Chapter 10

Conditioning of Problems and Stability of Algorithms

You should be familiar with

- Solution of a linear system $Ax = b$ using Gaussian elimination
- Matrix inverse
- Vector and matrix norms
- Eigenvalues
- Singular values

This chapter begins with the question "Why do we need numerical linear algebra?" and answers it by presenting six examples, and there will be more later in this book. Some algorithms can be very sensitive to their input, meaning that the algorithm can produce poor results with perfectly good data. These algorithms are unstable. On the other hand, a stable algorithm can have input that cause the output to be poor. Such input is termed ill-conditioned. The chapter presents concrete examples of these issues and develops tools that are useful in detecting and dealing with them.

10.1 WHY DO WE NEED NUMERICAL LINEAR ALGEBRA?

We have studied fundamental concepts in linear algebra. We know how to solve linear systems using Gaussian elimination, how to find eigenvalues and eigenvectors, and that orthogonal matrices are important. We are familiar with subspaces, linear independence, and matrix rank. So why is it necessary for us to study numerical linear algebra, and what is it? This chapter provides some answers. It is not adequate to run an algorithm and accept the results, and we know this from our study of floating point arithmetic in Chapter 8. A computer does not perform exact floating point arithmetic, and this can cause even a time-honored algorithm to produce bad results. A good example is using the quadratic equation in its standard form and suffering very serious cancellation errors. Chapter 9 introduces the concept of an algorithm and algorithm efficiency. Suppose you have a problem to solve that involves matrices, and a computer must be used to obtain a solution. You may be faced with a choice from among a number of competing algorithms. In this case, you must consider efficiency, one aspect of which is flop count. Understanding issues of algorithm efficiency is one aspect of numerical linear algebra that sets it aside from theoretical linear algebra. We will see that certain algorithms are more prone to bad behavior from roundoff error than others, and we must avoid using them. Also, an algorithm that normally is very effective may not give good results for certain data, and this leads to the subject of conditioning, particularly as involves matrices. In short, numerical linear algebra is the study of how to accurately and efficiently solve linear algebra problems on a computer. Here are some classic examples that illustrate the issues.

a. *Using Gaussian elimination to solve a nonsingular $n \times n$ system $Ax = b$.* Chapter 2 discusses Gaussian elimination. During the process, if a 0 is encountered in the pivot position, a row exchange solved the problem. As we will see in Chapter 11, Gaussian elimination can perform very poorly unless we incorporate row exchange into the algorithm so that the pivot a_{ii} is the element of largest absolute value among the elements $\{a_{ki}\}, k \leq i \leq n$.

b. *Dealing with $m \times n$ systems, $m \neq n$.* A theoretical linear algebra course shows that systems $Ax = b$, where x is an $n \times 1$ vector and b is $m \times 1$, have an infinite number of solutions or none. This is done by transforming A to what is called reduced row echelon form. In numerical linear algebra, systems such as these arise in least-squares problems. Under the right conditions, there is a unique solution satisfying the requirement that $\|b - Ax\|_2$ is a minimum. Very seldom does $Ax = b$.

c. *Solving a linear algebraic system using Cramer's Rule.* We presented Cramer's Rule in Theorem 4.6 and mentioned that is was intended primarily for theoretical purposes. In practice, it is frequently necessary to solve square systems of size greater than or equal to 50×50. Using Cramer's Rule for a 50×50 matrix involves computing 51 determinants of 50×50 matrices. If we use expansion by minors for each determinant, it will require evaluating $51\,(50!) \simeq 1.6 \times 10^{66}$ permutations to solve the system. Each permutation requires 49 multiplications. Assume a supercomputer can execute 10^{15} flops/s. The number of seconds required to do the multiplications is

$$\frac{51\,(50!)\,49}{10^{15}}\text{s} \simeq 2.4 \times 10^{45}$$

years!

d. *Computing the solution to a linear system $Ax = b$ by first finding A^{-1} and then computing $x = A^{-1}b$.* Computing A^{-1} takes more operations than using Gaussian elimination if you are solving one system $Ax = b$. But, is it effective to find A^{-1} if the solution to multiple systems $Ax_i = b_i, 1 \leq i \leq k$ is required? The solutions are $x_i = A^{-1}b_i$, so only k matrix-vector products need be computed. It can be shown that computing A^{-1} to solve the problem requires four (4) times as many flops as using Gaussian elimination to factor A into a product of a lower- and an upper-triangular matrix and then using forward and back substitution for each of the k systems. It is also the case that some inverses are very hard to compute accurately, and using A^{-1} gives very poor results.

e. *Computing the eigenvalues of a matrix by finding the roots of its characteristic polynomial.* There are long-standing algorithms for finding the roots of a polynomial, but remember that the coefficients of the characteristic polynomial will likely be corrupted by roundoff error. Section 10.3.1 demonstrates that even a slight change to one or more coefficients of a polynomial can cause large changes in its roots. If such a polynomial is the characteristic polynomial for the matrix, the eigenvalue computation can be disastrous. This method for computing eigenvalues should not be used.

f. *Finding the singular values of a matrix A by computing the eigenvalues of $A^{\mathrm{T}}A$.* In computing the singular values by finding the eigenvalues of $A^{\mathrm{T}}A$, errors introduced by matrix multiplication, followed by errors in computing the eigenvalues may be significant. We will show in Chapter 15 that the rank of a matrix is equal to the number of its nonzero singular values. Consider the example in Ref. [25]

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

where $\mu < \sqrt{\text{eps}}$, so $1 + \mu^2 < 1 + \text{eps}$, and thus $\text{fl}\left(1 + \mu^2\right) = 1$. As a result,

$$\text{fl}\left(A^{\mathrm{T}}A\right) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

its singular values are $\sigma_1 = \sqrt{2}$, $\sigma_2 = 0$, and the computed rank of A is 1. Using exact arithmetic,

$$A^{\mathrm{T}}A = \begin{bmatrix} 1+\mu^2 & 1 \\ 1 & 1+\mu^2 \end{bmatrix},$$

and the singular values of A are

$$\sigma_1 = \sqrt{2+\mu^2}, \quad \sigma_2 = |\mu|.$$

The true rank of A is 2. By computing $A^{\mathrm{T}}A$, the term μ^2 entered the computations, giving rise to the roundoff error $\text{fl}\left(1 + \mu^2\right) = 1$. MATLAB does floating point arithmetic using 64-bits.The following MATLAB statements demonstrate the problem.

```
>> mu = sqrt(eps);
>> A = [1 1;mu 0;0 mu];
>> rank(A)
ans =
      2
```

```
>> B = A'*A;
>> eig(B)

ans =
   0.000000000000000
   2.000000000000000

>> rank(B)

ans =
     1
```

10.2 COMPUTATION ERROR

Our aim is to define criteria that help us decide what algorithm to use for a particular problem, or when the data for a problem may cause computational problems. In order to do this, we need to understand the two types of errors, forward error and backward error. Consider the problem to be solved by an algorithm as a function f mapping the input data x to the solution $y = f(x)$. However, due to inaccuracies during floating point computation, the computed result is $\hat{y} = \hat{f}(x)$. Before defining the types of errors and providing examples, it is necessary to introduce a new notation.

Definition 10.1. If A is an $m \times n$ matrix, $|A| = \left(\left|a_{ij}\right|\right)$; in other words, $|A|$ is the matrix consisting of the absolute values of a_{ij}, $1 \le i \le m$, $1 \le j \le n$.

