Chapter 4

More on Linear Systems of Equations

4.1 Introduction and Least Squares Problems

In the previous chapter we saw that two-point boundary value problems led to solving systems of linear equations. These linear equations were of a very special form: the coefficient matrix had only three non-zero diagonals. Such matrices are a special case of *banded* matrices, to be discussed in Section 4.2. Banded matrices in turn are special cases of *full* or *dense* matrices in which all elements of the matrix are non-zero. The primary purpose of this chapter is to discuss solution techniques for banded or full matrices.

Most approaches to the solution of ordinary or partial differential equations give rise to linear systems in which the coefficient matrix is banded or very *sparse*; that is, it has few non-zero elements. How such matrices arise for partial differential equations will be discussed in Chapter 9. In the remainder of this section we will consider another important class of problems that generally lead to full coefficient matrices. These *least square problems* are not usually related to differential equations, although they may be. For example, given data on the number of predators and prey at different times, can we estimate the coefficients α, β, γ, and δ in the predator-prey equations (2.1.3)? Least squares techniques would be one approach to making such estimates.

Least Squares Polynomials

Recall from Section 2.3 that if $x_0, x_1, \ldots, x_n$ are $n+1$ distinct points and f is a given function, then there is a unique polynomial p of degree n such that

$$p(x_i) = f(x_i), \qquad i = 0, \ldots, n.$$

Now suppose that f itself is a polynomial of degree n and that we wish to determine its coefficients. Then, by the preceding interpolation result, it suffices to know f at $n+1$ distinct points, provided that the determination of the values $f(x_i)$ can be made exactly. In many situations, however, the values of f can be found only by measurements and may be in error. In this case it is common to take many more than $n+1$ measurements in the hope that these measurement errors will "average out." The way in which these errors average out is determined by the method used to combine the measurements to obtain the coefficients of f. For both computational and statistical reasons, the method of choice is often that of least squares, and this method also enjoys an elegant mathematical simplicity.

We now assume that we have m points $x_1, \ldots, x_m$ where $m \geq n+1$ and at least $n+1$ of the points are distinct. Let $f_1, \ldots, f_m$ be approximate values of a function f, not necessarily a polynomial, at the points $x_1, \ldots x_m$. Then we wish to find a polynomial $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ such that

$$\sum_{i=1}^{m} w_i [f_i - p(x_i)]^2 \tag{4.1.1}$$

is a minimum over all polynomials of degree n. That is, we wish to find $a_0, a_1, \ldots, a_n$ so that the weighted sum of the squares of the "errors" $f_i - p(x_i)$ is minimized. In (4.1.1) the w_i are given positive constants, called *weights*, that may be used to assign greater or lesser emphasis to the terms of (4.1.1). For example, if the f_i are measurements and we have great confidence in, say, $f_1, \ldots, f_{10}$, but rather little confidence in the rest, we might set $w_1 = w_2 = \cdots = w_{10} = 5$ and $w_{11} = \cdots = w_m = 1$.

The simplest case of a least squares problem is when $n = 0$, so that p is just a constant. Suppose, for example, that we have m measurements $l_1, \ldots, l_m$ of the length of some object, where the measurements are obtained from m different rulers. Here the points $x_1, \ldots, x_m$ are all identical and, indeed, do not enter explicitly. If we invoke the least-squares principle, then we wish to minimize

$$g(l) = \sum_{i=1}^{m} w_i (l_i - l)^2.$$

From the calculus we know that g takes on a (relative) minimum at a point $\hat{l}$ that satisfies $g'(\hat{l}) = 0$ and $g''(\hat{l}) \geq 0$. Since

$$g'(l) = -2\sum_{i=1}^{m} w_i (l_i - l), \qquad g''(l) = 2\sum_{i=1}^{m} w_i,$$

it follows that

$$\hat{l} = \frac{1}{s}\sum_{i=1}^{m} w_i l_i, \qquad s = \sum_{i=1}^{m} w_i,$$

and because this is the only solution of $g'(l) = 0$ it must be the unique point that minimizes g. Thus if the weights w_i are all 1, so that $s = m$, the least-squares approximation to l is just the average of the measurements $l_1, \ldots, l_m$.

The next-simplest situation is if we use a linear polynomial $p(x) = a_0 + a_1 x$. Problems of this type arise very frequently under the assumption that the data are obeying some linear relationship. In this case the function (4.1.1) is

$$g(a_0, a_1) = \sum_{i=1}^{m} w_i (f_i - a_0 - a_1 x_i)^2, \tag{4.1.2}$$

which we wish to minimize over the coefficients a_0 and a_1. Again from the calculus, we know that a necessary condition for g to be minimized is that the partial derivatives of g at the minimizer must vanish:

$$\begin{aligned} \frac{\partial g}{\partial a_0} &= -2\sum_{i=1}^{m} w_i (f_i - a_0 - a_1 x_i) = 0, \\ \frac{\partial g}{\partial a_1} &= -2\sum_{i=1}^{m} w_i x_i (f_i - a_0 - a_1 x_i) = 0. \end{aligned}$$

Collecting coefficients of a_0 and a_1 gives the system of two linear equations

$$\begin{aligned} \left(\sum_{i=1}^{m} w_i\right) a_0 + \left(\sum_{i=1}^{m} w_i x_i\right) a_1 &= \sum_{i=1}^{m} w_i f_i \\ \left(\sum_{i=1}^{m} w_i x_i\right) a_0 + \left(\sum_{i=1}^{m} w_i x_i^2\right) a_1 &= \sum_{i=1}^{m} w_i x_i f_i \end{aligned} \tag{4.1.3}$$

for the unknowns a_0 and a_1.

For polynomials of degree n, the function (4.1.1) that we wish to minimize is

$$g(a_0, a_1, \ldots, a_n) = \sum_{i=1}^{m} w_i (a_0 + a_1 x_i + \cdots + a_n x_i^n - f_i)^2. \tag{4.1.4}$$

Proceeding as in the $n = 2$ case, we know from the calculus that a necessary condition for g to be minimized is that

$$\frac{\partial g}{\partial a_j}(a_0, a_1, \ldots, a_n) = 0, \qquad j = 0, 1 \ldots, n.$$

Writing these partial derivatives out explicitly gives the conditions

$$\sum_{i=1}^{m} w_i x_i^j (a_0 + a_1 x_i + \cdots + a_n x_i^n - f_i) = 0, \qquad j = 0, 1, \ldots, n,$$

which is a system of $n+1$ linear equations in the $n+1$ unknowns $a_0, a_1, \ldots, a_n$; these equations are known as the *normal equations*. Collecting the coefficients of the a_j and rewriting the equations in matrix-vector form gives the system

$$\begin{bmatrix} s_0 & s_1 & s_2 & \cdots & s_n \\ s_1 & s_2 & & & \\ s_2 & & \ddots & & \vdots \\ \vdots & & & & \\ s_n & & \cdots & & s_{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ \\ a_n \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ \\ c_n \end{bmatrix}, \tag{4.1.5}$$

where

$$s_j = \sum_{i=1}^{m} w_i x_i^j, \qquad c_j = \sum_{i=1}^{m} w_i x_i^j f_i. \tag{4.1.6}$$

Equation (4.1.3) is the special case of (4.1.5) for $n = 1$. Note that the matrix in (4.1.5) is determined by only $2n + 1$ quantities $s_0, \ldots, s_{2n}$, and the "cross-diagonals" of the matrix are constant. Such a matrix is called a *Hankel matrix* and has many interesting properties.

The system (4.1.5) can also be written in the form

$$E^T W E \mathbf{a} = E^T W \mathbf{f}, \tag{4.1.7}$$

where W is a diagonal matrix containing the weights and

$$E = \begin{bmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & & & \vdots \\ 1 & x_m & \cdots & x_m^n \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}. \tag{4.1.8}$$

The matrix E is $m \times (n + 1)$ and is of Vandermonde type; in particular, if $m = n + 1$, it is precisely the Vandermonde matrix of (2.3.11), which we showed was non-singular. We now extend that argument to show that the matrix of (4.1.7) is non-singular provided that at least $n + 1$ of the points x_i are distinct.

A symmetric matrix A is *positive definite* if

$$\mathbf{x}^T A \mathbf{x} > 0 \quad \text{for all} \quad \mathbf{x} \neq 0. \tag{4.1.9}$$

A positive definite matrix is nonsingular, since otherwise there would be an $\mathbf{x} \neq 0$ so that $A\mathbf{x} = 0$ and thus (4.1.9) would be violated.

We now show that the matrix E^TWE of (4.1.7) is symmetric positive definite. Clearly it is symmetric since it is the matrix of (4.1.5). Let $\mathbf{y}$ be any m-vector and consider

$$\mathbf{y}^TW\mathbf{y} = \sum_{i=1}^{m} w_i y_i^2. \tag{4.1.10}$$

Since the w_i are assumed positive, $\mathbf{y}^TW\mathbf{y} \geq 0$ and is equal to zero if and only if $\mathbf{y} = 0$. Thus, if we set $\mathbf{y} = E\mathbf{a}$, we conclude that

$$\mathbf{a}^TE^TWE\mathbf{a} > 0 \quad \text{for all} \quad \mathbf{a} \neq 0$$

provided that $E\mathbf{a} \neq 0$. But if $E\mathbf{a} = 0$, this implies that

$$a_0 + a_1x_i + \cdots + a_nx_i^n = 0, \qquad i = 1, \ldots, m.$$

Recalling that we have assumed that at least $n+1$ of the x_i are distinct, the nth degree polynomial $a_0 + a_1x + \cdots + a_nx^n$ would have at least $n+1$ distinct roots. This contradiction proves that E^TWE is positive definite. Thus the system (4.1.7) has a unique solution, and the resulting polynomial with coefficients a_i is the unique polynomial of degree n which minimizes (4.1.1).

General Least Squares Problems

We next consider more general least-squares problems in which the approximating function is not necessarily a polynomial but is a linear combination

$$\phi(x) = \sum_{i=0}^{n} a_i\phi_i(x) \tag{4.1.11}$$

of given functions $\phi_0, \phi_1, \ldots, \phi_n$. If $\phi_j(x) = x^j$, $j = 0, \ldots, n$, then ϕ is the polynomial considered previously. Other common choices for the "basis functions" ϕ_j are

$$\phi_j(x) = \sin j\pi x, \qquad j = 0, 1 \ldots, n,$$

and

$$\phi_j(x) = e^{\alpha_j x}, \qquad j = 0, 1, \ldots, n,$$

where the α_j are given real numbers.

The *general linear least squares problem* is to find $a_0, \ldots, a_n$ such that

$$g(a_0, a_1, \cdots, a_n) = \sum_{i=1}^{m} w_i[\phi(x_i) - f_i]^2 \tag{4.1.12}$$

is minimized, where ϕ is given by (4.1.11). We can proceed exactly as before in obtaining the normal equations for (4.1.12). The first partial derivatives of g are

$$\frac{\partial g}{\partial a_j} = 2\sum_{i=1}^{m} w_i\phi_j(x_i)[a_0\phi_0(x_i) + a_1\phi_1(x_1) + \cdots + a_n\phi_n(x_i) - f_i].$$

Setting these equal to zero, collecting coefficients of the a_i, and writing the resulting linear system in matrix-vector form gives the system

$$\begin{bmatrix} s_{00} & s_{10} & \cdots & s_{n0} \\ s_{01} & s_{11} & & \\ \vdots & & \ddots & \vdots \\ s_{0n} & \cdots & & s_{nn} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}, \tag{4.1.13}$$

where

$$s_{ij} = \sum_{k=1}^{m} w_k\phi_i(x_k)\phi_j(x_k), \qquad c_j = \sum_{k=1}^{m} w_k\phi_j(x_k)f_k.$$

Note that in this case the coefficient matrix of (4.1.13) is not necessarily a Hankel matrix. As before, we can write (4.1.13) in the form (4.1.7) where now

$$E = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_n(x_2) \\ \vdots & & & \vdots \\ \phi_0(x_m) & & \cdots & \phi_n(x_m) \end{bmatrix}, \mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}.$$

Clearly, E^TWE is again symmetric, but in order to conclude that it is positive definite, suitable conditions must be imposed on the functions $\phi_0, \ldots, \phi_n$ as well as $x_1, \ldots, x_m$.

Orthogonal Polynomials

The normal equations are very useful for theoretical purposes or for computation purposes when n is small. But they have a tendency to become very ill-conditioned (see Section 4.4) for n at all large, even $n \geq 5$. We now describe an alternative approach to computing the least squares polynomial by means of orthogonal polynomials. In what follows we assume that $w_i = 1$, although the inclusion of weights presents no problem (Exercise 4.1.5).

