2

Sensitivity of Linear Systems; Effects of Roundoff Errors

When we solve a system of linear equations, we seldom solve exactly the system we intended to solve; rather we solve one that approximates it. In a system $Ax = b$, the coefficients in A and b will typically be known from some measurements and will therefore be subject to measurement error. For example, in the electrical circuit problems introduced in Section 1.2, the entries of the coefficient matrices depend on the values of the resistances, numbers that are known only approximately in practice. Thus the A and b with which we work are slightly different from the true A and b. Additional approximations must be made when the numbers are entered into a computer; the real or complex entries of A and b must be approximated by numbers from the computer's finite set of floating point numbers. (However, this error is usually much smaller than the measurement error.)

Because errors are inevitable, it is important to ask what effect small perturbations in the coefficients have on the solution of a system. When we change the system slightly, does this cause only a slight change in the solution or an enormous, unacceptable change? As we shall see, not all systems are alike in this regard; some are much more sensitive than others. The most important task of this chapter is to study the *sensitivity* of linear systems.

A second question is one that was already mentioned in Chapter 1. If we solve a system by Gaussian elimination on a computer, the result will be contaminated by the roundoff errors made during the computation. How do these errors affect the accuracy of the computed solution? We shall see that this question is closely related to the sensitivity issue.

The most comprehensive work on this subject is N. J. Higham's book, *Accuracy and Stability of Numerical Algorithms* [41].

Fundamentals of Matrix Computations, Second Edition. By David S. Watkins
 ISBN: 0-471-21394-2

2.1 VECTOR AND MATRIX NORMS

In order to study the effects of perturbations in vectors (such as b) and matrices (such as A), we need to be able to measure them. For this purpose we introduce vector and matrix norms.

Vector Norms

The vectors used in this book are generally n-tuples of real (or complex) numbers. Recall that the set of all real n-tuples is denoted $\mathbb{R}^n$. It is useful to visualize the members of $\mathbb{R}^2$ as points in a plane or as geometric vectors (arrows) in a plane with their tails at the origin. Likewise the elements of $\mathbb{R}^3$ can be viewed as points or vectors in space. Any two elements of $\mathbb{R}^n$ can be added in the obvious manner to yield an element of $\mathbb{R}^n$, and any element of $\mathbb{R}^n$ can be multiplied by any real number (scalar) to yield an element of $\mathbb{R}^n$. The vector whose components are all zero will be denoted 0. Thus the symbol 0 can stand for either a number or a vector. The careful reader will not be confused by this.

The set of all n-tuples of complex numbers is denoted $\mathbb{C}^n$. In this chapter, as in Chapter 1, we will restrict our attention to real numbers. However, everything that we do here can be carried over to complex numbers.

A *norm* (or *vector norm*) on $\mathbb{R}^n$ is a function that assigns to each $x \in \mathbb{R}^n$ a non-negative real number $\|x\|$, called the norm of x, such that the following three properties are satisfied for all x, $y \in \mathbb{R}^n$ and all $\alpha \in \mathbb{R}$:

$$\|x\| > 0 \text{ if } x \neq 0, \quad \text{and} \quad \|0\| = 0 \qquad \text{(positive definite property)} \quad (2.1.1)$$

$$\|\alpha x\| = |\alpha| \, \|x\| \qquad \text{(absolute homogeneity)} \quad (2.1.2)$$

$$\|x + y\| \leq \|x\| + \|y\| \qquad \text{(triangle inequality)} \quad (2.1.3)$$

Exercise 2.1.4

(a) In the equation $\|0\| = 0$, what it the nature of each zero (number or vector)?

(b) Show that the equation $\|0\| = 0$ actually follows from (2.1.2). Thus it need not have been stated explicitly in (2.1.1).

□

Any norm can be used to measure the lengths or magnitudes (in a generalized sense) of vectors in $\mathbb{R}^n$. In other words, we think of $\|x\|$ as the (generalized) *length* of x. The (generalized) *distance* between two vectors x and y is $\|x - y\|$.

Example 2.1.5 The *Euclidean norm* is defined by

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}.$$

You can easily verify that this function satisfies (2.1.1) and (2.1.2). The triangle inequality (2.1.3) is not so easy. It follows from the Cauchy-Schwarz inequality, which we will prove shortly (Theorem 2.1.6). The Euclidean distance between two vectors x and y is given by

$$\|x - y\|_2 = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}.$$

In the cases $n = 1, 2$, and 3, this measure coincides with our usual notion of distance between points in a line, in a plane, or in space, respectively.

Notice that the absolute value signs in the formula for $\|x\|_2$ are redundant, as $|x_i|^2 = x_i^2$ for any real number x_i. However, for complex numbers it is not generally true that $|x_i|^2 = x_i^2$, and the absolute value signs would be needed. Thus the inclusion of the absolute value signs gives a formula for $\|x\|_2$ that is correct for both real and complex vectors. □

Theorem 2.1.6 *(Cauchy-Schwarz inequality) For all* $x, y \in \mathbb{R}^n$

$$\left|\sum_{i=1}^{n} x_i y_i\right| \leq \left(\sum_{i=1}^{n} x_i^2\right)^{1/2} \left(\sum_{i=1}^{n} y_i^2\right)^{1/2}.$$

Proof. For every real number t we have

$$\begin{aligned} 0 \leq \sum_{i=1}^{n} (x_i + t y_i)^2 &= \sum_{i=1}^{n} x_i^2 + 2t \sum_{i=1}^{n} x_i y_i + t^2 \sum_{i=1}^{n} y_i^2 \\ &= c + bt + at^2, \end{aligned}$$

where $a = \sum_{i=1}^{n} y_i^2$, $b = 2\sum_{i=1}^{n} x_i y_i$, and $c = \sum_{i=1}^{n} x_i^2$. Since $at^2 + bt + c \geq 0$ for all real t, the quadratic polynomial $at^2 + bt + c$ cannot have two distinct real zeros. Therefore the discriminant satisfies $b^2 - 4ac \leq 0$. Rewriting this inequality as

$$(b/2)^2 \leq ac$$

and taking square roots, we obtain the desired result. □

Now we are ready to prove that the triangle inequality holds for the Euclidean norm. Thus the Euclidean norm is indeed a norm.

Theorem 2.1.7 *For all* $x, y \in \mathbb{R}^n$, $\|x + y\|_2 \leq \|x\|_2 + \|y\|_2$.

Proof. It suffices to show that $\|x + y\|_2^2 \leq \|x\|_2^2 + \|y\|_2^2$.

$$\|x + y\|_2^2 = \sum_{i=1}^{n} (x_i + y_i)^2 = \sum_{i=1}^{n} x_i^2 + 2\sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} y_i^2.$$

Applying the Cauchy-Schwarz inequality to the middle term on the right-hand side, we find that

$$\begin{aligned} \|x+y\|^2 &\leq \sum_{i=1}^{n} x_i^2 + 2\left(\sum_{i=1}^{n} x_i^2\right)^{1/2}\left(\sum_{i=1}^{n} y_i^2\right)^{1/2} + \sum_{i=1}^{n} y_i^2 \\ &= \left[\left(\sum_{i=1}^{n} x_i^2\right)^{1/2} + \left(\sum_{i=1}^{n} y_i^2\right)^{1/2}\right]^2 \\ &= (\|x\|_2 + \|y\|_2)^2. \end{aligned}$$

□

Example 2.1.8 Generalizing Example 2.1.5, we introduce the p-norms. For any real number $p \geq 1$, we define

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

Again it is easy to verify (2.1.1) and (2.1.2), but not (2.1.3). This is the *Minkowski Inequality*, which we will not prove because we are not going to use it. □

The most important p-norm is the 2-norm, which is just the Euclidean norm.

Example 2.1.9 Another important p-norm is the 1-norm

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|.$$

In this case (2.1.3) is not hard to prove; it follows directly from the triangle inequality for real numbers. □

Exercise 2.1.10 Prove that the 1-norm is a norm. □

Exercise 2.1.11 Let $x, y \in \mathbb{R}^2$. With respect to the 1-norm, the "distance" between x and y is $\|x-y\|_1 = |x_1 - y_1| + |x_2 - y_2|$. Explain why the 1-norm is sometimes called the *taxicab norm* (or *Manhattan metric*). □

Example 2.1.12 The ∞-norm is defined by

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|.$$

□

Exercise 2.1.13 Prove that the ∞-norm is a norm. □

Exercise 2.1.14 Given any norm on $\mathbb{R}^2$, the *unit circle* with respect to that norm is the set $\{x \in \mathbb{R}^2 \mid \|x\| = 1\}$. Thinking of the members of $\mathbb{R}^2$ as points in the plane,

the unit circle is just the set of points whose distance from the origin is 1. On a single set of coordinate axes, sketch the unit circle with respect to the p-norm for $p = 1, 3/2, 2, 3, 10$, and ∞. □

The analytically inclined reader might like to prove that for all $x \in \mathbb{R}^n$, $\|x\|_\infty = \lim_{p\to\infty} \|x\|_p$.

Example 2.1.15 Given a positive definite matrix $A \in \mathbb{R}^{n\times n}$, define the A-norm on $\mathbb{R}^n$ by

$$\|x\|_A = (x^T A x)^{1/2}.$$

In Exercise 2.1.17 you will show that this is indeed a norm. □

Exercise 2.1.16 Show that when $A = I$, the A-norm is just the Euclidean norm. □

Exercise 2.1.17

(a) Let A be a positive definite matrix, and let R be its Cholesky factor, so that $A = R^T R$. Verify that for all $x \in \mathbb{R}^n$, $\|x\|_A = \|Rx\|_2$.

(b) Using the fact that the 2-norm is indeed a norm on $\mathbb{R}^n$, prove that the A-norm is a norm on $\mathbb{R}^n$.

□

Matrix Norms

The set of real $m \times n$ matrices is denoted $\mathbb{R}^{m\times n}$. Like vectors, the matrices in $\mathbb{R}^{m\times n}$ can be added and multiplied by scalars in the obvious manner. In fact the matrices in $\mathbb{R}^{m\times n}$ can be viewed simply as vectors in $\mathbb{R}^{mn}$ with the components arranged differently. In the case $m = n$, the theory becomes richer. Unlike ordinary vectors, two matrices in $\mathbb{R}^{n\times n}$ can be multiplied together (using the usual matrix multiplication) to yield a product in $\mathbb{R}^{n\times n}$. A *matrix norm* is a function that assigns to each $A \in \mathbb{R}^{n\times n}$ a real number $\|A\|$, called the norm of A, such that the three vector norm properties hold, as well as one additional property, *submultiplicativity*, which relates the norm function to the operation of matrix multiplication. Specifically, for all $A, B \in \mathbb{R}^{n\times n}$ and all $\alpha \in \mathbb{R}$,

$$\|A\| > 0 \text{ if } A \neq 0 \tag{2.1.18}$$

$$\|\alpha A\| = |\alpha| \|A\| \tag{2.1.19}$$

$$\|A + B\| \leq \|A\| + \|B\| \tag{2.1.20}$$

$$\|AB\| \leq \|A\| \|B\| \quad \text{(submultiplicativity)} \tag{2.1.21}$$

Example 2.1.22 The *Frobenius norm* is defined by

$$\|A\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}.$$

Because it is the same as the Euclidean norm on vectors, we already know that it satisfies (2.1.18), (2.1.19), and (2.1.20). The submultiplicativity (2.1.21) can be deduced from the Cauchy-Schwarz inequality as follows. Let $C = AB$. Then $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. Thus

$$\| AB \|_F^2 = \| C \|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} | c_{ij} |^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \left| \sum_{k=1}^{n} a_{ik} b_{kj} \right|^2 .$$

Applying the Cauchy-Schwarz inequality to the expression $\sum_{k=1}^{n} a_{ik} b_{kj}$, we find that

$$\begin{aligned} \| AB \|_F^2 &\leq \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\sum_{k=1}^{n} | a_{ik}^2 | \sum_{k=1}^{n} | b_{kj}^2 | \right) \\ &= \left(\sum_{i=1}^{n} \sum_{k=1}^{n} | a_{ik} |^2 \right) \left(\sum_{j=1}^{n} \sum_{k=1}^{n} | b_{kj} |^2 \right) \\ &= \| A \|_F^2 \| B \|_F^2 . \end{aligned}$$

Thus the Frobenius norm is a matrix norm. □

Exercise 2.1.23 Define $\| A \|_{\max} = \max_{1 \leq i, j \leq n} | a_{ij} |$. Clearly this function satisfies (2.1.18), (2.1.19), and (2.1.20). Show by example that it violates (2.1.21) and is therefore not a matrix norm. □

Every vector norm on $\mathbb{R}^n$ can be used to define a matrix norm on $\mathbb{R}^{n \times n}$ in a natural way. Given a vector norm $\| \cdot \|_v$, the matrix norm *induced* by $\| \cdot \|_v$ is defined by

$$\| A \|_M = \max_{x \neq 0} \frac{\| Ax \|_v}{\| x \|_v} .$$

Theorem 2.1.26 will show that the induced norm is indeed a matrix norm. Another name for the induced norm is the *operator norm*.

The induced norm has geometric significance that can be understood by viewing A as a linear operator that maps $\mathbb{R}^n$ into $\mathbb{R}^n$: Each $x \in \mathbb{R}^n$ is mapped by A to the vector $Ax \in \mathbb{R}^n$. The ratio $\| Ax \|_v / \| x \|_v$ is the magnification that takes place when x is transformed to Ax. The number $\| A \|_M$ is then the maximum magnification that A can cause.

It is a common practice not to use distinguishing suffixes v and M. One simply uses the same symbol for both the vector norm and the matrix norm and writes, for example,

$$\| A \| = \max_{x \neq 0} \frac{\| Ax \|}{\| x \|} .$$

We will adopt this practice. This need not lead to confusion, because the meaning of the norm can always be deduced from the context.

Before proving that the induced norm is indeed a matrix norm, it is useful to make note of the following simple but important fact.

Theorem 2.1.24 *A vector norm and its induced matrix norm satisfy the inequality*

$$\|Ax\| \leq \|A\| \|x\| \tag{2.1.25}$$

for all $A \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$. This inequality is sharp in the following sense. For all $A \in \mathbb{R}^{n\times n}$ there exists a nonzero $x \in \mathbb{R}^n$ for which equality holds.

Proof. If $x = 0$, equality holds trivially. Otherwise

$$\frac{\|Ax\|}{\|x\|} \leq \max_{\hat{x}\neq 0} \frac{\|A\hat{x}\|}{\|\hat{x}\|} = \|A\|.$$

Thus $\|Ax\| \leq \|A\| \|x\|$. Equality holds if and only if x is a vector for which the maximum magnification is attained. (That such a vector exists is actually not obvious. It follows from a compactness argument that works because $\mathbb{R}^n$ is a finite-dimensional space. We omit the argument.) □

The fact that equality is attained in (2.1.25) is actually less important than the (obvious) fact that there exist vectors for which equality is approached as closely as one pleases.

Theorem 2.1.26 *The induced norm is a matrix norm.*

Proof. The proof is not particularly difficult. You are encouraged to provide one of your own before reading on. Only the submultiplicativity property (2.1.21) depends upon (2.1.25).

Each of the first three norm properties follows from the corresponding property of the vector norm. To prove (2.1.18), we must show that $\|A\| > 0$ if $A \neq 0$. If $A \neq 0$, there exists a (nonzero) vector $\hat{x} \in \mathbb{R}^n$ for which $A\hat{x} \neq 0$. Since the vector norm satisfies (2.1.1), we have $\|A\hat{x}\| > 0$ and $\|\hat{x}\| > 0$. Thus

$$\|A\| = \max_{x\neq 0} \frac{\|Ax\|}{\|x\|} \geq \frac{\|A\hat{x}\|}{\|\hat{x}\|} > 0.$$

To prove (2.1.19), we note that for every $x \in \mathbb{R}^n$ and every $\alpha \in \mathbb{R}$, $\|\alpha(Ax)\| = |\alpha| \|Ax\|$. This is because the vector norm satisfies (2.1.2). (*Remember:* Ax is a vector, not a matrix.) Thus

$$\begin{aligned} \|\alpha A\| &= \max_{x\neq 0} \frac{\|(\alpha A)x\|}{\|x\|} = \max_{x\neq 0} \frac{\|\alpha(Ax)\|}{\|x\|} = \max_{x\neq 0} \frac{|\alpha| \|Ax\|}{\|x\|} \\ &= |\alpha| \max_{x\neq 0} \frac{\|Ax\|}{\|x\|} = |\alpha| \|A\|. \end{aligned}$$

Applying similar ideas we prove (2.1.20).

$$\begin{aligned} \|A+B\| &= \max_{x\neq 0} \frac{\|(A+B)x\|}{\|x\|} = \max_{x\neq 0} \frac{\|Ax+Bx\|}{\|x\|} \\ &\leq \max_{x\neq 0} \frac{\|Ax\| + \|Bx\|}{\|x\|} \leq \max_{x\neq 0} \frac{\|Ax\|}{\|x\|} + \max_{x\neq 0} \frac{\|Bx\|}{\|x\|} \\ &= \|A\| + \|B\|. \end{aligned}$$

Finally we prove (2.1.21). Replacing x by Bx in (2.1.25), we have $\|ABx\| \leq \|A\|\,\|Bx\|$ for any x. Applying (2.1.25) again, we have $\|Bx\| \leq \|B\|\,\|x\|$. Thus

$$\|ABx\| \leq \|A\|\,\|B\|\,\|x\|.$$

For nonzero x we can divide both sides by the positive number $\|x\|$ and conclude that

$$\|AB\| = \max_{x\neq 0} \frac{\|ABx\|}{\|x\|} \leq \|A\|\,\|B\|.$$

□

Exercise 2.1.27

(a) Show that for any nonzero vector x and scalar c, $\|A(cx)\|/\|cx\| = \|Ax\|/\|x\|$. Thus rescaling a vector does not change the amount by which it is magnified under multiplication by A. In geometric terms, the magnification undergone by x depends only on its direction, not on its length.

(b) Prove that the induced matrix norm satisfies

$$\|A\| = \max_{\|x\|=1} \|Ax\|.$$

This alternative characterization is often useful.

□

Some of the most important matrix norms are induced by p-norms. For $1 \leq p \leq \infty$, the norm induced by the p-norm is called the *matrix p-norm*:

$$\|A\|_p = \max_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

The matrix 2-norm is also known as the *spectral norm*. As we shall see in subsequent chapters, this norm has great theoretical importance. Its drawback is that it is expensive to compute; it is *not* the Frobenius norm.

Exercise 2.1.28

(a) Calculate $\|I\|_F$ and $\|I\|_2$, where I is the $n \times n$ identity matrix, and notice that they are different.

(b) Use the Cauchy-Schwarz inequality (Theorem 2.1.6) to show that for all $A \in \mathbb{R}^{n\times n}$, $\|A\|_2 \le \|A\|_F$.

□

Other important cases are $p = 1$ and $p = \infty$. These norms can be computed easily.

Theorem 2.1.29 *(a)* $\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|$. *(b)* $\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$.

Thus $\|A\|_1$ is found by summing the absolute values of the entries in each column of A and then taking the largest of these column sums. Therefore the matrix 1-norm is sometimes called the *column-sum norm*. Similarly, the matrix ∞-norm is called the *row-sum norm*.

Proof. We will prove part (a), leaving part (b) as an exercise. We first show that $\|A\|_1 \le \max_j \sum_{i=1}^{n} |a_{ij}|$. For all $x \in \mathbb{R}^n$,

$$\begin{aligned}
\|Ax\|_1 &= \sum_{i=1}^{n} |(Ax)_i| = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} a_{ij}x_j \right| \le \sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|\,|x_j| \\
&= \sum_{j=1}^{n}\sum_{i=1}^{n} |a_{ij}|\,|x_j| \le \sum_{j=1}^{n} \left(\max_k \sum_{i=1}^{n} |a_{ik}| \right) |x_j| \\
&= \left(\max_k \sum_{i=1}^{n} |a_{ik}| \right) \left(\sum_{j=1}^{n} |x_j| \right) = \left(\max_k \sum_{i=1}^{n} |a_{ik}| \right) \|x\|_1 .
\end{aligned}$$

Therefore $\|Ax\|_1/\|x\|_1 \le \max_k \sum_{i=1}^{n} |a_{ik}|$ for all $x \ne 0$. From this $\|A\|_1 \le \max_k \left(\sum_{i=1}^{n} |a_{ik}| \right)$. To prove equality, we must simply find an $\hat{x} \in \mathbb{R}^n$ for which

$$\frac{\|A\hat{x}\|_1}{\|\hat{x}\|_1} = \max_k \left(\sum_{i=1}^{n} |a_{ik}| \right).$$

Suppose that the maximum is attained in the mth column of A. Let $\hat{x}$ be the vector with a 1 in position m and zeros elsewhere. Then $\|\hat{x}\|_1 = 1$, $A\hat{x} = [a_{1m}\ a_{2m}\ \cdots\ a_{nm}]^T$, and $\|A\hat{x}\|_1 = \sum_{i=1}^{n} |a_{im}|$. Thus

$$\frac{\|A\hat{x}\|_1}{\|\hat{x}\|_1} = \sum_{i=1}^{n} |a_{im}| = \max_j \left(\sum_{i=1}^{n} |a_{ij}| \right).$$

□

Exercise 2.1.30 Prove part (b) of Theorem 2.1.29. (The argument is generally similar to that of the proof of part (a), but your special vector $\hat{x}$ should be chosen to have components ± 1, with the sign of each component chosen carefully.) □

Additional Exercises

Exercise 2.1.31 Show that for all $x \in \mathbb{R}^n$

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2 \leq n\|x\|_\infty.$$

The two outer inequalities are fairly obvious. The inequality $\|x\|_2 \leq \|x\|_1$ becomes obvious on squaring both sides. the inequality $\|x\|_1 \leq \sqrt{n}\|x\|_2$ is obtained by applying the Cauchy-Schwarz inequality (Theorem 2.1.6) to the vectors x and $y = [1, 1, \cdots, 1]^T$. □

Exercise 2.1.32 Make systematic use of the inequalities from Exercise 2.1.31 to prove that for all $A \in \mathbb{R}^{n\times n}$

$$\|A\|_1 \leq \sqrt{n}\|A\|_2 \leq n\|A\|_1$$

and

$$\|A\|_\infty \leq \sqrt{n}\|A\|_2 \leq n\|A\|_\infty.$$

□

2.2 CONDITION NUMBERS

In this section we introduce and discuss the *condition number* of a matrix A. This is a simple but useful measure of the sensitivity of the linear system $Ax = b$.