Example 10.1.
a. If

$$A = \begin{bmatrix} -1 & 2 & 1.8 \\ -0.45 & 1.23 & 14.5 \\ -1.89 & 0.0 & -12.45 \end{bmatrix},$$

$$|A| = \begin{bmatrix} 1 & 2 & 1.8 \\ 0.45 & 1.23 & 14.5 \\ 1.89 & 0.0 & 12.45 \end{bmatrix}.$$

b. Let $x = \begin{bmatrix} -1 \\ 3 \\ -4 \end{bmatrix}$, $y = \begin{bmatrix} -8 \\ -1 \\ 2 \end{bmatrix}$. Then,

$$|x|^{\mathrm{T}}\,|y| = \begin{bmatrix} 1 & 3 & 4 \end{bmatrix} \begin{bmatrix} 8 \\ 1 \\ 2 \end{bmatrix} = 19.$$

■

10.2.1 Forward Error

Forward error deals with rounding errors in the solution of a problem.

Definition 10.2. The *forward error* in computing $f(x)$ is $\left|\hat{f}(x) - f(x)\right|$. This measures errors in computation for input x.

The forward error is a natural quantity to measure but, in general, we do not know the true value $f(x)$ so we can only get an upper bound on this error.

Example 10.2. The *outer product* of vectors x and y in $\mathbb{R}^n$ is the $n \times n$ matrix

$$A = xy^{\mathrm{T}} = \begin{bmatrix} x_1y_1 & x_1y_2 & \dots & x_1y_n \\ x_2y_1 & x_2y_2 & \dots & x_2y_n \\ \vdots & \vdots & \vdots & \vdots \\ x_ny_1 & x_ny_2 & \dots & x_ny_n \end{bmatrix}. \tag{10.1}$$

Each entry of the computed result, $\hat{a_{ij}} = x_iy_j$ satisfies

$$\hat{a_{ij}} = x_iy_j\left(1+\epsilon_{ij}\right), \quad \left|\epsilon_{ij}\right| \le \text{eps},$$

and so

$$\hat{A} = \text{fl}\left(xy^{\mathrm{T}}\right) = xy^{\mathrm{T}} + \Delta,$$

where

$$\Delta = \begin{bmatrix} x_1y_1\epsilon_{11} & x_1y_2\epsilon_{12} & & x_1y_n\epsilon_{1n} \\ x_2y_1\epsilon_{21} & x_2y_2\epsilon_{22} & & x_2y_n\epsilon_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_ny_1\epsilon_{n1} & x_ny_2\epsilon_{n2} & \dots & x_ny_n\epsilon_{nn} \end{bmatrix}.$$

Since $\left|\epsilon_{ij}\right| \le$ eps, it follows that

$$|\Delta| \le \text{eps}\left|xy^{\mathrm{T}}\right|.$$

This tells us that we can compute the outer product with forward error Δ, and that the error is bounded by eps $\left|xy^{\mathrm{T}}\right|$. ■

As another example, consider the important problem of computing the inner product of vectors, u, v.

Example 10.3. If x and y are $n \times 1$ vectors and $n \times \text{eps} \le 0.01$, then,

$$\left|\text{fl}\left(x^{\mathrm{T}}y\right) - x^{\mathrm{T}}y\right| \le 1.01\, n\, \text{eps}\, |x|^{\mathrm{T}}\, |y|.$$

The result shows that the forward error for the inner product is small. For a proof see Ref. [2, p. 99]. ■

10.2.2 Backward Error

Backward error relates the rounding errors in the computation to the errors in the data rather than its solution. Generally, a backward error analysis is preferable to a forward analysis for this reason.

Consider the addition of n floating point numbers. In Section 8.3.2, we determined that

$$s_n = x_1\left(1+\eta_1\right) + x_2\left(1+\eta_2\right) + x_3\left(1+\eta_3\right) + \dots + x_{n-1}\left(1+\eta_{n-1}\right) + x_n\left(1+\eta_n\right). \tag{10.2}$$

This says that s_n is the exact sum of the data perturbed by small errors, η_i. In other words, the result shows how errors in the data affect errors in the result.

Definition 10.3. Roundoff or other errors in the data have produced the result $\hat{y}$. The *backward error* is the smallest Δx for which $\hat{y} = f\left(x + \Delta x\right)$; in other words, backward error tells us what problem we actually solved.

Figure 10.1 is an adaptation of Figure 1.1 from Ref. [16, p. 7] and illustrates the concepts of forward and backward error.

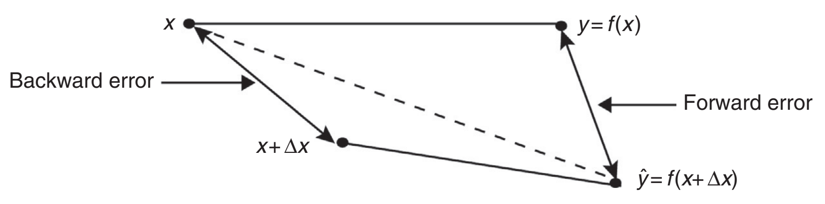

FIGURE 10.1 Forward and backward errors.

Example 10.4. The McLaurin series for $f(x) = 1/(1-x)$ is

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots = \sum_{n=0}^{\infty} x^n, \quad |x| < 1.$$

For $x = 0.57000$, approximate $f(x)$ by

$$\hat{f}(x) = 1 + x + x^2 + \cdots + x^7.$$

Then,

$$f(0.57000) = 2.3256, \quad \hat{f}(0.5700) = 2.2997.$$

The forward error is $|2.2997 - 2.3256| = 0.025900$. To find the backward error, we must find $\hat{x}$ such that

$$\frac{1}{1-\hat{x}} = 2.2997.$$

Solving for $\hat{x}$, we obtain $\hat{x} = 0.56516$, and the backward error is $\left|\hat{x} - x\right| = 0.0048400$. ■

For another example of a backward error analysis, consider the computation of $\langle u, v\rangle$, where u and v are $n \times 1$ vectors. A proof of the following theorem can be found in Ref. [16, pp. 62-63].

Theorem 10.1. *In the computation of the inner product, $\langle u, v\rangle$, where u and v are $n \times 1$ vectors,*

$$\text{fl}\left(\langle u, v\rangle\right) = x_1y_1\left(1 + \eta_n\right) + x_2y_2\left(1 + \eta_n'\right) + x_3y_3\left(1 + \eta_{n-1}\right) + \cdots + x_ny_n\left(1 + \eta_2\right),$$

where $|\eta_i| \le \frac{n\,\text{eps}}{1-n\,\text{eps}}$ is very small.

This backward error result has an interpretation as follows:

The computed inner product is the exact inner product for a perturbed set of data $x_1, x_2, \ldots, x_n, y_1\left(1 + \eta_n\right)$, $y_2\left(1 + \eta_n'\right), \ldots, y_n\left(1 + \eta_2\right)$.

10.3 ALGORITHM STABILITY

Now that we are familiar with backward and forward error analysis, we are in a position to define algorithm stability. Intuitively, an algorithm is stable if it performs well in general, and an algorithm is unstable if it performs badly in significant cases. In particular, an algorithm should not be unduly sensitive to errors in its input or errors during its execution. In Section 8.4.2, we saw that using the quadratic equation in its classical form

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

can produce poor results when $\sqrt{b^2 - 4ac} \approx b$. This is an *unstable algorithm*. Section 8.4.2 provided another algorithm that produces satisfactory results with the same values of a, b, and c. We will see in Chapter 11 that Gaussian elimination as we know it is unstable, but when a technique known as partial pivoting is added, the algorithm is stable in all but pathological cases. Given the same data, an unstable algorithm may produce poor results, while another, stable, algorithm produces good results. There are two types of stable algorithms, backward stable and forward stable.

Definition 10.4. An algorithm is *backward stable* if for any x, it computes $\hat{f}(x)$ with small backward error, Δx. In other words, it computes the exact solution to a nearby problem,

$$f(x + \Delta x) = \hat{f}(x),$$

so that the solution is not sensitive to small perturbations in x.

By virtue of Equation 10.2, the addition of floating point numbers is backward stable, and by Example 10.3 so is the inner product of two vectors.

We use the process of back substitution as the final step of Gaussian elimination, so we need to know its stability properties. A proof that back substitution is backward stable can be found in Ref. [26, pp. 122-127].