Let $q_0, q_1, \ldots, q_n$ be polynomials of degree $0, 1, \ldots, n$, respectively. Then we will say that the q_i are mutually *orthogonal* with respect to the points $x_1, \ldots, x_m$ if

$$\sum_{i=1}^{m} q_k(x_i)q_j(x_i) = 0, \qquad k, j = 0, 1, \ldots, n, \quad k \neq j. \tag{4.1.14}$$

We shall return shortly to the question of how one obtains such a set of orthogonal polynomials. For the moment assume that we have them and take $\phi_i = q_i$, $i = 0, 1, \ldots, n$, in the normal equations (4.1.13), in which the weights w_i are all equal to 1. Then, because of (4.1.14), all elements of the coefficient matrix of (4.1.13) off the main diagonal are zero, and the system of equations reduces to

$$\sum_{i=1}^{m} [q_k(x_i)]^2 a_k = \sum_{i=1}^{m} q_k(x_i) f_i, \qquad k = 0, 1, \ldots, n.$$

Thus

$$a_k = \frac{1}{\gamma_k} \sum_{i=1}^{m} q_k(x_i) f_i, \qquad k = 0, 1, \ldots, n, \tag{4.1.15}$$

where

$$\gamma_k = \sum_{i=1}^{m} [q_k(x_i)]^2. \tag{4.1.16}$$

Therefore the least squares polynomial is

$$q(x) = \sum_{k=0}^{n} a_k q_k(x). \tag{4.1.17}$$

An obvious question is whether the polynomial q of (4.1.17) is the same as the polynomial obtained from the normal equations (4.1.5) (with the $w_i = 1$). The answer is yes, under our standard assumption that at least $n + 1$ of the points x_i are distinct. This follows from the fact that – as shown earlier – there is a unique polynomial of degree n or less that minimizes (4.1.1). Therefore to show that the polynomial q of (4.1.17) is this same minimizing polynomial, it suffices to show that any polynomial of degree n can be written as a linear combination of the q_i; that is, given a polynomial

$$\hat{p}(x) = b_0 + b_1 x + \cdots + b_n x^n, \tag{4.1.18}$$

we can find coefficients $c_0, c_1, \ldots, c_n$ so that

$$\hat{p}(x) = c_0 q_0(x) + c_1 q_1(x) + \cdots + c_n q_n(x). \tag{4.1.19}$$

This can be done as follows. Let

$$q_i(x) = d_{i,0} + d_{i,1} x + \cdots + d_{i,i} x^i, \qquad i = 0, 1, \ldots, n,$$

where $d_{i,i} \neq 0$. Equating the right-hand sides of (4.1.18) and (4.1.19) gives

$$\begin{aligned} &b_0 + b_1 x + \cdots + b_n x^n \\ &\qquad = c_0 d_{0,0} + c_1(d_{1,0} + d_{1,1} x) + \cdots + c_n(d_{n,0} + \cdots + d_{n,n} x^n), \end{aligned}$$

and equating coefficients of powers of x then gives

$$\begin{aligned} b_n &= c_n d_{n,n} \\ b_{n-1} &= c_n d_{n,n-1} + c_{n-1} d_{n-1,n-1} \\ &\vdots \\ b_0 &= c_n d_{n,0} + c_{n-1} d_{n-1,0} + \cdots + c_0 d_{0,0}. \end{aligned} \tag{4.1.20}$$

These are necessary and sufficient conditions that the polynomials of (4.1.18) and (4.1.19) be identical. Given $b_0, b_1, \ldots, b_n$, (4.1.20) is a triangular linear system of equations for the c_i and is solvable since $d_{i,i} \neq 0$, $i = 0, 1, \ldots, n$. Hence the polynomial of (4.1.17) is just another representation of the unique least-squares polynomial obtained by solving the normal equations (4.1.5).

The use of orthogonal polynomials reduces the normal equations to a diagonal system of equations which is trivial to solve. However, the burden is now shifted to the computation of the q_i. There are several possible ways to construct orthogonal polynomials; we will describe one which is particularly suitable for computation.

Let

$$q_0(x) \equiv 1, \qquad q_1(x) \equiv x - \alpha_1, \tag{4.1.21}$$

where α_1 is to be determined so that q_0 and q_1 are orthogonal with respect to the x_i. Thus we must have

$$0 = \sum_{i=1}^{m} q_0(x_i) q_1(x_i) = \sum_{i=1}^{m} (x_i - \alpha_1) = \sum_{i=1}^{m} x_i - m\alpha_1,$$

so that

$$\alpha_1 = \frac{1}{m} \sum_{i=1}^{m} x_i. \tag{4.1.22}$$

Now let

$$q_2(x) = x q_1(x) - \alpha_2 q_1(x) - \beta_1,$$

where α_2 and β_1 are to be determined so that q_2 is orthogonal to both q_0 and q_1; that is,

$$\begin{aligned} \sum_{i=1}^{m} [x_i q_1(x_i) - \alpha_2 q_1(x_i) - \beta_1] &= 0 \\ \sum_{i=1}^{m} [x_i q_1(x_i) - \alpha_2 q_1(x_i) - \beta_1] q_1(x_i) &= 0. \end{aligned}$$

Noting that $\sum q_1(x_i) = 0$, these relations reduce to

$$\sum_{i=1}^{m} x_i q_1(x_i) - m\beta_1 = 0, \qquad \sum_{i=1}^{m} x_i [q_1(x_i)]^2 - \alpha_2 \gamma_1 = 0,$$

where γ_1 is given by (4.1.16). Thus

$$\beta_1 = \frac{1}{m}\sum_{i=1}^{m} x_i q_1(x_i), \qquad \alpha_2 = \frac{1}{\gamma_1}\sum_{i=1}^{m} x_i [q_1(x_i)]^2.$$

The computation for the remaining q_i proceeds in an analogous fashion. Assume that we have determined $q_0, q_1, \ldots, q_j$, and define q_{j+1} by the *three-term recurrence relation*

$$q_{j+1}(x) = xq_j(x) - \alpha_{j+1}q_j(x) - \beta_j q_{j-1}(x), \tag{4.1.23}$$

where α_{j+1} and β_j are to be determined from the orthogonality requirements

$$\sum_{i=1}^{m} q_{j+1}(x_i)q_j(x_i) = 0, \qquad \sum_{i=1}^{m} q_{j+1}(x_i)q_{j-1}(x_i) = 0. \tag{4.1.24}$$

If these two relations are satisfied, then q_{j+1} must also be orthogonal to all the previous q_k, $k < j-1$, since by (4.1.23)

$$\begin{aligned}\sum_{i=1}^{m} q_{j+1}(x_i)q_k(x_i) &= \sum_{i=1}^{m} x_i q_j(x_i)q_k(x_i) - \alpha_{j+1}\sum_{i=1}^{m} q_j(x_i)q_k(x_i) \\ &\quad -\beta_j \sum_{j=1}^{m} q_{j-1}(x_i)q_k(x_i).\end{aligned} \tag{4.1.25}$$

The last two terms in (4.1.25) are zero by assumption, whereas $xq_k(x)$ is a polynomial of degree $k+1$ and can be expressed as a linear combination of $q_0, q_1, \ldots, q_{k+1}$. Hence the first term on the right-hand side of (4.1.25) is also zero.

Returning to the conditions (4.1.24) and inserting q_{j+1} from (4.1.23) leads to the expressions

$$\alpha_{j+1} = \frac{1}{\gamma_j}\sum_{i=1}^{m} x_i [q_j(x_i)]^2, \tag{4.1.26}$$

$$\beta_j = \frac{1}{\gamma_{j-1}}\sum_{i=1}^{m} x_i q_j(x_i)q_{j-1}(x_i),$$

for α_{j+1} and β_j, where the γ's are given by (4.1.16). The β_i, however, can be computed in a better way if we substitute for $x_i q_{j-1}(x_i)$ using (4.1.23), and then note that

$$\sum_{i-1}^{m} q_j(x_i)[q_j(x_i) + \alpha_j q_{j-1}(x_i) + \beta_{j-1}q_{j-2}(x_i)] = \sum_{i=1}^{m}[q_j(x_i)]^2 = \gamma_j$$

by the orthogonality of the q's. Thus

$$\beta_j = \frac{\gamma_j}{\gamma_{j-1}}. \tag{4.1.27}$$

Note that the denominator in (4.1.27) can vanish only if $q_{j-1}(x_i) = 0$, $i = 1, \ldots, m$. But since at least $n + 1$ of the x_i are assumed to be distinct, this would imply that q_{j-1} is identically zero, which contradicts the definition of q_{j-1}. Hence $\gamma_{j-1} \neq 0$.

We can summarize the orthogonal polynomial algorithm as follows:

1. Set $q_0(x) \equiv 1$, $q_1(x) = x - \frac{1}{m} \sum_{i=1}^{m} x_i$.

2. For $j = 1, \ldots, n-1$, define q_{j+1} by (4.1.23), where α_{j+1} is given by (4.1.26), and β_j by (4.1.27).

3. Compute the coefficients $a_0, a_1, \ldots, a_n$ of the least squares polynomial $a_0q_0(x) + a_1q_1(x) + \cdots + a_nq_n(x)$ by (4.1.15).

As mentioned previously, this approach is preferred numerically because it avoids the necessity of solving the (possibly) ill-conditioned system (4.1.5). Another advantage is that we are able to build up the least-squares polynomial degree by degree. For example, if we do not know what degree polynomial we wish to use, we might start with a first-degree polynomial, then a second-degree, and so on, until we obtain a fit that we believe is suitable. With the orthogonal polynomial algorithm the coefficients a_i are independent of n, and as soon as we compute q_j we can compute a_j, and hence the least squares polynomial of degree j.

A Numerical Example

We now give a simple example, using the data

$$\begin{array}{lllll} x_1 = 0 & x_2 = \frac{1}{4} & x_3 = \frac{1}{2} & x_4 = \frac{3}{4} & x_5 = 1 \\ f_1 = 1 & f_2 = 2 & f_3 = 1 & f_4 = 0 & f_5 = 1. \end{array}$$

Since there are five points x_i, these data will uniquely determine a fourth-degree interpolating polynomial. We now compute the linear and quadratic least-squares polynomials by both the normal equations and the orthogonal polynomial approaches.

For the normal equations for the linear polynomial, we will need the following quantities:

$$\sum_{i=1}^{5} x_i = \tfrac{5}{2}, \qquad \sum_{i=1}^{5} x_i^2 = \tfrac{15}{8}, \qquad \sum_{i=1}^{5} f_i = 5, \qquad \sum_{i=1}^{5} x_i f_i = 2. \tag{4.1.28}$$

Then the coefficients a_0 and a_1 are the solutions of the system (4.1.3) (with the $w_i = 1$):

$$a_0 = \frac{7}{5}, \qquad a_1 = \frac{-4}{5}.$$

Thus the linear least squares polynomial is

$$p_1(x) = \tfrac{7}{5} - \tfrac{4}{5}x. \tag{4.1.29}$$

If we compute the same polynomial by orthogonal polynomials, the polynomial is given in the form

$$a_0q_0(x) + a_1q_1(x) = a_0 + a_1(x - \alpha_1), \tag{4.1.30}$$

where a_0 and a_1 are given by (4.1.15), and α_1 by (4.1.22):

$$a_0 = 1, \qquad a_1 = \frac{-4}{5}, \qquad \alpha_1 = \tfrac{1}{2}.$$

Therefore the polynomial (4.1.30) is $1 - \frac{4}{5}(x - \frac{1}{2})$, which is, as expected, the same as (4.1.29).

To compute the least squares quadratic polynomial by the normal equations, we need to solve the system (4.1.5) for $n = 2$, which for our data is

$$\begin{bmatrix} 640 & 320 & 240 \\ 320 & 240 & 200 \\ 240 & 200 & 177 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 640 \\ 256 \\ 176 \end{bmatrix}.$$

The solution of this system is

$$a_0 = \tfrac{7}{5}, \qquad a_1 = \tfrac{-4}{5}, \qquad a_2 = 0. \tag{4.1.31}$$

Thus the best least squares polynomial approximation turns out to be the linear least-squares polynomial; that is, no improvement of the linear approximation can be made by adding a quadratic term. That (4.1.31) is correct is verified by computing the least squares quadratic by orthogonal polynomials. The orthogonal polynomial representation will be

$$a_0q_0(x) + a_1q_1(x) + a_2q_2(x) = \tfrac{7}{5} - \tfrac{4}{5}x + a_2[x(x - \tfrac{1}{2}) - \alpha_2(x - \tfrac{1}{2}) - \beta_1],$$

where α_2 and β_1 are computed from (4.1.26) and (4.1.27) as

$$\alpha_2 = \frac{\sum x_i(x_i - \frac{1}{2})^2}{\sum (x_i - \frac{1}{2})^2} = \tfrac{1}{2}, \qquad \beta_1 = \tfrac{1}{5}\sum x_i(x_i - \tfrac{1}{2}) = \tfrac{1}{8}.$$

Thus $q_2(x) = x^2 - x - \frac{1}{8}$, and from (4.1.15) we find that $a_2 = 0$.

The orthogonal polynomial approach can be extended to other basis functions that are orthogonal, such as trigonometric functions. However, for most basis functions the system (4.1.13) must be solved and, in general, all elements of this coefficient matrix will be non-zero. Hence we must solve a "full" linear system, which is the topic of the next two sections.

Supplementary Discussion and References: 4.1

An alternative approach to solving the least squares polynomial problem is to deal directly with the system of linear equations $E\mathbf{a} = \mathbf{f}$, where E and $\mathbf{f}$ are given in (4.1.8). This is an $m \times (n+1)$ system, where m is usually greater that $n+1$; hence the matrix E is not square. Techniques for dealing with this type of system are given in Section 4.5. For further discussions of least squares problems see Golub and Van Loan [1989] and Lawson and Hanson [1974].

EXERCISES 4.1

4.1.1. Assume that f is a given function for which the following values are known: $f(1) = 2$, $f(2) = 3$, $f(3) = 5$, $f(4) = 3$. Find the constant, linear, and quadratic least-squares polynomials by both the normal-equation and orthogonal polynomial approaches.

4.1.2. Write a computer program to obtain the least-squares polynomial of degree n using $m \geq n+1$ data points by the orthogonal-polynomial approach. Test your program on the polynomials of Exercise 4.1.1.

4.1.3. Let $y_1 \dots y_m$ be a set of data that we wish to approximate by a constant c. Determine c so that

a. $\displaystyle\sum_{i=1}^{m} |c - y_i| = \text{minimum}.$

b. $\displaystyle\max_{1\leq i\leq m} |c - y_i| = \text{minimum}.$

4.1.4. If $y_1, \dots, y_m$ are m observations, define the mean and variance by

$$\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i, \qquad v = \sum_{i=1}^{m}(y_i - \bar{y})^2.$$

Let $y_{m+1}, \dots, y_{m+n}$ be n additional observations with mean and variance y_a and v_a. Show how to combine v and v_a by $\alpha_1 v + \alpha_2 v_a$ to obtain the variance of the combined set of observations. Specialize this to $n = 1$ so as to obtain an updating formula for the variance each time a new observation is added.

4.1.5. Modify the orthogonal polynomial approach to handle the problem (4.1.1) with given weights w_i. Replace the relation (4.1.14) by $\sum w_i q_k(x_i) q_j(x_i)$ and then modify (4.1.15), (4.1.26), and (4.1.27) accordingly.

4.1.6. Let f be a twice differentiable function on $[0,8]$, and let $x_i = (i-9)$, $i = 1, \ldots, 17$, and $f_i = f(x_i)$.

a. Using orthogonal polynomials, find the polynomial of degree 5 which minimizes (4.1.1) with the weights given by $w_i = \gamma|i-9|$ where $0 < \gamma \leq 1$.

b. Show how $f'(0)$ and $f''(0)$ can be approximated by using the recursion relations for the orthogonal polynomials. In particular, show how to obtain coefficients a_i so that $f'(0)$ is approximated by $\sum_{i=1}^{17} a_i f_i$.