Consider a linear system $Ax = b$, where A is nonsingular and b is nonzero. There is a unique solution x, which is nonzero. Now suppose we add a small vector δb to b and consider the perturbed system $A\hat{x} = b + \delta b$. This system also has a unique solution $\hat{x}$, which is hoped to be not too far from x. Let δx denote the difference between $\hat{x}$ and x, so that $\hat{x} = x + \delta x$. We would like to say that if δb is small, then δx is also small. A more precise statement would involve relative terms: when we say that δb is small, we really mean that it is small in comparison with b; when we say δx is small, we really mean small compared with x. In order to quantify the size of vectors, we introduce a vector norm $\|\cdot\|$. The size of δb relative to b is then given by $\|\delta b\|/\|b\|$, and the size of δx relative to x is given by $\|\delta x\|/\|x\|$. We would like to say that if $\|\delta b\|/\|b\|$ is small, then $\|\delta x\|/\|x\|$ is also small.

The equations $Ax = b$ and $A(x + \delta x) = b + \delta b$ imply that $A\delta x = \delta b$, that is, $\delta x = A^{-1}\delta b$. Whatever vector norm we have chosen, we will use the induced matrix norm to measure matrices. The equation $\delta x = A^{-1}\delta b$ and Theorem 2.1.24 imply that

$$\|\delta x\| \leq \|A^{-1}\| \, \|\delta b\|. \tag{2.2.1}$$

Similarly the equation $b = Ax$ and Theorem 2.1.24 imply $\|b\| \leq \|A\| \, \|x\|$, or equivalently

$$\frac{1}{\|x\|} \leq \|A\| \frac{1}{\|b\|}. \tag{2.2.2}$$

Multiplying inequality (2.2.1) by (2.2.2), we arrive at

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\| \, \|A^{-1}\| \frac{\|\delta b\|}{\|b\|}, \tag{2.2.3}$$

which provides a bound for $\|\delta x\|/\|x\|$ in terms of $\|\delta b\|/\|b\|$. The factor $\|A\| \, \|A^{-1}\|$ is called the *condition number* of A and denoted $\kappa(A)$ [72]. That is,

$$\kappa(A) = \|A\| \, \|A^{-1}\|.$$

With this new notation, we rephrase (2.2.3) as the conclusion of a theorem.

Theorem 2.2.4 *Let A be nonsingular, and let x and $\hat{x} = x + \delta x$ be the solutions of $Ax = b$ and $A\hat{x} = b + \delta b$, respectively. Then*

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\delta b\|}{\|b\|}. \tag{2.2.5}$$

Since (2.2.1) and (2.2.2) are sharp, (2.2.5) is also sharp; that is, there exist b and δb (and associated x and δx) for which equality holds in (2.2.5).

Exercise 2.2.6

(a) Show that $\kappa(A) = \kappa(A^{-1})$.

(b) Show that for any nonzero scalar c, $\kappa(cA) = \kappa(A)$.

□

From (2.2.5) we see that if $\kappa(A)$ is not too large, then small values of $\|\delta b\|/\|b\|$ imply small values of $\|\delta x\|/\|x\|$. That is, the system is not overly sensitive to perturbations in b. Thus if $\kappa(A)$ is not too large, we say that A is *well conditioned.* If, on the other hand, $\kappa(A)$ is large, a small value of $\|\delta b\|/\|b\|$ does not guarantee that $\|\delta x\|/\|x\|$ will be small. Since (2.2.5) is sharp, we know that in this case there definitely are choices of b and δb for which the resulting $\|\delta x\|/\|x\|$ is much larger than the resulting $\|\delta b\|/\|b\|$. In other words, the system is potentially very sensitive to perturbations in b. Thus if $\kappa(A)$ is large, we say that A is *ill conditioned.*

Proposition 2.2.7 *For any induced matrix norm, (a) $\|I\| = 1$ and (b) $\kappa(A) \geq 1$.*

Proof. Part (a) follows immediately from the definition of the induced matrix norm. To prove part (b), we note that $I = AA^{-1}$, so $1 = \|I\| = \|AA^{-1}\| \leq \|A\| \, \|A^{-1}\| = \kappa(A)$. □

Thus the best possible condition number is 1. Of course the condition number depends on the choice of norm. While it is possible to concoct bizarre norms such

that a given matrix has a large condition number with respect to one norm and a small condition number with respect to another, we will use mainly the 1-, 2-, and ∞-norms, which typically give roughly comparable values for the condition numbers of matrices. We will use the notation

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p \quad \text{for } 1 \leq p \leq \infty.$$

So far we have said that a matrix that has a large condition number is ill conditioned, but we have not said anything about where the cutoff line between well-conditioned and ill-conditioned matrices lies. Of course there is no point in looking for a precise boundary. Furthermore the (fuzzy) boundary depends upon a number of factors, including the accuracy of the data being used, the precision of the floating point numbers, and the amount of error we are willing to tolerate in our computed solution. Suppose, for example, that the components of b are correct to about four decimal places. We do not know the exact value of b; in the computation we actually use $b + \delta b$, where $\|\delta b\|/\|b\| \approx 10^{-4}$. If we solve the problem accurately, we get not x but $x + \delta x$, where an upper bound for $\|\delta x\|/\|x\|$ is given by (2.2.5).

Now suppose $\kappa(A) \leq 10^2$. Then by (2.2.5) the worst that can happen is $\|\delta x\|/\|x\| \approx 10^{-2}$. That is, the error in x is not bigger than about one hundredth the size of x. In many problems, this much error in the solution is acceptable. By contrast, if $\kappa(A) \approx 10^4$, then (2.2.5) tells us that it can happen that $\|\delta x\|/\|x\| \approx 1$; that is, the error could be as big as the solution itself. In this case we would have to say that the condition number is unacceptably high. Thus it appears that in this problem the boundary between well-conditioned and ill-conditioned matrices lies somewhere in the range 10^2 to 10^4.

Sometimes the accuracy of the floating point arithmetic can be the deciding factor. It may be that we know b with extreme accuracy, but if numbers are only stored to about seven decimal places accuracy in the computer, then we will be forced to work with $b + \delta b$, where $\|\delta b\|/\|b\| \approx 10^{-7}$. Then if we have $\kappa(A) \approx 10^7$, we cannot be sure to get a reasonable answer, even if we solve the system very accurately. On the other hand a condition number of 10^3, 10^4, or even 10^5 may be small enough, depending on how accurate we require the solution to be.

We are overdue for an example of an ill-conditioned matrix.

Example 2.2.8 Let $A = \begin{bmatrix} 1000 & 999 \\ 999 & 998 \end{bmatrix}$. You can easily verify that

$$A^{-1} = \begin{bmatrix} -998 & 999 \\ 999 & -1000 \end{bmatrix}.$$

Thus $\|A\|_\infty = \|A\|_1 = 1999 = \|A^{-1}\|_\infty = \|A^{-1}\|_1$, and

$$\kappa_\infty(A) = \kappa_1(A) = (1999)^2 = 3.996 \times 10^6.$$

The process of computing $\kappa_2(A)$ is more involved; we are not yet ready to describe it. However, on this small matrix, MATLAB can easily do the job. Using the command `cond(A)` or `cond(A,2)`, we find that $\kappa_2(A) \approx 3.992 \times 10^6$.

This matrix would be considered ill conditioned by most standards. We will discuss it further in Example 2.2.15. □

Example 2.2.9 The the most famous examples of ill conditioned matrices are the *Hilbert matrices*, defined by $h_{ij} = 1/(i + j - 1)$. If we let H_n denote the $n \times n$ Hilbert matrix, then

$$H_4 = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix},$$

for example. These matrices are symmetric, can be shown to be positive definite, and are increasingly ill conditioned as n is increased. For example, according to MATLAB, $\kappa_2(H_4) \approx 1.6 \times 10^4$ and $\kappa_2(H_8) \approx 1.5 \times 10^{10}$. □

Exercise 2.2.10 In MATLAB you can type `A = hilb(7)` to get the 7×7 Hilbert matrix, for example. Type `help cond` to find out how to use MATLAB's condition number function. Use it to calculate $\kappa_1(H_n)$, $\kappa_2(H_n)$, and $\kappa_\infty(H_n)$ for $n = 3, 6, 9$, and 12. □

Geometric Interpretation of the Condition Number

We begin by introducing some new terms. The *maximum* and *minimum magnification* by A are defined by

$$\text{maxmag}(A) = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

and

$$\text{minmag}(A) = \min_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

Of course, maxmag(A) is nothing but the induced matrix norm $\|A\|$.

Exercise 2.2.11 Prove that if A is a nonsingular matrix, then

$$\text{maxmag}(A) = \frac{1}{\text{minmag}(A^{-1})} \quad \text{and} \quad \text{maxmag}(A^{-1}) = \frac{1}{\text{minmag}(A)}.$$

□

From this exercise it follows easily that $\kappa(A)$ is just the ratio of the maximum magnification to the minimum magnification.

Proposition 2.2.12 $\kappa(A) = \dfrac{\text{maxmag}(A)}{\text{minmag}(A)}$ *for all nonsingular* A.

Exercise 2.2.13 Prove Proposition 2.2.12. □

An ill-conditioned matrix is one for which the maximum magnification is much larger than the minimum magnification.

If the matrix A is nonzero but singular, then there exists $x \neq 0$ such that $Ax = 0$. Thus $\text{minmag}(A) = 0$, and it is reasonable to say that $\kappa(A) = \infty$. That is, we view singularity as the extreme case of ill conditioning. Reversing the viewpoint, we can say that an ill-conditioned matrix is one that is "nearly" singular.

Since a matrix A is singular if and only if $\det(A) = 0$, it is natural to expect that the determinant is somehow connected to the condition number. This turns out to be wrong. There is no useful relationship between the condition number and the determinant. See Exercise 2.2.14. When it comes to assessing sensitivity of linear systems, the condition number is useful and the determinant is not.

Exercise 2.2.14

(a) Let α be a positive number, and define

$$A_\alpha = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix}.$$

Show that for any induced matrix norm we have $\|A_\alpha\| = \alpha$, $\|A_\alpha^{-1}\| = 1/\alpha$, and $\kappa(A_\alpha) = 1$. Thus A_α is well conditioned. On the other hand, $\det(A_\alpha) = \alpha^2$, so we can make it as large or small as we please by choosing α appropriately.

(b) More generally, given any nonsingular matrix A, discuss the condition number and determinant of αA, where α is any positive real number.

□

Example 2.2.15 Let us take another look at the ill-conditioned matrices

$$A = \begin{bmatrix} 1000 & 999 \\ 999 & 998 \end{bmatrix} \quad \text{and} \quad A^{-1} = \begin{bmatrix} -998 & 999 \\ 999 & -1000 \end{bmatrix}$$

from Example 2.2.8. Notice that

$$A \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1999 \\ 1997 \end{bmatrix}. \tag{2.2.16}$$

If we use the ∞-norm to measure lengths, the magnification factor $\|Ax\|_\infty / \|x\|_\infty$ is 1999, which equals $\|A\|_\infty$. Thus $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is a vector that is magnified maximally by A. Since the amount by which a vector is magnified depends only on its direction and not on its length, we say that $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is in a *direction of maximum magnification* by A. Equivalently we can say that $\begin{bmatrix} 1999 \\ 1997 \end{bmatrix}$ lies in a *direction of minimum magnification*

by A^{-1}. Looking now at A^{-1}, we note that

$$A^{-1}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1997 \\ -1999 \end{bmatrix}.$$

The magnification factor $\|A^{-1}x\|_\infty / \|x\|_\infty$ is 1999, which equals $\|A^{-1}\|_\infty$, so $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ is in a direction of maximum magnification by A^{-1}. Equivalently

$$A\begin{bmatrix} 1997 \\ -1999 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \tag{2.2.17}$$

and $\begin{bmatrix} 1997 \\ -1999 \end{bmatrix}$ is in a direction of minimum magnification by A. We will use the vectors in (2.2.16) and (2.2.17) to construct a spectacular example.

Suppose we wish to solve the system

$$\begin{bmatrix} 1000 & 999 \\ 999 & 998 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1999 \\ 1997 \end{bmatrix}, \tag{2.2.18}$$

that is, $Ax = b$, where $b = \begin{bmatrix} 1999 \\ 1997 \end{bmatrix}$. Then by (2.2.16) the unique solution is $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Now suppose that we solve instead the slightly perturbed system

$$\begin{bmatrix} 1000 & 999 \\ 999 & 998 \end{bmatrix}\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 1998.99 \\ 1997.01 \end{bmatrix}. \tag{2.2.19}$$

This is $A\hat{x} = b + \delta b$, where $\delta b = \begin{bmatrix} -.01 \\ .01 \end{bmatrix} = .01\begin{bmatrix} -1 \\ 1 \end{bmatrix}$, which is in a direction of maximum magnification by A^{-1}. By (2.2.17), $A\delta x = \delta b$, where $\delta x = \begin{bmatrix} 19.97 \\ -19.99 \end{bmatrix}$. Therefore $\hat{x} = x + \delta x = \begin{bmatrix} 20.97 \\ -18.99 \end{bmatrix}$. Thus the nearly identical problems (2.2.18) and (2.2.19) have very different solutions. □

It is important to recognize that this example was concocted in a special way. The vector b was chosen to be in a direction of minimum magnification by A^{-1}, so that the resulting x is in a direction of maximum magnification by A, and equality is attained in (2.2.2). The vector δb was chosen in a direction of maximum magnification by A^{-1}, so that equality holds in (2.2.1). As a consequence, equality also holds in (2.2.5). Had we not made such special choices of b and δb, the result would have been less spectacular.

In some applications, for example, numerical solution of partial differential equations, if the solution is known to be a smooth function, it can be shown that b must

necessarily lie in a direction of large magnification by A^{-1}. Under that restriction it is impossible to create an example that is anywhere near as spectacular as Example 2.2.15.

We have seen that if a system is ill conditioned, we can build examples where $\|\delta x\|/\|x\|$ is much larger than $\|\delta b\|/\|b\|$. In fact it can also happen that $\|\delta x\|/\|x\|$ is much smaller than $\|\delta b\|/\|b\|$. Inequality (2.2.5) has a companion inequality

$$\frac{\|\delta b\|}{\|b\|} \leq \kappa(A)\frac{\|\delta x\|}{\|x\|}, \tag{2.2.20}$$

which can be obtained by interchanging the roles of x and δx with b and δb, respectively, and which is also sharp.

Exercise 2.2.21 Prove Inequality (2.2.20). Under what conditions does equality hold in (2.2.20)? □

Example 2.2.22 We use the same ill-conditioned matrix A as in the two recent examples. By (2.2.17) the system

$$\begin{bmatrix} 1000 & 999 \\ 999 & 998 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

has $x = \begin{bmatrix} 1997 \\ -1999 \end{bmatrix}$ as its unique solution. If we now perturb this solution by $\delta x = \begin{bmatrix} 0.01 \\ 0.01 \end{bmatrix}$, we obtain $x + \delta x = \begin{bmatrix} 1997.01 \\ -1998.99 \end{bmatrix}$, which hardly differs from x at all. However, using (2.2.16),

$$A(x + \delta x) = Ax + A\delta x = \begin{bmatrix} 18.99 \\ 20.97 \end{bmatrix},$$

which is nowhere near Ax. □

Because of their great sensitivity it is generally futile, even meaningless, to try to solve ill-conditioned systems in the presence of uncertainty in the data.

Exercise 2.2.23 Let $A = \begin{bmatrix} 375 & 374 \\ 752 & 750 \end{bmatrix}$.

(a) Calculate A^{-1} and $\kappa_\infty(A)$.

(b) Find b, δb, x, and δx such that $Ax = b$, $A(x + \delta x) = b + \delta b$, $\|\delta b\|_\infty/\|b\|_\infty$ is small, and $\|\delta x\|_\infty/\|x\|_\infty$ is large.

(c) Find b, δb, x, and δx such that $Ax = b$, $A(x + \delta x) = b + \delta b$, $\|\delta x\|_\infty/\|x\|_\infty$ is small, and $\|\delta b\|_\infty/\|b\|_\infty$ is large.

□

Ill Conditioning Caused by Poor Scaling

Some linear systems are ill conditioned simply because they are out of scale. Consider the following example.

Example 2.2.24 Let ϵ be a small positive number. The system

$$\begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \epsilon \end{bmatrix}$$

has the unique solution $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. You can easily check that if $\epsilon \ll 1$, then then the coefficient matrix is ill conditioned with respect to the usual norms. In fact $\kappa_1(A) = \kappa_2(A) = \kappa_\infty(A) = 1/\epsilon$. This system is subject to everything we have said so far about ill-conditioned systems. For example, one can find a small perturbation in b that that causes a large perturbation in x: Just take $b + \delta b = \begin{bmatrix} 1 \\ 2\epsilon \end{bmatrix}$, for which $\|\delta b\|_\infty / \|b\|_\infty = \epsilon$, to get $x + \delta x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, which is far from $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Notice that this perturbation of b is small with respect to $\|b\|$ but not small compared to the component that was perturbed.

If we multiply the second equation of our system by $1/\epsilon$, we get a new system

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

which is perfectly well conditioned. Thus the ill conditioning was just a consequence of poor scaling. □

Theorem 2.2.25 *Let A be any nonsingular matrix, and let $a_1, a_2, \ldots, a_n$ denote its columns. Then for any i and j,*

$$\kappa_p(A) \geq \frac{\|a_i\|_p}{\|a_j\|_p}, \quad 1 \leq p \leq \infty.$$

Proof. Clearly $a_i = Ae_i$, where e_i is the vector with a one in the ith position and zeros elsewhere. Thus

$$\text{maxmag}(A) = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \geq \frac{\|Ae_i\|_p}{\|e_i\|_p} = \|a_i\|_p,$$

$$\text{minmag}(A) = \min_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \leq \frac{\|Ae_j\|_p}{\|e_j\|_p} = \|a_j\|_p,$$

and

$$\kappa_p(A) = \frac{\text{maxmag}(A)}{\text{minmag}(A)} \geq \frac{\|a_i\|_p}{\|a_j\|_p}.$$

□

Theorem 2.2.25 implies that any matrix that has columns that differ by several orders of magnitude is ill conditioned. The same can be said of rows, since A is ill conditioned if and only if A^T is. (You can easily verify that $\kappa_\infty(A) = \kappa_1(A^T)$. In Section 4.2 (Corollary 4.2.2) we will see that $\kappa_2(A) = \kappa_2(A^T)$.) Thus a necessary condition for a matrix to be well conditioned is that all of its rows and columns be of roughly the same magnitude. This condition is not sufficient, as the matrices in Example 2.2.8 and Exercise 2.2.23 show.

If a system is ill conditioned because its rows or columns are badly out of scale, one must refer back to the underlying physical problem in order to determine whether the ill conditioning is inherent in the problem or simply a consequence of poor choices of measurement units. The system in Example 2.2.24 was easy to handle only because it really consists of two independent problems

$$\begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} x_1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \epsilon \end{bmatrix} \begin{bmatrix} x_2 \end{bmatrix} = \begin{bmatrix} \epsilon \end{bmatrix},$$

each of which is well conditioned. In general a more careful analysis is required. Although the rows and columns of any matrix can easily be rescaled so that all of the rows and columns have about the same norm, there is no unique way of doing it, nor is it guaranteed that the resulting matrix is well conditioned. Issues associated with scaling will be discussed in Section 2.8, but no definite advice will be given. Any decision about whether to rescale or not, and how to rescale, should be guided by the underlying physical problem.

For a summary of some of the most important known results on scaling to (nearly) minimize the condition number, see Higham [41]. A different kind of condition number that is invariant under row scaling is introduced in Section 2.9.

Another Geometric View of Ill Conditioning

We have already seen one geometric interpretation of ill conditioning. For those matrices whose rows and columns are not badly out of scale, a second useful geometric picture of ill conditioning can be developed. Recall that a matrix is singular if and only if its columns are linearly dependent (Theorem 1.2.3). We will show that the columns of an ill-conditioned matrix are "nearly" linearly dependent. This is consistent with the idea that ill-conditioned matrices are "nearly" singular. Let A be a nonsingular matrix whose rows and columns are not severely out of scale, and supposed A has been normalized so that $\|A\| = 1$. That is, if $\|A\| \neq 1$, we multiply A by the scalar $1/\|A\|$ to obtain a new matrix whose norm is 1. We have already seen that multiplying an entire matrix by a scalar does not change its condition number (Exercise 2.2.6). This normalization procedure is not essential to our argument, but it makes it simpler and clearer. Since $\|A\| = \text{maxmag}(A)$, we have

$$1 \ll \kappa(A) = \frac{\text{maxmag}(A)}{\text{minmag}(A)} = \frac{1}{\text{minmag}(A)},$$

so minmag$(A) \ll 1$. This implies that there is a $c \in \mathbb{R}^n$ such that $\|Ac\|/\|c\| \ll 1$. Since the ratio $\|Ac\|/\|c\|$ depends only on the direction of c and not on the length, we can choose c so that $\|c\| = 1$, and consequently $\|Ac\| \ll 1$. Letting $a_1, a_2, \ldots, a_n$ denote the columns of of A, we have

$$Ac = \sum_{i=1}^{n} a_i c_i.$$

Thus we see that there is a linear combination of the columns of A that adds up to something small (Ac). If there were a linear combination that added up exactly to zero, the columns would be linearly dependent. Since A is nonsingular, this cannot occur. Since there is a linear combination that adds up to something that is "almost" zero, we say that the columns of A are "nearly" linearly dependent.

A singular matrix had not only linearly dependent columns but also linearly dependent rows. This suggests that an ill-conditioned matrix should have rows that are "nearly" linearly dependent. That this is indeed the case follows from the fact that A is ill conditioned if and only if A^T is.

Example 2.2.26 The matrices from Example 2.2.8 and Exercise 2.2.23 have rows and columns that are nearly linearly dependent. □

The geometric interpretation of ill conditioning is based on the idea that the rows of an ill-conditioned matrix are nearly linearly dependent. Consider the case $n = 2$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2. \end{aligned}$$

The solution set of each of these equations is a line in the (x_1, x_2) plane. The solution of the system is the point at which the two lines intersect. The first line is perpendicular to the (row) vector $[a_{11}\ a_{12}]$, and the second line is perpendicular to $[a_{21}\ a_{22}]$. If A is ill conditioned, then these two vectors are nearly dependent; that is, they point in nearly the same (or opposite) direction. Therefore the lines determined by them are nearly parallel, as depicted in Figure 2.1. The point labelled p is the solution of the system. A small perturbation in b_1 (for example) causes a small parallel shift in the first line. The perturbed line is represented by the dashed line in Figure 2.1. Since the two lines are nearly parallel, a small shift in one of them causes a large shift in the solution from point p to point q. In contrast, in the well-conditioned case, the rows are not nearly dependent, and the lines determined by the two equations are not nearly parallel. A small perturbation in one or both of the lines gives rise to a small perturbation in the solution.