Theorem 10.2. *Let back substitution be applied to the system, whose entries are floating point numbers. For any matrix norm, the computed solution $\hat{x}$ satisfies*

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

for some upper-triangular matrix δR with

$$\frac{\|\delta R\|}{\|R\|} = O\,(\text{eps})\,.$$

Specifically, for each i, j,

$$\frac{\left|\delta r_{ij}\right|}{\left|r_{ij}\right|} \le n\,\text{eps} + O\left(\text{eps}^2\right).$$

Definition 10.5. An algorithm is *forward stable* if whenever $f\,(x)$ is the true solution, the difference between the computed and true solutions is small. In other words,

$$\left|\hat{f}\,(x) - f\,(x)\right|$$

is small.

We have seen that the inner and outer product of two vectors is forward stable.

We know from our discussion in Chapter 8 that floating-point arithmetic does not follow the laws of real arithmetic. This can make forward error analysis difficult. In backward error analysis, however, real arithmetic is employed, since it is assumed that the computed result is the exact solution to a nearby problem. This is one reason why backward error analysis is often preferred, so we will refer to *stability* to mean backward stability. We have seen that floating point addition and inner product are stable, but we should provide examples of unstable algorithms.

10.3.1 Examples of Unstable Algorithms

The matrix xy^{T} has rank 1 (see Problem 10.3). For the computation of the outer product to be backward stable,

$$\hat{A} = (x + \Delta x)\,(y + \Delta y)^{\mathrm{T}}\,,$$

must have rank 1. However,

$$\hat{A} = xy^{\mathrm{T}} + \Delta,$$

and Δ in general does not have rank 1. The outer product is not backward stable, leading to the remark 10.1.

Remark 10.1. It is possible for an algorithm to be forward stable but not backward stable. In other words, it is possible for $\left|\hat{f}\,(x) - f\,(x)\right|$ to be small but the perturbation in the data, Δx, may be large.

We commented that one should not compute eigenvalues by finding the roots of the characteristic equation. Another example of an unstable algorithm is polynomial root finding.

Example 10.5. The matrix

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

has characteristic polynomial

$$p\,(\lambda) = \lambda^3 - 6\lambda^2 + 12\lambda - 8 = (\lambda - 2)^3$$

with three equal roots. Suppose roundoff error causes the coefficient of λ^2 to become 5.99999, a perturbation of 10^{-5}. The roots of the characteristic equation are now

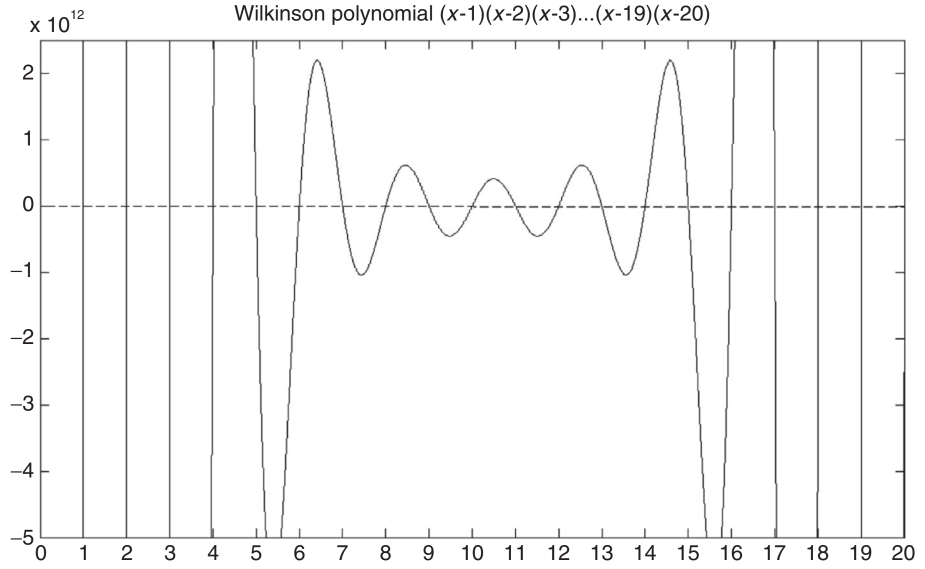

FIGURE 10.2 The Wilkinson polynomial.

```
2.016901481104199 + 0.029955270323774i
2.016901481104199 - 0.029955270323774i
1.966187037791599
```
■

Equal roots or roots very close together normally make the root finding problem unstable. The problem is actually more serious. If the roots are not close together, the root finding problem can still be unstable. A very clever example of this is the *Wilkinson polynomial*

$$p(x) = (x-1)(x-2)(x-3)\dots(x-20).$$

Of course, all the roots are distinct and separated by 1. Expanded, $p(x)$ is

$$x^{20} - 210x^{19} + 20,615x^{18} - 1,256,850x^{17} + 53,327,946x^{16} - 1,672,280,820x^{15}$$
$$+ \cdots - 8,752,948,036,761,600,000x + 2,432,902,008,176,640,000.$$

Figure 10.2 shows a graph of the Wilkinson polynomial. Wilkinson found that when -210, the coefficient of x^{19}, was perturbed to $-210 - 2^{-23}$ the roots at $x = 16$ and $x = 17$ became approximately $16.73 \pm 2.81i$, quite a change for a perturbation of $2^{-23} \approx 1.192 \times 10^{-7}$!

This discussion definitely verifies that computing eigenvalues by finding the roots of the characteristic equation is a bad idea. Round-off errors when computing the coefficients of the characteristic polynomial may cause large errors in the determination of the roots.

10.4 CONDITIONING OF A PROBLEM

Even when using a stable algorithm to solve a problem, the problem may be sensitive to small changes (perturbations) in the data. The perturbations can come from roundoff error, small measurement errors when collecting experimental data, noise that is not filtered out of a signal, or truncation error when approximating the sum of an infinite series. If a problem falls in this category, we say it is *ill-conditioned*.

Definition 10.6. A problem is *ill-conditioned* if a small relative change in its data can cause a large relative error in its computed solution, regardless of the algorithm used to solve the problem. If small perturbations in problem data lead to small relative errors in the solution, a problem is said to be *well-conditioned*.

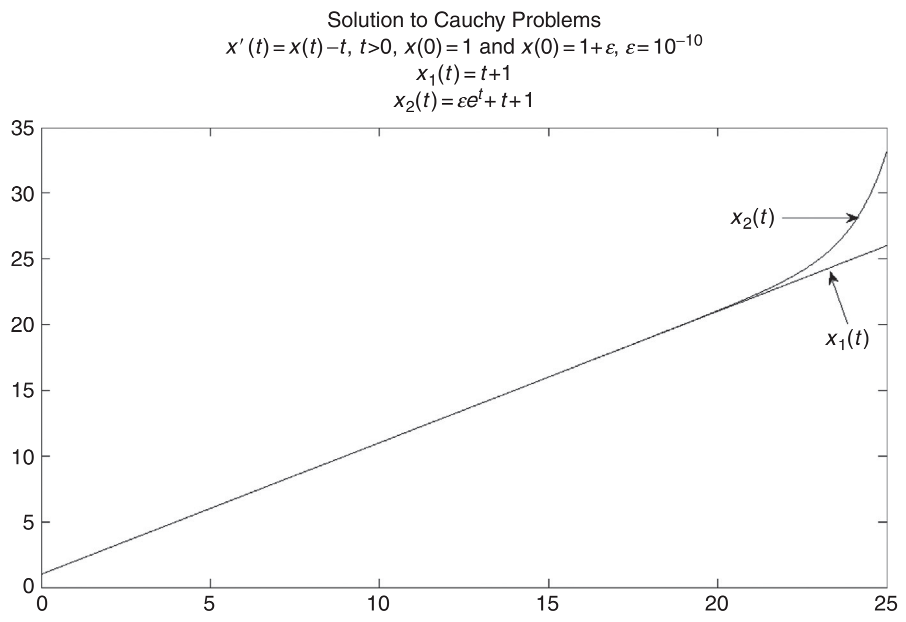

FIGURE 10.3 Ill-conditioned Cauchy problem.