4.2 Gaussian Elimination

In the previous chapter we discussed how to solve a system of tridiagonal linear equations by the Gaussian elimination method. We now apply Gaussian elimination to a general linear system

$$A\mathbf{x} = \mathbf{b}, \tag{4.2.1}$$

where A is a given $n \times n$ matrix assumed to be nonsingular, $\mathbf{b}$ is a given column n vector, and $\mathbf{x}$ is the solution vector to be determined.

We write the system (4.2.1) as

$$\begin{array}{lcl} a_{11}x_1 & + \cdots + & a_{1n}x_n = b_1 \\ a_{21}x_1 & + \cdots + & a_{2n}x_n = b_2 \\ \vdots & & \vdots \\ a_{n1}x_1 & + \cdots + & a_{nn}x_n = b_n. \end{array} \tag{4.2.2}$$

As with tridiagonal systems we first subtract a_{21}/a_{11} times the first equation from the second equation to eliminate the coefficient of x_1 in the second equation. For tridiagonal systems this completed the first stage but in the general case we wish to eliminate the coefficient of x_1 in all the remaining equations. Thus we subtract a_{31}/a_{11} times the first equation from the third equation, a_{41}/a_{11} times the first equation from the fourth equation, and so on, until the coefficients of x_1 in the last $n-1$ equations have all been eliminated. This gives the reduced system of equations

$$\begin{array}{rl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = b_1 \\ a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n & = b_2^{(1)} \\ \vdots \qquad\qquad & \\ a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n & = b_n^{(1)}, \end{array} \tag{4.2.3}$$

where

$$a_{ij}^{(1)} = a_{ij} - a_{1j}\frac{a_{i1}}{a_{11}}, \qquad b_i^{(1)} = b_i - b_1\frac{a_{i1}}{a_{11}}, \quad i,j = 2, \ldots, n. \tag{4.2.4}$$

Precisely the same process is now applied to the last $n-1$ equations of the system (4.2.3) to eliminate the coefficients of x_2 in the last $n-2$ equations, and so on, until the entire system has been reduced to the *triangular form*

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & . & & a_{nn}^{(n-1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2^{(1)} \\ \vdots \\ b_n^{(n-1)} \end{bmatrix}. \tag{4.2.5}$$

The superscripts indicate the number of times the elements have, in general, been changed. This completes the *forward reduction* (or *forward elimination* or *triangular reduction*) phase of the Gaussian elimination algorithm. Note that we have tacitly assumed that a_{11} and the $a_{ii}^{(i-1)}$ are all non-zero since we divide by these elements. In the following section we will consider the important question of how to handle zero or small divisors.

The Gaussian elimination method is based on the fact (usually established in an introductory linear algebra course) that replacing any equation of the original system (4.2.2) by a linear combination of itself and another equation does not change the solution of (4.2.2). Thus the triangular system (4.2.5) has the same solution as the original system. The purpose of the forward reduction is to reduce the original system to one which is easy to solve; this is a common theme in much of scientific computing. The second part of the Gaussian elimination method then consists of the solution of (4.2.5) by *back substitution*, in which the equations are solved in reverse order:

$$\begin{aligned} x_n &= \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}} \\ x_{n-1} &= \frac{b_{n-1}^{(n-2)} - a_{n-1,n}^{(n-2)} x_n}{a_{n-1,n-1}^{(n-2)}} \\ &\vdots \\ x_1 &= \frac{b_1 - a_{12}x_2 - \cdots - a_{1n}x_n}{a_{11}}. \end{aligned} \tag{4.2.6}$$

The Gaussian elimination algorithm can be written in algorithmic form as follows:

Forward Reduction

```
For $k = 1, \ldots, n-1$
  For $i = k+1, \ldots, n$
    $l_{ik} = \frac{a_{ik}}{a_{kk}}$                    (4.2.7)
    For $j = k+1, \ldots, n$
      $a_{ij} = a_{ij} - l_{ik}a_{kj}$
    $b_i = b_i - l_{ik}b_k$
```

Back Substitution

```
For $k = n, n-1, \ldots, 1$                             (4.2.8)
  $x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{kj}x_j}{a_{kk}}$.
```

To translate the preceding algorithm into a computer code one should note that the $a_{ij}^{(k)}$ can be overwritten on the same storage spaces occupied by the original elements a_{ij}. If this is done, the original matrix will, of course, be destroyed during the process. Similarly, the new $b_i^{(k)}$ may be overwritten on the original storage spaces of the b_i. The multiplier l_{ik} can be written into the corresponding storage space for a_{ik}, which is no longer needed after l_{ik} is computed.

LU Factorization

Gaussian elimination is related to a factorization

$$A = LU \tag{4.2.9}$$

of the matrix A. Here U is the upper triangular matrix of (4.2.5) obtained in the forward reduction, and L is a *unit* lower triangular matrix (all main diagonal elements are 1) in which the subdiagonal element l_{ij} is the multiplier used for eliminating the jth variable from the ith equation. For example, if the original system is

$$\begin{bmatrix} 4 & -9 & 2 \\ 2 & -4 & 4 \\ -1 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}, \tag{4.2.10}$$

then the reduced system (4.2.5) is

$$\begin{bmatrix} 4 & -9 & 2 \\ 0 & 0.5 & 3 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2.5 \end{bmatrix}. \tag{4.2.11}$$

The multipliers used to obtain (4.2.11) from (4.2.10) are 0.5, −0.25, and −0.5. Thus

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.25 & -0.5 & 1 \end{bmatrix}, \tag{4.2.12}$$

and A is the product of (4.2.12) and the matrix U of (4.2.11). The calculations needed for the above assertions are left to Exercise 4.2.1.

More generally, it is easy to verify (Exercise 4.2.2) that the elimination step that produces (4.2.3) from (4.2.2) is equivalent to multiplying (4.2.2) by the matrix

$$L_1 = \begin{bmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{bmatrix}. \tag{4.2.13}$$

Continuing in this fashion, the reduced system (4.2.5) may be written as

$$\hat{L}A\mathbf{x} = \hat{L}\mathbf{b}, \qquad \hat{L} = L_{n-1} \cdots L_2 L_1, \tag{4.2.14}$$

where

$$L_i = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{i+1,i} & & \\ & & \vdots & \ddots & \\ & & -l_{n,i} & & 1 \end{bmatrix}. \tag{4.2.15}$$

Each of the matrices L_i has determinant equal to 1 and so is non-singular. Therefore the product $\hat{L}$ is non-singular. Moreover, $\hat{L}$ is unit lower triangular and, therefore, so is $\hat{L}^{-1}$. By construction, the coefficient matrix of (4.2.14) is U, and if we set $L = \hat{L}^{-1}$ we have

$$U = \hat{L}A = L^{-1}A,$$

which is equivalent to (4.2.9). The verification of the above statements is left to Exercise 4.2.3.

The factorization (4.2.8) is known as the *LU decomposition* (or *LU factorization*) of A. The right-hand side of (4.2.13) is $L^{-1}\mathbf{b}$, which is the solution of $L\mathbf{y} = \mathbf{b}$. Thus the Gaussian elimination algorithm for solving $A\mathbf{x} = \mathbf{b}$ is mathematically equivalent to the three-step process:

$$\begin{array}{lll} 1. & \text{Factor} & A = LU. \\ 2. & \text{Solve} & L\mathbf{y} = \mathbf{b}. \\ 3. & \text{Solve} & U\mathbf{x} = \mathbf{y}. \end{array} \tag{4.2.16}$$

This matrix-theoretic view of Gaussian elimination is very useful for theoretical purposes and also forms the basis for some computational variants of the elimination process. In particular, the classical Crout and Doolittle forms of LU decomposition are based on formulas for the elements of L and U obtained from equating LU and A. Another use of (4.2.16) is when there are many right hand sides; in this case, step 1. is done once and the factors saved so that steps 2. and 3. may be performed repeatedly. This will be discussed in more detail later.

Operation Counts

An important question is the efficiency of the Gaussian elimination algorithm, and we next estimate the number of arithmetic operations needed to compute the solution vector $\mathbf{x}$. The major part of the work is in updating elements of A and we first count the number of operations to carry out the arithmetic statement $a_{ij} = a_{ij} - l_{ik}a_{kj}$ in (4.2.7). There is one addition and one multiplication in this statement and we count the number of additions. In the j loop of (4.2.7) there are $n-k$ additions. The i loop repeats these $n-k$ times so that the total over the k loop is

$$\sum_{k=1}^{n-1}(n-k)^2 = \sum_{k=1}^{n-1} k^2.$$

Thus using the summation formula in Exercise 4.2.4 we obtain

$$\text{Number of additions} = \sum_{k=1}^{n-1} k^2 = \frac{(n-1)(n)(2n-1)}{6} \doteq \frac{n^3}{3}, \tag{4.2.17}$$

and the same count holds for the number of multiplications. We also need the number of divisions to compute the l_{ik}, and the number of operations to modify the right hand side **b** and to do the back substitution. All of these involve no more than order n^2 operations (see Exercise 4.2.5). Hence for sufficiently large n the work involved in the triangular factorization makes the dominant contribution to the computing time and is proportional to n^3. Note that this implies that if n is doubled, the amount of work increases by approximately a factor of 8.

To obtain an understanding of the amount of time that Gaussian elimination might require on a moderate-size problem, suppose that $n = 100$ and that addition and multiplication times are $1\mu s$ and $2\mu s$, respectively (μs= microsecond = 10^{-6} second). Then the time for the additions and multiplications in the factorization is approximately

$$\frac{100^3}{3}(3\mu s) = 10^6 \mu s = 1 \textit{ second.}$$

There will also be time required for the other arithmetical operations of the complete elimination process, but this will be much less than 1 second. More importantly, there will be various "overhead" costs such as moving data back and forth from storage, and indexing. This could easily double or triple the total computing time, but on a computer of this speed only a few seconds would be required to solve a 100×100 system.

Banded Matrices

The previous discussion has assumed that the matrix of the system is "full," that is, it has few zero elements. For many matrices that arise in practice, and particularly in the solution of differential equations, the elements of the matrix are primarily zero. Perhaps the simplest nontrivial examples of this are the tridiagonal matrices discussed in Section 3.1; here there are no more than three non-zero elements in each row regardless of the size of n.

Tridiagonal matrices are special cases of *banded* matrices in which the non-zero elements are all contained in diagonals about the main diagonal, as illustrated in Figure 4.1. A matrix $A = (a_{ij})$ is a banded matrix of *bandwidth* $p+q+1$ if $a_{ij} = 0$ for all i, j such that $i-j > p$ or $j-i > q$. Such a matrix has all of its non-zero elements on the main diagonal, the closest p subdiagonals, and the closest q superdiagonals. If $p = q = 1$, there are only three non-zero diagonals, and the matrix is tridiagonal. In Section 3.1 tridiagonal matrices arose in the finite difference solution of two-point boundary-value problems. When derivatives are approximated by higher-order difference approximations, the matrices will have larger bandwidths. For example, the fourth-order approximation given by (3.1.42) leads to a matrix with bandwidth 5 ($p = q = 2$).

Matrices with larger bandwidths will be discussed in Chapter 9 in connection with the numerical solution of partial differential equations. In most cases of interest $p = q$, and p is called the *semi-bandwidth.* We will restrict ourselves to such matrices in the sequel, although the case $p \neq q$ presents no difficulties.

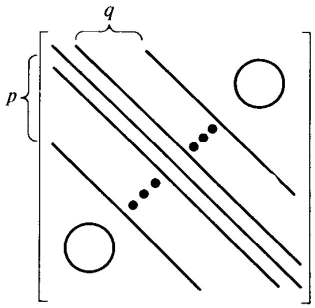

Figure 4.1: *Matrix with Bandwidth* $p + q + 1$

Just as for tridiagonal matrices, the Gaussian elimination algorithm for banded matrices benefits by not having to deal with the zero elements outside of the band. If A has semi-bandwidth p, there will be p coefficients to be eliminated in the first column, and this elimination will alter only the elements in the second through $(p + 1)$st rows and columns of A. The number of operations required will be p^2 multiplications and additions, and p divisions (not counting operations on the right-hand side of the system). The first reduced matrix will be a banded matrix with the same bandwidth, and hence the same count applies. After $n - p - 1$ of these reductions, there will remain a full $(p + 1) \times (p + 1)$ matrix to be reduced. Hence the number of additions (or multiplications) required in the triangular reduction is

$$(n - p - 1)p^2 + (\tfrac{1}{6}p)(p + 1)(2p + 1) \doteq np^2 - \tfrac{2}{3}p^3.$$

If n is large with respect to p (for example, $n = 1000, p = 7$), the dominant term in this operation count is np^2. As with full matrices, the number of operations to modify the right-hand side and perform the back substitution is of lower order, namely $0(np)$; see Exercise 4.2.7. However, as the bandwidth decreases, the number of operations for the right-hand side, the back substitution, and the divisions in the forward reduction constitute a larger fraction of the total operation count. In particular, for tridiagonal matrices, as discussed in Section 3.2, the forward reduction requires only $n - 1$ addition/multiplication pairs and $n-1$ divisions, whereas the right-hand side and back substitution require $(2n-1)$ additions/multiplications and n divisions.

The storage of a banded matrix does not require reserving a full $n \times n$ two-dimensional array, which would be very inefficient. All that is needed is $2p+1$ one-dimensional arrays, each holding one of the non-zero diagonals. In particular, a tridiagonal matrix may be stored in three one-dimensional arrays. However, if p is at all large it is probably better to store the diagonals of A as columns in a $(2p+1) \times n$ two-dimensional array, as indicated in Figure 4.2 for $p=2$ and $n=6$. In any case, the total storage required, including the right-hand side and the solution vector, is no more than $(2p+3)n$; in particular, for tridiagonal systems it is no more than $5n$.

$$\begin{bmatrix} 0 & 0 & a_{11} & a_{12} & a_{13} \\ 0 & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{53} & a_{54} & a_{55} & a_{56} & 0 \\ a_{64} & a_{65} & a_{66} & 0 & 0 \end{bmatrix}$$

Figure 4.2: *Storage for a Banded Matrix*

For a banded matrix, the factors L and U of the LU decomposition retain the same bandwidth (Exercise 4.2.8). In particular, for tridiagonal matrices

$$L = \begin{bmatrix} 1 & & & \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ & & l_n & 1 \end{bmatrix}, U = \begin{bmatrix} u_1 & a_{12} & & \\ & u_2 & a_{23} & \\ & & \ddots & \ddots \\ & & & a_{n-1,n} \\ & & & u_n \end{bmatrix}. \quad (4.2.18)$$

Note that the superdiagonal elements in U are the original elements of A. Matrices of the form (4.2.18) are called *bidiagonal*.