Example 2.2.27 Consider the system

$$\begin{aligned} 1000x_1 + 999x_2 &= b_1 \\ 999x_1 + 998x_2 &= b_2, \end{aligned}$$

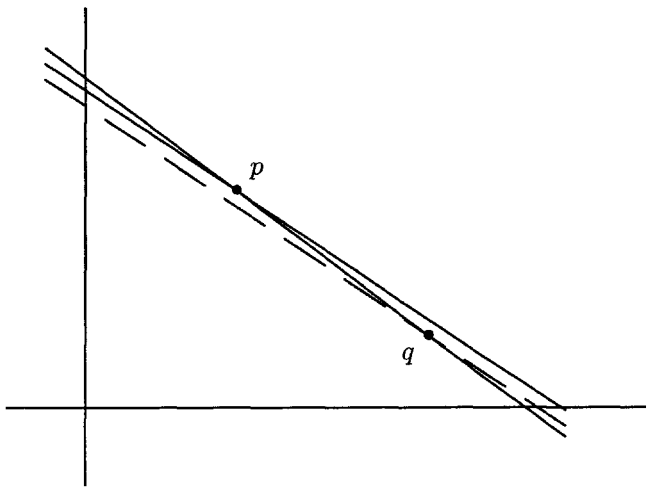

Fig. 2.1 Perturbing an ill-conditioned system

for which the coefficient matrix is the same as in Example 2.2.8. the slopes of the two lines are

$$m_1 = -\frac{1000}{999} = -1.001001 \quad \text{and} \quad m_2 = -\frac{999}{998} = -1.001002.$$

They are so nearly parallel that they are virtually indistinguishable in the vicinity of their intersection point. Therefore the intersection point is hard to find. □

The system depicted in Figure 2.1 is actually not very ill conditioned at all. It is not possible to draw a good picture of a truly ill-conditioned system; the lines would be so nearly parallel as to be indistinguishable.

It is also useful to visualize the case of three equations in three unknowns. The solution set of each equation is a plane in three-dimensional space. The plane determined by the ith equation is perpendicular to the row vector $[a_{i1}\ a_{i2}\ a_{i3}]$. Each pair of planes intersects in a line, and the three planes together have a common intersection point, which is the solution of the system.

In the ill-conditioned case, the rows of the matrix are nearly linearly dependent, so one of the rows is nearly a linear combination of the other two rows. For the sake of argument let us say that the third row of A is nearly a linear combination of the other two rows. This means that the vector $[a_{31}\ a_{32}\ a_{33}]$ nearly lies in the plane spanned by $[a_{11}\ a_{12}\ a_{13}]$ and $[a_{21}\ a_{22}\ a_{23}]$. Therefore the plane of solutions of the third equation is nearly parallel to the line of intersection of the first and second planes. In the vicinity of the solution this line appears nearly to lie in the third plane. Thus the exact location of the solution is hard to distinguish, and a small perturbation of any of the planes will cause a large perturbation in the solution.

A better description would treat all equations equally rather than distinguishing the third equation. Such a description is harder to write, but the situation is not hard to visualize. Think first of a singular system that has infinitely many solutions; picture three planes that intersect in a line. Now perturb the picture slightly so that there is only one intersection point, but the three lines determined by intersecting the planes in pairs remain nearly parallel. This is the ill-conditioned case.

Estimating the Condition Number

The developments of this section have made clear the importance of the condition number of a matrix. Obviously we would like to be able to compute, or at least estimate, this quantity at will. In principle the condition number is not hard to calculate; one simply finds A^{-1} and then calculates $\|A\|\ \|A^{-1}\|$. (Or, if one is using MATLAB, one simply types `cond(A)` .) This is fine if A is not too large. However, for really large A, we would prefer to save the considerable expense of computing A^{-1}. For our purposes we do not need to know the condition number exactly; an order-of-magnitude estimate is good enough. What is needed is an inexpensive estimate of $\kappa(A)$.

Let us suppose that we have already solved the system $Ax = b$ by Gaussian elimination, and now we would like to estimate $\kappa(A)$ in order to help us assess

the quality of our computed solution. Suppose we choose to estimate $\kappa_1(A) = \|A\|_1\|A^{-1}\|_1$. From Theorem 2.1.29 we know that it is easy to compute $\|A\|_1$. What is more challenging is to get an estimate of $\|A^{-1}\|_1$. We begin by noting that for any nonzero $w \in \mathbb{R}^n$

$$\frac{\|A^{-1}w\|_1}{\|w\|_1} \leq \max_{y \neq 0} \frac{\|A^{-1}y\|_1}{\|y\|_1} = \|A^{-1}\|_1.$$

Thus, taking $w = b$, we have $A^{-1}w = x$,

$$\frac{\|x\|_1}{\|b\|_1} \leq \|A^{-1}\|_1 \quad \text{and} \quad \kappa_1(A) \geq \frac{\|A\|_1\|x\|_1}{\|b\|_1}.$$

This gives an inexpensive lower bound for $\kappa_1(A)$. More generally, for any nonzero $w \in \mathbb{R}^n$,

$$\kappa_1(A) \geq \frac{\|A\|_1\|A^{-1}w\|_1}{\|w\|_1}.$$

Since we already have an LU decomposition of A at hand, we can calculate $A^{-1}w$ by solving $Ac = w$ at a cost of only some $2n^2$ flops. If w is chosen in a direction of near maximum magnification by A^{-1}, the estimate

$$\kappa_1(A) \approx \frac{\|A\|_1\|A^{-1}w\|_1}{\|w\|_1} \tag{2.2.28}$$

will be quite good. Actually any w chosen at random is likely to have a significant component in the direction of maximum magnification by A^{-1} and therefore to give a reasonable estimate in (2.2.28). Since a random w will occasionally give a severe underestimate of $\kappa_1(A)$, the cautious operator might like to try several different choices of w.

More sophisticated approaches conduct systematic searches for a w that points nearly in the direction of maximum magnification. The most successful method to date has been the method of Hager, as modified by Higham (see [41]), which uses ideas from convex optimization to search for a w that maximizes $\|A^{-1}w\|_1/\|w\|_1$. This method, which usually gives an excellent estimate, is the basis of the condition number estimators in LAPACK [1] and MATLAB.

Exercise 2.2.29 Recall that in MATLAB you can type `A = hilb(3)` to get the 3×3 Hilbert matrix H_3, for example. Use MATLAB's condition number estimator `condest` to estimate $\kappa_1(H_n)$ for $n = 3, 6, 9$, and 12. Compare it with the true condition number, as computed by `cond(A,1)`. Note the excellent agreement. □

Exercise 2.2.30 Try MATLAB's condition number estimator on a larger matrix. For example, try

```
m = 42;  % Make m larger or smaller, as needed.
A = delsq(numgrid('N',m)); % sparse matrix
size(A)                    % of dimension (m-2)^2.
```

```
issparse(A)
B = full(A);                    % nonsparse version of A
issparse(B)
tic; c1 = condest(A), toc
tic; c2 = cond(B,1), toc
```

Comment on the speed and accuracy of `condest`. You might also like to try `tic; c3 = cond(B,2), toc`, which computes $\kappa_2(B)$. However, you may find that this takes too long unless you decrease the size of the problem by decreasing m. This is a time consuming calculation, because it requires the singular values of B (Section 4.2). □

MATLAB's `condest` function has to compute the LU decomposition of the matrix, since it cannot assume that an LU decomposition is available. Thus `condest` is less efficient than it would be if the decomposition were assumed available. However, if the matrix under consideration is sparse, `condest` will do a sparse LU decomposition (Section 1.9), thereby saving a lot of work. This explains the good outcome in Exercise 2.2.30.

2.3 PERTURBING THE COEFFICIENT MATRIX

Up to this point we have considered only the effect of perturbing b in the system $Ax = b$. We must also consider perturbations of A, as A is also known and represented only approximately. Thus, let us compare two systems $Ax = b$ and $(A + \delta A)\hat{x} = b$, where $\|\delta A\|/\|A\|$ is small. Our first task is to establish a condition that guarantees that the system$(A + \delta A)\hat{x} = b$ has a unique solution, given that the system $Ax = b$ does. This is given by the following theorem, which, along with the subsequent theorems in this section, is valid for any vector norm and its induced matrix norm and condition number.

Theorem 2.3.1 *If A is nonsingular and*

$$\frac{\|\delta A\|}{\|A\|} < \frac{1}{\kappa(A)}, \tag{2.3.2}$$

then $A + \delta A$ is nonsingular.

Proof. The hypothesis $\|\delta A\|/\|A\| < 1/\kappa(A)$ can be rewritten in various ways, for example, $\|\delta A\| < 1/\|A^{-1}\|$ and $\|\delta A\|\,\|A^{-1}\| < 1$. We'll use this last form of the inequality, and we'll prove the contrapositive form of the theorem: If $A + \delta A$ is singular, then $\|\delta A\|\,\|A^{-1}\| \geq 1$.

Suppose $A + \delta A$ is singular. Then, by Theorem 1.2.3, there is a nonzero vector y such that $(A + \delta A)y = 0$. Reorganizing this equation, we obtain $y = -A^{-1}\delta A y$, which implies $\|y\| = \|A^{-1}\delta A y\| \leq \|A^{-1}\|\,\|\delta A\|\,\|y\|$. Since $\|y\| > 0$, we can divide both sides of the inequality by $\|y\|$ to obtain $1 \leq \|A^{-1}\|\,\|\delta A\|$, which is the desired result. □

Theorem 2.3.1 demonstrates another important function of the condition number; it gives us an idea of the distance from A to the nearest nonsingular matrix: If $A+\delta A$ is singular, then $\|\delta A\|/\|A\|$ must be at least $1/\kappa(A)$. It turns out that for the spectral norm this result is exact: If $A+\delta A$ is the singular matrix closest to A, in the sense that $\|\delta A\|_2$ is as small as possible, then $\|\delta A\|_2/\|A\|_2$ is exactly $1/\kappa_2(A)$. We will prove this in Corollary 4.2.22.

As long as (2.3.2) is satisfied, we are assured that the equation $(A+\delta A)\hat{x}=b$ has a unique solution. Notice that (2.3.2) is hard to satisfy if A is ill conditioned; that is, it is satisfied only for very small perturbations δA. If, on the other hand, A is well conditioned, (2.3.2) holds even for relatively large perturbations.

Now let us consider the relationship between the solutions of $Ax=b$ and $(A+\delta A)\hat{x}=b$. Let $\delta x=\hat{x}-x$, so that $\hat{x}=x+\delta x$. Under what conditions can we conclude that $\|\delta x\|/\|x\|$ is small? We would like an upper bound on $\|\delta x\|/\|x\|$ in the spirit of Theorem 2.2.4. We will obtain such a bound eventually, but it turns out to be easier to bound $\|\delta x\|/\|\hat{x}\|$. In most cases there will not be much difference between $\|x\|$ and $\|\hat{x}\|$, so it makes little difference which one we use in the denominator.

Theorem 2.3.3 *Let A be nonsingular, let $b\neq 0$, and let x and $\hat{x}=x+\delta x$ be solutions of $Ax=b$ and $(A+\delta A)\hat{x}=b$, respectively. Then,*

$$\frac{\|\delta x\|}{\|\hat{x}\|}\leq \kappa(A)\frac{\|\delta A\|}{\|A\|}. \tag{2.3.4}$$

Proof. Rewriting the equation $(A+\delta A)\hat{x}=b$ as $Ax+A\delta x+\delta A\hat{x}=b$, using the equation $Ax=b$, and reorganizing the resulting equation, we obtain $\delta x=-A^{-1}\delta A\hat{x}$. Thus

$$\|\delta x\|\leq \|A^{-1}\|\,\|\delta A\|\,\|\hat{x}\|. \tag{2.3.5}$$

Dividing through by $\|\hat{x}\|$ and using the definition $\kappa(A)=\|A\|\,\|A^{-1}\|$, we obtain the desired result. □

Theorem 2.3.3 shows that once again the condition number of A plays the decisive role. If $\kappa(A)$ is not too large, then a small perturbation in A results in a small perturbation in x, in the sense that $\|\delta x\|/\|\hat{x}\|$ is small.

It is interesting to note that Theorem 2.3.3 does not rely on nonsingularity of $A+\delta A$, nor on any assumption to the effect that δA is small. In contrast, the next theorem, which provides a bound on $\|\delta x\|/\|x\|$, does make such an assumption.

Theorem 2.3.6 *If A is nonsingular, $\|\delta A\|/\|A\|<1/\kappa(A)$, $b\neq 0$, $Ax=b$, and $(A+\delta A)(x+\delta x)=b$, then*

$$\frac{\|\delta x\|}{\|x\|}\leq \frac{\kappa(A)\dfrac{\|\delta A\|}{\|A\|}}{1-\kappa(A)\dfrac{\|\delta A\|}{\|A\|}}. \tag{2.3.7}$$

If A is well conditioned and $\|\delta A\|/\|A\|$ is sufficiently small, then $\|\delta A\|/\|A\| \ll 1/\kappa(A)$. In this case the denominator on the right side of (2.3.7) is approximately 1. Then (2.3.7) states *roughly* that

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta A\|}{\|A\|},$$

which is almost the same as (2.3.4). This shows that if A is well conditioned and $\|\delta A\|/\|A\|$ is small, then $\|\delta x\|/\|x\|$ is small.

If, on the other hand, A is ill conditioned, then (2.3.7) allows (but does not prove) that $\|\delta x\|/\|x\|$ could be large, even if $\|\delta A\|/\|A\|$ is small.

Proof. The proof of Theorem 2.3.6 is the same as that of Theorem 2.3.3, up to (2.3.5). Rewriting $\hat{x}$ as $x + \delta x$ in (2.3.5) and using the triangle inequality, we find that

$$\begin{aligned}\|\delta x\| &\leq \|A^{-1}\|\,\|\delta A\|\,(\|x\| + \|\delta x\|) \\ &= \kappa(A)\frac{\|\delta A\|}{\|A\|}\,(\|x\| + \|\delta x\|)\,.\end{aligned}$$

Now rewrite this inequality so that all of the terms involving $\|\delta x\|$ are on the left-hand side.

$$\left(1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}\right)\|\delta x\| \leq \kappa(A)\frac{\|\delta A\|}{\|A\|}\|x\|$$

The assumption $\|\delta A\|/\|A\| < 1/\kappa(A)$ guarantees that the factor that multiplies $\|\delta x\|$ is positive, so we can divide by it without reversing the inequality. If we also divide through by $\|x\|$, we obtain the desired result. □

So far we have considered the effects of perturbing b and A separately. This was done not out of necessity but from a desire to keep the analysis simple. The combined effects of perturbations in A and b can be expressed in a single inequality, as the next two theorems show. The first is in the spirit of Theorem 2.3.3, and the second is in that of Theorem 2.3.6.

Theorem 2.3.8 *Let A be nonsingular, and suppose x and $\hat{x}$ satisfy $Ax = b$ and $\hat{A}\hat{x} = \hat{b}$, respectively, where $\hat{A} = A + \delta A$, $\hat{x} = x + \delta x \neq 0$, and $\hat{b} = b + \delta b \neq 0$. Then*

$$\frac{\|\delta x\|}{\|\hat{x}\|} \leq \kappa(A)\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|\hat{b}\|} + \frac{\|\delta A\|}{\|A\|}\frac{\|\delta b\|}{\|\hat{b}\|}\right).$$

Of the terms on the right-hand side, the product term is usually negligible. For example, if $\dfrac{\|\delta A\|}{\|A\|} = 10^{-5}$ and $\dfrac{\|\delta b\|}{\|\hat{b}\|} = 10^{-5}$, then $\dfrac{\|\delta A\|}{\|A\|}\dfrac{\|\delta b\|}{\|\hat{b}\|} = 10^{-10}$.

Theorem 2.3.9 *If A is nonsingular, $\|\delta A\|/\|A\| < 1/\kappa(A)$, $b \neq 0$, $Ax = b$, and $(A + \delta A)(x + \delta x) = b + \delta b$, then*

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)\left(\dfrac{\|\delta A\|}{\|A\|} + \dfrac{\|\delta b\|}{\|b\|}\right)}{1 - \kappa(A)\dfrac{\|\delta A\|}{\|A\|}}. \tag{2.3.10}$$

Example 2.3.11 In Example 1.2.6 we considered an electrical circuit that leads to the linear system

$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 1.5 & 0 & -.5 \\ -1 & 0 & 1.7 & -.2 \\ 0 & -.5 & -.2 & 1.7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 3 \\ 0 \end{bmatrix},$$

which we solved to determine the voltages at the nodes of the circuit. If we solve the system using MATLAB or any other reliable software, we obtain an extremely accurate solution (See Example 2.4.2 below). It is the accurate solution of the given system, but what if the entries of A and b are incorrect? The entries of A depend on the resistances in the circuit, and the one nonzero entry of b depends also on the voltage of the battery. None of these quantities are known exactly. Theorem 2.3.9 gives us information about the effects of inaccuracies. Suppose, for example, the resistances and the voltage are known to be in error by less than one one hundredth of one percent. This means that the relative error is less than 10^{-4}, so, roughly speaking,

$$\frac{\|\delta A\|_2}{\|A\|_2} \approx 10^{-4} \quad \text{and} \quad \frac{\|\delta b\|_2}{\|b\|_2} \approx 10^{-4}.$$

Using MATLAB's `cond` function, we get $\kappa_2(A) = 12.7$. Substituting these values into (2.3.10), we find that

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq 2.5 \times 10^{-3}.$$

Thus the computed nodal voltages are off by at most one quarter of one percent. It should be noted that the actual error is likely to be much less than this. Results obtained using an upper bound like the one in Theorem 2.3.9 tend to be quite pessimistic. □

Exercise 2.3.12 Prove Theorem 2.3.8. Do it your way, or use the following outline.

(a) Show that

$$\delta x = A^{-1}(\delta b - \delta A \hat{x}),$$

$$\|\delta x\| \leq \|A^{-1}\| (\|\delta b\| + \|\delta A\| \, \|\hat{x}\|),$$

and

$$\frac{\|\delta x\|}{\|\hat{x}\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|A\| \, \|\hat{x}\|} \right).$$

(b) Show that

$$\|\hat{b}\| \leq (\|A\| + \|\delta A\|) \, \|\hat{x}\|,$$

and therefore

$$\frac{1}{\|\hat{x}\|} \leq \frac{(\|A\| + \|\delta A\|)}{\|\hat{b}\|}.$$

(c) Combine the results of (a) and (b) to finish the proof.

□

Exercise 2.3.13 Prove Theorem 2.3.9 by combining elements of the proofs of Theorems 2.2.4 and 2.3.6. □

Geometric pictures of two- and three-dimensional ill-conditioned systems such as those we developed to visualize the effects of perturbations in b are also useful for visualizing perturbations in A. Whereas perturbations in b cause parallel shifts of the lines or planes, perturbations in A cause nonparallel shifts.

2.4 A POSTERIORI ERROR ANALYSIS USING THE RESIDUAL

So far we have been studying the sensitivity of the solution of $Ax = b$ to perturbations in the data. Now we switch to a related question. If we solve the system $Ax = b$ using Gaussian elimination or some other method, how do the roundoff (and other) errors affect the accuracy of the computed solution? Before getting into the details of floating-point arithmetic and roundoff error analysis, let us pause to make note of a simple error test that uses the residual and the condition number.

Suppose we have computed a solution of the system $Ax = b$ by any method whatsoever. Call this computed solution $\hat{x}$. Regardless of how we obtained $\hat{x}$, we can easily compute the *residual* $\hat{r} = b - A\hat{x}$, which gives a measure of how well $\hat{x}$ fits the equations. The fit is good if $\hat{r}$ is small (more precisely, if $\|\hat{r}\|/\|b\|$ is small); $\hat{r} = 0$ if and only if $\hat{x}$ is the true solution of $Ax = b$.

A tiny residual is reassuring. It guarantees that $\hat{x}$ is the solution of a system that is close to $Ax = b$: If we define $\delta b = -\hat{r}$, then $\hat{x}$ is the exact solution of the system $A\hat{x} = b + \delta b$, which is just a slight perturbation of the system $Ax = b$. Unfortunately this does not guarantee that $\hat{x}$ is close to x; we have to take the condition number of A into account. Writing $\hat{x} = x + \delta x$, as in Section 2.2, we see that Theorem 2.2.4 gives us an upper bound on the relative error $\|\delta x\|/\|x\|$. Restating Theorem 2.2.4 as a statement about residuals, we have the following result.

Theorem 2.4.1 *Let A be nonsingular, let $b \neq 0$, and let $\hat{x}$ be an approximation to the solution of $Ax = b$. (In other words, let $\hat{x}$ be any vector whatsoever.) Let $\hat{r} = b - A\hat{x}$. Then*

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \kappa(A)\frac{\|\hat{r}\|}{\|b\|}.$$

From this simple theorem we see that if the residual is tiny and A is well conditioned, then $\hat{x}$ is an extremely accurate approximation to x. The cost of calculating $\hat{r}$ is only about $2n^2$ flops if A is full and even less if A is sparse. If we also have an efficient means of calculating or estimating the condition number (as discussed at the end of Section 2.2), then we may be able to use this theorem to guarantee the accuracy of our computed solution.

Theorem 2.4.1 is an example of an *a posteriori* error bound. It is a bound we obtain after we have solved the problem (i.e. computed $\hat{x}$). In contrast, an *a priori*

error analysis attempts to determine, before solving a problem, whether the method is going to produce an accurate solution. *A posteriori* analyses are generally easier and more informative than *a priori* analyses because they can make use of the computed solution and any other information that was obtained in the course of the computation. In Section 2.7 we will develop an *a priori* analysis of the accuracy of the solution of a linear system by Gaussian elimination in the presence of roundoff errors.