Example 10.6. The initial-value problem

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x(t) - t, \quad t > 0, \quad x(0) = 1$$

is an example of a *Cauchy problem*. The unique solution to this problem is

$$x_1(t) = t + 1.$$

Now perturb the initial condition by a small amount ϵ, to obtain the Cauchy problem

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x(t) - t, \quad t > 0, \quad x(0) = 1 + \epsilon.$$

The unique solution to this problem is

$$x_2(t) = \epsilon e^t + t + 1,$$

as is easily verified. Figure 10.3 is a graph of both solutions over the interval $0 \le t \le 25$ with $\epsilon = 10^{-10}$. A very small change in the initial condition created a very different solution for $x > 20$. This Cauchy problem is ill-conditioned. The L^2 function norm will give us the relative error over the interval:

$$\sqrt{\frac{\int_0^{25} (x_1(t) - x_2(t))^2 \,\mathrm{d}t}{\int_0^{25} x_1(t)^2 \,\mathrm{d}t}} = \sqrt{\frac{3}{2}\left(10^{-10}\right)^2 \left(\frac{e^{50}-1}{26^3-1}\right)} = 0.06652,$$

a very poor result. ■

It is easy to confuse the concepts of stability and conditioning. Remark 10.2 provides a summary of the differences.

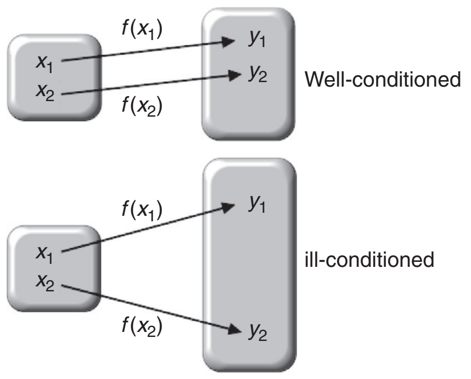

FIGURE 10.4 Conditioning of a problem.

Remark 10.2. Summary

- Stable or unstable refers to an algorithm.
- Well or ill-conditioned refers to the particular problem, not the algorithm used.

Clearly, mixing roundoff error with an unstable algorithm is asking for disaster.

As mentioned in the introduction this section, a stable algorithm for the solution of a problem can produce poor results if the data are ill-conditioned. Assume that $f(x)$ represent an algorithm that takes data x and produces a solution $f(x)$. Here are some examples:

- In Example 10.6, f is an initial-value problem solver, and $x(0)$ is the initial condition.
- f is Gaussian elimination applied to solve a linear algebraic system $Ax = b$. The data are A and b.
- f is a function that takes a real number x and returns a real number y.

We can define ill and well-conditioning in terms of $f(x)$. Use the notation $\|\cdot\|$ to indicate a measure of size, such as the absolute value of a real number or a vector or matrix norm.

Definition 10.7. Let x and $\bar{x}$ be the original and slightly perturbed data, and let $f(x)$ and $f(\bar{x})$ be the respective solutions. Then,

The problem is well-conditioned with respect to x if whenever $|x - \bar{x}|$ is small, $|f(x) - f(\bar{x})|$ is small.

The problem is ill-conditioned with respect to x if whenever $|x - \bar{x}|$ is small, $|f(x) - f(\bar{x})|$ can be large (Figure 10.4).

The sensitivity of a problem to data perturbations is measured by defining the *condition number*. The larger the condition number, the more sensitive a problem is to changes in data. For a particular x, assume there is a small error in the data so that the input to the problem is $\bar{x} = x + \Delta x$, and the computed value is $f(\bar{x})$ instead of $f(x)$. Form the relative error of the result divided by the relative error of the input:

$$\frac{\dfrac{|f(x) - f(\bar{x})|}{|f(x)|}}{\dfrac{|x - \bar{x}|}{|x|}} \tag{10.3}$$

The ratio measures how sensitive a function is to changes or errors in the input.

Remark 10.3. In mathematics, the *supremum* (sup) of a subset S of an ordered set X is the least element of X that is greater than or equal to all elements of S. It differs from the maximum, in that it does not have to be a member of subset S.

This leads us to a mathematical definition of the condition number for a problem f.

Definition 10.8. The condition number of problem f with input x is

$$C_f(x) = \lim_{\epsilon \to 0^+} \sup_{\|\delta x\| \le \epsilon} \frac{\dfrac{\|f(x + \delta x) - f(x)\|}{\|f(x)\|}}{\dfrac{\|\delta x\|}{\|x\|}}.$$

It is sufficient to think of the condition number as the limiting behavior of Equation 10.3 as the error δx becomes small.

As a first example of computing a condition number, let $f(x)$ be a function from $\mathbb{R}$ to $\mathbb{R}$. The question is whether evaluating the function is well-conditioned. The function $f(x) = x^2$ is clearly well-conditioned, but $f(x) = (1/(1-x))$ has serious problems near $x = 1$. In the general case, assume that f is differentiable so that the mean value theorem, proved in any calculus text, applies to f.

Theorem 10.3. *(Mean value theorem) If a function $f(x)$ is continuous on the closed interval $a \le x \le b$ and differentiable on the open interval $a < x < b$, then there exists a point ξ in $a < x < b$ such that*

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}.$$

Fix x and let $\overline{x} = x + \delta x$. Applying the mean value theorem

$$\frac{\frac{|f(x)-f(x+\delta x)|}{|f(x)|}}{\frac{|\delta x|}{|x|}} = \frac{|x|}{|\delta x|}\frac{|f(x) - f(x+\delta x)|}{|f(x)|} = \frac{|x|}{|f(x)|}\frac{|f(x) - f(x+\delta x)|}{|\delta x|} = |x|\frac{\left|f'(\xi)\right|}{|f(x)|},$$

where ξ is between x and $x + \delta x$. As a result, $C_f(x) = |x|\left|f'(x)\right|/|f(x)|$.

Example 10.7. The example finds the condition number for three functions

a. $f(x) = x^2$, $C_f(x) = |x|\,|2x|/\left|x^2\right| = 2$, so f is well-conditioned.

b. $f(x) = 1/(1-x)$, $C_f(x) = |x|\left|1/(1-x)^2\right|/|1/(1-x)| = |x/(1-x)|$. $f(x)$ is ill-conditioned near $x = 1$ and well-conditioned everywhere else.

c. $f(x) = e^x$, $C_f(x) = |x|\,|e^x/e^x| = |x|$. $f(x)$ is ill-conditioned large x.

Parts (b) and (c) indicate that whenever x is within a certain range, a small relative error in x can cause a large relative error in the computation of $f(x)$. ■

10.5 PERTURBATION ANALYSIS FOR SOLVING A LINEAR SYSTEM

For $A = \begin{bmatrix} 1.0001 & 1 \\ 1 & 1 \end{bmatrix}$, the problem $Ax = b$ is ill-conditioned. A good reason to suspect this is that $\det(A) = 0.0001$, so the matrix is almost singular. You will note that the exact solution to the following system is $x_1 = 1$ and $x_2 = 1$.

$$\begin{bmatrix} 1.0001 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2.0001 \\ 2 \end{bmatrix}$$

Now replace the right-hand side value 2.0001 by 2, and solve the system

$$\begin{bmatrix} 1.0001 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}.$$

The result is $x_1 = 0.0000$, $x_2 = 2.0000$. A very small change in the right-hand side caused a very large change in the solution.

In the remainder of this section, we will study the effect on the solution x if the elements of the linear system $Ax = b$ are slightly perturbed. This can occur in three ways:

a. One or more elements of b are perturbed, but the elements of A are exact.

b. One or more entries in A are perturbed, but the elements of b are exact.

c. There are perturbations in both A and b.