Determinants and Inverses

We return now to general matrices (not necessarily banded). We note first that the determinant of the coefficient matrix A, denoted by $\det A$, is an easy by-product of the elimination process. By the LU decomposition of A, and using the facts that the determinant of a product of two matrices is the product of the determinants and that the determinant of a triangular matrix is the product of its diagonal elements, we have

$$\det A = \det LU = \det L \det U = u_{11}u_{22}\cdots u_{nn} \ . \tag{4.2.19}$$

Thus, the determinant is just the product of the diagonal elements of the reduced triangular matrix and is computed by an additional $n-1$ multiplications. Even if only the determinant of the matrix is desired – and not the solution of a linear system – the Gaussian elimination reduction to triangular form is still the best general method for its computation.

The Gaussian elimination process is also the best way, in general, to compute the inverse of A, if that is needed. Let $\mathbf{e}_i$ be the vector with 1 in the ith position and zeros elsewhere. Then $\mathbf{e}_i$ is the ith column of the identity matrix I, and from the basic relation $AA^{-1} = I$ it follows that the ith column of A^{-1} is the solution of the linear system of equations $A\mathbf{x} = \mathbf{e}_i$. Hence we can obtain A^{-1} by solving the n systems of equations

$$A\mathbf{x}_i = \mathbf{e}_i, \qquad i = 1, \dots, n, \tag{4.2.20}$$

where the solution vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ will be the columns of A^{-1}.

We note that one does *not* wish to solve a system $A\mathbf{x} = \mathbf{b}$ by computing A^{-1} and then forming $\mathbf{x} = A^{-1}\mathbf{b}$. This would generally require considerably more work than just solving the system.

Several Right-Hand Sides

The above procedure for computing A^{-1} extends to the more general problem of solving several systems with the same coefficient matrix:

$$A\mathbf{x}_i = \mathbf{b}_i, \qquad i = 1, \dots, m. \tag{4.2.21}$$

Recall that this was the case is carrying out the Sherman-Morrison or Sherman-Morrison-Woodbury formulas in Section 3.2 (See Exercise 3.2.8 and (3.2.24)). In terms of the LU decomposition of A, (4.2.21) can be carried out efficiently by the following modification of (4.2.16):

$$\begin{aligned} &1.\quad \text{Factor } A = LU. \\ &2.\quad \text{Solve } L\mathbf{y}_i = \mathbf{b}_i, \qquad i = 1, \dots, m. \\ &3.\quad \text{Solve } U\mathbf{x}_i = \mathbf{y}_i, \qquad i = 1, \dots, m. \end{aligned} \tag{4.2.22}$$

Note that the matrix A is factored only once, regardless of the number of right-hand sides. Hence the operation count is $0(n^3) + 0(mn^2)$, the latter term representing parts 2 and 3 in (4.2.22). Only when m becomes nearly as large as n does the amount of work in parts 2 and 3 approach that of the factorization, at least for full matrices. In the case of computing A^{-1}, $m = n$, but the total operation count is still $0(n^3)$.

To carry out the intent of (4.2.21) in terms of the elimination process, we can either do parts 1 and 2 simultaneously, modifying the right-hand sides as the elimination proceeds, or we can first complete the factorization and save the multipliers l_{ij} to do 2.

The various algorithms in this section have all been predicated on the assumption that a_{11} and the subsequent diagonal elements of the reduced matrix do not vanish. In practice, it is not sufficient that these divisors be non-zero; they must also be large enough in some sense or severe rounding error problems may occur. In Section 4.3 we will consider these problems and the modifications that are necessary for the elimination process to be a viable procedure.

Supplementary Discussion and References: 4.2

There are many books on numerical linear algebra and these, as well as more elementary books on numerical methods, all discuss the problem of solving linear systems of equations. For an advanced treatment and more references, see Golub and van Loan [1989].

Although the Gaussian elimination method is very efficient, there are algorithms that have a lower operation count as a function of n. In particular, the number of multiplications required to solve a linear system of size n by a method due to Strassen [1969] is $0(n^{2.8\cdots})$. But the constant multiplying the high-order term is larger than for Gaussian elimination, and the method is more complicated; consequently it has not been a serious competitor to Gaussian elimination for practical computation. Recent research, however (Higham [1990]), indicates that this approach can be superior to Gaussian elimination, at least in certain circumstances.

EXERCISES 4.2

4.2.1. Verify the forward reduction of (4.2.10) to obtain (4.2.11). Then verify that $A = LU$, where L is given by (4.2.12) and U is the upper triangular matrix in (4.2.11)

4.2.2. Verify that multiplication of (4.2.2) by (4.2.13) gives (4.2.3). Then verify that (4.2.14) holds.

4.2.3. Verify the following statements:

a. The determinants of the L_i of (4.2.15) are all 1.

b. Products of lower (upper) triangular matrices are lower (upper) triangular.

c. Inverses of lower (upper) triangular matrices are lower (upper) triangular.

d. The inverse of L_i of (4.2.15) is the same matrix with the signs of the off-diagonal elements changed.

4.2.4. Verify by induction the following summation formulas:

$$\sum_{i=1}^{n} i = \tfrac{1}{2}n(n+1), \qquad \sum_{i=1}^{n} i^2 = \tfrac{1}{6}n(n+1)(2n+1).$$

4.2.5. Show that the following operation counts are correct for Gaussian elimination:

a. Number of additions (multiplications) to compute new right hand side = $n(n-1)/2$.

b. Number of divisions to compute the multipliers $l_{ik} = n(n-1)/2$.

c. Number of divisions in back substitution = n.

d. Number of additions (multiplications) in back substitution = $n(n-1)/2$.

4.2.6. Using a Gaussian elimination code (either a package or one that you write), measure the time to solve full linear systems of sizes $n = 50$ and $n = 100$. Discuss why the larger time is not exactly a factor of 8 larger than the smaller, as suggested by the $0(n^3)$ operation count.

4.2.7. If A is a banded matrix with semi-bandwidth p, show that $0(pn)$ operations are required to modify the right-hand side and do the back substitution.

4.2.8. Let A be a matrix with bandwidth $p+q+1$, as illustrated in Figure 4.1. If $A = LU$ is the LU decomposition, show that L has bandwidth $p+1$ and U has bandwidth $q+1$. In particular, for tridiagonal matrices show that L and U have the form (4.2.18).

4.2.9. Verify that the product of L and U in (4.2.18) is tridiagonal.

4.2.10. Write a computer program to implement Gaussian elimination for a banded matrix with p subdiagonals and q superdiagonals. Use the storage pattern of Figure 4.2.

4.2.11. If $A = LU$, show that $A^{-1} = U^{-1}L^{-1}$. Using this, describe an algorithm for computing A^{-1}, taking advantage of the triangular structure of U^{-1} and L^{-1}. How does your algorithm compare with solving the systems (4.2.20)?

4.2.12. We have shown that Gaussian elimination on a full matrix requires $0(n^3)$ operations and on a tridiagonal matrix it requires $0(n)$ operations. For what size semi-bandwidth will it require $0(n^2)$ operations?

4.2.13. Consider the $n \times n$ complex system $(A + iB)(\mathbf{u} + i\mathbf{v}) = \mathbf{b} + i\mathbf{c}$ where A, B, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{b}$, and $\mathbf{c}$ are real. Show how to write this as a real $2n \times 2n$ system. Compare the operation counts of Gaussian elimination applied to the real system and the original complex system, assuming that complex multiplication takes four real multiplications and two additions. (What does complex division require?)

4.2.14. (Uniqueness of LU Decomposition). Let A be non-singular and suppose that $A = LU = \hat{L}\hat{U}$ where L and $\hat{L}$ are unit lower triangular and U and $\hat{U}$ are upper triangular. Show that $L = \hat{L}$ and $U = \hat{U}$.

4.2.15. Let $A = LU$. Use Exercise 4.2.17 to give the LU decomposition of AT, where T is upper triangular. Specialize this to $T = D$, a diagonal matrix. What does this imply about the multipliers in Gaussian elimination when the columns of A are scaled?

4.2.16. (Jordan elimination). Show that it is possible to eliminate the elements above the main diagonal as well as below, so that the reduced system is $D\mathbf{x} = \hat{\mathbf{b}}$ where D is diagonal. How many operations does this method require to solve a linear system?

4.2.17. Show that if Gaussian elimination is done on the matrix (3.1.41) the factor L of the LU decomposition has the same non-zero structure as the lower triangular part of A except that the last row has, in general, all non-zero elements. Assuming that no operations are done on elements known to be zero, show that the operation count for this algorithm is $8(n-1)$ multiplications plus $6(n-1)$ additions plus $3n-2$ divisions.

4.2.18. Again for the matrix (3.1.41) show that solving $A\mathbf{x} = \mathbf{b}$ by the Sherman-Morrison formula (3.2.21) requires the following operations, assuming that the LU decomposition of T is done only once: $6n-1$ multiplications, $6n-1$ additions, and $3n-2$ divisions. Compare this operation count with that of Exercise 4.2.20. Show also that the use of the Sherman-Morrison-Woodbury formula (3.2.26) to solve this system, as described in Exercise 3.2.8, requires $8n-6$ multiplications, $8n-5$ additions, and $4n-3$ divisions.

4.2.19. Let T be a symmetric nonsingular tridiagonal matrix. Give an algorithm to find numbers $p_1, \cdots, p_n$ and $q_1, \cdots, q_n$ so that the i, j element of T^{-1} is $p_i q_j$ if $i \geq j$ and $p_j q_i$ if $i < j$. (Hint: Look at the first column of T^{-1}.)

4.3 Interchanges

In our discussion of the Gaussian elimination process in the previous section we assumed that a_{11} and all subsequent divisors were non-zero. However, we do not need to make such an assumption provided that we revise the algorithm so as to interchange equations if necessary, as we shall now describe.

We assume, as usual, that the coefficient matrix A is nonsingular. Suppose that $a_{11} = 0$. Then some other element in the first column of A must be non-zero or else A is singular (see Exercise 4.3.1). If, say, $a_{k1} \neq 0$, then we interchange the first equation in the system with the kth; clearly, this does not change the solution. In the new system the $(1,1)$ coefficient is now non-zero, and the elimination process can proceed. Similarly, an interchange can be done if any computed diagonal element that is to become a divisor in the next stage should vanish. Suppose, for example, that the elimination has progressed to the point

$$\begin{matrix} a_{11} & & & & \\ & \ddots & & & \\ & & a_{ii}^{(i-1)} & \cdots & a_{in}^{(i-1)} \\ & & \vdots & & \vdots \\ & & a_{ni}^{(i-1)} & \cdots & a_{nn}^{(i-1)} \end{matrix},$$

and that $a_{ii}^{(i-1)} = 0$. If all of the remaining elements below $a_{ii}^{(i-1)}$ in the ith column are also zero, this matrix is singular (see Exercise 4.3.2). Since the operations of adding a multiple of one row to another row, which produced this reduced matrix, do not affect the determinant, and an interchange of rows only changes the sign of the determinant, it follows that the original matrix is also singular, contrary to assumption. Hence at least one of the elements $a_{ki}^{(i-1)}$, $k = i+1, \ldots, n$, is non-zero, and we can interchange a row that contains a non-zero element with the ith row, thus ensuring that the new i,i element is non-zero. Again, this interchange of rows does not change the solution of the system. However, an interchange of rows does change the sign of the determinant of the coefficient matrix, so that if the determinant is to be computed a record must be kept of whether the number of interchanges is even or odd. In any case, in exact arithmetic we can ensure that Gaussian elimination can be carried out by interchanging rows so that no divisions are zero.

Rounding Error and Instability

Since in exact arithmetic the Gaussian elimination process produces the solution of the linear system in a finite number of steps, and there is no discretization error associated with the process, the only thing that can affect the accuracy of our computed solution is rounding error. There are two possibilities. The first is an accumulation of rounding errors during a large number of arithmetic operations. For example, if $n = 1{,}000$, the operation count of the previous section shows that on the order of $n^3 = 10^9$ operations will be required; even though the error in each individual operation may be small, the total buildup could be large. We shall see later that this potential accumulation of rounding error is not as serious as one might expect.

An Example of Instability

The second possibility involves catastrophic rounding errors. If an algorithm has this unfortunate characteristic, it is called *numerically unstable* and is not suitable as a general method. Although the interchange process described previously ensures that Gaussian elimination can be carried out mathematically for any nonsingular matrix, the algorithm can give rise to catastrophic rounding errors and is numerically unstable. We shall analyze a simple 2×2 example in order to see how this can occur.

Consider the system

$$\begin{bmatrix} -10^{-5} & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{4.3.1}$$

whose exact solution is

$$x_1 = -0.4999975\cdots, \qquad x_2 = 0.999995.$$

Now suppose that we have a decimal computer with a word length of four digits; that is, numbers are represented in the form $0.**** \times 10^p$. Let us carry out Gaussian elimination on this hypothetical computer. First, we note that $a_{11} \neq 0$, and no interchange is needed. The multiplier is

$$l_{21} = \frac{-0.2 \times 10^1}{0.1 \times 10^{-4}} = -0.2 \times 10^6,$$

which is exact, and the calculation for the new a_{22} is

$$\begin{aligned} a_{22}^{(1)} &= 0.1 \times 10^1 - (-0.2 \times 10^6)(0.1 \times 10^1) \\ &= 0.1 \times 10^1 + 0.2 \times 10^6 = 0.2 \times 10^6. \end{aligned} \tag{4.3.2}$$

The exact sum in (4.3.2) is, of course, 0.200001×10^6, but since the computer has a word length of only four digits this must be represented as 0.2000×10^6; this is the first error in the calculation.

