf{z} \end{bmatrix}. \tag{2.72}$$

Therefore by (2.72), (2.71) reduces to the simple expression

$$A^TA\mathbf{x}^* = A^T\mathbf{b}. \tag{2.73}$$

This is an $(n \times n)$ linear system commonly known as the *normal equations*, and a *least-squares solution*, $\mathbf{x}^*$, may be obtained by Gauss elimination or Gauss-Seidel. (We note here that if $A\mathbf{x} = \mathbf{b}$ has an exact solution, $\mathbf{x}^*$, that is, $\mathbf{b} \in Y$, then $\mathbf{x}^*$ is still a solution of $A^TA\mathbf{x}^* = A^T\mathbf{b}$.)

A familiar problem that arises in this setting is the following: assume that a function $f(x)$ is known at the m distinct points $\{x_0, x_1, \ldots, x_{m-1}\}$, and we wish to find an $(n-1)$st degree polynomial $p(x) = a_0x^{n-1} + a_1x^{n-2} + \cdots + a_{n-2}x + a_{n-1}$ that minimizes the expression $\sum_{i=0}^{m-1}(f(x_i) - p(x_i))^2$ where $m > n$ [$p(x)$ is the $(n-1)$st degree best discrete least-squares polynomial fit for $f(x)$]. The equations, $f(x_i) = p(x_i) = a_0x_i^{n-1} + a_1x_i^{n-2} + \cdots + a_{n-1}$ and $0 \le i \le m-1$, yield the $(m \times n)$ linear system

$$\begin{bmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & & & \\ 1 & x_{m-1} & \cdots & x_{m-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{m-1}) \end{bmatrix}, \tag{2.74}$$

and the least-squares solution for $\{a_i\}_{i=0}^{n-1}$ is obtained via (2.73).

EXAMPLE 2.19. Suppose we wish to draw the straight line in the plane that comes closest to "fitting" the points (0, 1), (1, 2.1), (2, 2.9), (3, 3.2). As we indicated above, if we set $f(0) = 1, f(1) = 2.1, f(2) = 2.9$ and $f(3) = 3.2$, then we want a_0 and a_1 to minimize $\sum_{i=0}^{3}[f(x_i) - p(x_i)]^2$ where $p(x) = a_0x + a_1$ and $x_i = i$, $i = 0, 1, 2, 3$. Hence we are led to

the overdetermined system

$$\begin{aligned} a_1 &= 1. \\ a_0 + a_1 &= 2.1 \\ 2a_0 + a_1 &= 2.9 \\ 3a_0 + a_1 &= 3.2 \end{aligned} \quad \text{or} \quad \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2.1 \\ 2.9 \\ 3.2 \end{bmatrix}.$$

Then

$$A^{\mathrm{T}}A\mathbf{x}^* = A^{\mathrm{T}}\mathbf{b} \quad \text{is} \quad \begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 17.5 \\ 9.2 \end{bmatrix}.$$

Thus we find $a_1 = 1.19$ and $a_0 = 0.74$. (See Fig. 2.3.)

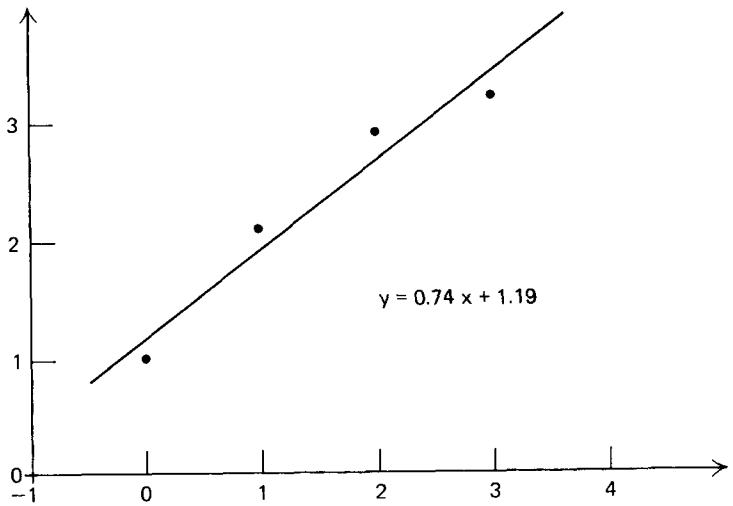

Figure 2.3 Least-squares straight-line fit (Example 2.19).