Theorem 10.4 specifies bounds for the errors involved in each case. If the reader does not desire to read through the proof, carefully look at Equations 10.4–10.6, and note the presence of the factor $\|A\|\left\|A^{-1}\right\|$. Our definition of the very important matrix condition is motivated by these results.

Remark 10.4. Recall that if $\|\cdot\|$ is a subordinate matrix norm, then $\|Ax\| \le \|A\|\,\|x\|$ (Equation 7.9)

Theorem 10.4. *Assume A is a nonsingular matrix, $b \ne 0$ is a vector, x is the solution to the system $Ax = b$, and $\|\cdot\|$ is a subordinate norm.*

1. *If $x + \delta x$ is the solution to the perturbed system $A(x + \delta x) = b + \delta b$, then*
$$\frac{\|\delta x\|}{\|x\|} \le \|A\| \left\|A^{-1}\right\| \frac{\|\delta b\|}{\|b\|}. \tag{10.4}$$
2. *If $x + \delta x$ is the solution to the perturbed system $(A + \delta A)(x + \delta x) = b$, then*
$$\frac{\|\delta x\|}{\|x + \delta x\|} \le \|A\| \left\|A^{-1}\right\| \frac{\|\delta A\|}{\|A\|}. \tag{10.5}$$
3. *If $x + \delta x$ is the solution to the perturbed system $(A + \delta A)(x + \delta x) = b + \delta b$, then*
$$\frac{\|\delta x\|}{\|x\|} \le \|A\| \left\|A^{-1}\right\| \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|A\| \, \|x + \delta x\|}\right). \tag{10.6}$$

Proof. To prove part 1, note that $A(x + \delta x) = Ax + A(\delta x) = b + A(\delta x) = b + \delta b$, so
$$A(\delta x) = \delta b.$$

It follows that $\delta x = A^{-1}(\delta b)$, so
$$\|\delta x\| \le \left\|A^{-1}\right\| \|\delta b\|. \tag{10.7}$$

$Ax = b$, and thus
$$\|b\| \le \|A\| \, \|x\|. \tag{10.8}$$

Multiply inequalities 10.7 and 10.8 together, and $\|\delta x\| \, \|b\| \le \|A\| \left\|A^{-1}\right\| \|\delta b\| \, \|x\|$, from which we obtain the result
$$\frac{\|\delta x\|}{\|x\|} \le \|A\| \left\|A^{-1}\right\| \frac{\|\delta b\|}{\|b\|}.$$

For part 2, $(A + \delta A)(x + \delta x) = Ax + A(\delta x) + \delta A(x + \delta x) = b$, and $b = Ax$, so
$$A(\delta x) + \delta A(x + \delta x) = 0. \tag{10.9}$$

Now, multiply Equation 10.9 by A^{-1} to obtain
$$\delta x = -A^{-1}\delta A(x + \delta x),$$

and so $\|\delta x\| \le \left\|A^{-1}\right\| \|\delta A\| \, \|x + \delta x\|$, and
$$\frac{\|\delta x\|}{\|x + \delta x\|} \le \left\|A^{-1}\right\| \|\delta A\|. \tag{10.10}$$

Multiply the right-hand side of Equation 10.10 by $\|A\| / \|A\|$ to obtain
$$\frac{\|\delta x\|}{\|x + \delta x\|} \le \|A\| \left\|A^{-1}\right\| \left(\frac{\|\delta A\|}{\|A\|}\right).$$

For part 3, since $Ax = b$ and $(A + \delta A)(x + \delta x) = b + \delta b$ it follows that
$$A(\delta x) + \delta A(x + \delta x) = \delta b. \tag{10.11}$$

Multiply Equation 10.11 by A^{-1} to get
$$\delta x = A^{-1}(\delta b - \delta A(x + \delta x)),$$

so
$$\|\delta x\| \le \left\|A^{-1}\right\| \|\delta b - \delta A(x + \delta x)\|. \tag{10.12}$$

Apply the triangle inequality as well as Equation 7.9 to $\|\delta b - \delta A(x + \delta x)\|$, and Equation 10.12 becomes
$$\|\delta x\| \le \left\|A^{-1}\right\| (\|\delta b\| + \|\delta A\| \, \|x + \delta x\|). \tag{10.13}$$

Divide both sides of Equation 10.13 by $\|x+\delta x\|$ and multiply the right-hand side by $\|A\|/\|A\|$, and we obtain the required result

$$\frac{\|\delta x\|}{\|x+\delta x\|} \leq \|A\| \left\|A^{-1}\right\| \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|A\| \, \|x+\delta x\|}\right). \qquad \square$$

A question that naturally arises is whether these bounds are realistic. Is the upper bound far larger than any possible value for the actual relative error? As it turns out, the three inequalities are what is termed *optimal*.

Definition 10.9. An upper bound in an inequality is *optimal* if there exist parameters for the inequality that attain the upper bound. In other words, if *expr* depends on a number of parameters and $expr \leq upperBound$, then there are parameters such that $expr = upperBound$.

It can be shown that each of the inequalities Equations 10.4–10.6 is optimal [27, pp. 80-81]. For instance, for any matrix A, there exists a perturbation δA, a right-hand side b and a vector x such that $\|\delta x\|/\|x+\delta x\| = \|A\| \left\|A^{-1}\right\| \|\delta A\|/\|A\|$.

Remark 10.5. Even though the inequalities in Theorem 10.2 are optimal, they are *pessimistic*. This means that under most circumstances the upper bound is considerably larger than the relative error [27, p. 82].

Notice that Equations 10.4–10.6 involve the factor $\|A\| \left\|A^{-1}\right\|$, which has particular significance. It is the condition number of matrix A.

Definition 10.10. The number $\|A\| \left\|A^{-1}\right\|$ is called the *condition number* of A and we denote it by $\kappa(A)$.

Theorem 10.4 says that relative change in the solution is bounded above by the product of $\kappa(A)$ and another factor that will be small if $\|\delta A\|$ and $\|\delta b\|$ are small. If $\kappa(A)$ is small, the relative change in the solution will be small, but if $\kappa(A)$ is large, then even small changes in A or b might drastically change the solution.

Remark 10.6. Any matrix norm can be used to compute a condition number. We will assume that $\kappa(A)$ refers to the condition number relative to the 2-norm. The notation $\kappa_\infty(A)$, $\kappa_1(A)$, and $\kappa_F(A)$ refer to the ∞-, 1-, and Frobenius-norms, respectively.

In the following examples, $\|\cdot\|$ will refer to the 2-norm.

Example 10.8. Let $A = \begin{bmatrix} 1 & 3 & 8 \\ -1 & 2 & 6 \\ 2 & -1 & 7 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. The solution to $Ax = b$ is $x = \begin{bmatrix} -0.1320755 \\ -0.075471698113208 \\ 0.169811320754717 \end{bmatrix}$. Let $\delta b = \begin{bmatrix} 0.0001 \\ -0.0001 \\ 0.0005 \end{bmatrix}$, so $b + \delta b = \begin{bmatrix} 1.0001 \\ 0.9999 \\ 1.0005 \end{bmatrix}$. The solution to the perturbed system is $x + \delta x = \begin{bmatrix} -0.131964150943396 \\ -0.075550943396226 \\ 0.169839622641509 \end{bmatrix}$.
The relative perturbation is $\|\delta b\|/\|b\| = 3.0000 \times 10^{-4}$, which is quite small. The relative error of the solution is $\|\delta x\|/\|x\| = 6.1209 \times 10^{-4}$. The condition number of the matrix A is 9.6978, so good behavior should be expected.
Verify inequality 10.4:

$$\frac{\|\delta x\|}{\|x\|} = 6.1209 \times 10^{-4} \leq \|A\| \left\|A^{-1}\right\| \frac{\|\delta b\|}{\|b\|} = 9.6978 \left(3.0000 \times 10^{-4}\right) = 2.9093 \times 10^{-3} \qquad \blacksquare$$

Example 10.9. Let A be the matrix obtained in MATLAB using the command $A = \text{gallery}(3)$.

$$A = \begin{bmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{bmatrix}$$

If $b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$, the solution is $x = \begin{bmatrix} 324.3333 \\ -1035.8333 \\ 22.5000 \end{bmatrix}$. Perturb A by $\delta A = \begin{bmatrix} 0.00001 & 0 & -0.00001 \\ 0 & 0 & 0 \\ 0 & 0.00003 & 0 \end{bmatrix}$ so that $A + \delta A = \begin{bmatrix} -148.99999 & -50 & -154.00001 \\ 537 & 180 & 546 \\ -27 & -8.99997 & -25 \end{bmatrix}$. The solution to the perturbed system is

$$x + \delta x = \begin{bmatrix} 326.31236 \\ -1042.17016 \\ 22.64266 \end{bmatrix},$$

which is very different from the solution to the unperturbed system relative to the small change in A. This is not surprising, since $\kappa\,(A) = 275848.6$.