The new b_2 is

$$b_2^{(1)} = -(-0.2 \times 10^6)(0.1 \times 10^1) = 0.2 \times 10^6. \tag{4.3.3}$$

No rounding errors occurred in this computation, nor do any occur in the back substitution:

$$\begin{aligned} x_2 &= \frac{b_2^{(1)}}{a_{22}^{(1)}} = \frac{0.2 \times 10^6}{0.2 \times 10^6} = 0.1 \times 10^1, \\ x_1 &= \frac{0.1 \times 10^1 - 0.1 \times 10^1}{-0.1 \times 10^{-4}} = 0. \end{aligned}$$

The computed x_2 agrees excellently with the exact x_2, but the computed x_1 has no digits of accuracy. Note that the only error made in the calculation is in $a_{22}^{(1)}$, which has an error in the sixth decimal place. Every other operation was exact. How, then, can this one "small" error cause the computed x_1 to deviate so drastically from its exact value?

Backward Error Analysis

The answer lies in the principle of *backward error analysis*, one of the most important concepts in scientific computing. The basic idea of backward error analysis is to "ask not what the error is, but what problem have we really solved." We shall invoke this principle here in the following form. Note that the quantity 0.000001×10^6 that was dropped from the computed $a_{22}^{(1)}$ in (4.3.2) is the original element a_{22}. Since this is the only place that a_{22} enters the calculation, the computed solution would have been the same if a_{22} were zero. Therefore the calculation on our four-digit computer has computed the exact solution of the system

$$\begin{bmatrix} -10^{-5} & 1 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{4.3.4}$$

Intuitively, we would expect the two systems (4.3.1) and (4.3.4) to have rather different solutions, and this is indeed the case.

But why did this occur? The culprit is the large multiplier l_{21}, which made it impossible for a_{22} to be included in the sum in (4.3.2) because of the word length of the machine. The large multiplier was due to the smallness of a_{11} relative to a_{21}, and the remedy is, again, an interchange of the equations. Indeed, if we solve the system

$$\begin{bmatrix} 2 & 1 \\ -10^{-5} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.3.5}$$

on our hypothetical four-digit computer, we obtain

$$\begin{aligned}
l_{21} &= \frac{0.1 \times 10^{-4}}{0.2 \times 10^1} = -0.5 \times 10^{-5} \\
a_{22}^{(1)} &= 0.1 \times 10^1 - (-0.5 \times 10^{-5})(1) = 0.1 \times 10^1 \\
b_2^{(1)} &= 0.1 \times 10^1 - (-0.5 \times 10^{-5})(0) = 0.1 \times 10^1 \\
x_2 &= \frac{0.1 \times 10^1}{0.1 \times 10^1} = 1.0 \\
x_1 &= \frac{-(0.1 \times 10^1)(1)}{0.2 \times 10^1} = -0.5 .
\end{aligned}$$

The computed solution now agrees excellently with the exact solution.

Partial Pivoting

By a relatively simple strategy we can always arrange to keep the multipliers in the elimination process less than or equal to 1 in absolute value. This is known as *partial pivoting*: at the kth stage of the elimination process an interchange of rows is made, if necessary, to place in the main diagonal position the element of largest absolute value from the kth column below or on the main diagonal. If we include this interchange strategy in the forward reduction algorithm (4.2.7), we have:

Forward Reduction with Partial Pivoting

```
For $k = 1, \dots, n-1$
    Find $m \geq k$ such that $|a_{mk}| = \max\{|a_{ik}| : i \geq k\}$.
    If $a_{mk} = 0$, then $A$ is singular, and stop.
            else interchange $a_{kj}$ and $a_{mj}$, $j = k, k+1, \dots, n$.
            interchange $b_k$ and $b_m$.                                (4.3.6)
    For $i = k+1, k+2, \dots, n$
        $l_{ik} = a_{ik}/a_{kk}$
        For $j = k+1, k+2, \dots, n$
            $a_{ij} = a_{ij} - l_{ik}a_{kj}$
        $b_i = b_i - l_{ik}b_k$.
```

Gaussian elimination with partial pivoting has proved to be an extremely reliable algorithm in practice. However, there are two major precautions that should be kept in mind. First, the matrix must be properly scaled before the algorithm is used. To illustrate this point, consider the system

$$\begin{bmatrix} 10 & -10^6 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -10^6 \\ 0 \end{bmatrix}. \tag{4.3.7}$$

No interchange is called for by the partial pivoting strategy since the (1, 1) element is already the largest in the first column. However, if we carry out the elimination on our hypothetical four-digit computer (see Exercise 4.3.5), we will encounter exactly the same problem that we did with the system (4.3.1). Indeed, (4.3.7) is just (4.3.1) with the first equation multiplied by -10^6.

The use of the partial pivoting strategy is predicated on the coefficient matrix being properly scaled so that the maximum element in each row and column is the same order of magnitude. This scaling is called *equilibration* or *balancing* of the matrix. Unfortunately, there is no known foolproof general procedure for such scaling, but usually it will be clear that some rows or columns of the matrix need scaling, and this can be done before the elimination starts. For example, if we were given the system (4.3.7), we should scale the first row so that its maximum element is approximately 1. Then a_{11} will be small, and the partial pivoting strategy will cause an interchange of the first and second rows.

The second precaution regarding the partial pivoting strategy is that even with an equilibrated matrix, it can be numerically unstable. Examples in which this can happen have been given, but the occurrence of such matrices in practical computations seems to be sufficiently rare that the danger can be safely ignored. (For additional remarks, see the Supplementary Discussion.)

LU with Interchanges

If row interchanges are made, the Gaussian elimination process is not equivalent to a factorization of the matrix A into the product of lower- and upper-triangular matrices; the lower-triangular matrix must be modified in the following way. A *permutation matrix*, P, is an $n \times n$ matrix that has exactly one element equal to 1 in each row and column and zeros elsewhere. A 4×4 example is

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \qquad (4.3.8)$$

Interchange of rows of a matrix can be effected by multiplication on the left by a permutation matrix. For example, multiplication of a 4×4 matrix by the permutation matrix (4.3.8) will leave the first and third rows the same and interchange the second and fourth rows (see Exercise 4.3.6). Thus the row interchanges of the coefficient matrix A that are required by the partial pivoting strategy can be represented by multiplication of A on the left by suitable permutation matrices. If P_i denotes the permutation matrix corresponding to the interchange required at the ith stage, then conceptually we are generating the triangular factorization of the matrix

$$P_{n-1}P_{n-2}\cdots P_2P_1A = PA = LU \qquad (4.3.9)$$

rather than A itself. Thus the factorization is $A = (P^{-1}L)U$. Since the product of permutation matrices is again a permutation matrix and the inverse of a permutation matrix is a permutation matrix (Exercise 4.3.7), the first factor is a permutation of a lower-triangular matrix, whereas the second is again upper-triangular. Note that if no interchange is required at the ith stage, the permutation matrix P_i is simply the identity matrix.

Banded Systems

Row interchanges require additional time and, in the case of banded matrices, also complicate the storage. Consider first a tridiagonal system. If an interchange is made at the first stage, the elements in the first two rows will be

$$\begin{array}{l} * \; * \; * \; 0 \cdots \\ * \; * \; 0 \cdots \end{array}$$

The elimination process will then reenter a (generally) non-zero element into the $(2, 3)$ position, and the reduced $(n-1) \times (n-1)$ matrix will again be tridiagonal. Thus, the effect of the interchanges will be to introduce possible non-zero elements into the second superdiagonal of the reduced triangular matrix. Then the factor U in the decomposition of A will no longer be bidiagonal but will have, in general, three non-zero diagonals. Perhaps the simplest way of handling the storage is to add an additional one-dimensional array to hold these elements in the second superdiagonal.

For a banded matrix with semibandwidth p the same kind of problem occurs. If an interchange is made at the first stage between the first and $(p+1)$st rows, an additional p elements will be introduced into the first row, and these, in turn, will be propagated into rows 2 through $p+1$ during the elimination process. Thus we need to provide storage for a possible additional p superdiagonals. The simplest way to handle this is to allow for an additional $n \times p$ array of storage at the outset. Of course this requires an additional np storage locations. An alternative method is based on the observation that the amount of additional storage needed is no more than the amount of storage required for the non-zero subdiagonals. As the subdiagonals are eliminated, we no longer will need that storage, and the new superdiagonals elements can be stored in those positions. However, it is this subdiagonal space that is normally used to store the multipliers if their retention is desired; in this case we have no alternative but to set aside additional storage.

Diagonally Dominant and Positive Definite Matrices

Although for general nonsingular matrices it is necessary to use the partial pivoting strategy, there are some types of matrices for which it is known that no interchanges are necessary. The most important of these are diagonally dominant matrices [recall (3.1.24) and (3.1.25)] and symmetric positive definite matrices [recall (4.1.9)]. In both cases, it is safe to use Gaussian elimination with no interchanges at all (see the Supplementary Discussion), although for positive definite matrices accuracy is often improved slightly by using interchanges. Not needing to interchange is especially advantageous for banded matrices, and it is a fortunate fact that most banded matrices arising from differential equations are either diagonally dominant or symmetric and positive definite. In particular, no interchanges are needed for those tridiagonal matrices that were shown to be diagonally dominant in Chapter 3.

In this section, we have discussed various questions concerning the accuracy of computed solutions of systems of linear equations. In the following section we will consider additional questions, including the important topic of "ill-conditioning."

Supplementary Discussion and References: 4.3

The interchange of rows required by partial pivoting need not be done explicitly. Instead, the interchanges may be carried out implicitly by using a permutation vector that keeps track of which rows are interchanged. Whether one should use explicit interchanges depends on the computer's "interchange" time, time required for indexing, program clarity, and other considerations.

In those cases in which partial pivoting is not sufficient to guarantee accuracy, we can use another strategy called *complete pivoting*, in which both rows and columns are interchanged so as to bring into the diagonal divisor position the largest element in absolute value in the remaining submatrix to be processed. This adds a significant amount of time to the Gaussian elimination process and is rarely incorporated into a standard program. See Wilkinson [1961] for further discussion.

In a very important paper, Wilkinson [1961] showed that the effect of rounding errors in Gaussian elimination is such that the computed solution is the exact solution of a perturbed system $(A + E)\mathbf{x} = \mathbf{b}$ (see also, for example, Golub and van Loan [1989] and Ortega [1990] for textbook discussions). A bound on the matrix E is of the form

$$||E||_\infty \leq p(n)g(n)\varepsilon||A||_\infty,$$

where $p(n)$ is a cubic polynomial in the size of the matrix, ε is the basic rounding error of the computer (for example, 2^{-27}), and $g(n)$ is the *growth factor* defined by

$$g = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{a}, \qquad a = \max_{i,j} |a_{ij}|,$$

where the $a_{ij}^{(k)}$ are the elements of the successive reduced matrices formed in the elimination process. The growth factor depends crucially on the interchange strategy used. With no interchanges, g may be arbitrarily large. With partial pivoting (and in exact arithmetic), $g(n)$ is bounded by 2^{n-1}, which for large n completely dominates $p(n)$. Wilkinson has exhibited matrices for which $g(n) = 2^{n-1}$, but such matrices seem to be very rare in practice; indeed, the actual size of g has been monitored extensively by Wilkinson and others for a large number of practical problems and has seldom exceeded 10, regardless of the size of the matrix. For the complete pivoting strategy a complicated but much better bound for g has been given by Wilkinson, and a long-standing conjecture was that $g(n) \leq n$. This conjecture has recently been shown by Gould [1991] to be false if rounding error in Gaussian elimination is allowed. Subsequently Edelman and Ohlrich [1991] used Mathematica to show that the conjecture is also false in exact arithmetic. The form of the best bound for g using complete pivoting remains an open question.

For matrices that are (column) diagonally dominant, the growth factor g is bounded by 2, without any interchanges. For symmetric positive definite matrices, g is equal to 1. This explains why for these two important classes of matrices, no interchange strategy is necessary.

The partial pivoting strategy ensures that the multipliers are all less than or equal to one in magnitude. However, for problems where the matrix has few non-zero elements it is sometimes desirable to require only that $|l_{ij}| \leq \alpha$ for some "threshold" parameter $\alpha > 1$. For further discussion, see Section 9.2.

EXERCISES 4.3

4.3.1. Suppose that the ith column of the matrix A consists of zero elements. Show that A is singular by the following different arguments:

a. The determinant of A is zero.

b. $A\mathbf{e}_i = 0$, where $\mathbf{e}_i$ is the vector with 1 in the ith position and zeros elsewhere.

c. A has, at most, $n - 1$ linearly independent columns.

4.3.2. Let A be a matrix of the form

$$\begin{bmatrix} a_{11} & \cdots & & & & a_{1n} \\ & \ddots & & & & \vdots \\ & & a_{i-1,i-1} & & & \\ & & & a_{ii} & & \\ & & & \vdots & \ddots & \\ & & & a_{i+1,i} & & \\ & & & a_{ni} & \cdots & a_{nn} \end{bmatrix}.$$

If $a_{ii} = a_{i+1,i} = \cdots = a_{ni} = 0$, show that A is singular.

4.3.3. Solve the following 3×3 system by Gaussian elimination by making row interchanges where needed to avoid division by zero:

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

4.3.4. Translate the algorithm (4.3.6) into a computer program for Gaussian elimination using partial pivoting. Include back substitution.

4.3.5. Apply Gaussian elimination to the system (4.3.7) using the four-digit decimal computer of the text. Repeat the calculation after interchanging the equations.

4.3.6. **a.** Show that multiplication of a 4×4 matrix on the left by the permutation matrix (4.3.8) interchanges the second and fourth rows and leaves the first and third rows the same.

b. Show that multiplication on the right by the permutation matrix interchanges the second and fourth columns.

c. Give the 4×4 permutation matrix that interchanges the first and third rows and leaves the second and fourth rows the same.

4.3.7. *Show that the product of two $n \times n$ permutation matrices is a permutation* matrix. Show that the inverse of a permutation matrix is a permutation matrix.

4.3.8. Let $A = LL^T$ be a factorization of a symmetric positive definite matrix A, where L is lower-triangular and has positive main diagonal elements. Show that if $\hat{L}$ is obtained from L by changing the sign of every element of the ith row, then $A = \hat{L}\hat{L}^T$. (This shows that the LL^T factorization is not unique, although there is a unique L with positive main diagonal elements.)