We have noted previously that matrices of the form $A^{\mathrm{T}}A$ can be ill-conditioned. For this reason, the particular problem of discrete least-squares curve fitting is usually attacked by a different method, which we shall investigate in a later section on polynomial approximations for functions.

PROBLEMS, SECTION 2.5

1. Use the methods of this section to find the best least-squares solution of the system

$$\begin{aligned} x_1 + x_2 &= 1 \\ 2x_1 + x_2 &= 0 \\ x_1 - x_2 &= 0. \end{aligned}$$

2. If A denotes the coefficient matrix of the (3×2) system of Problem 1, show that $A^{\mathrm{T}}A$ is symmetric and positive-definite as in Problem 12, Section 2.4.

3. Consider the (3 × 3) system

$$\begin{aligned} x_1 + 3x_2 + 7x_3 &= 1 \\ 2x_1 + x_2 - x_3 &= 1 \\ x_1 + 2x_2 + 4x_3 &= 1. \end{aligned}$$

Show the system has no solution (so that coefficient matrix A must be singular). Next, find the best least-squares solution by the techniques of this section. Note that even though A^TA is singular, the equation corresponding to (2.73) is solvable.

4. For the matrix A in Problem 3, find a nonzero vector $\mathbf{x}$ such that $\mathbf{x}^T(A^TA)\mathbf{x} = 0$, and thus conclude that A^TA cannot be positive-definite.

5. If $y = a_0x + a_1$, choose a_0 and a_1 such that this straight line is the best least-squares fit to these (x, y) data points:

   a) (1, 1), (4, 2), (8, 4), (11, 5)

   b) (−1, 0), (0, 1), (1, 2), (2, 4)

   c) (−2, 2), (−1, 1), (1, 0), (2, −1).

6. If $y = a_0x^2 + a_1x + a_2$, choose a_0, a_1, and a_2 such that this quadratic is the best least-squares fit to these (x, y) data points:

   a) (−2, 2), (−1, 1), (1, 1), (2, 2)

   b) (0, 0), (1, 0), (2, 1), (3, 2).

*7. Find c_0, c_1, and c_2 so that the expression $y = c_0 + c_1 \cos x + c_2 \sin x$ is a best least-squares fit to the (x, y) data points $(-\pi/2, 1)$, $(0, 0)$, $(\pi/2, 1/2)$, $(\pi, 1)$.

8. An $(n \times n)$ matrix B is called *positive semidefinite* if $\mathbf{x}^TB\mathbf{x} \geq 0$ for all $(n \times 1)$ vectors $\mathbf{x}$. Show that any matrix B of the form $B = A^TA$ is positive semidefinite.

9. Write a computer program to find a fourth-degree polynomial approximation to $f(x) = \cos(x)$, for $0 \leq x \leq \pi$, as follows.

   a) Let $x_j = j\pi/20$; $j = 0, 1, \ldots, 20$.

   b) Set up Eq. (2.74) with $m = 21$ and $n = 5$, to find a best fourth-degree least-squares approximation to $\cos(x)$ on $x_0, x_1, \ldots, x_{20}$.

   c) Having (2.74), form the system (2.73) and use a Gauss elimination routine to solve for $\mathbf{x}^*$.

   d) Let $p(x)$ denote the resulting best fourth-degree polynomial approximation. Print a table, listing $p(y_i), f(y_i), |p(y_i) - f(y_i)|/|f(y_i)|$ for $y_i = (0.01)j\pi; j = 0, 1, 2, \ldots, 100; j \neq 50$.

*2.6 THE CAUCHY-SCHWARZ INEQUALITY.†

Let $\mathbf{x}$ and $\mathbf{y}$ be vectors in R^n, with $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$. We recall the familiar *scalar* or *dot* product, given by

$$\mathbf{x}^T\mathbf{y} = x_1y_1 + x_2y_2 + \cdots + x_ny_n.$$

†This brief section is included for the sake of completeness and may be omitted without loss of continuity. The ambitious reader will find that the ideas contained in this section have wide-ranging application.