Example 2.4.2 Consider the linear system

$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 1.5 & 0 & -.5 \\ -1 & 0 & 1.7 & -.2 \\ 0 & -.5 & -.2 & 1.7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

from Example 1.2.6. The components of the solution x are the nodal voltages of the circuit shown in that example. Entering the matrix A and vector b into MATLAB and using the commands

```
format long
xhat = A\b
```

we find that

$$\hat{x} = \begin{bmatrix} 3.06382978723404 \\ 2.42553191489362 \\ 3.70212765957447 \\ 1.14893617021277 \end{bmatrix}.$$

We write $\hat{x}$ to indicate that this result is not exactly the true solution, since roundoff errors occurred during the calculation. To check the accuracy of the result, we perform the additional operations

```
rhat = b - A*xhat;
nr = norm(rhat)
ca = cond(A)
errbound = ca*nr/norm(b)
```

to find that $\|\hat{r}\|_2 = 1.05 \times 10^{-15}$, $\kappa_2(A) = 12.7$, and

$$\frac{\|x - \hat{x}\|_2}{\|x\|_2} \leq \frac{\kappa_2(A)\|\hat{r}\|_2}{\|b\|_2} = 4.45 \times 10^{-15}.$$

This shows that our computed solution is very accurate. Roughly speaking, its entries agree with the correct nodal voltages to at least fourteen decimal places.

This assumes, of course, that the data (A and b) are correct. However, since A is well-conditioned, we know from results in Section 2.3 that slight errors in A and b will perturb the solution only slightly. See Example 2.3.11. □

Exercise 2.4.3 Consider the system

$$\begin{bmatrix} 9 & -5 \\ -5 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

from Example 1.2.8. The components of the solution are the loop currents in the electrical circuit in that example. Use MATLAB to solve the system, compute the residual and condition number, and show that the computed solution is extremely accurate. □

Exercise 2.4.4 Rework Exercise 1.2.17. Compute the solution using MATLAB. Compute the residual and condition number, and conclude that the computed solution is extremely accurate. □

Exercise 2.4.5 If the matrix is large, we prefer to estimate the condition number instead of trying to compute it exactly. Using MATLAB, solve a system $Ax = b$, where A is a large discrete Laplacian operator:

```
m = 42;  % Make m larger or smaller, as needed.
A = delsq(numgrid('N',m));
n = size(A,1)
b = ones(n,1); % ''all ones'' right-hand side.
xhat = A\b;
ca = condest(A)
```

Compute the residual, and use `condest` and Theorem 2.4.1 to estimate the error. Since `condest` estimates the 1-condition number $\kappa_1(A)$, use 1-norms in your estimate. □

2.5 ROUNDOFF ERRORS; BACKWARD STABILITY

This section begins with a discussion of floating-point arithmetic and the effects of roundoff errors. The accuracy of arithmetic operations in the presence of errors is studied. It is found that a sudden loss of (relative) accuracy can occur when a cancellation occurs in the addition or subtraction of two numbers. Because of the threat of cancellation, it is impossible to analyze the errors in a complicated algorithm like Gaussian elimination in a forward or direct way. Therefore a more modest approach, backward error analysis, is introduced.

Floating-point Arithmetic

Most scientific computations are performed on computers using *floating-point arithmetic*, which is the computer version of scientific notation. We will not define the term but instead give some examples. The number $.123456 \times 10^7$ is a six-digit decimal floating-point number. It has a *mantissa* .123456 and an *exponent* 7. It is called a decimal number because the number base is 10 (and of course the mantissa is interpreted as a base-10 fraction). Because the number has an exponent, the decimal point can "float" rather than remaining in a fixed position. For example, $.123456 \times 10^3 = 123.456$, $.123456 \times 10^8 = 12345600.$, and $.123456 \times 10^{-2} = .00123456$. The advantage of the floating-point representation is that it allows very large and very small numbers to be represented accurately. Other

examples of floating-point numbers are $.6542 \times 10^{36}$, a large four-digit decimal number, and $-.71236 \times 10^{-42}$, a small, five-digit decimal number. A floating-point number is said to be *normalized* if the first digit of its mantissa is nonzero. Thus the examples we have looked at so far are normalized, whereas $.0987 \times 10^{6}$ and $-.0012346 \times 10^{-4}$ are not normalized. With few exceptions, nonzero floating-point numbers are stored in normalized form.

Each mantissa and each exponent takes up space in the computer's memory. In a floating-point number system a fixed amount of space is allocated for each number, so there are limitations to the precision with which numbers can be represented. For example, if we are using a decimal (base-10) machine that stores four decimal digits, we cannot represent the number $.11112 \times 10^{5}$ exactly. We must approximate it by the nearest floating-point number, which is $.1111 \times 10^{5}$. Also, each arithmetic operation will be accompanied by a roundoff error. Suppose, for example, we multiply the floating point number $.1111 \times 10^{1}$ by $.1111 \times 10^{2}$. Multiplying the mantissas and adding the exponents, we obtain the exact result. $.01234321 \times 10^{3}$, which we normalize to $.1234321 \times 10^{2}$. The mantissa of this result has more digits than we can store, so we must approximate it by the nearest floating-point number $.1234 \times 10^{2}$. This is where the *roundoff error* occurs.

Since each number's exponent must also be stored within some prescribed amount of space, there is also a limit to the size of exponents that can be represented. Numbers that are past a certain threshold cannot be represented in the floating-point system. If a computation results in a number that is too big to be represented, an *overflow* is said to have occurred. For example, if our system can represent numbers up to about 10^{99}, an overflow will occur if we try to multiply, say, 10^{55} by 10^{50}. This undesired event may cause the computer to stop execution of the program (depending on how certain compiler flags have been set). If a number that is nonzero but too small to be represented is computed, an *underflow* results. For example, if our number system allows storage of numbers as small as about 10^{-99}, and underflow will occur if we multiply 10^{-55} by 10^{-50}. A common remedy for underflow is to set the result to zero. This action is usually (but not always) harmless. With one or two exceptions we will ignore the possibility of underflow or overflow.

All of our examples use base-10 arithmetic, because that is what we humans are used to. Hand-held calculators aside, most computers do not use a base-10 representation; a power of two is more convenient architecturally. In the early days of computing, each manufacturer made its own decisions about the characteristics of its computers' floating point arithmetic (e.g. number base, how many digits are allocated to the mantissa and how many to to the exponent). Thus there were many different floating point systems in use, some better than others. Since the adoption of the IEEE floating-point standard (ANSI/IEEE Standard 754-1985) in 1985, the situation has improved dramatically. Nowadays all of the inexpensive, widely-available microprocessors conform to this standard.

Here we will outline some of the basic properties of IEEE arithmetic. For more details see [15], [41], or [52]. The IEEE standard supports both single precision and double precision floating-point numbers. In both cases the number base is two. Single precision numbers are stored in 32-bit words, of which 24 bits are used for

the mantissa and the other 8 for the exponent. Since $2^{24} \approx 10^7$, 24 bits are enough to store approximately seven decimal digits worth of information. In other words, single precision numbers can be accurate to about seven decimal places. With eight bits for the exponent it is possible to represent exponents from about -128 to $+128$. Since these are exponents of the base 2 (not 10), the range of numbers that can be represented is from about $2^{-128} \approx 10^{-38}$ to $2^{128} \approx 10^{38}$. (These are very coarse approximations.)

Double precision IEEE floating-point numbers are allocated 64 bits, of which 53 are used for the mantissa and 11 for the exponent. Since $2^{53} \approx 10^{16}$, double precision numbers can be accurate to about 16 decimal places. Eleven bits of exponent allow the representation of numbers from about $2^{-1024} \approx 10^{-308}$ to $2^{1024} \approx 10^{308}$, an extremely wide range.

The IEEE floating-point standard also includes useful features for automatic handling of exceptional events such as overflow, underflow, and division by zero. See [15], [41], or [52].

Computing in the Presence of Errors

Our task is to assess the cumulative effects of roundoff errors on our calculations. To this end we introduce the notation $\mathrm{fl}(C)$ to denote the floating-point result of some computation C. For example, if we multiply x by y, the result calculated by the computer will be denoted $\mathrm{fl}(xy)$. We can apply this notation to more complicated expressions as well, as long as the order in which the computations are to be performed is clear. For example, $\mathrm{fl}(\sum_{i=1}^{n} x_i y_i)$ is a perfectly acceptable expression, as long as we have agreed on the order in which the terms are to be added.

Denoting the exact result of a computation also by the letter C, we have $\mathrm{fl}(C) = C + e$, where e is the *absolute error* of the computation. A more useful measure is the *relative error* $\epsilon = e/C$, provided that $C \neq 0$. You can easily verify that the relative error ϵ satisfies

$$\mathrm{fl}(C) = C(1 + \epsilon). \tag{2.5.1}$$

Example 2.5.2 One example suffices to show that the relative error is more meaningful than the absolute error. Suppose we perform a computation on a 7-digit decimal machine and get the result $\mathrm{fl}(C) = .9876572 \times 10^{17}$, whereas the correct value is $C = .98765432 \times 10^{17}$. The computed value is clearly a good approximation to the true value, but the absolute error is $e = C - \mathrm{fl}(C) = .288 \times 10^{12}$, which looks large unless it is compared with C. In the relative error the magnitude of C is automatically taken into account: $\epsilon = e/C = .291 \times 10^{-5}$. Now consider a different computation in which $\mathrm{fl}(C) = .9876572 \times 10^{-15}$ and $C = .98765432 \times 10^{-15}$. Now $e = .288 \times 10^{-20}$, which appears extremely small until it is compared with C. By contrast $\epsilon = e/C = .291 \times 10^{-5}$, the same as before. That the relative error is approximately 10^{-5} is reflected in the fact that C and $\mathrm{fl}(C)$ agree in their first five decimal places. Because of this agreement, the difference between C and $\mathrm{fl}(C)$ is about five powers of 10 smaller that C. That is, the relative error is approximately 10^{-5}. □

Another point worth mentioning is that the absolute error has the same units as C. If C is measured in volts (seconds, meters), then e is also measured in volts (seconds, meters). By contrast the relative error is a dimensionless number; it has no units.

In this text we will normally measure errors in relative terms. The relative error appears in various guises. For example, the expressions $\|\delta x\|/\|x\|$, $\|\delta b\|/\|b\|$, and $\|\delta A\|/\|A\|$, with which we worked in Section 2.2, are expressions of relative error. Also we have already observed that statements about the number of correct digits are actually vague statements about the relative error. Finally, statements about percent error are also statements about the relative error: percent error = | relative error | $\times 100$.

In our analysis we will not assume that our machine satisfies the IEEE standard. Instead we will assume an ideal computer that performs each operation exactly and then rounds the result to the nearest floating point number. The IEEE standard satisfies this assumption, so long as no overflows or underflows occur.[1] Our analysis will ignore the possibility of overflow or underflow. Each individual computation produces a roundoff error that is tiny in the relative sense. We will define the *unit roundoff* u to be the largest relative error that can occur in a rounding operation. The value of u depends on the system. In a system with a mantissa of s decimal digits, the value of u will be around 10^{-s}, since the rounded value and the original value agree to s decimal places. A careful analysis (Exercise 2.5.9) shows that $u = \frac{1}{2} \times 10^{1-s}$. For IEEE single precision numbers $u = 2^{-24} \approx 6 \times 10^{-8}$, and for double precision numbers $u = 2^{-53} \approx 10^{-16}$. Using the form (2.5.1), our ideal floating-point operations satisfy

$$\begin{aligned} \text{fl}(x \pm y) &= (x \pm y)(1+\epsilon_1) & |\epsilon_1| \le u \\ \text{fl}(xy) &= (xy)(1+\epsilon_2) & |\epsilon_2| \le u \\ \text{fl}(x/y) &= (x/y)(1+\epsilon_3) & |\epsilon_3| \le u. \end{aligned} \tag{2.5.3}$$

Our analysis will take the results (2.5.3) as a starting point. These can give one a false sense of security. Since the error made in each individual operation is small, one might think that a great many operations would have to be made before significant errors could accumulate. Unfortunately this turns out to be false.

To get a realistic idea of what can happen, we need to take account of the fact that the operands x and y normally have some error in them already. Instead of the correct values, x and y, the computer works with polluted values $\hat{x} = x(1+\epsilon_1)$ and $\hat{y} = y(1+\epsilon_2)$. Instead of calculating xy or $\text{fl}(xy)$, the computer calculates $\text{fl}(\hat{x}\hat{y})$. We need to compare $\text{fl}(\hat{x}\hat{y})$ with xy. We would like to be able to say that if $|\epsilon_1| \ll 1$ and $|\epsilon_2| \ll 1$, then $\text{fl}(\hat{x}\hat{y}) = xy(1+\epsilon)$, where $|\epsilon| \ll 1$. It turns out that such a result does hold for multiplication, and there is an analogous result for division. Unfortunately, addition and subtraction do not always behave so well.

Let us begin with the well behaved operations. The computer multiplies $\hat{x}$ by $\hat{y}$ to get $\text{fl}(\hat{x}\hat{y}) = \hat{x}\hat{y}(1+\epsilon_3)$, where the roundoff error ϵ_3 satisfies $|\epsilon_3| \le u \ll 1$ by

[1] This assumes that the default "round to nearest" rounding mode is being used. The IEEE standard also supports several other rounding modes. In all modes (2.5.3) continues to hold if we replace u by $2u$.

(2.5.3). Thus

$$\begin{aligned} \mathrm{fl}(\hat{x}\hat{y}) &= x(1+\epsilon_1)y(1+\epsilon_2)(1+\epsilon_3) \\ &= xy(1+\epsilon_1+\epsilon_2+\epsilon_3+\epsilon_1\epsilon_2+\epsilon_1\epsilon_2+\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_3) \\ &= xy(1+\epsilon), \end{aligned}$$

where $\epsilon = \epsilon_1+\epsilon_2+\epsilon_3+\epsilon_1\epsilon_2+\epsilon_1\epsilon_2+\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_3$. The terms involving products of two or more ϵ_i are negligible because all ϵ_i are small. Thus $\epsilon \approx \epsilon_1+\epsilon_2+\epsilon_3$. Since $|\epsilon_1| \ll 1$, $|\epsilon_2| \ll 1$, and $|\epsilon_3| \ll 1$, it also holds that $|\epsilon| \ll 1$. We conclude that multiplication is well behaved in the presence of errors in the operands.

In order to analyze division, we begin by recalling from the theory of geometric series that

$$\frac{1}{1+\epsilon_2} = 1-\epsilon_2+\epsilon_2^2-\epsilon_2^3+\cdots$$

Since $|\epsilon_2| \ll 1$, the approximation $1/(1+\epsilon_2) \approx 1-\epsilon_2$, obtained by ignoring quadratic and higher terms, is good. Thus

$$\begin{aligned} \mathrm{fl}\left(\frac{\hat{x}}{\hat{y}}\right) &= \frac{x(1+\epsilon_1)}{y(1+\epsilon_2)}(1+\epsilon_3) \\ &\approx \frac{x}{y}(1+\epsilon_1)(1-\epsilon_2)(1+\epsilon_3) \\ &\approx \frac{x}{y}(1+\epsilon_1-\epsilon_2+\epsilon_3). \end{aligned}$$

Therefore $\mathrm{fl}(x/y) = (x/y)(1+\epsilon)$, where $\epsilon \approx \epsilon_1-\epsilon_2+\epsilon_3$. We conclude that division is well behaved in the presence of errors.

Our analysis of addition will be a little bit different. We know that the difference between $\mathrm{fl}(\hat{x}+\hat{y})$ and $\hat{x}+\hat{y}$ is relatively small, so we will simply compare $\hat{x}+\hat{y}$ with $x+y$. (We could have done the same in our analyses of multiplication and division.) This simplifies the analysis slightly and has the advantage of making it clear that any serious damage that is done is attributable not to the roundoff error from the current operation but to the errors that had been made previously.

$$\begin{aligned} \hat{x}+\hat{y} &= x(1+\epsilon_1)+y(1+\epsilon_2) \\ &= (x+y)+x\epsilon_1+y\epsilon_2 \\ &= (x+y)\left(1+\frac{x}{x+y}\epsilon_1+\frac{y}{x+y}\epsilon_2\right). \end{aligned}$$

Thus $\hat{x}+\hat{y} = (x+y)(1+\epsilon)$, where

$$\epsilon = \frac{x}{x+y}\epsilon_1+\frac{y}{x+y}\epsilon_2.$$

Given that $|\epsilon_1| \ll 1$ and $|\epsilon_2| \ll 1$, we can say that $|\epsilon| \ll 1$ provided that neither x nor y is large compared with $x+y$. If, on the other hand, x or y is large compared with $x+y$, then ϵ can and probably will be large. That is, the computed result

is probably inaccurate. This occurs when (and only when) x and y are are almost exactly opposites of each other, so that they nearly cancel one another out when they are added.

An identical analysis holds for subtraction.

Exercise 2.5.4 Show that if $\hat{x} = x(1+\epsilon_1)$ and $\hat{y} = y(1+\epsilon_2)$, where $|\epsilon_1| \ll 1$ and $|\epsilon_2| \ll 1$, then $\hat{x} - \hat{y} = (x-y)(1+\epsilon)$, where $|\epsilon| \ll 1$ unless x or y is much larger than $x - y$. □

If x and y are nearly equal, so that $x - y$ is much smaller than both x and y, then the computed result $\hat{x} - \hat{y}$ can and probably will be inaccurate.

We conclude that both addition and subtraction are well behaved in the presence of errors, except when the operands nearly cancel one another out. Because cancellation generally signals a sudden loss of accuracy, it is sometimes called *catastrophic cancellation.*

Example 2.5.5 It is easy to see intuitively how cancellation leads to inaccurate results. Suppose an eight-digit decimal machine is to calculate $x - y$, where $x = .31415927\ldots \times 10^1$ and $y = .31415916\ldots \times 10^1$. Due to errors in the computation of x and y, the numbers that are actually stored in the computer's memory are $\hat{x} = .31415929\ldots \times 10^1$ and $\hat{y} = .31415914\ldots \times 10^1$. Clearly these numbers are excellent approximations to x and y; they are correct in the first seven decimal places. That is, the relative errors ϵ_1 and ϵ_2 are of magnitude about 10^{-7}. Since $\hat{x}$ and $\hat{y}$ are virtually equal, all but one of the seven accurate digits are canceled off when $\hat{x} - \hat{y}$ is formed: $\hat{x} - \hat{y} = .00000015 \times 10^1 = .15000000 \times 10^{-5}$. In the normalized result only the first digit is correct. The second digit is inaccurate as are all of the zeros that follow it. Thus the computed result is a poor approximation to the true result $x + y = .11\ldots \times 10^{-5}$. the relative error is about 36 percent. □

This example demonstrates a relatively severe case of cancellation. In fact a whole range of severities is possible. Suppose, for example, two numbers are accurate to seven decimal places and they agree with each other in the first three places. then when their difference is taken, three accurate digits will be lost, and the result will be accurate to four decimal places.

We have demonstrated not only that cancellation can cause a sudden loss of relative accuracy, but also that it is the only mechanism by which a sudden loss of accuracy can occur. The only other way an inaccurate result can occur is by gradual accumulation of small errors. Although it is possible to concoct examples where this happens, it is seldom a problem in practice. The small errors that occur are just as likely to cancel each other out, at least in part, as they are to reinforce one another, so they tend to accumulate very slowly. Thus as a practical matter we can say that if a computation has gone bad, there must have been at least one cancellation at some crucial point. In other words, if no cancellations occur during a computation (and the original operands were accurate), the result will generally be accurate.

Unfortunately it is usually difficult to verify that no cancellation will occur in a given computation, and this makes it hard to prove that roundoff errors will not

spoil the computation. The first attempts at error analysis took the *forward* or *direct* approach, in which one works through the algorithm, attempting to bound the error in each intermediate result. In the end one gets a bound for the error in the final result. This approach usually fails, because each time an addition or subtraction is performed, one must somehow prove either that cancellation cannot occur or that a cancellation at that point will not cause any damage in the subsequent computations. This is usually impossible.

Backward Error Analysis

Because of the threat of sudden loss of accuracy through cancellation, the pioneers of scientific computing were quite pessimistic about the possible effects of roundoff errors on their computations. It was feared that any attempt to solve, say, a system of 50 equations in 50 unknowns would yield an inaccurate result. The early attempts to solve systems of linear equations on computers turned out generally better than expected, although disasters did sometimes occur. The issues were not well understood until a new approach, called *backward error analysis*, was developed. The new approach does not attempt to bound the error in the result directly. Instead it pushes the effects of the errors back onto the operands.

Suppose, for example, we are given three floating-point numbers x, y, and z, and we wish to calculate $C = (x + y) + z$. The computer actually calculates $\hat{C} = \mathrm{fl}(\mathrm{fl}(x + y) + z)$. Even if the operands are exact, we cannot assert that the relative error in $\hat{C}$ is small: $\mathrm{fl}(x + y)$ is (probably) slightly in error, so there can be large relative error in $\hat{C}$ if cancellation takes place when $\mathrm{fl}(x + y)$ is added to z. However, there is something else we can do. We have

$$\hat{C} = [(x + y)(1 + \epsilon_1) + z]\,(1 + \epsilon_2),$$

where $|\epsilon_1|$, $|\epsilon_2| \leq u \ll 1$. Define ϵ_3 by $(1 + \epsilon_3) = (1 + \epsilon_1)(1 + \epsilon_2)$, so that $|\epsilon_3| \approx |\epsilon_1 + \epsilon_2| \ll 1$. Then $\hat{C} = (x+y)(1+\epsilon_3)+z(1+\epsilon_2)$. Defining $\overline{x} = x(1+\epsilon_3)$, $\overline{y} = y(1 + \epsilon_3)$, and $\overline{z} = z(1 + \epsilon_2)$, we have

$$\hat{C} = (\overline{x} + \overline{y}) + \overline{z}.$$

Notice that $\overline{x}$, $\overline{y}$, and $\overline{z}$ are extremely close to x, y, and z, respectively. This shows that $\hat{C}$ is the exact result of performing the computation $(x+y)+z$ using the slightly perturbed data $\overline{x}$, $\overline{y}$, and $\overline{z}$. The errors have been shoved back onto the operands. The same can be done with subtraction, multiplication, division, and (with a bit of ingenuity) longer computations.