4.3.9. A matrix H is called *Hessenberg* (see Chapter 7) if $h_{ij} = 0$ when $i > j + 1$. How many operations are required to solve $H\mathbf{x} = \mathbf{b}$ by Gaussian elimination? If H is normalized so that $|h_{ij}| \le 1$ for all i, j, and partial pivoting is used in Gaussian elimination, show that the elements of U are less than n in magnitude.

4.3.10. Consider a matrix of the form

$$\begin{bmatrix} * & * & \cdots & * \\ * & * & & \\ \vdots & & \ddots & \\ * & & & * \end{bmatrix},$$

in which all elements are zero except in the first row and column and the main diagonal. How many operations will Gaussian elimination require if no pivoting is used? Does pivoting change this? Can you find a reordering of the equations and unknowns so that only $0(n)$ operations are required?

4.3.11. Assume that the positive definite condition $\mathbf{x}^T A\mathbf{x} > 0$ if $\mathbf{x} \neq 0$ holds for all real $\mathbf{x}$ even though A is not symmetric (such matrices are sometimes called *positive real*). Show that an LU decomposition of A exists.

4.4 Ill-Conditioning and Error Analysis

The Gaussian elimination algorithm with partial pivoting has proved to be an efficient and reliable method in practice. Nevertheless, it may fail to compute accurate solutions of systems of equations that are "ill-conditioned". A linear system of equations is said to be *ill-conditioned* if small changes in the elements of the coefficient matrix and/or right-hand side cause large changes in the solution. In this case no numerical method can be expected to produce an accurate solution, nor, in many cases, should a solution even be attempted.

An Example of an Ill-conditioned System

We begin with a 2×2 example. Consider the system

$$\begin{aligned} 0.832x_1 + 0.448x_2 &= 1.00 \\ 0.784x_1 + 0.421x_2 &= 0, \end{aligned} \tag{4.4.1}$$

and assume that we use a three-digit decimal computer to carry out Gaussian elimination. Since a_{11} is the largest element of the matrix no interchange is required, and the computation of the new elements $a_{22}^{(1)}$ and $b_1^{(1)}$ is

$$\begin{aligned} l_{21} &= \frac{0.784}{0.832} = 0.942 \mid 308\cdots = 0.942 \\ a_{22}^{(1)} &= 0.421 - 0.942 \times 0.448 = 0.421 - 0.422 \mid 016 = -0.001 \\ b_2^{(1)} &= 0 - 1.00 \times 0.942 = -0.942, \end{aligned} \tag{4.4.2}$$

where we have indicated by the vertical bars those digits lost in the computation. Hence the computed triangular system is

$$\begin{aligned} 0.832x_1 + 0.448x_2 &= 1.00 \\ -0.001x_2 &= -0.942, \end{aligned}$$

and the back substitution produces the approximate solution

$$x_1 = -506, \qquad x_2 = 942. \tag{4.4.3}$$

But the exact solution of (4.4.1), correct to three figures, is

$$x_1 = -439, \qquad x_2 = 817, \tag{4.4.4}$$

so the computed solution is incorrect by about 15%. Why has this occurred?

The first easy answer is that we have lost significance in the calculation of $a_{22}^{(1)}$. Indeed, it is clear that the computed value of $a_{22}^{(1)}$ has only one significant figure, so our final computed solution will have no more than one significant figure. But this is only the manifestation of the real problem. We invoke again the principle of backward error analysis. By carrying out a more detailed computation we can show that the computed solution (4.4.3) is the exact solution of the system

$$\begin{aligned} 0.832x_1 + 0.447974\cdots x_2 &= 1.00 \\ 0.783744\cdots x_1 + 0.420992\cdots x_2 &= 0. \end{aligned} \tag{4.4.5}$$

The maximum percentage change between the elements of this system and the original system (4.4.1) is only 0.03%; therefore errors in the data are magnified by a factor of about 500.

The root cause of this ill-conditioning is that the coefficient matrix of (4.4.1) is "almost singular." Geometrically, this means that the lines defined by the two equations (4.4.1) are almost parallel, as indicated in Figure 4.3. Consider now the system of equations

$$\begin{aligned} 0.832x_1 + 0.448x_2 &= 1.00 \\ 0.784x_1 + (0.421 + \varepsilon)x_2 &= 0. \end{aligned} \tag{4.4.6}$$

The second equation defines a family of lines depending on the parameter ε. As ε increases from zero to approximately 0.0012, the line rotates counterclockwise and its intersection with the line defined by the first equation recedes to infinity until the two lines become exactly parallel and no solution of the linear system exists.

For only one value of ε, say ε_0, is the coefficient matrix of (4.4.6) singular, *but for infinitely many values of ε near ε_0 the matrix is almost singular.* In general, the probability of a matrix being exactly singular is very small unless it was constructed in such a way that singularity is ensured. For example, we saw in Section 3.2 that periodic boundary conditions can give rise to singular coefficient matrices. In many situations, however, it may not be obvious in the formulation of the problem that the resulting matrix will be singular or almost singular. This must be detected during the course of the solution and a warning issued. We will consider such detection mechanisms later, but we point out here that, in general, it is extremely difficult to ascertain computationally if a given matrix is exactly singular. For example, if LU is the computed factorization of A and $u_{nn} = 0$, then U is singular. But u_{nn} may be contaminated by rounding error so we cannot claim that A itself is singular. Conversely, if the computed u_{nn} is not zero, this does not guarantee that A is non-singular. The fundamental problem is the detection of zero in the presence of rounding error.

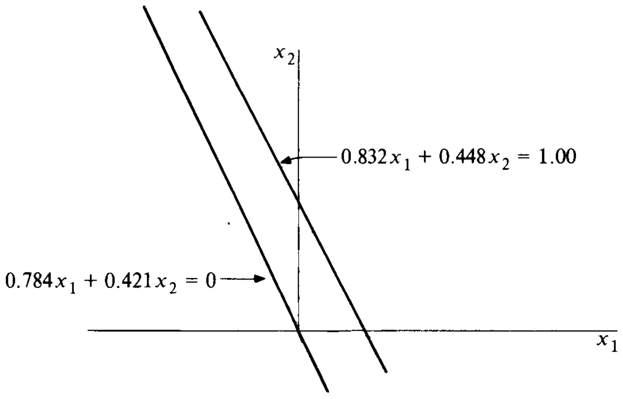

Figure 4.3: *Almost-Parallel Lines Defined by* (4.4.1). *The Intersection of the Lines is at* (-439,817)

However, near-singularity is easier to detect, and if that is the case, it is likely that the problem should be reformulated. For example, we may have chosen variables that are close to being dependent, and we should remove some of them or choose another set of variables.

Determinants and Ill-conditioning

Since a matrix is singular if its determinant is zero, it is sometimes suggested that the smallness of the determinant is a measure of the ill-conditioning of the system. This is not however generally true as the following example shows:

$$\det\begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix} = 10^{-20}, \qquad \det\begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix} = 10^{20}. \tag{4.4.7}$$

The values of the two determinants are very different, but the lines defined by the two corresponding sets of equations

$$\begin{aligned} 10^{-10}x_1 &= 0 \qquad & 10^{10}x_1 &= 0 \\ 10^{-10}x_2 &= 0 \qquad & 10^{10}x_2 &= 0 \end{aligned} \tag{4.4.8}$$

are the same and are the coordinate axes. As we shall see more clearly in a moment, if the lines defined by the equations of a system are perpendicular, that system is "perfectly conditioned." Thus the magnitude of the determinant of the coefficient matrix is not a good measure of the near-singularity of the matrix. It can, however, become the basis of such a measure if the matrix is suitably scaled, as we shall now see.

For two equations it is clear that a good measure of the "almost parallelness" of the corresponding two lines is the angle between them. An essentially equivalent measure is the area of the parallelogram shown in Figure 4.4, in which the sides of the parallelogram are of length 1 and the height is denoted by h. The area of the parallelogram is then equal to h since the base is 1, and the angle θ between the lines defined by the two equations is related to h by $h = \sin\theta$. The area, h, varies between zero, when the lines coalesce, and 1, when they are perpendicular.

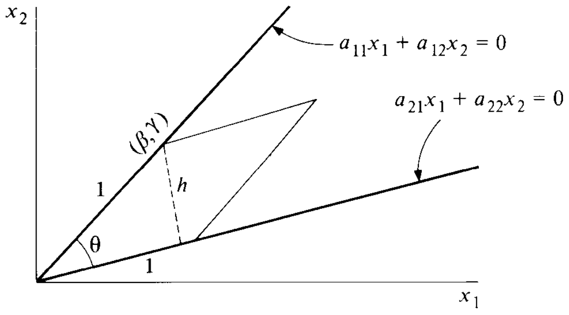

Figure 4.4: *The Unit Parallelogram*

From analytic geometry, the distance from the point (β, γ) to the line $a_{21}x_1 + a_{22}x_2 = 0$ is

$$h = \frac{|a_{21}\beta + a_{22}\gamma|}{\alpha_2}, \qquad \alpha_2 = (a_{21}^2 + a_{22}^2)^{1/2}.$$

If we assume that $a_{11} \geq 0$, the coordinates (β, γ) are given by

$$\beta = \frac{-a_{12}}{\alpha_1}, \qquad \gamma = \frac{a_{11}}{\alpha_1}, \qquad \alpha_1 = (a_{11}^2 + a_{12}^2)^{1/2},$$

so that

$$h = \frac{|a_{11}a_{22} - a_{21}a_{12}|}{\alpha_1\alpha_2} = \frac{|\det A|}{\alpha_1\alpha_2}. \tag{4.4.9}$$

Hence we see that the area, h, is just the determinant divided by the product $\alpha_1\alpha_2$.

This measure easily extends to n equations. Let $A = (a_{ij})$ be the coefficient matrix, and set

$$V = \frac{|\det A|}{\alpha_1\alpha_2\cdots\alpha_n} = \left|\det \begin{bmatrix} a_{11}/\alpha_1 & \cdots & a_{1n}/\alpha_1 \\ \vdots & & \vdots \\ a_{n1}/\alpha_n & \cdots & a_{nn}/\alpha_n \end{bmatrix}\right|, \tag{4.4.10}$$

where

$$\alpha_i = (a_{i1}^2 + a_{i2}^2 + \cdots + a_{in}^2)^{1/2}.$$

We have called the quantity in (4.4.10) V instead of h because it is the volume of the n-dimensional unit parallelepiped circumscribed by the lines defined by the rows of matrix A; that is,

$$\frac{1}{\alpha_i}(a_{i1}, a_{i2}, \ldots, a_{in}), \qquad i = 1, \ldots, n,$$

are the coordinates of n points in n-dimensional space located Euclidean distance 1 from the origin, and these n points define a parallelepiped whose sides are of length 1. It is intuitively clear, and can be proved rigorously, that the volume of this parallelepiped is between zero, when two or more of the edges coincide, and 1, when the edges are all mutually perpendicular. If $V = 0$, then $\det A = 0$, and the matrix is singular. If $V = 1$, then the edges are as far from being parallel as possible, and in this case the matrix is called *perfectly conditioned.*

Ramifications of Ill-Conditioning

There are various ramifications of ill-conditioning of a matrix besides the difficulty in computing an accurate solution of the corresponding linear system. Consider again the system (4.4.1) and suppose that (4.4.5) is the "real" system that we wish to solve but that the coefficients of this system must be measured by some physical apparatus accurate to only the third decimal place. Thus (4.4.1) is not the system that we really wish to solve but is the best approximation to it that we can make. Suppose that we can also claim that the coefficients of (4.4.1) are accurate to at least 0.05%, as indeed they are, compared with (4.4.5). Then, it is an often-heard argument that we should be able to compute the solution of the system to about the same accuracy. But we have seen that this is not true; the ill-conditioning of the coefficient matrix magnifies small errors in the coefficients by a factor of about 500 in the case of (4.4.1). Hence no matter how accurately the system (4.4.1) is solved, we will still have the error that has come from the measurement error in the coefficients. If, for example, we need the solution of the "real" system

(4.4.5) accurate to less than 1%, we need to measure the coefficients much more accurately than three decimal places.

In some cases, however, the coefficient matrix may be exact. A famous example of a class of ill-conditioned matrices is the *Hilbert matrices* (or *Hilbert segments*), in which the elements of the matrix are exact rational numbers:

$$H_n = \begin{bmatrix} 1 & 1/2 & \cdots & 1/n \\ 1/2 & & & \\ \vdots & & & \vdots \\ 1/n & & \cdots & 1/(2n-1) \end{bmatrix}. \tag{4.4.11}$$

These matrices are increasingly ill-conditioned as n increases. If for $n = 8$ the coefficients are entered in the computer as binary fractions exact to the extent possible with 27 binary digits (equivalent to about 8 decimal digits), the exact inverse of the matrix in the computer differs from the exact inverse of H_8 in the first figure!

The following is another manifestation of ill-conditioning. Suppose that $\bar{\mathbf{x}}$ is a computed solution of the system $A\bar{\mathbf{x}} = \mathbf{b}$. One way to try to ascertain the accuracy of $\bar{\mathbf{x}}$ is to form the *residual vector*,

$$\mathbf{r} = \mathbf{b} - A\bar{\mathbf{x}}. \tag{4.4.12}$$

If $\bar{\mathbf{x}}$ were the exact solution, then $\mathbf{r}$ would be zero. Thus we would expect $\mathbf{r}$ to be "small" if $\bar{\mathbf{x}}$ were a good approximation to the exact solution, and, conversely, that if $\mathbf{r}$ were small, then $\bar{\mathbf{x}}$ would be a good approximation. This is true in some cases, but if A is ill-conditioned, the magnitude of $\mathbf{r}$ can be very misleading. As an example, consider the system

$$\begin{aligned} 0.780x_1 + 0.563x_2 &= 0.217 \\ 0.913x_1 + 0.659x_2 &= 0.254, \end{aligned} \tag{4.4.13}$$

and the approximate solution

$$\bar{\mathbf{x}} = \begin{bmatrix} 0.341 \\ -0.087 \end{bmatrix}. \tag{4.4.14}$$

Then, the residual vector is

$$\mathbf{r} = \begin{bmatrix} 10^{-6} \\ 0 \end{bmatrix}. \tag{4.4.15}$$

Now consider another very different approximate solution

$$\bar{\mathbf{x}} = \begin{bmatrix} 0.999 \\ -1.001 \end{bmatrix}, \tag{4.4.16}$$

and the corresponding residual vector

$$\mathbf{r} = \begin{bmatrix} 0.0013\cdots \\ -0.0015\cdots \end{bmatrix}. \tag{4.4.17}$$

By comparing the residuals (4.4.15) and (4.4.17) we could easily conclude that (4.4.14) is the better approximate solution. However, the exact solution of (4.4.13) is $(1, -1)$, so the residuals give completely misleading information.