Verify inequality 10.5:

$$\frac{\|\delta x\|}{\|x + \delta x\|} = \frac{6.64020}{1085.65605} = 0.0061163 \le \|A\| \left\|A^{-1}\right\| \frac{\|\delta A\|}{\|A\|} = 275848.64261 \left(3.66856 \times 10^{-8}\right) = 0.010120. \quad \blacksquare$$

Example 10.10. If $A = \begin{bmatrix} -3 & \frac{1}{2} & \frac{1}{3} \\ -36 & 8 & 6 \\ 30 & -7.5 & -6 \end{bmatrix}$ and $b = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}$, the solution is $x = \begin{bmatrix} -6.5000 \\ -66.0000 \\ 49.5000 \end{bmatrix}$. Perturb A by $\delta A = \begin{bmatrix} 0.0005 & 0 & 0.00001 \\ 0 & 0.0003 & 0 \\ 0 & 0 & -0.0007 \end{bmatrix}$ and b by $\delta b = \begin{bmatrix} -0.0001 \\ 0.0005 \\ -0.00002 \end{bmatrix}$, so $A + \delta A = \begin{bmatrix} -2.99950 & 0.50000 & 0.33334 \\ -36.00000 & 8.00030 & 6.00000 \\ 30.00000 & -7.50000 & -6.00070 \end{bmatrix}$ and $b + \delta b = \begin{bmatrix} 2.99990 \\ 3.00050 \\ 2.99998 \end{bmatrix}$. The solution to the perturbed system $x + \delta x = \begin{bmatrix} -6.48637 \\ -65.72708 \\ 49.22128 \end{bmatrix}$, so $\delta x = \begin{bmatrix} 0.013630 \\ 0.272920 \\ -0.27872 \end{bmatrix}$. The relative error is

$$\frac{\|\delta x\|}{\|x\|} = 0.0047166.$$

Considering that $\|\delta A\| / \|A\| = 1.43018 \times 10^{-5}$ and $\|\delta b\| / \|b\| = 9.82061 \times 10^{-5}$, this relative error is poor, indicating the A is ill-conditioned. Indeed, $\kappa\,(A) = 2.39661 \times 10^3$.

Verify inequality 10.6:

$$\begin{aligned} \frac{\|\delta x\|}{\|x\|} &= 0.0047166 \le \|A\| \left\|A^{-1}\right\| \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|A\|\ \|x + \delta x\|} \right) \\ &= 2.39661 \times 10^3 \left(1.43018 \times 10^{-5} + \frac{5.10294 \times 10^{-4}}{48.95519\,(82.37024)} \right) = 0.034579 \quad \blacksquare \end{aligned}$$

Definition 10.11. If the condition number of matrix A is large, we say A is *ill-conditioned*; otherwise, A is well-conditioned.

The term “large" is vague. Condition numbers in the range of 10^4 or more definitely indicate ill-conditioning. For some matrices, a smaller condition number can indicate ill-conditioning, so determining ill-conditioning is not an exact science.

10.6 PROPERTIES OF THE MATRIX CONDITION NUMBER

The matrix condition number has importance in various applications and in proving some highly useful results. This section develops properties of the condition number and provides examples that illustrate its properties.

Theorem 10.5. *Let A be a nonsingular matrix.*

1. $\kappa_p\,(A) \ge 1$ *for any p-norm.*
2. $\kappa_G\,(\alpha A) = \kappa_G\,(A)$*, where* $\alpha \ne 0$ *is a constant. Here* $\kappa_G\,(A)$ *refers to any matrix norm.*

3. *Let A be an orthogonal matrix. Then, $\kappa(A) = 1$ if and only if $A^{\mathrm{T}}A = \alpha I$, where $\alpha \neq 0$.*
4. $\kappa\left(A^{\mathrm{T}}A\right) = (\kappa(A))^2$.
5. $\kappa(A) = \kappa\left(A^{\mathrm{T}}\right); \kappa_1(A) = \kappa_\infty\left(A^{\mathrm{T}}\right)$.
6. *For any matrix norm, $\kappa(AB) \leq \kappa(A)\kappa(B)$.*
7. $\kappa(A) = \sigma_{\max}/\sigma_{\min}$ *where $\sigma_{\max}$ and $\sigma_{\min}$ are, respectively, the largest and smallest singular values of A.*

We will prove properties (2), (4), and (7). The remaining properties are left as exercises.

Proof. To prove (2), note that since $(\alpha A)\left(\frac{1}{\alpha}A^{-1}\right) = I$, $\frac{1}{\alpha}A^{-1}$ must be the inverse of αA, and so

$$\kappa_G(\alpha A) = \|\alpha A\| \left\|(\alpha A)^{-1}\right\| = |\alpha|\,\|A\| \left\|\frac{1}{\alpha}A^{-1}\right\| = \|A\| \left\|A^{-1}\right\| = \kappa_G(A).$$

To prove (4), the condition number of $A^{\mathrm{T}}A$ is $\kappa\left(A^{\mathrm{T}}A\right) = \left\|A^{\mathrm{T}}A\right\|_2 \left\|\left(A^{\mathrm{T}}A\right)^{-1}\right\|_2$, so we have to deal with the two factors $\left\|A^{\mathrm{T}}A\right\|_2$ and $\left\|\left(A^{\mathrm{T}}A\right)^{-1}\right\|_2$. In Theorem 7.9, part (4) we proved that

$$\left\|A^{\mathrm{T}}A\right\|_2 = \|A\|_2^2. \tag{10.14}$$

Recall the property $\left(A^{\mathrm{T}}\right)^{-1} = \left(A^{-1}\right)^{\mathrm{T}}$ from Theorem 1.6, part (5), so

$$\left\|\left(A^{\mathrm{T}}A\right)^{-1}\right\|_2 = \left\|A^{-1}\left(A^{\mathrm{T}}\right)^{-1}\right\|_2 = \left\|A^{-1}\left(A^{-1}\right)^{\mathrm{T}}\right\|_2 = \left\|A^{-1}\right\|_2^2. \tag{10.15}$$

From Equations 10.14 and 10.15, we have

$$\kappa\left(A^{\mathrm{T}}A\right) = \|A\|_2^2 \left\|A^{-1}\right\|_2^2 = \left(\|A\|_2 \left\|A^{-1}\right\|_2\right)^2 = (\kappa(A))^2.$$

Now consider property 7. By Theorem 7.7, $\|A\|_2 = \sqrt{\lambda_{\max}}$, where $\lambda_{\max}$ is the largest eigenvalue of $A^{\mathrm{T}}A$. From Theorem 7.9, part (5), $\left\|A^{-1}\right\|_2 = 1/\sqrt{\lambda_{\min}}$, where $\lambda_{\min}$ is the smallest eigenvalue of $A^{\mathrm{T}}A$. Then,

$$\|A\|_2 \left\|A^{-1}\right\|_2 = \frac{\sqrt{\lambda_{\max}}}{\sqrt{\lambda_{\min}}} = \frac{\sigma_{\max}}{\sigma_{\min}}.$$ □

Lemma 7.1 states that all norms on the vector space $\mathbb{R}^n$ are equivalent. That is, given any two norms $\|\cdot\|_a$ and $\|\cdot\|_b$, there exist constants C_l and C_h such that

$$C_l\|x\|_a \leq \|x\|_b \leq C_h\|x\|_a.$$

This same type of result also applies to the condition number of matrix norms. It answers the question "Can a condition number based on one norm be large and a condition number based on another norm be small?" The answer is "No." We state without proof the following relationships among norms [2, p. 88].