In general suppose we wish to analyze some long computation $C(z_1, \ldots, z_m)$ involving m operands or input data $z_1, \ldots, z_m$. Instead of trying to show directly that $\mathrm{fl}(C(z_1, \ldots, z_m))$ is close to $C(z_1, \ldots, z_m)$, a backward error analysis attempts to show that $\mathrm{fl}(C(z_1, \ldots, z_m))$ is the exact result of performing the computation with slightly perturbed input data; that is,

$$\mathrm{fl}(C(z_1, \ldots, z_m)) = C(\overline{z}_1, \ldots, \overline{z}_m),$$

where $\overline{z}_1, \ldots, \overline{z}_m$ are extremely close to $z_1, \ldots, z_m$. By *extremely close* we usually mean that the error is a modest multiple of the unit roundoff u. If such a result holds, the computation is said to be *backward stable*.

Of course the analysis does not end here. The backward error analysis has to be combined with a *sensitivity analysis* of the problem. If (1) the computation is backward stable, and (2) we can show that small perturbations in the operands lead to small perturbations in the results, then we can conclude that our computed result is accurate.

For example, if our problem is to solve a nonsingular linear system $Ax = b$, then the operands or inputs are the entries of A and b, and the output is x. We obtain x from A and b by some specified computation $x = C(A, b)$, for example, Gaussian elimination with complete pivoting. In this context we normally use the word *algorithm* instead of *computation*. If we perform the computations in exact arithmetic, we get x exactly, but if we use floating-point arithmetic, we obtain an approximate solution $\hat{x} = \mathrm{fl}(C(A, b))$. In this context the algorithm is backward stable if there exist $\hat{A}$ and $\hat{b}$ that are extremely close to A and b, respectively, such that $\hat{x} = C(\hat{A}, \hat{b})$, that is, $\hat{A}\hat{x} = \hat{b}$ exactly. Another way to say this is that $\hat{x}$ satisfies $(A+\delta A)\hat{x} = b+\delta b$ exactly for some δA and δb such that $\| \delta A \|/\| A \|$ and $\| \delta b \|/\| b \|$ are tiny, that is, a small multiple of u.

As we well know by now, backward stability does not imply that $\hat{x}$ is close to x. The sensitivity of the problem is given by the condition number of A. If the algorithm is backward stable, and the coefficient matrix is well conditioned, then the computed solution is accurate. If, on the other hand, the matrix is ill conditioned, the solution may well be inaccurate, even though the algorithm is backward stable.

The backward approach to error analysis separates clearly the properties of the problem (e.g. solve $Ax = b$) from the properties of the algorithm (e.g. Gaussian elimination with complete pivoting). The sensitivity analysis pertains to the problem, and the backward error analysis pertains to the algorithm. We say that the *problem* is *well conditioned* if small changes in the input lead to small changes in the results. Otherwise it is *ill conditioned*. An *algorithm* is backward stable if it returns answers that exactly solve some slightly perturbed problem. It follows that a backward stable algorithm will be able to solve well-conditioned problems accurately. We will adopt the attitude that it is unreasonable to expect any algorithm to solve ill-conditioned problems accurately. Therefore we will judge an algorithm to be satisfactory if it is backward stable.

The backward approach to error analysis succeeds because it is much less ambitious than the forward approach. The latter attempts to prove that a given algorithm always produces an accurate result, regardless of the sensitivity of the problem. This is usually impossible.

Small Residual Implies Backward Stability

If we say that an algorithm is backward stable (with no further qualification), we mean that it performs in a backward stable manner on all possible sets of input

data. However, the term can also be used in a much broader sense. If we use an algorithm, say Gaussian elimination, to solve a particular problem $Ax = b$ (for some specific choice of A and b), we will say the the algorithm is backward stable *on that particular problem* if it produces an $\hat{x}$ that is the exact solution of a nearby problem $(A + \delta A)\hat{x} = b + \delta b$.

For the linear system problem (and many other problems) there is a simple *a posteriori* method checking the backward stability of a computation: check the residual. The problem is to solve $Ax = b$. Whatever method we use to get a solution $\hat{x}$, we can easily calculate the residual $\hat{r} = b - A\hat{x}$. As we already noted in Section 2.4, $\hat{x}$ is the exact solution of $A\hat{x} = b + \delta b$, where $\delta b = -r$. If $\|\delta b\|/\|b\|$ is tiny, then $\hat{x}$ is indeed the solution of a nearby system. Thus the algorithm is backward stable on this problem. To summarize, a tiny residual implies backward stability.

The following exercise draws essentially the same conclusion by a different approach, in which the residual is associated with a perturbation in A instead of b.

Exercise 2.5.6 Let $\hat{x}$ be an approximation to the solution of $Ax = b$, and let $\hat{r} = b - A\hat{x}$. Define $\delta A \in \mathbb{R}^{n \times n}$ by $\delta A = \alpha r \hat{x}^T$, where $\alpha = \|\hat{x}\|_2^{-2}$.

(a) Show that $\hat{x}$ is the exact solution of $(A + \delta A)\hat{x} = b$.

(b) Show that $\|\delta A\|_2 = \|r\|_2/\|\hat{x}\|_2$ and

$$\frac{\|\delta A\|_2}{\|A\|_2} = \frac{\|r\|_2}{\|A\|_2\|\hat{x}\|_2}.$$

□

Thus if $\|r\|_2$ is tiny relative to $\|A\|_2\|\hat{x}\|_2$, then the algorithm (whichever algorithm was used) is backward stable for this problem.

Additional Exercises

Exercise 2.5.7 Learn more about your computer's arithmetic by running the following MATLAB programs.

(a) What do you learn from running the following program?

```
a = 1;
u = 1;
b = a + u;
while b ~= a
  u = .5*u;
  b = a + u;
end
u
```

(b) What does this one tell you?

```
a = 1;
while a ~= Inf
  a= 10*a
end
```

To get a more precise result, replace the 10 by a 2. You might find `help inf` informative.

(c) What does this one tell you?

```
a = 1;
while a ~= 0
  a= .1*a
end
```

The outcome of this one is probably different from what you expected, based on the information given in the text. The IEEE standard allows *gradual underflow* through the use of subnormal numbers, once the minimum exponent is reached.

(d) What kind of arithmetic does MATLAB appear to be using? To learn more about your computer (as used by MATLAB) try `help computer`, `help isieee`, `help inf`, and `help nan`, for example.

□

Exercise 2.5.8 Write Fortran or C programs that perform the same functions as the programs from Exercise 2.5.7. Make both single and double precision versions and try them out. □

Exercise 2.5.9

(a) Show that in a base-β floating-point number system the largest relative gap between two consecutive normalized numbers occurs between

$$.1000\ldots000 \times \beta^t \quad \text{and} \quad .1000\ldots001 \times \beta^t.$$

(The value of the exponent t is irrelevant.) Thus the largest relative error (the unit roundoff) occurs when one tries to represent the number that lies half way between these two.

(b) Show that the unit roundoff is $\frac{1}{2} \times \beta^{1-s}$, where β is the number base, and s is the number of base-β digits in the mantissa.

□

2.6 PROPAGATION OF ROUNDOFF ERRORS IN GAUSSIAN ELIMINATION

Now that we know something about floating-point arithmetic and roundoff errors, we are ready to analyze the effects of roundoff errors on Gaussian elimination. Our

formal tool will be backward error analysis, but we defer that to Section 2.7. In this section we will see that even without backward error analysis we can get some good insights through the study of well chosen examples.

Gaussian Elimination with Ill-Conditioned Matrices

Our studies of the sensitivity of linear systems have given us ample reason to believe that there is no point in trying to solve severely ill-conditioned systems. Further insight can be gained by taking a heuristic look at what happens when one tries to solve an ill-conditioned system by Gaussian elimination with partial pivoting. We will assume that the rows and columns of the coefficient matrix are not out of scale.

When we do row operations, we take linear combinations of rows in such a way that zeros are deliberately created. Since the rows of an ill-conditioned matrix are nearly linearly dependent, there is the possibility of entire rows being made almost exactly zero by row operations. This possibility is encouraged by the progressive introduction of zeros into the array. Let us say that a row is *bad* if it is nearly a linear combination of previous rows. Suppose the kth row of A is the first bad row. After $k-1$ steps of Gaussian elimination we will have subtracted multiples of the first $k-1$ rows from the kth row in such a way that there are now zeros in the first $k-1$ positions. If the kth row were exactly a linear combination of the previous rows (and exact arithmetic were used), the entire kth row would now be zero. (Why?) Since it is only approximately a linear combination of the previous rows, it will still contain nonzero entries, but these entries will typically be tiny. They are not only tiny but but also inaccurate, because they became tiny through cancellation, as multiples of the earlier rows were subtracted from row k.

One of these tiny, inaccurate entries is the potential pivot in the (k,k) position. Because it is small, the kth row will be interchanged with a lower row that has a larger entry in its kth position, if such a row exists. In this way the bad rows get shifted downward. Eventually a step will be reached at which only bad rows remain. At this point all choices of pivot are tiny and inaccurate. Although the presence of small, inaccurate numbers is not necessarily disastrous to the computation, the use of one as a pivot must be avoided if possible. In the present scenario we are forced to use a tiny, inaccurate pivot. This is used as a divisor in the computation of not-so-small, inaccurate multipliers, whose error pollutes all subsequent rows. The pivots are also used as divisors in the last step of the back-substitution process. Each component of the computed solution is a quotient whose divisor is a pivot. We cannot expect these quotients to be accurate if the divisors are not.

The above analysis is obviously heuristic and is not claimed to be universally valid. A different flavor of ill conditioning is exhibited by the *Kahan matrix* [41].

Example 2.6.1 Consider the ill-conditioned matrix

$$A = \begin{bmatrix} 1000 & 999 \\ 999 & 998 \end{bmatrix},$$

which we have discussed previously. Since the rows are nearly linearly dependent, when a zero is created in the $(2,1)$ position, the entry in the $(2,2)$ position should become nearly zero as well. Indeed the multiplier is $l_{21} = .999$, and the $(2,2)$ entry becomes

$$998 - (.999)(999) = 998 - 998.001 = -.001.$$

This is indeed small, and the result was obtained by severe cancellation. There is no error in the result, because it was computed by exact arithmetic. Consider what happens when five-digit decimal floating-point arithmetic is used. The computation yields

$$998.00 - (.99900)(999.00) = 998.00 - 998.00 = 0.$$

The matrix appears to be singular! □

This example might remind you of a remark that was made in Chapter 1. Not only can a nonsingular matrix appear singular (as just happened); the reverse can occur as well and is actually a much more common occurrence. We remarked that if a Gaussian elimination program attempts to calculate the LU decomposition of a singular matrix, it almost certainly will not recognize that the matrix is singular: Certain entries that should become zero in the course of the calculation will turn out to be nonzero because of roundoff errors. Thus in numerical practice it is impossible to distinguish between ill-conditioned matrices and singular matrices. In contrast to the theoretical situation, where there is a clear distinction between singular and nonsingular, we have instead a continuum of condition numbers, ranging from the well conditioned to the severely ill conditioned, with no clear dividing line in the middle. (Exceptions to this picture are certain matrices that are obviously singular, such as a matrix with a row or column of zeros or two equal rows.)

The next example shows that the distinction between good and bad rows is not always clear. It can happen that the accuracy of a computation deteriorates gradually over a number of steps.

Example 2.6.2 We introduced the ill-conditioned *Hilbert matrices*, defined by $h_{ij} = 1/(i+j-1)$, in Example 2.2.9. For example,

$$H_4 = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}.$$

The rows look very much alike, which suggests ill conditioning. According to MATLAB, $\kappa_2(H_4) \approx 1.6 \times 10^4$.

Let us see how the ill conditioning manifests itself during Gaussian elimination. For the first step, there is no need to make a row interchange, since the largest entry is already in the pivotal position. You can easily check that after one step the

transformed array is

$$\left[\begin{array}{c|ccc} 1 & 1/2 & 1/3 & 1/4 \\ \hline 1/2 & 1/12 & 1/12 & 3/40 \\ 1/3 & 1/12 & 4/45 & 1/12 \\ 1/4 & 3/40 & 1/12 & 9/112 \end{array}\right].$$

The second step operates on the submatrix

$$\begin{bmatrix} 1/12 & 1/12 & 3/40 \\ 1/12 & 4/45 & 1/12 \\ 3/40 & 1/12 & 9/112 \end{bmatrix},$$

all of whose entries are smaller than the original matrix entries. Thus each entry has undergone a small amount of cancellation. Of course these numbers are perfectly accurate because we calculated them by exact arithmetic. If we had used floating-point arithmetic, each of the entries would have suffered a slight loss of accuracy due to the cancellation. Notice that all of the entries of this submatrix are quite close to $1/12$; the rows are almost equal.

The potential pivots for the second step are smaller than those for the first step. Again there is no need for a row interchange, and after the step the transformed submatrix is

$$\left[\begin{array}{c|cc} 1/12 & 1/12 & 3/40 \\ \hline 1 & 1/180 & 1/120 \\ 9/10 & 1/120 & 9/700 \end{array}\right].$$

The entries of the submatrix

$$\begin{bmatrix} 1/180 & 1/120 \\ 1/120 & 9/700 \end{bmatrix}$$

are even smaller than before; more cancellation has taken place. The potential pivots for the third step, $1/180$ and $1/120$, are both quite small. Since the latter is larger, we interchange the rows (although this has little effect on the outcome). After the third step, we have

$$\left[\begin{array}{c|c} 1/120 & 9/700 \\ \hline 2/3 & -1/4200 \end{array}\right].$$

The final pivot is $-1/4200$, which is even smaller.

Now let us see what happens when the same operations are performed in three-digit decimal floating-point arithmetic. The original array is

$$\begin{bmatrix} 1.00 & .500 & .333 & .250 \\ .500 & .333 & .250 & .200 \\ .333 & .250 & .200 & .167 \\ .250 & .200 & .167 & .143 \end{bmatrix}.$$

Some of the entries are already slightly in error. On the first step the $(4,3)$ entry (for example) is modified as follows:

$$.167 - (.250)(.333) = .167 - .833 \times 10^{-1} = .837 \times 10^{-1}.$$

Comparing it with the correct value $1/12 \approx .833 \times 10^1$, we see that there is a substantial error in the third digit. The complete result of the first step is

$$\left[\begin{array}{c|ccc} 1.000 & .500 & .333 & .250 \\ \hline .500 & .830 \times 10^{-1} & .830 \times 10^{-1} & .750 \times 10^{-1} \\ .333 & .830 \times 10^{-1} & .890 \times 10^{-1} & .837 \times 10^{-1} \\ .250 & .750 \times 10^{-1} & .837 \times 10^{-1} & .805 \times 10^{-1} \end{array}\right].$$

The result of the second step is (ignoring the first row and column)

$$\left[\begin{array}{c|cc} .830 \times 10^{-1} & .830 \times 10^{-1} & .750 \times 10^{-1} \\ \hline 1.00 & .600 \times 10^{-2} & .870 \times 10^{-2} \\ .904 & .870 \times 10^{-2} & .127 \times 10^{-1} \end{array}\right].$$

Significant cancellations have now taken place; most of these numbers have only one correct digit. For example, the following computation produced the $(4,3)$ element:

$$.837 \times 10^{-1} - (.904)(.830 \times 10^{-1}) = (.837 - .750) \times 10^{-1} = .870 \times 10^{-2}.$$

Comparing this result with the correct value $1/120 \approx .833 \times 10^{-2}$, we see that it has only one correct digit. The result of the third and final step is

$$\left[\begin{array}{c|c} .870 \times 10^{-2} & .127 \times 10^{-1} \\ \hline .690 & -.600 \times 10^{-4} \end{array}\right].$$

The $(4,4)$ entry $-.600 \times 10^{-4}$ is not even close to the correct value $-1/4200 \approx -.238 \times 10^{-3}$. □

Exercise 2.6.3 Work through the computations in Example 2.6.2, observing the cancellations and the accompanying loss of accuracy. Remember that if you wish to use your calculator to simulate three-digit decimal floating-point arithmetic, it does not suffice simply to set the display to show three digits. Although only three digits are displayed, nine or more digits are stored internally. Correct simulation of three-digit arithmetic requires that each intermediate result be rounded off before it is used in the next computation. A simple way to do this is to write down each intermediate result and then enter it back into the calculator when it is needed. □

Exercise 2.6.4 Work Example 2.6.2 using four-digit arithmetic instead of three. You will see that the outcome is not nearly so bad. □

Exercise 2.6.5 Use MATLAB to explore the extent of cancellation when Gaussian elimination is performed on larger Hilbert matrices.

(a) In MATLAB, type `A = hilb(7)` to get H_7. To get the LU decomposition with partial pivoting, type `[L,U] = lu(A)` . Notice that the matrix L is not itself unit lower triangular, but it can be made unit lower triangular by permuting the rows. This is because MATLAB's `lu` command incorporates the row interchanges in the L matrix. Our real object of interest is the matrix

U, which is the triangular matrix resulting from Gaussian elimination. Notice that the further down in U you go, the smaller the numbers become. The ones at the bottom appear to be zero. To get a more accurate picture, type `format long` and redisplay U.

(b) Generate H_{12} and its LU decomposition. Observe the U matrix using `format long`.

□

Exercise 2.6.6 Let z denote the vector whose entries are all ones (Using MATLAB: `z = ones(n,1)`), and let $b = H_n z$, where H_n is again the $n \times n$ Hilbert matrix. If we now solve the system $H_n x = b$ for x, we should get z as the solution in theory. Using MATLAB (`xhat = A\b`), try solving $H_n x = b$ for $n = 4$, 8, 12, and 16, and see what you get. In each case compute the condition number $\kappa_2(H_n)$ and the norm of the difference: $\| \hat{x} - z \|_2$, where $\hat{x}$ is the computed solution. Calculate the residual $\hat{r} = b - H_n \hat{x}$, too. □

Exercise 2.6.7 The previous exercise exaggerates somewhat the intractability of Hilbert matrices. There we saw that already for $n = 12$ we get a bad solution (assuming double-precision IEEE arithmetic). Actually, how bad the solution is depends not only on the matrix, but also on the vector b. Most choices of b will not give nearly such bad results as those we just saw. Carry out the following computations with $n = 12$. Use double precision (which you get automatically from MATLAB) throughout.

(a) Let z denote the vector of ones, as before, and solve the system $H_{12} y = z$ for y. Note that $\| y \|$ is huge.

(b) Define a vector b by $b = H_{12} y$. In principle b should be the same as z, but in the presence of roundoff errors it is a little different. Calculate b and $\| b - z \|_2$.

(c) Now consider the system $H_{12} x = b$. The way b was defined, the solution x ought to be the same as y. However, our experience from the previous problem suggests that the computed solution $\hat{x}$ may be far from y. Go ahead and solve $H_{12} x = b$ and compare the computed $\hat{x}$ with y. Notice that they agree to almost two decimal places. This is not nearly as bad as what we saw in the previous problem. Calculate the relative error $\| \hat{x} - y \|_2 / \| y \|_2$.

(d) Calculate the norm of the residual $\hat{r} = b - A\hat{x}$, and use it and the condition number to compute an upper bound on the relative error (Theorem 2.4.1). Note that this bound is very pessimistic compared to the actual relative error.

□

Another family of ill-conditioned matrices is the *Lotkin matrices*. The $n \times n$ Lotkin matrix is identical to the Hilbert matrix H_n, except that its first row consist entirely of ones. These nonsymmetric matrices are just as ill conditioned as the Hilbert matrices. To get the 6×6 Lotkin matrix in MATLAB type `A =`

`gallery('lotkin',6)`. For more information about the gallery of Higham test matrices, type `help gallery`. For more information about Lotkin matrices, type `help private/lotkin`.

Exercise 2.6.8 Rework each of the following exercises, using Lotkin matrices in place of Hilbert matrices: (a) Exercise 2.6.5, (b) Exercise 2.6.6, (c) Exercise 2.6.7. □

Why Small Pivots Should Be Avoided

In Section 1.8 we introduced the partial pivoting strategy, in which the pivot for the kth step is chosen to be the largest in magnitude of the potential pivots in column k. The justification given at that time was that we wanted to avoid using as a pivot a small number that would have been zero except for roundoff errors made in previous steps. Since then we have studied cancellation and ill-conditioned matrices and can make a more general statement: We wish to avoid using a small pivot because it may have become small as the result of cancellations in previous steps, in which case it could be very inaccurate. The dangers of using an inaccurate pivot were stated in the first part of this section, in connection with ill conditioning.

In the following example we consider a different scenario. We show what can go wrong if a small pivot is used even though large pivots are available. Here the pivot is not inaccurate; it ruins the computation simply by being small.

Example 2.6.9 Consider the linear system

$$\begin{bmatrix} .002 & 1.231 & 2.471 \\ 1.196 & 3.165 & 2.543 \\ 1.475 & 4.271 & 2.142 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3.704 \\ 6.904 \\ 7.888 \end{bmatrix}. \tag{2.6.10}$$

This system is well conditioned ($\kappa_2(A) \approx 30$), and we will see that it can be solved accurately by Gaussian elimination with partial pivoting. But first let us observe what happens when we use Gaussian elimination without interchanges. We will see that the use of the exceptionally small $(1,1)$ entry as a pivot destroys the computation. The computations will be done in four-digit decimal floating-point arithmetic. You should think of the numbers in (2.6.10) as being exact. You can easily check that the exact solution is $x = [1\ 1\ 1]^T$. The roundoff error effects you are about to observe are caused not because the small pivot is inaccurate (It is not!), but simply because it is small.

The multipliers for the first step are

$$l_{21} = \frac{1.196}{.002} = 598.0 \quad \text{and} \quad l_{31} = \frac{1.475}{.002} = 737.5. \tag{2.6.11}$$

These multiplied by the first row are subtracted from the second and third rows, respectively. For example, the $(2,2)$ entry is altered as follows: 3.165 is replaced by

$$3.165 - (598.0)(1.231) = 3.165 - 736.1 = -732.9. \tag{2.6.12}$$

These equations are not exact; four-digit arithmetic was used. Notice that the resulting entry is much larger than the number it replaced and that the last two digits of 3.165 were lost (rounded off) when a large number was added to (i.e. subtracted from) it. This type of information loss is called *swamping*. The small number was *swamped* by the large one. You can check that swamping also occurs when the $(2,3)$, $(3,2)$, and $(3,3)$ entries are modified. In fact three digits are swamped in the $(2,3)$ and $(3,3)$ positions. At the end of the first step the modified coefficient matrix looks like

$$\begin{bmatrix} .002 & 1.231 & 2.471 \\ 598.0 & -732.9 & -1475. \\ 737.5 & -903.6 & -1820. \end{bmatrix}.$$

The second step of Gaussian elimination works with the submatrix

$$\tilde{A} = \begin{bmatrix} -732.9 & -1475. \\ -903.6 & -1820. \end{bmatrix}.$$

Since this matrix was obtained by subtracting very large multiples of the row $[1.231\ 2.471]$ from the much smaller numbers that originally occupied these rows, the two rows of $\tilde{A}$ are almost exact multiples of $[1.231\ 2.471]$. Thus the rows of $\tilde{A}$ are nearly linearly dependent; that is, $\tilde{A}$ is ill conditioned. You can easily check that $\kappa_2(\tilde{A}) \approx 6400$, which is huge, considering that four-digit arithmetic is being used.