Condition Numbers Based on Norms

We turn now to another way of measuring the ill-conditioning of a matrix by means of norms (see Appendix 2 for a review of vector and matrix norms). Suppose first that $\hat{\mathbf{x}}$ is the solution of $A\mathbf{x} = \mathbf{b}$ and that $\hat{\mathbf{x}} + \Delta\mathbf{x}$ is the solution of the system with the right-hand side $\mathbf{b} + \Delta\mathbf{b}$:

$$A(\hat{\mathbf{x}} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}. \tag{4.4.18}$$

Since $A\hat{\mathbf{x}} = \mathbf{b}$, it follows that $A(\Delta\mathbf{x}) = \Delta\mathbf{b}$ and $\Delta\mathbf{x} = A^{-1}(\Delta\mathbf{b})$, assuming, as usual, that A is nonsingular. Thus

$$||\Delta\mathbf{x}|| \le ||A^{-1}||\ ||\Delta\mathbf{b}||, \tag{4.4.19}$$

which shows that the change in the solution due to a change in the right-hand side is bounded by $||A^{-1}||$. Thus a small change in $\mathbf{b}$ may cause a large change in $\hat{\mathbf{x}}$ if $||A^{-1}||$ is large. The notion of "large" is always relative, however, and it is more useful to deal with the relative change $||\Delta\mathbf{x}||/||\hat{\mathbf{x}}||$. From $A\hat{\mathbf{x}} = \mathbf{b}$, it follows that

$$||\mathbf{b}|| \le ||A||\ ||\hat{\mathbf{x}}||,$$

and combining this with (4.4.19) yields

$$||\Delta\mathbf{x}||\ ||\mathbf{b}|| \le ||A||\ ||A^{-1}||\ ||\Delta\mathbf{b}||\ ||\hat{\mathbf{x}}||,$$

or, equivalently (if $\mathbf{b} \neq 0$),

$$\frac{||\Delta\mathbf{x}||}{||\hat{\mathbf{x}}||} \le ||A||\ ||A^{-1}||\frac{||\Delta\mathbf{b}||}{||\mathbf{b}||}. \tag{4.4.20}$$

This inequality shows that the relative change in $\hat{\mathbf{x}}$ due to a change in $\mathbf{b}$ is bounded by the relative change in $\mathbf{b}$, $||\Delta\mathbf{b}||/||\mathbf{b}||$, multiplied by $||A||\ ||A^{-1}||$. The latter quantity is of great importance and is called the *condition number* of A (with respect to the norm being used); it will be denoted by $\text{cond}(A)$. This is the condition number for the problem of solving $A\mathbf{x} = \mathbf{b}$ or computing A^{-1}. Other problems will have different condition numbers. For example, in Chapter 7 we discuss the computation of eigenvalues, and the condition number of an eigenvalue of A is different than cond(A).

Consider next the case in which the elements of A are changed so that the perturbed equations are

$$(A + \delta A)(\hat{\mathbf{x}} + \delta\mathbf{x}) = \mathbf{b}. \tag{4.4.21}$$

Thus, since $A\hat{\mathbf{x}} = \mathbf{b}$,

$$A\delta\mathbf{x} = \mathbf{b} - A\hat{\mathbf{x}} - \delta A(\hat{\mathbf{x}} + \delta\mathbf{x}) = -\delta A(\hat{\mathbf{x}} + \delta\mathbf{x}),$$

or

$$-\delta\mathbf{x} = A^{-1}\delta A(\hat{\mathbf{x}} + \delta\mathbf{x}).$$

Therefore

$$||\delta\mathbf{x}|| \leq ||A^{-1}||\ ||\delta A||\ ||\hat{\mathbf{x}} + \delta\mathbf{x}|| = \text{cond}(A)\frac{||\delta A||}{||A||}||\hat{\mathbf{x}} + \delta\mathbf{x}||,$$

so that

$$\frac{||\delta\mathbf{x}||}{||\hat{\mathbf{x}} + \delta\mathbf{x}||} \leq \text{cond}(A)\frac{||\delta A||}{||A||}. \tag{4.4.22}$$

Once again, the condition number plays a major role in the bound. Note that (4.4.22) expresses the change in $\hat{\mathbf{x}}$ relative to the perturbed solution, $\hat{\mathbf{x}} + \delta\mathbf{x}$, rather than $\hat{\mathbf{x}}$ itself, as in (4.4.20), although it is possible to obtain a bound relative to $\hat{\mathbf{x}}$.

The inequalities (4.4.20) and (4.4.22) need to be interpreted correctly. Note first that cond(A) ≥ 1 (see Exercise 4.4.2). If cond(A) is close to 1, then small relative changes in the data can lead to only small relative changes in the solution. In this case we say that the problem is *well-conditioned.* This also guarantees that the residual vector provides a valid estimate of the accuracy of an approximate solution $\bar{\mathbf{x}}$. From (4.4.12)

$$\mathbf{r} = A(A^{-1}\mathbf{b} - \bar{\mathbf{x}}), \tag{4.4.23}$$

so that, if $\mathbf{e} = A^{-1}\mathbf{b} - \bar{\mathbf{x}}$ is the error in the approximate solution,

$$\mathbf{e} = A^{-1}\mathbf{r}. \tag{4.4.24}$$

This is the fundamental relation between the residual and the error. Then

$$||\mathbf{e}|| \leq ||A^{-1}||\ ||\mathbf{r}|| = \text{cond}(A)\frac{||\mathbf{r}||}{||A||}, \tag{4.4.25}$$

so that the error is bounded by cond(A) times a normalized residual vector. Note that it is necessary to normalize the residual vector somehow since we can multiply the equation $A\mathbf{x} = \mathbf{b}$ by any constant without changing the solution, and such a multiplication would change the residual by the same amount.

On the other hand, if the condition number is large, then small changes in the data may cause large changes in the solution, but not necessarily, depending on the particular perturbation. The practical effect of a large condition number depends on the accuracy of the data and the word length of the computer being used. If, for example, $\text{cond}(A) = 10^6$, then the equivalent of 6 decimal digits could possibly be lost. On a computer with a word length equivalent to 8 decimal digits, this could be disastrous; on the other hand if the word length were the equivalent of 16 decimal digits, it might not cause much of a problem. If the data are measured quantities, however, the computed solution may not have any meaning even if computed accurately.

Computation of the Condition Number

In general, it is very difficult to compute the condition number $||A||\ ||A^{-1}||$ without knowing A^{-1}, although the packages LINPACK and LAPACK (see the Supplementary Discussion) are able to estimate $\text{cond}(A)$ in the course of solving a linear system. In some cases of interest, however, it is relatively easy to compute the condition number explicitly, and we give an example of this for the $(2, -1)$ tridiagonal matrix of (3.1.10).

As given in Appendix 2, the l_2 norm of a symmetric matrix is its spectral radius $\rho(A)$. Thus

$$\text{cond}_2(A) = ||A||_2||A^{-1}||_2 = \rho(A)\rho(A^{-1}). \tag{4.4.26}$$

For the matrix of (3.1.10), we can compute explicitly (Exercise 4.4.5) the eigenvalues as

$$\lambda_k = 2 - 2\cos\frac{k\pi}{n+1} = 2 - 2\cos kh, \tag{4.4.27}$$

where we have set $h = \pi/(n+1)$. Thus, the largest eigenvalue of A is

$$\rho(A) = \lambda_n = 2 - 2\cos nh,$$

and the smallest is

$$\lambda_1 = 2 - 2\cos h > 0.$$

(This shows, incidentally, that A is positive definite; see Appendix 2). Since the eigenvalues of A^{-1} are $\lambda_1^{-1}, \ldots, \lambda_n^{-1}$ (see Exercise 4.4.6), the spectral radius of A^{-1} is λ_1^{-1}. Thus

$$\text{cond}_2(A) = \lambda_n\lambda_1^{-1} = \frac{1-\cos nh}{1-\cos h} = \frac{1+\cos h}{1-\cos h}.$$

For small h we can approximate the cosine by the first-order Taylor expansion

$$\cos h \doteq 1 - \frac{h^2}{2}.$$

Thus

$$\text{cond}_2(A) \doteq \frac{4-h^2}{h^2} = 0(h^{-2}) = 0(n^2).$$

This shows that A is moderately ill-conditioned and that the condition number grows approximately as the square of the dimension of the matrix, or as h^{-2}. This is typical for matrices arising from boundary value problems.

Supplementary Discussion and References: 4.4

The state of the art in solving linear equations has now reached a very high level, especially for full and banded systems that can be stored in fast memory. Probably the best current set of codes is LINPACK, a package of FORTRAN subroutines. Recently LINPACK has evolved to a new package, LAPACK, specifically designed for use on vector and parallel computers. See Dongarra et al. [1979] for a discussion of LINPACK and Dongarra and Anderson et al. [1990] for LAPACK.

One way to attempt to obtain an accurate solution of ill-conditioned systems – and also to detect the ill-conditioning – is *iterative refinement*, which we now describe. Let $\mathbf{x}_1$ be the computed solution of the system $A\mathbf{x} = \mathbf{b}$ and $\mathbf{r}_1 = A\mathbf{x}_1 - \mathbf{b}$. If $\mathbf{x}_1$ is not the exact solution, then $\mathbf{r}_1 \neq 0$. Now solve the system $A\mathbf{z}_1 = -\mathbf{r}_1$. If $\mathbf{z}_1$ were the exact solution of this system, then

$$\mathbf{A}(\mathbf{x}_1 + \mathbf{z}_1) = A\mathbf{x}_1 - \mathbf{r}_1 = \mathbf{b}$$

so that $\mathbf{x}_1 + \mathbf{z}_1$ is the exact solution of the original system. Of course, we will not be able to compute $\mathbf{z}_1$ exactly, but we hope that $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{z}_1$ will be a better approximation to the exact solution than $\mathbf{x}_1$. For this to be the case, it is usual to compute the residual in double precision, although single precision can be sufficient in certain cases (see Skeel [1980] and Arioli et al. [1989]). The process can be repeated: form $\mathbf{r}_2 = A\mathbf{x}_2 - \mathbf{b}$, solve $A\mathbf{z}_2 = -\mathbf{r}_2$, set $\mathbf{x}_3 = \mathbf{x}_2 + \mathbf{z}_2$, and so on. One or two iterations will usually suffice to obtain an accurate solution, unless the problem is very ill-conditioned. For further discussion of iterative refinement, see Golub and Van Loan [1989].

For additional perturbation results such as (4.4.20) see Stewart and Sun [1990].

EXERCISES 4.4

4.4.1. Compute the determinant and the normalized determinant (4.4.10) for the matrix of (4.4.1) and for the matrix

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

4.4.2. Using properties of matrix norms, prove that $\text{cond}(A) \geq 1$.

4.4.3. Compute $\text{cond}(A)$ for the matrices in Exercise 4.4.1 using both the l_1 and l_∞ norms (see Appendix 2 for definitions of these norms).

4.4.4. Solve the system (4.4.1) for different right-hand sides. Compare the differences in these solutions to the bound (4.4.20), using the l_∞ norm.

4.4.5. Use the trigonometric identity $\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$ to verify that the eigenvalues of the matrix (3.1.10) are given by (4.4.27) with corresponding eigenvectors

$$\mathbf{x}_k = (\sin kh, \sin 2kh, \ldots, \sin nkh)^T.$$

That is, verify that $A\mathbf{x}_k = \lambda_k \mathbf{x}_k$, $k = 1, \ldots, n$.

4.4.6. If A is nonsingular and $A\mathbf{x} = \lambda\mathbf{x}$, show that $A^{-1}\mathbf{x} = \lambda^{-1}\mathbf{x}$.

4.4.7. If A has eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding eigenvectors $\mathbf{v}_n \ldots, \mathbf{v}_n$, show that $(A + cI)\mathbf{v}_k = (\lambda_k + c)\mathbf{v}_k$, $k = 1, \ldots, n$, so that $A + cI$ has eigenvalues $\lambda_1 + c, \ldots, \lambda_n + c$.

4.4.8. Let A be the matrix of (3.1.9) in which $c_1 = c_2 = \cdots = c_n = c$. Use Exercise 4.4.7 to show that A has eigenvalues $2 + c - 2coskh$, $k = 1, \ldots, n$, and use them to find $\text{cond}_2(A)$. Discuss how $\text{cond}_2(A)$ varies with c.

4.4.9 Let

$$A = \begin{pmatrix} 1.6384 & 0.8065 \\ 0.8321 & 0.4096 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 0.8319 \\ 0.4225 \end{pmatrix}.$$

Verify that $(1, -1)^T$ is the exact solution of $A\mathbf{x} = \mathbf{b}$. If $\mathbf{r} = A\bar{\mathbf{x}} - \mathbf{b}$, construct an $\bar{\mathbf{x}}$ for which $\mathbf{r} = (-10^{-8}, 10^{-8})$ exactly. Find $\text{cond}_\infty(A)$. If $\mathbf{b}$ is exact, how small should the relative error in A be so that the solution can be guaranteed to have a relative error which is $\leq 10^{-8}$?

4.4.10 Let $\mathbf{r} = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b}, \bar{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x}, \mathbf{A}\mathbf{x} = \mathbf{b}$ and $R = AC - I$, where C is an approximate inverse of A and $||R|| < 1$. Prove that

$$||\delta\mathbf{x}|| \leq \frac{||\mathbf{r}||||\mathbf{C}||}{\mathbf{1} - ||\mathbf{R}||}.$$

4.4.11 Let A be a non-singular diagonal matrix. Show that the quantity V of (4.4.10) is always equal to 1, but that $||A||||A^{-1}||$ may be arbitrarily large.

4.5 Other Factorizations

So far in this chapter we have considered only Gaussian elimination and the corresponding LU factorization. But there are other factorizations of the matrix A which are sometimes very useful.