Theorem 10.6. *If A is an $n \times n$ matrix, any two condition numbers $\kappa_\alpha(A)$ and $\kappa_\beta(A)$ are equivalent in that there are constants c_1 and c_2 such that*

$$c_1\kappa(A) \leq \kappa_\beta(A) \leq c_2\kappa(A).$$

For $\mathbb{R}^{n\times n}$,

$$\begin{aligned}
\frac{1}{n}\kappa_2(A) &\leq \kappa_1(A) \leq n\kappa_2(A), \\
\frac{1}{n}\kappa_\infty(A) &\leq \kappa_2(A) \leq n\kappa_\infty(A), \\
\frac{1}{n^2}\kappa_1(A) &\leq \kappa_\infty(A) \leq n^2\kappa_1(A).
\end{aligned}$$

Thus, if a matrix is ill-conditioned in one norm, it is ill-conditioned in another, taking into account the constants c_1 and c_2.

We stated in Section 6.3 that orthogonal matrices are the most beautiful of all matrices. Now we can add another reason for supporting this claim.

Lemma 10.1. *In the 2-norm, an orthogonal matrix, P, is perfectly conditioned, in that $\kappa(P) = 1$.*

Proof. By Theorem 10.5, part 7, $\kappa(P) = \sigma_{\max}/\sigma_{\min} = 1$, since the singular values of P are the square roots of the eigenvalues of $P^{\mathrm{T}}P = I$. □

10.7 MATLAB COMPUTATION OF A MATRIX CONDITION NUMBER

To determine the condition number of a matrix, use the function `cond`. Its arguments are the matrix and one of the following:

- None (default is the 2-norm)
- 1 (the 1-norm)
- 2 (the 2-norm)
- 'inf' (the ∞-norm)
- 'fro' (the Frobenius-norm)

Here are the results when each of these norms are applied to the 10×10 bidiagonal matrix $A = \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 \\ 0 & \vdots & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & 1 & 1 \\ 0 & 0 & 0 & \dots & \dots & 1 \end{bmatrix}$

```
>>  A = diag(ones(10,1)) + diag(ones(9,1),1);
>> cond(A)

ans =
   13.2320

>> cond(A,'inf')

ans =
    20

>> cond(A,1)

ans =
    20

>> cond(A, 'fro')

ans =
   32.3265
```

10.8 ESTIMATING THE CONDITION NUMBER

Since the condition number is so important, we must either compute its value or have a good approximation. To compute $\kappa(A) = \left\|A^{-1}\right\|_2 \|A\|_2$ requires that we compute the maximum and minimum singular values of A. Recall from Theorem 10.6 that

$$\frac{1}{n}\kappa_2(A) \le \kappa_1(A) \le n\kappa_2(A),$$

and so it is reasonable to use $\kappa_1(A)$, since the matrix 1-norm is much easier to compute.

$$\|A\|_1 = \max_{1\le k\le n} \sum_{i=1}^{m} |a_{ik}|.$$

However, the problem of accounting for A^{-1} still remains. It can be shown that

$$\kappa_1(A) \geq \frac{\|A\|_1 \left\|A^{-1}w\right\|_1}{\|w\|_1} \tag{10.16}$$

for any nonzero $w \in \mathbb{R}^n$ [23, pp. 131-133]. If we choose w so that $\left\|A^{-1}w\right\|_1 / \|w\|_1$ is close to its maximum, Equation 10.16 will give a sharp lower bound for $\kappa_1(A)$. The Hager algorithm is most frequently used to estimate w (see Refs. [1, pp. 50-54], [19, pp. 141-143], and [28]).

Example 10.11. The MATLAB function `condest(A)` estimates the condition number of a square matrix.

```
>> H = hilb(35);   % the Hilbert matrices are notoriously ill-conditioned
>> condest(H)
ans =
  4.7538e+019

>> A = [1 2 3;3 4 5;6 7 8.00001];
>> condest(A)
ans =
  1.6e+007

>> cond(A)
ans =

  1.0991e+007
```
■

10.9 INTRODUCTION TO PERTURBATION ANALYSIS OF EIGENVALUE PROBLEMS

The 10×10 bidiagonal matrix in Section 10.7 is well-conditioned, since its condition number is 13.232. As it turns out, a matrix can be well-conditioned and yet its eigenvalues are sensitive to perturbations. As a result, the conditioning problem for eigenvalues must be considered separately from matrix conditioning. Perturbation analysis of eigenvalue problems will be discussed in Chapter 18, but at this point it is instructive to present some examples.

Example 10.12. The bidiagonal matrix, A, of Section 10.7 is an upper-triangular matrix, so its eigenvalues lie on the diagonal, and are all 1. After computing the eigenvalues of A, perturb $A(5,1)$ by 10^{-10} and compute the eigenvalues.

```
>> A = diag(ones(10,1)) + diag(ones(9,1),1);
>> eig(A)
ans =
      1
      1
     ...
      1
>> A(5,1) = 1.0e-10;
>> eig(A)
ans =

   0.9919 + 0.0059i
   0.9919 - 0.0059i
   1.0031 + 0.0095i
   1.0031 - 0.0095i
   1.0100
   1.0000
   1.0000
   1.0000
   1.0000
   1.0000
```

A perturbation of 10^{-10} in one entry caused four of the eigenvalues to become complex. ■

10.10 CHAPTER SUMMARY

Reasons Why the Study of Numerical Linear Algebra Is Necessary

Floating point roundoff and truncation error cause many problems. We have learned how to perform Gaussian elimination in order to row reduce a matrix to upper-triangular form. Unfortunately, if the pivot element is small, this can lead to serious errors in the solution. We will solve this problem in Chapter 11 by using partial pivoting. Sometimes an algorithm is simply far too slow, and Cramer's Rule is an excellent example. It is useful for theoretical purposes but, as a method of solving a linear system, should not be used for systems greater than 2×2. Solving $Ax = b$ by finding A^{-1} and then computing $x = A^{-1}b$ is a poor approach. If the solution to a single system is required, one step of Gaussian elimination, properly performed, requires far fewer flops and results in less roundoff error. Even if the solution is required for many right-hand sides, we will show in Chapter 11 that first factoring A into a product of a lower- and an upper-triangular matrix and then performing forward and back substitution is much more effective. A classical mistake is to compute eigenvalues by finding the roots of the characteristic polynomial. Polynomial root finding can be very sensitive to roundoff error and give extraordinarily poor results. There are excellent algorithms for computing eigenvalues that we will study in Chapters 18 and 19. Singular values should not be found by computing the eigenvalues of $A^{\mathrm{T}}A$. There are excellent algorithms for that purpose that are not subject to as much roundoff error. Lastly, if $m \neq n$ a theoretical linear algebra course deals with the system using a reduction to what is called reduced row echelon form. This will tell you whether the system has infinitely many solutions or no solution. These types of systems occur in least-squares problems, and we want a single meaningful solution. We will find one by requiring that x be such that $\|b - Ax\|_2$ is minimum.