The multiplier for the second step is $l_{32} = (-903.6)/(-732.9) = 1.233$. It is used only to modify the $(3,3)$ entry as follows:

$$-1820. - (1.233)(-1475.) = -1820. + 1819. = -1.000.$$

Severe cancellation occurs here. This is just an attempt to recover the information that was lost through swamping in the previous step. Unfortunately, that information is gone, and consequently the result -1.000 is inaccurate. At the end of the second step, the LU decomposition is complete:

$$\begin{bmatrix} .002 & 1.231 & 2.471 \\ 598.0 & -732.9 & -1475. \\ 737.5 & 1.233 & -1.000 \end{bmatrix}.$$

The forward substitution step yields

$$\begin{aligned} y_1 &= 3.704 \\ y_2 &= 6.904 - (598.0)(3.704) = 6.904 - 2215. = -2208. \\ y_3 &= 7.888 - (737.5)(3.704) - (1.233)(-2208.) \\ &= 7.888 - 2732. + 2722. = -2724. + 2722. = -2.000 \end{aligned}$$

Notice that the last three digits of 6.904 were swamped in the computation of y_2, and severe cancellation occurred in the calculation of y_3. Thus y_3 is inaccurate.

The first step of back substitution is

$$x_3 = \frac{y_3}{u_{33}} = \frac{-2.000}{-1.000} = 2.000.$$

Being the quotient of two inaccurate numbers, x_3 is also inaccurate. Recall that the correct value is 1.000. You can carry out the rest of the back substitution process and find that the computed solution is $[4.000,\ -1.012,\ 2.000]^T$, which is nothing like the true solution.

Let us summarize what went wrong, speaking in general terms (and heuristically!). When a pivot that is much smaller than the other potential pivots is used, large multipliers will result. Thus very large multiples of the pivotal row will be subtracted from the remaining rows. In the process the numbers that occupied those rows will be swamped. The resulting submatrix (the matrix that will be operated on in the next step) will be ill conditioned because each of its rows is almost exactly a multiple of the pivotal row. Because of the ill conditioning, there will be cancellations in later steps. These cancellations are actually just an attempt to uncover the information that was lost due to swamping, but that information is gone.

Now let us see what happens when we solve (2.6.10) using partial pivoting. Interchanging rows 1 and 3, we obtain the system

$$\begin{bmatrix} 1.475 & 4.271 & 2.142 \\ 1.196 & 3.165 & 2.543 \\ .002 & 1.231 & 2.471 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7.888 \\ 6.904 \\ 3.704 \end{bmatrix}.$$

After one step the partially reduced matrix has the form

$$\left[\begin{array}{ccc} 1.475 & 4.271 & 2.142 \\ \hline .8108 & -.2980 & .8060 \\ 1.356 \times 10^{-3} & 1.225 & 2.468 \end{array}\right].$$

When you carry out this computation, you can see that the information in the $(2,2)$, $(2,3)$, $(3,2)$, and $(3,3)$ positions is not swamped. There is, however, a slight cancellation in the $(2,2)$ and $(2,3)$ positions. The partial pivoting strategy dictates that we interchange rows 2 and 3. In this way we avoid using the slightly inaccurate number $-.2980$ as a pivot. After step 2 the LU decomposition is complete:

$$\left[\begin{array}{ccc} 1.475 & 4.271 & 2.142 \\ \hline 1.356 \times 10^{-3} & 1.225 & 2.468 \\ .8108 & \multicolumn{1}{|c}{-.2433} & 1.407 \end{array}\right].$$

Forward substitution yields $y = [7.888,\ 3.693,\ 1.407]^T$, and back substitution gives the computed result

$$x = \begin{bmatrix} 1.000 \\ 1.000 \\ 1.000 \end{bmatrix}.$$

It is a matter of luck that the computed solution agrees with the true solution exactly, but it is not luck that the computation yielded an accurate result. Accuracy is guaranteed by the well-conditioned coefficient matrix together with Theorem 2.7.14. □

Exercise 2.6.13 Work through the details of the computations performed in Example 2.6.9. □

2.7 BACKWARD ERROR ANALYSIS OF GAUSSIAN ELIMINATION

A major operation in many algorithms, including Gaussian elimination, is the accumulation of sums

$$\sum_{j=1}^{n} w_j.$$

There are many ways to add n numbers together. For example, if we have four numbers, we can add them in the "natural" way: first we add w_1 to w_2, then we add on w_3, then w_4, that is, $((w_1 + w_2) + w_3) + w_4$, or we can apply this process to any reordering of $w_1, \ldots, w_4$. Another possibility is to add w_1 to w_2, add w_3 to w_4, then add the two intermediate sums, that is, $(w_1 + w_2) + (w_3 + w_4)$.

If we accumulate a sum in different ways using floating-point arithmetic, we will get different results, because the different ways have different roundoff errors. The relative differences in the computed sums will usually be tiny, but they can be large if there is a cancellation in the end (the summands add "nearly" to zero).

Our first task is to show that if a sum is accumulated in floating-point arithmetic, the computation is always backward stable, regardless of the manner in which the sum was accumulated. As before, we will let u be the unit roundoff. We will use the notation $O(u^2)$ to denote terms of order u^2. These are tiny terms that can be neglected. For example, $(1 + \alpha_1)(1 + \alpha_2) = (1 + \beta)$, where $\beta = \alpha_1 + \alpha_2 + \alpha_1\alpha_2$. If $|\alpha_1| \approx u$ and $|\alpha_2| \approx u$, then $\alpha_1\alpha_2 = O(u^2)$, and we write $\beta = \alpha_1 + \alpha_2 + O(u^2)$.

Proposition 2.7.1 *Suppose we compute the sum* $\sum_{j=1}^{n} w_j$ *using floating-point arithmetic with unit roundoff* u. *Then*

$$\mathrm{fl}\left(\sum_{j=1}^{n} w_j\right) = \sum_{j=1}^{n} w_j(1 + \gamma_j),$$

where $|\gamma_j| \leq (n-1)u + O(u^2)$, *regardless of the order in which the terms are accumulated.*

Proof. The proof is by induction on n. The proposition is trivially true when $n = 1$. Now let m be any positive integer. We shall show that the proposition holds for $n = m$, assuming that it holds for $n < m$, that is, for all sums of fewer than m terms.

Suppose we have reached the point in our computation of $\sum_{j=1}^{m} w_j$ at which we are within one addition of being done. At this point we will have accumulated two sums, one being the sum of, say, k of the w_j and the other being the sum of the other $m - k$. It might be that $k = 1$ or $k = m - 1$. In any event, $1 \leq k \leq m - 1$. For notational convenience relabel the terms so that the terms in the first sum are $w_1, \ldots, w_k$. Then the two sums that we have so far are

$$\sum_{j=1}^{k} w_j \quad \text{and} \quad \sum_{j=k+1}^{m} w_j.$$

Since each of these sums has fewer than m terms, we have, by the induction hypothesis,

$$\mathrm{fl}\left(\sum_{j=1}^{k} w_j\right) = \sum_{j=1}^{k} w_j(1+\alpha_j)$$

and

$$\mathrm{fl}\left(\sum_{j=k+1}^{m} w_j\right) = \sum_{j=k+1}^{m} w_j(1+\alpha_j),$$

where

$$|\alpha_j| \leq \begin{cases} (k-1)u + O(u^2) & \text{for } j = 1, \ldots, k, \\ (m-k-1)u + O(u^2) & \text{for } j = k+1, \ldots, m, \end{cases}$$

regardless of the order in which each of these sums was accumulated. Since $k-1 \leq m-2$ and $m-k-1 \leq m-2$, we have

$$|\alpha_j| \leq (m-2)u + O(u^2) \quad \text{for } j = 1, \ldots, m.$$

Adding the two sums together, we have

$$\begin{aligned} \mathrm{fl}\left(\sum_{j=1}^{m} w_j\right) &= \mathrm{fl}\left(\mathrm{fl}\left(\sum_{j=1}^{k} w_j\right) + \mathrm{fl}\left(\sum_{j=k+1}^{m} w_j\right)\right) \\ &= \left(\sum_{j=1}^{k} w_j(1+\alpha_j) + \sum_{j=k+1}^{m} w_j(1+\alpha_j)\right)(1+\beta) \\ &= \sum_{j=1}^{m} w_j(1+\alpha_j)(1+\beta). \end{aligned}$$

β is the roundoff error of the current addition and satisfies $|\beta| \leq u$ by (2.5.3). Let $\gamma_j = \alpha_j + \beta + \alpha_j\beta$, so that $(1+\gamma_j) = (1+\alpha_j)(1+\beta)$. Then

$$\begin{aligned} |\gamma_j| &\leq |\alpha_j| + |\beta| + O(u^2) \\ &\leq (m-2)u + O(u^2) + u + O(u^2) \\ &= (m-1)u + O(u^2). \end{aligned}$$

We have thus shown that

$$\mathrm{fl}\left(\sum_{j=1}^{m} w_j\right) = \sum_{j=1}^{m} w_j(1+\gamma_j),$$

where $|\gamma_j| \leq (m-1)u + O(u^2)$. This completes the proof. □

The numbers γ_j in Proposition 2.7.1 can be termed *relative backward errors*. There are at least three reasons why the inequalities $|\gamma_j| \leq (n-1)u + O(u^2)$

grossly overestimate $|\gamma_j|$. First of all, the factor $n-1$ reflects the fact that each summand participates in at most $n-1$ additions and is therefore subjected to at most $n-1$ roundoff errors. γ_j is approximately the sum of all the roundoff errors that occur in sums involving w_j (recall $\gamma_j \approx \alpha_j + \beta$). The exact number of roundoff errors that each term suffers depends on the method of summation and is, on average, much less than $n-1$. (You can clarify this for yourself by working Exercise 2.7.24.) Thus most γ_j are sums of many fewer than $n-1$ roundoffs.

Secondly, u is an upper bound on each roundoff error. A typical round off will be significantly less than u. Finally, and most importantly, when roundoff errors are added together, they sometimes reinforce one another and they sometimes (partially) cancel each other out. Bounds like $|\gamma_j| \leq (n-1)u + O(u^2)$ have to take into account the worst possible (and highly unlikely) case, where all roundoff errors are maximal and reinforce one another.

For these reasons, the γ_j are more likely to be much closer to u than $(n-1)u$, so the factor $(n-1)$ can be ignored in practice. Thus we consider Proposition 2.7.1 to be proof that the accumulation of sums is backward stable, regardless of how large n is.

Backward Stability of Forward and Back Substitution

Let G be a nonsingular, lower-triangular matrix, and let b be a nonzero vector. Then we can solve the system $Gy = b$ in about n^2 flops by forward substitution. Our next theorem shows that the forward substitution algorithm is backward stable. First we introduce some simplifying notation. Given an $n \times m$ matrix (or, in particular, a vector) C with (i,j) entry c_{ij}, we define $|C|$, the *absolute value* of C, to be the $n \times m$ matrix whose (i,j) entry is $|c_{ij}|$. Also, given two $n \times m$ matrices C and F, we will write $C \leq F$ if and only if $c_{ij} \leq f_{ij}$ for all i and j. With these new definitions, we can now make the following statement.

Theorem 2.7.2 *Let G be a nonsingular, lower-triangular matrix, and let $b \neq 0$. If the system $Gy = b$ is solved by any variant of forward substitution using floating-point arithmetic, then the computed solution $\hat{y}$ satisfies*

$$(G + \delta G)\hat{y} = b, \tag{2.7.3}$$

where δG satisfies

$$|\delta G| \leq 2nu|G| + O(u^2). \tag{2.7.4}$$

This inequality means that $|\delta g_{ij}| \leq 2nu|g_{ij}| + O(u^2)$ for all i and j. Thus the term $O(u^2)$ in (2.7.4) stands for an $n \times n$ matrix, each of whose entries is of order u^2.

This is not the tightest possible result. For a more careful argument that gets rid of the factor 2, see [41].

Proof. Once we have $y_1, \ldots, y_{i-1}$, we compute

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} g_{ij}y_j}{g_{ii}},$$

in principle. In practice we use the computed quantities $\hat{y}_1, \ldots, \hat{y}_{i-1}$ and make further rounding errors, so

$$\hat{y}_i = \mathrm{fl}\left(\frac{b_i - \sum_{j=1}^{i-1} g_{ij}\hat{y}_j}{g_{ii}} \right).$$

The numerator is just a sum of i terms, but before we can do any additions, we have to do the multiplications. We have $\mathrm{fl}(g_{ij}\hat{y}_j) = g_{ij}\hat{y}_j(1+\alpha_{ij})$, where $|\alpha_{ij}| \leq u$ by (2.5.3). Once we have the products, we can accumulate the numerator. Different variants of forward substitution will do this in different ways. Since it is a sum of i terms, Proposition 2.7.1 guarantees that no matter how it is done,

$$\mathrm{fl}\left(b_i - \sum_{j=1}^{i-1} g_{ij}\hat{y}_j(1+\alpha_{ij}) \right) = b_i(1+\gamma_{ii}) - \sum_{j=1}^{i-1} g_{ij}\hat{y}_j(1+\alpha_{ij})(1+\gamma_{ij}),$$

where $|\gamma_{ij}| \leq (i-1)u + O(u^2)$ for $j = 1, \ldots, i$. Once this is done, we obtain $\hat{y}_i$ by a division, which introduces one more rounding error:

$$\hat{y}_i = \left(\frac{b_i(1+\gamma_{ii}) - \sum_{j=1}^{i-1} g_{ij}\hat{y}_j(1+\alpha_{ij})(1+\gamma_{ij})}{g_{ii}} \right)(1+\beta_i), \tag{2.7.5}$$

where $|\beta_i| \leq u$.

We are aiming for the final result (2.7.3), in which all of the errors have been pushed back onto G and, in particular, not onto b. To obtain this effect we divide numerator and denominator in (2.7.5) by $(1+\gamma_{ii})(1+\beta_i)$ to get rid of the error terms that multiply b_i. If we define ϵ_{ij} through the equations

$$1+\epsilon_{ij} = \begin{cases} \frac{(1+\alpha_{ij})(1+\gamma_{ij})}{1+\gamma_{ii}} & \text{if } j < i, \\ \frac{1}{(1+\gamma_{ii})(1+\beta_i)} & \text{if } j = i, \end{cases} \tag{2.7.6}$$

we then have

$$\hat{y}_i = \frac{b_i - \sum_{j=1}^{i-1} g_{ij}(1+\epsilon_{ij})\hat{y}_j}{g_{ii}(1+\epsilon_{ii})}. \tag{2.7.7}$$

This last equation can be rewritten as

$$\sum_{j=1}^{i} g_{ij}(1+\epsilon_{ij})\hat{y}_j = b_i.$$

Since this holds for all i, we can write it as a single matrix equation

$$(G+\delta G)\hat{y} = b,$$

where δG is the lower-triangular matrix defined by $\delta g_{ij} = \epsilon_{ij} g_{ij}$ for $i \geq j$. To complete the proof we just have to show that $|\epsilon_{ij}| \leq 2nu + O(u^2)$ for all i and j.

Referring back to (2.7.6), we see that we need to deal with the factor $1/(1+\gamma_{ii})$. Since $\gamma_{ii} = O(u)$, we have

$$\frac{1}{1+\gamma_{ii}} = 1 - \gamma_{ii} + \gamma_{ii}^2 - \gamma_{ii}^3 + \cdots = 1 - \gamma_{ii} + O(u^2).$$

Thus, when $j < i$,

$$\begin{aligned} 1 + \epsilon_{ij} &= (1+\alpha_{ij})(1+\gamma_{ij})(1-\gamma_{ii}+O(u^2)) \\ &= 1 + \alpha_{ij} + \gamma_{ij} - \gamma_{ii} + O(u^2), \end{aligned}$$

and

$$\begin{aligned} |\epsilon_{ij}| &\leq |\alpha_{ij}| + |\gamma_{ij}| + |\gamma_{ii}| + O(u^2) \\ &\leq u + (i-1)u + (i-1)u + O(u^2) \\ &= (2i-1)u + O(u^2) \leq 2nu + O(u^2). \end{aligned}$$

By a similar analysis we get that $|\epsilon_{ii}| \leq iu + O(u^2) \leq 2nu + O(u^2)$. □

Exercise 2.7.8 Check that (2.7.7) is valid for $i = 1$. □

Corollary 2.7.9 *Under the conditions of Theorem 2.7.2 the computed solution $\hat{y}$ satisfies*

$$(G + \delta G)\hat{y} = b,$$

where

$$\|\delta G\|_\infty \leq 2nu\|G\|_\infty + O(u^2). \tag{2.7.10}$$

Proof. The bound (2.7.10) follows directly from (2.7.4), using the properties proved in Exercise 2.7.11. □

Exercise 2.7.11

(a) Show that if $|C| \leq |F|$ (elementwise), then $\|C\|_\infty \leq \|F\|_\infty$.

(b) Show that $\|C\|_\infty = \| \, |C| \, \|_\infty$.

□

The properties established in Exercise 2.7.11 hold also for the matrix 1-norm and the Frobenius norm. Corollary 2.7.9 holds for any norm that satisfies these properties.

We have already noted that the factor 2 in the bounds (2.7.4) and (2.7.10)) can be eliminated. The factor n can also be ignored, just as in Proposition 2.7.1.

Corollary 2.7.9 shows that forward substitution is *normwise backward stable*; that is, the computed solution $\hat{y}$ is the exact solution of a nearby problem $(G + \delta G)\hat{y} = b$, where $\|\delta G\|_\infty / \|G\|_\infty$ is tiny.

Theorem 2.7.2 is actually a much stronger result. It states not just that $\|\delta G\|$ is tiny relative to $\|G\|$, but each element perturbation δg_{ij} is tiny relative to g_{ij}, the element it is perturbing. This property is called *componentwise backward stability*.

Exercise 2.7.12 Construct an example of 2×2 matrices G and δG such that $\| \delta G \|_\infty / \| G \|_\infty$ is tiny but $| \delta g_{ij} | / | g_{ij} |$ is not tiny for at least one component (i, j). □

Theorems essentially identical to Theorem 2.7.2 and Corollary 2.7.9 hold for back substitution applied to upper-triangular systems. We see no need to state these results.

Backward Error of Gaussian Elimination

Gaussian elimination is sometimes backward stable, sometimes not, depending on circumstances. The basic results are the following two theorems, which are statements about Gaussian elimination without pivoting. However, both of these results are applicable to Gaussian elimination with row and column interchanges, since the latter is equivalent to Gaussian elimination without interchanges, applied to a matrix whose rows and columns were interchanged in advance. Thus these results can be applied to Gaussian elimination with partial pivoting, complete pivoting, or any other pivoting strategy.

Theorem 2.7.13 *Suppose the LU decomposition of A is computed by Gaussian elimination in floating-point arithmetic, and suppose no zero pivots are encountered in the process. Let $\hat{L}$ and $\hat{U}$ denote the computed factors. Then*

$$A + E = \hat{L}\hat{U},$$

where

$$| E | \leq 2nu \, | \hat{L} | \, | \hat{U} | + O(u^2)$$

and

$$\| E \|_\infty \leq 2nu \, \| \hat{L} \|_\infty \, \| \hat{U} \|_\infty + O(u^2).$$

Theorem 2.7.14 *Under the conditions of Theorem 2.7.13, suppose we solve $Ax = b$ numerically by performing forward substitution with $\hat{L}$ followed by back substitution with $\hat{U}$ in floating-point arithmetic. Then the computed solution $\hat{x}$ satisfies*

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

where

$$| \delta A | \leq 6nu \, | \hat{L} | \, | \hat{U} | + O(u^2)$$

and

$$\| \delta A \|_\infty \leq 6nu \, \| \hat{L} \|_\infty \, \| \hat{U} \|_\infty + O(u^2).$$

We will defer the proofs of these results until after we have discussed their implications.[2]

[2]These theorems have been stated in the form in which they will be proved. They are not the best possible results; a factor of 2 can be removed from each of the bounds.

Looking at either of these Theorems, we see that whether or not Gaussian elimination is backward stable depends upon how big $\hat{L}$ and $\hat{U}$ are. If $\| \hat{L} \|_\infty \| \hat{U} \|_\infty$ is only a modest multiple of $\| A \|_\infty$, then we can conclude that $\| \delta A \|_\infty / \| A \|_\infty$ is a modest multiple of the unit roundoff, and the operation is backward stable. (As before, we ignore the factor n.) If, on the other hand, $\| \hat{L} \|_\infty \| \hat{U} \|_\infty$ is much larger than $\| A \|_\infty$, then we can draw no such conclusion. In this case the computation is probably not backward stable.

Let us first consider Gaussian elimination without pivoting. The use of small pivots can result in large multipliers, which are entries of $\hat{L}$. Thus $\| \hat{L} \|_\infty$ can be arbitrarily large. The large multipliers cause large multiples of some rows to be added to other rows, with the effect that $\| \hat{U} \|_\infty$ is also large. These effects are seen in Example 2.6.9. There we demonstrated how the unnecessary use of small pivots can destroy the accuracy of Gaussian elimination. See Exercise 2.7.25 as well. We conclude that Gaussian elimination without pivoting is unstable.

Now consider partial pivoting. This guarantees that all of the multipliers $\hat{l}_{ij}$ have modulus less than or equal to 1 and has the tendency of keeping the norm of $\hat{L}$ from being too large. In fact $\| \hat{L} \|_\infty \leq n$. Thus, to guarantee backward stability, we need only show that $\| \hat{U} \|_\infty / \| A \|_\infty$ cannot be too large. Unfortunately there exist matrices for which $\| \hat{U} \|_\infty / \| A \|_\infty \approx 2^{n-1}$. An example is given in Exercise 2.7.26. Because 2^{n-1} is enormous even for modest values of n (e.g. $2^{n-1} > 10^{29}$ when $n = 100$), we cannot claim that Gaussian elimination with partial pivoting is unconditionally backward stable, except when n is quite small.