Cholesky Factorization

In the case of a symmetric positive definite matrix there is an important variant of Gaussian elimination, *Cholesky's method*, which is based on a factorization (or decomposition) of the form

$$A = LL^T. \tag{4.5.1}$$

Here L is a lower-triangular matrix but does not necessarily have 1's on the main diagonal as in the LU factorization. The factorization (4.5.1) is unique, provided that L is required to have positive diagonal elements (see Exercise 4.5.1).

The product in (4.5.1) is

$$\begin{bmatrix} l_{11} & & & \\ \vdots & \ddots & & \\ l_{i1} & l_{ii} & & \\ \vdots & & \ddots & \\ l_{n1} & \cdots & & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & \cdots & l_{i1} & \cdots & l_{n1} \\ & \ddots & & & \\ & & l_{ii} & & \vdots \\ & & & \ddots & \\ & & & & l_{nn} \end{bmatrix}. \tag{4.5.2}$$

By equating elements of the first column of (4.5.2) with corresponding elements of A, we see that $a_{i1} = l_{i1}l_{11}$, so the first column of L is determined by

$$l_{11} = (a_{11})^{1/2}, \qquad l_{i1} = \frac{a_{i1}}{l_{11}}, \qquad i = 2, \ldots, n. \tag{4.5.3}$$

In general,

$$a_{ii} = \sum_{k=1}^{i} l_{ik}^2, \qquad a_{ij} = \sum_{k=1}^{j} l_{ik}l_{jk}, \qquad j < i, \tag{4.5.4}$$

which forms the basis for determining the columns of L in sequence . Once L is computed, the solution of the linear system can proceed just as in the LU decomposition (4.2.16): solve $L\mathbf{y} = \mathbf{b}$ and then solve $L^T\mathbf{x} = \mathbf{y}$. The algorithm for the factorization is given in Figure 4.5. For the Cholesky decomposition to be carried out, it is necessary that the quantities $a_{jj} - \sum l_{jk}^2$ all be positive so that the square roots may be taken. If the coefficient matrix A is positive definite, these quantities are indeed positive; moreover, the algorithm is numerically stable.

```
For $j = 1, \ldots, n$
    $l_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2\right)^{1/2}$
    For $i = j+1, \ldots, n$
        $l_{ij} = \dfrac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}$
```

Figure 4.5: *Cholesky Factorization*

The Cholesky factorization enjoys two advantages over *LU* factorization. First, there are approximately half as many arithmetic operations (Exercise 4.5.2). Square roots are also required in the Cholesky factorization, although there is a variant of the algorithm which avoids these (see Exercise 4.5.3). The second advantage is that by utilizing symmetry only the lower triangular part of A needs to be stored. As with the *LU* factorization, the l_{ij} can be overwritten onto the corresponding portions of A as they are computed. Finally, the Cholesky factorization extends readily to banded matrices and preserves the bandwidth, just as *LU* factorization without interchanges (see Exercise 4.5.4).

The QR Factorization

The Cholesky factorization applies only to symmetric positive definite matrices. We next consider a factorization that applies to any matrix. Indeed, later in this section we will use this factorization for rectangular matrices, but for the moment we will assume that A is $n \times n$ and real. The *QR factorization* (or *decomposition* or *reduction*) is then

$$A = QR, \tag{4.5.5}$$

where Q is an orthogonal matrix (See Appendix 2) and R is upper triangular. (We could denote R by U, but historically R has always been used in this context.)

We will obtain the matrix Q as a product of simpler orthogonal matrices based upon the rotation matrix

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \tag{4.5.6}$$

We generalize such rotation matrices to $n \times n$ matrices of the form

$$P_{ij} = \begin{bmatrix} 1 &&&&&&&&& \\ & \ddots &&&&&&&& \\ && 1 &&&&&&& \\ &&& c_{ij} &&&& s_{ij} && \\ &&&& 1 &&&&& \\ &&&&& \ddots &&&& \\ &&&&&& 1 &&& \\ &&& -s_{ij} &&&& c_{ij} && \\ &&&&&&&& 1 & \\ &&&&&&&&& \ddots & \\ &&&&&&&&&& 1 \end{bmatrix}, \tag{4.5.7}$$

where $c_{ij} = \cos\theta_{ij}$ and $s_{ij} = \sin\theta_{ij}$ are located in the ith and jth rows and columns, as indicated. Such matrices are called *plane rotation matrices* or *Givens transformations.* Just as (4.5.6) defines a rotation of the plane, a matrix of the form (4.5.7) gives a rotation in the (i,j) plane in n-space. It is easy to show (Exercise 4.5.5) that the matrices (4.5.6) and (4.5.7) are orthogonal.

We now use the matrices P_{ij} in the following way to achieve the QR factorization (4.5.5). Let $\mathbf{a}_i$ denote the ith row of A. Then multiplication of A by P_{12} gives the matrix

$$A_1 = P_{12}A = \begin{bmatrix} c_{12}\mathbf{a}_1 + s_{12}\mathbf{a}_2 \\ -s_{12}\mathbf{a}_1 + c_{12}\mathbf{a}_2 \\ \mathbf{a}_3 \\ \vdots \\ \mathbf{a}_n \end{bmatrix}. \tag{4.5.8}$$

Note that only the first two rows of A are changed by this multiplication. If we choose s_{12} and c_{12} so that

$$-a_{11}s_{12} + a_{21}c_{12} = 0, \tag{4.5.9}$$

then A_1 has a zero in the $(2,1)$ position, and the other elements in the first two rows of A_1 differ, in general, from those of A. We do not actually compute the angle θ_{12} to achieve (4.5.9) since we can obtain the desired sine and cosine directly by

$$c_{12} = a_{11}(a_{11}^2 + a_{21}^2)^{-1/2}, \qquad s_{12} = a_{21}(a_{11}^2 + a_{21}^2)^{-1/2}. \tag{4.5.10}$$

The denominator in (4.5.10) is non-zero unless both a_{11} and a_{21} are zero; but if $a_{21} = 0$, this step can be bypassed since a zero is already in the desired position.

The transformation (4.5.8) is analogous to the first step of an LU factorization in which a multiple of the first row of A is subtracted from the second row to achieve a zero in the $(2, 1)$ position. We next proceed, as in LU, to obtain zeros in the remaining positions of the first column. We form $A_2 = P_{13}A$, which modifies the first and third rows of A_1 while leaving all other rows the same; in particular, the zero produced in the first stage in the $(2, 1)$ position remains unchanged. The elements c_{13} and s_{13} of P_{13} are chosen analogously to (4.5.10) so that the $(3, 1)$ element of A_2 is zero. We continue in this fashion, zeroing the remaining elements in the first column one after another and then zeroing elements in the second column in the order $(3, 2), (4, 2), \ldots, (n, 2)$, and so on. In all, we will use $(n - 1) + (n - 2) + \cdots + 1$ plane rotation matrices P_{ij}, and the result is that

$$PA \equiv P_{n-1,n} \cdots P_{12}A = R \tag{4.5.11}$$

is upper triangular.

We now need two basic facts about orthogonal matrices. First, if U and V are orthogonal, then

$$(UV)^T UV = V^T U^T UV = V^T V = I,$$

so that the product of orthogonal matrices is orthogonal. Each of the matrices P_{ij} in (4.5.11), and thus their product, P, is orthogonal. The second fact is that the inverse of an orthogonal matrix is orthogonal. This follows from the definition $U^T U = I$ since this implies that $I = (U^T U)^{-1} = U^{-1}U^{-T}$. Thus, if we set $Q = P^{-1}$, then Q is orthogonal, and multiplying (4.5.11) by Q gives (4.5.5).

We next count the operations to carry out the above QR factorization. The majority of the work is in modifying the elements of the two rows that are changed at each rotation. From (4.5.8) it is clear that modification of the first two rows requires $4n$ multiplications and $2n$ additions. (For simplicity we have also counted the operations used to produce the zero in the $(2, 1)$ position even though they do not need to be performed.) The same count is true for producing zeros in the remaining $n - 2$ elements in the first column. Hence the first stage requires $4n(n - 1)$ multiplications and $2n(n - 1)$ additions. In each of the subsequent $n - 2$ stages, n decreases by 1 in this count so that the total is

$$4\sum_{k=2}^{n} k(k-1) \text{ mult } + 2\sum_{k=2}^{n} k(k-1) \text{ add } \doteq \frac{4}{3}n^3 \text{ mult } + \frac{2}{3}n^3 \text{ add}, \tag{4.5.12}$$

where we have used the summation formulas of Exercise 4.2.4. Computation of the c_{ij} and s_{ij} is also needed, but this requires only $0(n^2)$ operations. Thus we see by (4.5.12) that this QR factorization requires approximately four times the multiplications and twice the additions of LU factorization (see Section 4.2). We will discuss the relative merits of QR and LU factorization shortly, but first we show that by using other orthogonal transformations the QR factorization can be computed more economically.

Householder Transformations

A *Householder tranfsormation* is a matrix of the form $I - 2\mathbf{w}\mathbf{w}^T$, where $\mathbf{w}^T\mathbf{w} = 1$. It is easy to see (Exercise 4.5.6) that such matrices are symmetric and orthogonal. They are also called *elementary reflection* matrices (see Exercise 4.5.6). We now show how Householder transformations can be used to obtain the QR factorization of A. Let $\mathbf{a}_1$ be the first column of A and define (see Exercise 4.5.11)

$$\mathbf{w}_1 = \mu_1\mathbf{u}_1, \quad \mathbf{u}^T = (a_{11} - s_1, a_{21}, \ldots, a_{n1}), \tag{4.5.13}$$

where

$$s_1 = \pm(\mathbf{a}_1^T\mathbf{a}_1)^{1/2}, \qquad \mu_1 = (2s_1^2 - 2a_{11}s_1)^{-1/2}. \tag{4.5.14}$$

The sign of s_1 must be chosen to be opposite that of a_{11} so that there is no cancellation in the computation of μ_1; otherwise, the algorithm would be numerically unstable. The vector $\mathbf{w}_1$ satisfies

$$\mathbf{w}_1^T\mathbf{w}_1 = \mu_1^2[(a_{11} - s_1)^2 + \sum_{j=2}^n a_{j1}^2] = \mu_1^2(a_{11}^2 - 2a_{11}s_1 + 2s_1^2 - a_{11}^2) = 1,$$

so that $P_1 = I - 2\mathbf{w}_1\mathbf{w}_1^T$ is a Householder transformation. Moreover,

$$\mathbf{w}_1^T\mathbf{a}_1 = \mu[(a_{11} - s_1)a_{21} + \sum_{j=2}^n a_{j1}^2] = \mu_1(s_1^2 - a_{11}s_1) = \frac{1}{2\mu_1},$$

so that

$$a_{11} - 2w_1\mathbf{w}_1^T\mathbf{a}_1 = a_{11} - \frac{2(a_{11} - s_1)\mu_1}{2\mu_1} = s_1,$$

and

$$a_{i1} - 2w_i\mathbf{w}_1^T\mathbf{a}_1 = a_{i1} - \frac{2a_{i1}\mu_1}{2\mu_1} = 0, \qquad i = 2, 3, \ldots, n.$$

This shows that the first column of P_1A is

$$P_1\mathbf{a}_1 = \mathbf{a}_1 - 2\mathbf{w}_1^T\mathbf{a}_1\mathbf{w}_1 = (s_1, 0, \ldots, 0)^T.$$

Thus with this one orthogonal transformation, zeros have been introduced into the subdiagonal positions of the first column, as was done by $n-1$ Givens transformations.

The second stage is analogous. A Householder transformation $P_2 = I - 2\mathbf{w}_2\mathbf{w}_2^T$ is defined by a vector $\mathbf{w}_2$ whose first component is zero and whose remaining components are defined as in (4.5.13, 4.5.14), using now the second column of P_1A from the main diagonal element down. The matrix P_2P_1A then has zeros below the main diagonal in each of its first two columns. We continue in this way to zero elements below the main diagonal by Householder matrices $P_i = I - 2\mathbf{w}_i\mathbf{w}_i^T$, where $\mathbf{w}_i$ has zeros in its first $i-1$ components. Thus

$$P_{n-1}\cdots P_1A = R,$$

where R is upper triangular. The matrices P_i are all orthogonal so that $P = P_{n-1}\cdots P_1$ and P^{-1} are also orthogonal. Therefore, $Q = P^{-1}$ is the orthogonal matrix of (4.5.5).

The above discussion has considered only the formation of the vectors $\mathbf{w}_i$ that define the Householder transformations, and we next consider the remainder of the computation. If $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ are the columns of A, then

$$P_1A = A - 2\mathbf{w}_1\mathbf{w}_1^TA = A - 2\mathbf{w}_1(\mathbf{w}_1^T\mathbf{a}_1, \mathbf{w}_1^T\mathbf{a}_2, \ldots, \mathbf{w}_1^T\mathbf{a}_n). \tag{4.5.15}$$

Thus, the ith column of P_1A is

$$\mathbf{a}_i - 2\mathbf{w}_1^T\mathbf{a}_i\mathbf{w}_1 = \mathbf{a}_i - \gamma_1\mathbf{u}_1^T\mathbf{a}_i\mathbf{u}_1, \tag{4.5.16}$$

where

$$\gamma_1 = 2\mu_1^2 = (s_1^2 - s_1a_{11})^{-1}.$$

It is more efficient computationally not to form the vector $\mathbf{w}_1$ explicitly but to work with γ_1 and $\mathbf{u}_1$ as shown in (4.5.16). Analogous computations are performed to obtain the remaining reduced matrices $P_2P_1A, \ldots$; the complete algorithm is summarized in Figure 4.6.

We next count the operations in the Householder reduction. The bulk of the work is in the formation of the new columns of the reduced matrices. Referring to the inner loop in Figure 4.6, at the kth stage the inner product $\mathbf{u}_k^T\mathbf{a}_j$ requires $n-k+1$ multiplications and $n-k$ additions and the operation $\mathbf{a}_j - \alpha_j\mathbf{u}_k$ requires $n-k+1$ additions and multiplications. Since there are $n-k$ columns to update at the kth stage, this gives approximately $2(n-k)$ additions and multiplications. Summing this over all $n-1$ stages, we obtain

$$2\sum_{k=1}^{n-1}(n-k)^2 = \frac{n(n-1)(2n-1)}{3} \doteq \frac{2}{3}n^3 \tag{4.5.17}$$

additions and multiplications. The number of other operations is no more than order n^2. Comparing this count with (4.5.12) for the QR factorization using