Despite this bad news, partial pivoting is now and will continue to be widely used. For example, it is the main method for solving $Ax = b$ in MATLAB and LAPACK. Years of testing and experience have shown that the type of element growth exhibited by the matrix in Exercise 2.7.26 is extremely rare in practice. Typically we see

$$\frac{\| \hat{U} \|_\infty}{\| A \|_\infty} \approx \sqrt{n}.$$

Hence, for practical purposes, Gaussian elimination with partial pivoting is considered to be a stable algorithm and is used with confidence.

A statistical explanation of the good behavior of partial pivoting is given by Trefethen and Bau [71]. See also Exercise 2.7.27.

Since partial pivoting might occasionally perform badly, one would like to have a way of checking whether one's results are good or not. Fortunately such tests exist. One such test would be simply to compute the ratio

$$\| \hat{L} \|_\infty \| \hat{U} \|_\infty / \| A \|_\infty.$$

If this is not too large, the computation was backward stable. An even simpler test, once $\hat{x}$ has been computed, is to calculate the residual $\hat{r} = b - A\hat{x}$. As we have remarked before, $A\hat{x} = b + \delta b$, where $\delta b = -r$. (See also Exercise 2.5.6.) Thus a small residual implies backward stability.

Another test is simply to compute the backward error in the LU decomposition: $E = \hat{L}\hat{U} - A$. If $\| E \|$ is tiny, the computation was backward stable. However, this

test is expensive. The computation of $\hat{L}\hat{U}$ costs $O(n^3)$ flops, even if the triangular form of the matrices is taken into account (Exercise 2.7.28).

Example 2.7.15 Using MATLAB we computed the LU decomposition (with partial pivoting) of the 12×12 Hilbert matrix (`A = hilb(12); [L,U] = lu(A);`) and then computed $E = LU - A$. We found that $\|E\|_\infty / \|A\|_\infty \approx 2.7 \times 10^{-17}$. Also, $\|L\|_\infty \approx 5.6$, $\|U\|_\infty \approx 3.1$, and $\|L\|_\infty \|U\|_\infty / \|A\|_\infty \approx 5.6$. All of these results signal stability. This shows that the poor results obtained in Exercises 2.6.7 and (especially) 2.6.6 are due entirely to the ill conditioning of the Hilbert matrix, not to instability of the algorithm. In those exercises you also computed residuals and found that they are tiny. This also demonstrates backward stability. □

Exercise 2.7.16

(a) Perform the computations indicated in Example 2.7.15 for Hilbert matrices of dimension 12, 24, and 48. Notice that we have stability in every case.

(b) Repeat part (a) using Lotkin Matrices (`A = gallery('lotkin',n)`).

□

Complete pivoting is better behaved than partial pivoting. Element growth of the type exhibited by partial pivoting in Exercise 2.7.26 is impossible. In the worst known cases $\|\hat{U}\|/\|A\| = O(n)$. It is also true that $\|\hat{L}_\infty\| \leq n$, as for partial pivoting. Thus Gaussian elimination with complete pivoting is considered to be a backward stable algorithm.

In spite of the theoretical superiority of complete pivoting over partial pivoting, the latter is much more widely used. The reasons are simple: 1.) Partial pivoting works well in practice, and it is significantly less expensive. 2.) Inexpensive *a posteriori* stability tests exist.

Theorem 2.7.14 is also important for Gaussian elimination in sparse matrices. Here there are conflicting objectives: one wishes not only to perform the elimination in a stable manner, but also to keep fill-in as small as possible. One might therefore pursue a strategy that does not always select the largest possible pivots. The stability of the decomposition can be monitored by checking the size of the entries of $\hat{L}$ and $\hat{U}$ as they are produced.

For symmetric, positive definite systems a result like Theorem 2.7.14 holds for Cholesky's method with $\hat{L}$ and $\hat{U}$ replaced by $\hat{R}^T$ and $\hat{R}$, respectively. It can also be shown that $\|\hat{R}^T\|_F \|\hat{R}\|_F$ cannot be large relative to $\|A\|_F$. Therefore Cholesky's method is unconditionally backward stable. You can work out the details in Exercise 2.7.29.

In summary, Gaussian elimination with partial or complete pivoting and Cholesky's method for positive definite systems are stable in practice. For partial pivoting this assertion is based upon years of experience; for Cholesky's method it is an iron-clad fact. The factors of n that appear in the theorems are gross overestimates and can be ignored. In practice we get a computed solution $\hat{x}$ satisfying $(A + \delta A)\hat{x} = b$, where $\|\delta A\|/\|A\| \approx Cu$, with C a modest multiple of 1. Thus the total effect of roundoff

errors is not much greater than that of the initial roundoff errors in the representation of A. These errors are usually much smaller than the original measurement errors in A and b. These considerations lead to a useful rule of thumb.

Rule of Thumb 2.7.17 *Suppose the linear system $Ax = b$ is solved by Gaussian elimination with partial or complete pivoting (or by Cholesky's method in the positive definite case). If the entries of A and b are accurate to about s decimal places and $\kappa(A) \approx 10^t$, where $t < s$, then the entries of the computed solution are accurate to about $s - t$ decimal places.*

"Proof." We intended to solve $Ax = b$, but our computed solution $\hat{x}$ satisfies a perturbed equation $(A + \delta A)\hat{x} = b + \delta b$, where δA is the sum of measurement error, initial rounding error, and the effect of roundoff errors made during the computation, and δb is the sum of measurement error and initial rounding error. We assume that measurement error dominates. Since the entries of A and b are accurate to about s decimal places,

$$\frac{\|\delta A\|}{\|A\|} \approx 10^{-s} \quad \text{and} \quad \frac{\|\delta b\|}{\|b\|} \approx 10^{-s}.$$

Preparing to apply Theorem 2.3.9, we note that

$$\kappa(A)\frac{\|\delta A\|}{\|A\|} \approx 10^{t-s} \ll 1 \quad \text{so} \quad 1 - \kappa(A)\frac{\|\delta A\|}{\|A\|} \approx 1.$$

Thus Theorem 2.3.9 gives roughly

$$\frac{\|\delta x\|}{\|x\|} \approx \kappa(A)\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right) \approx 10^{t-s},$$

where $\hat{x} = x + \delta x$. That is, the entries of $\hat{x}$ are accurate to about $s - t$ decimal places. □

Proofs of Theorems 2.7.13 and 2.7.14

Proof of Theorem 2.7.13. We begin by recalling (cf. (1.7.22) and (1.7.24)) that if $A = LU$, then L and U are given by the formulas

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}}{u_{jj}} \quad \text{for } i > j \tag{2.7.18}$$

and

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} \quad \text{for } i \leq j. \tag{2.7.19}$$

All versions of Gaussian elimination perform the computations indicated in (2.7.18) and (2.7.19), but different versions organize the computations differently.

Let us first consider the computation of l_{ij}. In practice the computed values $\hat{l}_{ik}$ and $\hat{u}_{kj}$ are used, and further roundoff errors occur. Thus the computed $\hat{l}_{ij}$ satisfies

$$\hat{l}_{ij} = \mathrm{fl}\left(\frac{a_{ij} - \sum_{k=1}^{j-1} \hat{l}_{ik}\hat{u}_{kj}}{\hat{u}_{jj}} \right).$$

Different versions of Gaussian elimination accumulate the sum in the numerator in different ways. Proposition 2.7.1 shows that no matter how it is done,

$$\hat{l}_{ij} = \frac{a_{ij}(1+\gamma_{ij}) - \sum_{k=1}^{j-1} \hat{l}_{ik}\hat{u}_{kj}(1+\alpha_k)(1+\gamma_{ik})}{\hat{u}_{jj}}(1+\beta), \tag{2.7.20}$$

where $|\gamma_{ij}| \leq (j-1)u + O(u^2)$. The quantities α_k and β are the roundoff errors associated with the multiplications and the division, respectively, and they satisfy $|\alpha_k| \leq u$ and $|\beta| \leq u$.

Proceeding just as in the proof of Theorem 2.7.2, we divide the numerator and denominator in (2.7.20) by $(1+\gamma_{ij})(1+\beta)$ to remove the error from the a_{ij} term. This move is not strictly necessary, but it yields a more elegant result. We then simplify the resulting equation by consolidating the errors. Define δ_{ik} by

$$1 + \delta_{ik} = \begin{cases} \frac{(1+\alpha_k)(1+\gamma_{ik})}{1+\gamma_{ij}} & \text{if } k < j, \\ \frac{1}{(1+\gamma_{ij})(1+\beta)} & \text{if } k = j. \end{cases}$$

Recalling that $1/(1+\gamma_{ij}) = 1 - \gamma_{ij} + \gamma_{ij}^2 - \gamma_{ij}^3 + \cdots$, we have, for $k < j$, $\delta_{ik} = \alpha_k + \gamma_{ik} - \gamma_{ij} + O(u^2)$, and

$$\begin{aligned} |\delta_{ik}| &\leq |\alpha_k| + |\gamma_{ik}| + |\gamma_{ij}| + O(u^2) \\ &\leq u + (j-1)u + O(u^2) + (j-1)u + O(u^2) = (2j-1)u + O(u^2). \end{aligned}$$

Similarly, $\delta_{ij} = -\gamma_{ij} - \beta + O(u^2)$, and

$$\begin{aligned} |\delta_{ij}| &\leq |\gamma_{ij}| + |\beta| + O(u^2) \\ &\leq (j-1)u + O(u^2) + u + O(u^2) = ju + O(u^2). \end{aligned}$$

Thus

$$|\delta_{ik}| \leq 2nu + O(u^2), \quad k = 1, \ldots, j.$$

In terms of δ_{ik}, (2.7.20) becomes

$$\hat{l}_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} \hat{l}_{ik}\hat{u}_{kj}(1+\delta_{ik})}{\hat{u}_{jj}(1+\delta_{ij})}.$$

Multiplying through by $\hat{u}_{jj}(1+\delta_{ij})$ and rewriting the resulting expression so that all of the error terms are consolidated into a single term e_{ij}, we have

$$a_{ij} + e_{ij} = \sum_{k=1}^{j} \hat{l}_{ik}\hat{u}_{kj} \tag{2.7.21}$$

for $i > j$ (because we started from (2.7.18), which holds only for $i > j$), where

$$e_{ij} = -\sum_{k=1}^{j} \hat{l}_{ik}\hat{u}_{kj}\delta_{ik}.$$

Since $|\delta_{ik}| \leq 2nu + O(u^2)$,

$$|e_{ij}| \leq 2nu \sum_{k=1}^{j} |\hat{l}_{ik}|\,|\hat{u}_{kj}| + O(u^2) \tag{2.7.22}$$

for $i > j$.

We obtained (2.7.21) and (2.7.22) for $i > j$ by starting from (2.7.18). The same results can be obtained for $i \leq j$ by performing a similar analysis starting from (2.7.19). Thus (2.7.21) and (2.7.22) hold for all i and j. Writing (2.7.21) as a matrix equation, we have

$$A + E = \hat{L}\hat{U}.$$

Writing (2.7.22) as a matrix inequality, we have

$$|E| \leq 2nu|\hat{L}|\,|\hat{U}| + O(u^2).$$

The bound on $\|E\|_\infty$ follows immediately from the bound on $|E|$, using the properties of the matrix ∞-norm established in Exercise 2.7.11. □

Exercise 2.7.23 Starting from (2.7.19), demonstrate that (2.7.21) and (2.7.22) hold for $i \leq j$. □

Proof of Theorem 2.7.14. In the forward substitution phase we compute $\hat{y}$ such that $(\hat{L} + \delta\hat{L})\hat{y} = b$, where

$$|\delta\hat{L}| \leq 2nu\,|\hat{L}|$$

by Theorem 2.7.2. In the back substitution phase we compute $\hat{x}$ such that $(\hat{U} + \delta\hat{U})\hat{x} = \hat{y}$, where

$$|\delta\hat{U}| \leq 2nu\,|\hat{U}|$$

by the upper-triangular analogue of Theorem 2.7.2. Thus $\hat{x}$ satisfies exactly

$$(\hat{L} + \delta\hat{L})(\hat{U} + \delta\hat{U})\hat{x} = b.$$

Multiplying out the product of matrices and using the fact (Theorem 2.7.13) that $\hat{L}\hat{U} = A + E$, where $|E| \leq 2nu\,|\hat{L}|\,|\hat{U}| + O(u^2)$, we have

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

where

$$\delta A = E + (\delta\hat{L})\hat{U} + \hat{L}(\delta\hat{U}) + (\delta\hat{L})(\delta\hat{U}).$$

Applying the upper bounds that we have for E, $\delta\hat{L}$, and $\delta\hat{U}$, we obtain

$$|\delta A| \leq 6nu\,|\hat{L}|\,|\hat{U}| + O(u^2).$$

The bound on $\|\delta A\|_\infty$ follows immediately from this bound. □

Additional Exercises

Exercise 2.7.24 In Proposition 2.7.1 we showed that summation is backward stable, regardless of the order. In the notation of Proposition 2.7.1 the backward errors γ_j are bounded by $|\gamma_j| \leq (n-1)u + O(u^2)$. In this exercise we obtain tighter bounds on $|\gamma_j|$ for two specific orders of summation of $\sum_{j=1}^{n} w_j$.

(a) First we consider the "obvious" order, reversed for notational convenience. Show that if we perform the computation in the order

$$(\cdots(((w_n + w_{n-1}) + w_{n-2})) + \cdots + w_2) + w_1$$

in floating-point arithmetic, then

$$|\gamma_j| \leq \begin{cases} ju + O(u^2) & \text{if } j < n, \\ (n-1)u + O(u^2) & \text{if } j = n. \end{cases}$$

(Look at small cases like $n = 3$ and $n = 4$, and observe the pattern.)

(b) Now consider summing by pairs. Calculate $w_1 + w_2$, $w_3 + w_4$, $w_5 + w_6$, and so on. If there is an odd term, just let it sit. Now you have a new list, which you can sum by pairs. Keep summing by pairs until you have a single sum. This is easiest to discuss when n is a power of 2, say $n = 2^k$. In this case, how many additions does each term participate in? Express your answer in terms of n. Show that if we carry out this process in floating-point arithmetic, we have

$$|\gamma_j| \leq (\log_2 n)u + O(u^2).$$

This is an excellent result, as $\log_2 n$ grows much more slowly than n.

□

Exercise 2.7.25 In this exercise you will assess the backward stability of Gaussian elimination by calculating backward error in the LU decomposition: $E = \hat{L}\hat{U} - A$. Write a MATLAB program that does Gaussian elimination without pivoting, for example

```
A = randn(n);
L = zeros(n); U = zeros(n);
for k = 1:n
  U(k,k:n) = a(k,k:n) - L(k,1:k-1)*U(1:k-1,k:n);
```

```
    L(k+1:n,k) = (a(k+1:n,k) - ...
                  L(k+1:n,1:k-1)*U(1:k-1,k))/U(k,k);
end
L = L + eye(n);
```

(a) Calculate the LU decomposition of random matrices (`A = randn(n);`) for several choices of n (e.g. $n = 40, 80, 160$), and note $\|L\|_\infty$, $\|U\|_\infty$, and the norm of the backward error: $\|E\|_\infty = \|LU - A\|_\infty$. On the same matrices do Gaussian elimination with partial pivoting (`[L,U] = lu(A);`) and calculate the same quantities. Notice that partial pivoting decreases the backward error and the norms of L and U, but the performance of Gaussian elimination without pivoting is usually not conspicuously bad. That is, usually Gaussian elimination without pivoting is able to calculate the LU decomposition (of a random matrix) more or less stably.

(b) To demonstrate the weakness of Gaussian elimination without pivoting, give it a matrix for which at least one of the pivots is guaranteed to be small. The easiest way to do this is to use matrices whose $(1,1)$ entry is tiny. Repeat the experiments from part (a) using matrices for which a_{11} is tiny. For example, take `A = randn(n); A(1,1) = 50*eps*A(1,1);`. MATLAB's *machine epsilon* `eps` equals $2u$, twice the unit roundoff. For these experiments n need not be large. Try several choices of n, but $n = 2$ is already big enough.

□

Exercise 2.7.26 Let A_n denote the $n \times n$ matrix whose form is illustrated by

$$A_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}.$$

Show that if Gaussian elimination with partial pivoting is used, then A_n can be reduced to upper-triangular form without row interchanges, and the resulting matrix U has $u_{nn} = 2^{n-1}$. Thus $\|U\|_\infty / \|A\|_\infty = 2^{n-1}/n$. □

Exercise 2.7.27 It is pointed out in [71] that the LU decomposition of a random matrix is anything but random. If $A = LU$, then $U = L^{-1}A$. We have stability if $\|U\|$ is not too much larger than $\|A\|$, and this will be the case if $\|L^{-1}\|$ is not large. In other words, a necessary condition for instability is that $\|L^{-1}\|$ be large. In this exercise we will see by experiment that Gaussian elimination with partial pivoting tends to return matrices for which L^{-1} is not large.

(a) Write a MATLAB program that generates random unit lower-triangular matrices with entries between 1 and -1. For example, you can start with the identity

matrix (`L = eye(n)`) and then fill in the lower triangular part. The command `L(i,j) = 2*rand-1` gives a random number uniformly distributed in $[-1, 1]$. Calculate the norm of L^{-1} for several such matrices with $n = 40$, 80, and 160.

(b) Now generate the LU factors of random $n \times n$ matrices by

```
A = randn(n);
[L,U,P] = lu(A);
```

MATLAB's `lu` command does Gaussian elimination with partial pivoting. This way of using it produces a truly unit lower-triangular matrix L; the row interchanges are incorporated into the permutation matrix P. Calculate $\| L^{-1} \|$ for several matrices generated in this way with $n = 40$, 80, and 160. Contrast your results with those of part (a).

(c) Extend your code from part (b) so that it computes not only $\| L^{-1} \|$ but also $\| M^{-1} \|$, where M is generated from L by reversing the signs of the pivots: $M = 2I - L$.

□

Exercise 2.7.28 Show that if $L \in \mathbb{R}^{n \times n}$ and $U \in \mathbb{R}^{n \times n}$ are full lower and upper triangular, respectively, the flop count for computing the product LU is $\frac{2}{3}n^3$. □

Exercise 2.7.29 Here we prove the backward stability of Cholesky's method in detail. Let A be a positive definite matrix, and let $\hat{R}$ denote its Cholesky factor computed by some variant of Cholesky's method in floating-point arithmetic. Assume that square roots are calculated accurately: $\mathrm{fl}(\sqrt{s}) = \sqrt{s}\,(1 + \epsilon)$, where $|\epsilon| \leq u$.

(a) Using the proof of Theorem 2.7.13 as a model, prove that $A + E = \hat{R}^T \hat{R}$, where

$$|E| \leq 2nu\,|\hat{R}|^T\,|\hat{R}| + O(u^2)$$

and

$$\| E \|_F \leq 2nu \| \hat{R} \|_F^2 + O(u^2).$$

(b) The *trace* of a matrix $B \in \mathbb{R}^{n \times n}$ is $\mathrm{tr}(B) = \sum_{i=1}^{n} b_{ii}$. Use the Cauchy-Schwarz inequality to prove that $|\mathrm{tr}(B)| \leq \sqrt{n}\, \| B \|_F$. (Notice that equality is attained when $B = I$. More commonly $|\mathrm{tr}(B)| \approx \| B \|_F$.)

(c) Prove that if $A + E = \hat{R}^T \hat{R}$, then $\| \hat{R} \|_F^2 = \mathrm{tr}(A + E) = \mathrm{tr}(A) + \mathrm{tr}(E)$. (This holds regardless of whether or not $\hat{R}$ is triangular.) Thus $\| \hat{R} \|_F^2 \leq \sqrt{n}\,(\| A \|_F + \| E \|_F)$.

(d) Substituting this last inequality into the result of part (a), show that

$$\| E \|_F \leq 2n^{3/2} u\,(\| A \|_F + \| E \|_F) + O(u^2)$$

and, if $2n^{3/2}u < 1$,

$$\|E\|_F \leq \frac{2n^{3/2}u}{1 - 2n^{3/2}u}\|A\|_F + O(u^2).$$

For realistic values of n and u we generally have $2n^{3/2}u \ll 1$, so the denominator in this expression is about 1. (Consider, e.g., the huge value $n = 10^6$ and IEEE double precision's $u \approx 10^{-16}$.) Furthermore, the factor $2n^{3/2}$ in the numerator is based on a pessimistic worst-case analysis and can be ignored. Thus we have, in practice,

$$\|E\|_F \approx Cu\|A\|_F,$$

where C is a modest multiple of 1, not $n^{3/2}$. Thus Cholesky's method for positive definite matrices is stable.

□

2.8 SCALING

In a linear system $Ax = b$, any equation can be multiplied by any nonzero constant without changing the solution of the system. Such an operation is called a *row scaling* operation. A similar operation can be applied to a column of A. In contrast to row scaling operations, *column scaling* operations do change the solution.

Exercise 2.8.1 Show that if the nonsingular linear system $Ax = b$ is altered by multiplication of its jth column by $c \neq 0$, then the solution is altered only in the jth component, which is multiplied by $1/c$. □

Scaling operations can be viewed as changes of measurement units. suppose the entries of the jth column of A are masses expressed in grams, and x_j is an acceleration measured in meters/sec^2. Multiplication of the jth column by $1/1000$ is the same as changing the units of its entries from grams to kilograms. At the same time x_j is multiplied by 1000, which is the same as changing its units from meters/second^2 to millimeters/second^2.

A discussion of scaling operations is made necessary by the fact that these operations affect the numerical properties of the system. This discussion has been placed near the end of the chapter because in most cases rescaling is unnecessary and undesirable; usually an appropriate scaling is determined by the physical units of the problem. Consider for example the electrical circuit problem in Example 1.2.6. In the linear system derived there, all entries of the coefficient matrix have the same units (1/ohm), all components of the solution have the same units (volts), and all components of the right-hand side have the same units (amperes). One could rescale this system so that, for example, one of the unknowns is expressed in millivolts while the others remain in volts, but this should not be done without a good reason. In most cases it is best not to rescale.

Let us look at some examples that illustrate some of the effects of scaling.

Example 2.8.2 The first example shows that a small pivot cannot be "cured" by multiplying its row by a large number. Consider the system

$$\begin{bmatrix} 2.000 & 1231. & 2471. \\ 1.196 & 3.165 & 2.543 \\ 1.475 & 4.271 & 2.142 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3704. \\ 6.904 \\ 7.888 \end{bmatrix},$$

which was obtained from the system (2.6.10) of Example 2.6.9 by multiplying the first row by 1000. We used (2.6.10) to illustrate the damaging effects of using a small number as a pivot. Now that the first row has been multiplied by 1000, the $(1,1)$ entry is no longer small. It is now the largest entry in the first column. Let us see what happens when it is used as a pivot.

Using four-digit decimal arithmetic, the multipliers for the first step are

$$l_{21} = \frac{1.196}{2.000} = .5980 \quad \text{and} \quad l_{31} = \frac{1.475}{2.000} = .7375.$$

Comparing these with (2.6.11), we see that they are 1000 times smaller than before. In step 1 the $(2,2)$ entry is altered as follows:

$$3.165 - (.5980)(1231.) = 3.165 - 736.1 = -732.9.$$

Comparing this with (2.6.12), we see that in spite of the smaller pivot, the outcome is the same as before. This time the number 3.165 is swamped by the large entry 1231. Notice that this computation is essentially identical with (2.6.12). The result is *exactly* the same, including the roundoff errors. You can check that when the $(2,3)$, $(3,2)$, and $(3,3)$ entries are modified, swamping occurs just as before, and indeed the computations are essentially the same as before and yield exactly the same result. Thus, after the first step, the modified coefficient matrix is

$$\begin{bmatrix} 2.000 & 1231. & 2471. \\ \hline .5980 & -732.9 & -1475. \\ .7375 & -903.6 & -1820. \end{bmatrix}.$$

The submatrix for the second step is

$$\begin{bmatrix} -732.9 & -1475. \\ -903.6 & -1820. \end{bmatrix},$$

which is exactly the same as before. If we continue the computation, we will have the same disastrous outcome. This time swamping occurred not because large multiples of the first row were subtracted from the other rows, but because the first row itself is large.

How could this disaster have been predicted? Looking at the coefficient matrix, we can see that it is ill conditioned: the rows (and the columns) are out of scale. It is interesting that we have two different explanations for the same disaster. With

the original system we blamed a small pivot; with the rescaled system we blame ill conditioning. □

This example illustrated an interesting theorem of F. L. Bauer. Suppose we solve the system $Ax = b$ by Gaussian elimination, using some specified sequence of row and column interchanges, on a computer that uses base β floating-point arithmetic. If the system is then rescaled by multiplying the rows and columns by powers of β and solved again using the same sequence of row and column interchanges, the result will be exactly the same as before, including roundoff errors. All roundoff errors at all steps are the same as before. In our present example $\beta = 10$; multiplying the first row by 10^3 has no effect on the arithmetic. It is not hard to prove Bauer's theorem; you might like to do so as an exercise or at least convince yourself that it is true. The examples that follow should help.

Bauer's theorem has an interesting consequence. If the scaling factors are always chosen to be powers of β, then the only way rescaling affects the numerical properties of Gaussian elimination is by changing the choices of pivot. If scaling factors that are not powers of β are used, there will be additional roundoff errors associated with the rescaling, but it remains true that the principal effect of rescaling is to alter the pivot choices.

Example 2.8.3 Let us solve the system

$$\begin{bmatrix} .003 & .217 \\ .277 & .138 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} .437 \\ .553 \end{bmatrix} \tag{2.8.4}$$

using three-digit decimal floating-point arithmetic without row or column interchanges. The exact solution is $x = [1,\ 2]^T$. The multiplier is $l_{21} = .277/.003 = 92.3$, and $u_{22} = .138 - (92.3)(.217) = .138 - 20.0 = -19.9$, so the computed LU decomposition is

$$\begin{bmatrix} 1 & 0 \\ 92.3 & 1 \end{bmatrix} \begin{bmatrix} .003 & .217 \\ 0 & -19.9 \end{bmatrix}.$$

The forward substitution gives $y_1 = .437$ and $y_2 = .553 - (92.3)(.437) = .553 - 40.3 = -39.7$. Finally the back substitution gives $x_2 = (-39.7)/(-19.9) = 1.99$ and $x_1 = .437 - (.217)(1.99)]/(.003) = (.437 - .432)/(.003) = (.005)/(.003) = 1.67$. Thus the computed solution is $\hat{x} = [1.67,\ 1.99]^T$, whose first component is inaccurate. □

Exercise 2.8.5

(a) Calculate $\kappa_\infty(A)$, where A is the coefficient matrix of (2.8.4). Observe that A is well conditioned.

(b) Perform Gaussian elimination on (2.8.4) with the rows interchanged, using three-digit decimal floating-point arithmetic, and note that an accurate solution is obtained. (Remember to round each intermediate result to three decimal places before using it in the next calculation.)

□

Example 2.8.6 Now let us solve

$$\begin{bmatrix} .300 & 21.7 \\ .277 & .138 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 43.7 \\ .553 \end{bmatrix}, \tag{2.8.7}$$

which was obtained by multiplying the first row of (2.8.4) by 10^2. Now the $(1,1)$ entry is the largest entry in the first column. Again we use three-digit decimal arithmetic and no row or column interchanges. By Bauer's theorem the outcome should be the same as in Example 2.8.3. Let us check that it is. The multiplier is $l_{21} = .277/.300 = .923$, and $u_{22} = .138 - (.923)(21.7) = .138 - 20.0 = -19.9$, so the computed LU decomposition is

$$\begin{bmatrix} 1 & 0 \\ .923 & 1 \end{bmatrix} \begin{bmatrix} .300 & 21.7 \\ 0 & -19.9 \end{bmatrix}.$$

The forward substitution gives $y_1 = 43.7$ and $y_2 = .553 - (.923)(43.7) = .553 - 40.3 = -39.7$. Finally the back substitution yields $x_2 = (-39.7)/(-19.9) = 1.99$ and $x_1 = 43.7 - (21.7)(1.99)]/(.300) = (43.7 - 43.2)/(.300) = (.500)/(.300) = 1.67$. Thus the computed solution is again $\hat{x} = [1.67,\ 1.99]^T$. All intermediate results are identical to those in Example 2.8.3, except for powers of 10. □

Exercise 2.8.8

(a) Calculate $\kappa_\infty(A)$, where A is the coefficient matrix of (2.8.7). A is ill conditioned (relative to three-digit decimal arithmetic) because its rows (and columns) are out of scale.

(b) Perform Gaussian elimination on (2.8.7) with the rows interchanged, using three-digit decimal arithmetic. Note that, as guaranteed by Bauer's theorem, the computations and outcome are identical to those in part (b) of Exercise 2.8.5. Thus an ill-conditioned coefficient matrix does not absolutely guarantee an inaccurate result. (However, if the partial-pivoting strategy had been used, the row interchange would not have been made, and the outcome would have been bad.)

□

Exercise 2.8.9 Solve (2.8.7) by Gaussian elimination with the columns interchanged, using three-digit decimal arithmetic. This is the complete pivoting strategy. Note that a good result is obtained. □

Example 2.8.10 Now let us solve

$$\begin{bmatrix} .300 & .217 \\ .277 & .00138 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 43.7 \\ .553 \end{bmatrix}, \tag{2.8.11}$$

which was obtained from (2.8.7) by multiplying the second column by 1/100. The exact solution is therefore $x = [1,\ 200]^T$. The $(1,1)$ entry is now the largest

entry in the matrix. Again we use three-digit decimal arithmetic and no row or column interchanges (which is the choice that both the partial and complete pivoting strategies would make). By Bauer's theorem the outcome should be the same as in Examples 2.8.3 and 2.8.6. The multiplier is $l_{21} = .277/.300 = .923$, and $u_{22} = .00138 - (.923)(.217) = .00138 - .200 = -.199$, so the computed LU decomposition is

$$\begin{bmatrix} 1 & 0 \\ .923 & 1 \end{bmatrix} \begin{bmatrix} .300 & .217 \\ 0 & -.199 \end{bmatrix}.$$

The forward substitution gives $y_1 = 43.7$ and $y_2 = .553 - (.923)(43.7) = .553 - 40.3 = -39.7$. Finally the back substitution yields $x_2 = (-39.7)/(-.199) = 199.$, and $x_1 = 43.7 - (.217)(199.)]/(.300) = (43.7 - 43.2)/(.300) = (.500)/(.300) = 1.67$. Thus the computed solution is $\hat{x} = [1.67,\ 199.]^T$. All computations were identical to those in Examples 2.8.3 and 2.8.6.

Although the computed solution $\hat{x} = [1.67,\ 199.]^T$ has an inaccurate first component, it should not necessarily be viewed as a bad result. The inaccurate entry is much smaller than the accurate one, and in fact $\| \delta x \|_\infty / \| x \|_\infty = .005$, where $\delta x = \hat{x} - x$. This is an excellent outcome for three-digit decimal arithmetic. The small value of $\| \delta x \|_\infty / \| x \|_\infty$ is guaranteed by the well-conditioned coefficient matrix, together with the fact that the computed $\hat{L}$ and $\hat{U}$ do not have large entries (cf. Theorems 2.7.14 and 2.3.6).

It is easy to imagine situations in which the computed result $\hat{x} = [1.67\ 199.]^T$ is acceptable. Suppose for example that x_1 and x_2 represent voltages expressed in the same units. If all that matters is the voltage difference, then the result is okay, since the computed difference $\hat{x}_2 - \hat{x}_1 = 197.33$ differs from the correct difference 199 by only about one percent. □

Exercise 2.8.12

(a) Calculate $\kappa_\infty(A)$, where A is the coefficient matrix of (2.8.11).

(b) Perform Gaussian elimination on (2.8.11) with the rows interchanged, using three-digit decimal arithmetic.

(c) Perform Gaussian elimination on (2.8.11) with the columns interchanged, using three-digit decimal arithmetic.

□

2.9 COMPONENTWISE SENSITIVITY ANALYSIS

In this chapter we have taken the oldest and simplest approach to sensitivity analysis, in which everything is measured by norms. It is called *normwise sensitivity analysis*, and it is accompanied by *normwise backward error analysis*. This style of error analysis has been very successful, but there are some situations in which a different type of analysis, *componentwise sensitivity analysis*, is more appropriate. In the

normwise analysis, δA is considered a small perturbation of A if $\|\delta A\|/\|A\|$ is small. This criterion does not force all of the ratios $|\delta a_{ij}|/|a_{ij}|$, which are the perturbations of the components, to be small.

Example 2.9.1 Suppose

$$A = \begin{bmatrix} 1.04 & 2.35 \\ 4.26 \times 10^{-6} & 6.32 \end{bmatrix} \quad \text{and} \quad \delta A = \begin{bmatrix} 1.32 \times 10^{-5} & 5.46 \times 10^{-6} \\ 1.02 \times 10^{-5} & 8.29 \times 10^{-6} \end{bmatrix}.$$

Then

$$A + \delta A = \begin{bmatrix} 1.0400132 & 2.35000546 \\ 1.446 \times 10^{-5} & 6.32000829 \end{bmatrix}.$$

We have $\|\delta A\|_\infty/\|A\|_\infty < 10^{-5}$, but this does not force $|\delta a_{21}|/|a_{21}|$ to be small. Obviously this can happen to any entry for which $|a_{ij}| \ll \|A\|$. □

In componentwise sensitivity analysis, perturbations are considered small only if the perturbation in each component is small relative to that component, that is,

$$\max_{i,j} \frac{|\delta a_{ij}|}{|a_{ij}|}$$

is small. Since we often encounter matrices with zero entries, we prefer the following reformulation, in which a_{ij} does not appear in the denominator: The perturbation δA is *componentwise ϵ-small* with respect to A if there is a positive $\epsilon \ll 1$ such that

$$|\delta a_{ij}| \leq \epsilon |a_{ij}| \quad \text{for } i, j = 1, \ldots, n. \tag{2.9.2}$$

Notice that under this condition, if $a_{ij} = 0$, then $\delta a_{ij} = 0$. Thus sparse matrices stay sparse and sparsity patterns are preserved under perturbations of this type.

Recall the following notation, which we introduced in Section 2.7. If B is a matrix (or vector) with (i, j) entry b_{ij}, then $|B|$ is the matrix with the same dimensions whose (i, j) entry is $|b_{ij}|$. We write $|B| \leq |C|$ to mean that $|b_{ij}| \leq |c_{ij}|$ for all i and j. With these notational conventions we can rewrite the condition (2.9.2) as

$$|\delta A| \leq \epsilon |A|. \tag{2.9.3}$$

In the componentwise sensitivity analysis we can ask the same sort of questions as we do in the normwise analysis. For example, if $Ax = b$ and $(A + \delta A)(x + \delta x) = b$, what is the largest δx can be relative to x? Before considering this and related questions, we pause to establish some basic facts about the matrix absolute value notation and matrix inequalities.

Exercise 2.9.4

(a) Show that if $A = BC$, then $|A| \leq |B||C|$. (This is a matrix inequality.) In particular, $|Ax| \leq |A||x|$.

(b) (Review) Show that if $y = |x|$, then $\|y\|_\infty = \|x\|_\infty$.

□

If $A \in \mathbb{R}^{n\times n}$ is nonsingular, we can build various other $n \times n$ matrices from A, for example, $|A|$, A^{-1}, $|A^{-1}|$, and $K = |A^{-1}|\,|A|$. This last one appears in various error bounds, as we shall see, and we use it to define a new type of condition number, the *Skeel condition number* of A:

$$\text{skeel}(A) = \|K\|_\infty = \| \, |A^{-1}|\,|A| \, \|_\infty.$$

Now we are ready to prove some theorems. The first is the componentwise analogue of Theorem 2.2.4

Theorem 2.9.5 *Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, let $b \in \mathbb{R}^n$ be nonzero, and let x be the unique solution of $Ax = b$. Suppose $\hat{x} = x + \delta x$ is the solution of $A\hat{x} = b + \delta b$, where*

$$|\delta b| \le \epsilon |b|.$$

Then

$$|\delta x| \le \epsilon \, |A^{-1}|\,|A|\,|x| \tag{2.9.6}$$

and

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \le \epsilon \, \text{skeel}(A). \tag{2.9.7}$$

Exercise 2.9.8 Prove Theorem 2.9.5 as follows (or do it your own way).

(a) Prove that $|\delta x| \le |A^{-1}|\,|\delta b|$ and $|b| \le |A|\,|x|$. Then deduce (2.9.6).

(b) Deduce (2.9.7) from (2.9.6).

□

Now consider this componentwise analogue of Theorems 2.3.3 and 2.3.6.

Theorem 2.9.9 *Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, let $b \in \mathbb{R}^n$ be nonzero, and let x be the unique solution of $Ax = b$. Suppose $\hat{x} = x + \delta x$ satisfies $(A + \delta A)\hat{x} = b$, where*

$$|\delta A| \le \epsilon \, |A|.$$

Then

$$|\delta x| \le \epsilon \, |A^{-1}|\,|A|\,|\hat{x}| \tag{2.9.10}$$

and

$$\frac{\|\delta x\|_\infty}{\|\hat{x}\|_\infty} \le \epsilon \, \text{skeel}(A). \tag{2.9.11}$$

If $\epsilon \, \text{skeel}(A) < 1$, then also

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \le \frac{\epsilon \, \text{skeel}(A)}{1 - \epsilon \, \text{skeel}(A)}. \tag{2.9.12}$$

Exercise 2.9.13 Prove Theorem 2.9.9 as follows (or do it your own way).

(a) Prove that $\delta x = -A^{-1}\delta A\hat{x}$ and $|\delta x| \leq |A^{-1}|\,|\delta A|\,|\hat{x}|$. Then deduce (2.9.10).

(b) Deduce (2.9.11) from (2.9.10).

(c) Multiply (2.9.11) through by $\|\hat{x}\|_\infty$, apply the triangle inequality to break $\|\hat{x}\|_\infty$ into two parts, and deduce (2.9.12), remembering to point out where you are using the added hypothesis $\epsilon\,\mathrm{skeel}(A) < 1$.

□

Componentwise Backward Error Analysis

Componentwise sensitivity analysis can be combined with componentwise backward stability analysis, whenever the latter is successful. One example of componentwise backward stability that we have already encountered is the forward substitution algorithm for solving triangular systems. Theorem 2.7.2 states that if we solve the triangular system $Gy = b$ by forward substitution using floating-point arithmetic, then the computed solution $\hat{y}$ satisfies $(G + \delta G)\hat{y} = b$, where $|\delta G| \leq 2nu|G| + O(u^2)$. This is a componentwise backward stability result, since it says that each component of δG is tiny relative to the corresponding component of G. As a practical matter we have $|\delta G| \leq Cu|G|$, where C is a modest multiple of 1. We can now apply Theorem 2.9.9 with $\epsilon = Cu$, to conclude that $\hat{y}$ is accurate (in the sense that $\|\delta y\|_\infty / \|y\|_\infty$ is tiny) if the Skeel condition number $\mathrm{skeel}(G)$ is not too large.

Iterative Refinement

Another process that yields componentwise backward stability is iterative refinement. This is an old procedure that was originally used to improve the accuracy of solutions to ill-conditioned systems. Let $\hat{x}$ denote an approximation to the solution of the system $Ax = b$, and let $\hat{r}$ be the associated residual: $\hat{r} = b - A\hat{x}$. The approximation $\hat{x}$ may have been obtained by Gaussian elimination, for example. If we could solve the residual system $Az = \hat{r}$ exactly, then the vector $x = \hat{x} + z$ would be the exact solution of $Ax = b$, as you can easily check. If we did obtain $\hat{x}$ by Gaussian elimination, then an LU decomposition is available, so we can solve $Az = \hat{r}$ inexpensively. Of course the computed solution $\hat{z}$ is not exact. If A is somewhat ill conditioned, it may be far from exact. Nevertheless it is not unreasonable to hope that $\hat{x} + \hat{z}$ will be an improvement over $\hat{x}$. If this is the case, then perhaps we can improve the solution even more by calculating the residual associated with $\hat{x} + \hat{z}$ and repeating the process. In fact we can repeat it as many times as we wish. This gives the following *iterative refinement* algorithm.

Iterative Refinement Algorithm

for $k = 1, \ldots, m$

$\quad \hat{r} \leftarrow b - A\hat{x}$

$\quad$ Calculate $\hat{z}$, an approximate solution of $Az = \hat{r}$.

$\quad\quad$ (Use the LU decomposition that was computed previously.) (2.9.14)

$\quad \hat{x} \leftarrow \hat{x} + \hat{z}$

$\quad$ if ($\| \hat{z} \| / \| \hat{x} \|$ is sufficiently small) exit (successful completion)

Set flag indicating failure.

This is called an *iterative* algorithm because the number of steps or *iterations* to be performed is not known in advance. The iterations are terminated as soon as the corrections become sufficiently small. Any iterative algorithm should have an upper bound on the number of iterations it is willing to attempt before abandoning the process as a failure. In (2.9.14) that number is denoted by m. Notice that in order to carry out this procedure, we must save copies of A and b to use in the computation of the residuals.

Up until about 1980 it was thought that (2.9.14) cannot hope to succeed unless the residuals are calculated in extended-precision arithmetic. This means that if we are using single-precision arithmetic, the step $\hat{r} \leftarrow b - A\hat{x}$ should be done in double precision. The reason for this is that severe cancellation occurs when $A\hat{x}$ is subtracted from b. The smaller $\hat{r}$ becomes, the worse the cancellation is. The objective of the extended-precision computation is to preserve as many significant digits as possible in the face of this cancellation.

If iterative refinement is carried out in this way and the system is not too badly conditioned (say $\kappa(A) \ll 1/u$), (2.9.14) actually does converge to the true solution of $Ax = b$. Thus iterative refinement can be used to solve the problem to full precision ($\| x - \hat{x} \| \approx u$). There is a catch, however. The system that is solved so precisely is the one whose coefficient matrix A and right-hand-side vector b are exactly what is stored in the computer. Because of measurement and representation errors, the stored A and b are mere approximations to the true data for the physical problem that we are trying to solve. If the problem is ill conditioned, the exact solution of $Ax = b$ can be a very unsatisfactory approximation to the solution of the problem we really wish to solve.

Around 1980 it was realized that iterative refinement is useful even if the residuals are not computed with extended precision arithmetic. Full precision ($\| x - \hat{x} \| \approx u$) cannot be attained, but some improvement in accuracy is possible. Furthermore, iterative refinement has a good effect on the backward error. Skeel [61] showed that if the system is not too badly conditioned and not too badly out of scale, then *one step* of iterative refinement is usually enough to ensure a *componentwise* backward stable solution. See [41] or [61] for details.

Componentwise backward stability means that the computed solution satisfies $(A + \delta A)\hat{x} = b$, where $|\delta A| \leq Cu|A|$, with C not too big. This can then be

combined with Theorem 2.9.9 to get the bound

$$\frac{\|\delta x\|_\infty}{\|\hat{x}\|_\infty} \leq C\, u\, \text{skeel}(A).$$

This is sometimes significantly better than bounds that can be obtained using the normwise error analysis, because skeel(A) can be much smaller than $\kappa_\infty(A)$ (See Exercise 2.9.15). The greatest advantage comes with matrices that are ill-conditioned simply because the rows are out of scale. The Skeel condition number skeel(A) is insensitive to row scaling, so it remains small while the normwise condition number $\kappa_\infty(A)$ becomes large in proportion to the badness of the scaling.

There are inexpensive methods for estimating skeel(A) that work on the same principal as condition estimators for $\kappa_1(A)$. See [41].

Exercise 2.9.15

(a) Show that for every nonsingular matrix A, skeel$(A) \leq \kappa_\infty(A)$.

(b) Show that if D is a nonsingular diagonal matrix, then $|DA| = |D|\,|A|$ and $|D|^{-1} = |D^{-1}|$.

(c) Show that the Skeel condition number is invariant under row scaling; that is, skeel$(DA) =$ skeel(A) for all nonsingular diagonal matrices D.

(d) From part (a) we know that the ratio $\kappa_\infty(A)/\text{skeel}(A)$ is always at least one. Show by example that it can be made arbitrarily large.

□

Exercise 2.9.16

(a) Prove that if A is nonsingular, $A + \delta A$ is singular, and $|\delta A| \leq \epsilon |A|$, then ϵ skeel$(A) \geq 1$ (cf. Theorem 2.3.1).

(b) Prove that if A is nonsingular and $|\delta A| \leq \epsilon |A|$, where ϵ skeel$(A) < 1$, then $A + \delta A$ is nonsingular.

(c) Discuss the relationship between the result in part (b) and Theorem 2.3.1. Is one stronger than the other?

□