Forward and Backward Error Analysis

Forward error deals with rounding errors in the solution of the problem. If the input is x, the solution is $f(x)$, and the result obtained is $\hat{f}(x)$, the forward error is $\left|f(x) - \hat{f}(x)\right|$. Normally we do not know the true solution $f(x)$, so we must resort to bounding the forward error. For instance, the forward error for the computation of the outer product of vectors x and y, xy^{T}, is bounded by eps $\times$ $\left|xy^{\mathrm{T}}\right|$, where $|.|$ returns the matrix with each entry replaced by its absolute value.

Backward error relates the rounding errors in the computation to the errors in the data rather than its solution. Generally, a backward error analysis is preferable to a forward analysis for this reason. Roundoff or other errors in the data have produced the result $\hat{y}$. The backward error is the smallest Δx for which $\hat{y} = f(x + \Delta x)$; in other words, backward error tells us what problem we actually solved. For instance, it can be shown that the inner product, $\langle x, y\rangle$, is the exact inner product for a perturbed set of data, where the perturbations are very small.

Algorithm Stability

Intuitively, an algorithm is stable if it performs well in general, and an algorithm is unstable if it performs badly in significant cases. In particular, an algorithm should not be unduly sensitive to errors in its input or errors during its execution. We saw in Chapter 8 that using the quadratic equation in its natural form is subject to serious cancellation error. There are two types of stable algorithms, backward stable, and forward stable.

An algorithm is backward stable if for any x, it computes $f(x)$ with small backward error, Δx. In other words, it computes the exact solution to a nearby problem,

$$f(x + \Delta x) = \hat{f}(x)$$

so that the solution is not sensitive to small perturbations in x. The addition of floating point numbers is backward stable, as is the computation of the inner product, and back substitution.

An algorithm is forward stable if whenever $f(x)$ is the true solution, the difference between the computed and true solutions is small. In other words,

$$\left|\hat{f}(x) - f(x)\right|$$

is small. The computation of the inner and outer product of two vectors is forward stable. We know from our discussion in Chapter 8 that floating-point arithmetic does not follow the laws of real arithmetic. This can make forward error analysis difficult. In backward error analysis, however, real arithmetic is employed, since it is assumed that the computed result is the exact solution to a nearby problem. This is one reason why backward error analysis is often preferred, so we will refer to stability to mean backward stability. We have seen that floating point addition and inner product are stable.

Computing the outer product is not backward stable. The root finding problem can be unstable, particularly in the presence of multiple roots. The Wilkinson polynomial is an example where all the roots are distinct, but their computation is unstable.

Conditioning of a Problem

A problem is ill-conditioned if a small relative change in its data can cause a large relative error in its computed solution, regardless of the algorithm used to solve the problem. If small perturbations in problem data lead to small relative errors in the solution, a problem is said to be well-conditioned. Let x and $\bar{x}$ be the original and slightly perturbed data, and let $f(x)$ and $f(\bar{x})$ be the respective solutions. Then,

- The problem is well-conditioned with respect to x if whenever $|x - \bar{x}|$ is small, $|f(x) - f(\bar{x})|$ is small.
- The problem is ill-conditioned with respect to x if whenever $|x - \bar{x}|$ is small, $|f(x) - f(\bar{x})|$ can be large.

The condition number defines the degree of conditioning for a particular problem. The chapter gives a mathematical definition of the condition number as the limiting ratio of the relative rate of change of the function divided by the relative change in the input. The larger the condition number, the more sensitive the problem is to errors.

Remark 10.7.

- Stable or unstable refers to an algorithm.
- Well or ill-conditioned refers to the particular problem, not the algorithm used.

Perturbation Analysis for Solving a Linear System

When solving a linear system, there are three situations we must consider, errors in the right-hand side, errors in the coefficient matrix, and errors in both. The text discusses each of these cases and, in each case, the expression

$$\|A\| \left\|A^{-1}\right\|$$

appears, where $\|.\|$ is any subordinate norm. This value is called the condition number of the matrix and is denoted by $\kappa(A)$. If the condition number is large, solving a linear system accurately is difficult.

Properties of the Matrix Condition Number

There are a number of important properties of the 2-norm matrix condition number. We list some of the most useful:

- If P is orthogonal, $\kappa(A) = 1$.
- $\kappa(A) = \sigma_{\max}/\sigma_{r\min}$

Using MATLAB to Compute the Matrix Condition Number

In MATLAB, compute the condition number using the function cond in one of the forms

`cond(A)`	2-norm
`cond(A,1)`	1-norm
`cond(A,'inf')`	∞-norm
`cond(A,'fro')`	Frobenius-norm

Approximating the Matrix Condition Number

The MATLAB function `condest` approximates the condition number of a matrix using the matrix 1-norm. We normally use this approximation with large sparse matrices, since the computation of the condition number is too costly.

Perturbation Analysis of Eigenvalue Problems

A matrix can be well-conditioned and yet its eigenvalues can be sensitive to perturbations. As a result, the conditioning problem for eigenvalues must be considered separately from matrix conditioning. Perturbation analysis of eigenvalue problems will be discussed in Chapter 18,

10.11 PROBLEMS

10.1 Show that the floating point multiplication of two numbers is backward stable.

10.2 Prove that a small residual for the solution of $Ax = b$ implies backward stability.

10.3 Prove that if x and y are in $\mathbb{R}^n$, the $n \times n$ matrix xy^{T} has rank 1.

10.4 If A and B are matrices and α is a floating point number, prove the following forward error results.

a. $\mathrm{fl}(\alpha A) = \alpha A + E,\ |E| \le \mathrm{eps}\ |\alpha A|$.

b. $\mathrm{fl}(A + B) = (A + B) + E,\ |E| \le \mathrm{eps}\ |A + B|$.

10.5 Show that the roots of the polynomial $x^3 - 12x^2 + 48x - 64$ are ill-conditioned and explain why.

10.6 Let $f(x) = \ln x$.

a. Show that the condition number of f at x is $c(x) = 1/|\ln x|$.

b. Using the above result, show that $\ln x$ is ill-conditioned near $x = 1$.

10.7 Show that computing $\sqrt{x}$ for $x > 0$ is well-conditioned.

10.8 What is the condition number for $f(x) = x/(x-1)$ at x? Where is it ill-conditioned?

10.9 Let A be a nonsingular square matrix.

a. Prove that if A is a scalar multiple of an orthogonal matrix, then $\kappa_2(A) = 1$.

b. Prove that if $\kappa_2(A) = 1$, then A is a scalar multiple of an orthogonal matrix. You may assume the singular value decomposition that says

$$A = U\Sigma V^{\mathrm{T}},$$

where U and V are orthogonal and Σ is a diagonal matrix of singular values.

10.10 If A is an $m \times n$ matrix, and x is an $n \times 1$ vector, then the linear transformation $y = Ax$ maps $\mathbb{R}^n$ to $\mathbb{R}^m$, so the linear transformation should have a condition number, $\mathrm{cond}_{Ax}(x)$. Assume that $\|\cdot\|$ is a subordinate norm.

a. Show that we can define $cond_{Ax}(x) = \|A\|\,\|x\|/\|Ax\|$ for every $x \neq 0$.

b. Find the condition number of the linear transformation at $x = \begin{bmatrix} 1 & -1 & 2 \end{bmatrix}^{\mathrm{T}}$ using the ∞-norm.

$$T = \begin{bmatrix} 1 & 7 & -1 \\ 3 & 2 & 1 \\ 5 & -9 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

c. Show that $cond_{Ax}(x) \le \|A\|\,\|A^{-1}\|$ for all x.

d. Verify the result of part (c) for the matrix and vector of part (b).

10.11 Prove the following properties of the condition number.

a. $\kappa_p(A) \ge 1$ for any p-norm.

b. $\kappa_2(A) = \kappa_2(A^{\mathrm{T}})$; $\kappa_1(A) = \kappa_\infty(A^{\mathrm{T}})$.

c. For any sub-multiplicative matrix norm, $\kappa(AB) \le \kappa(A)\,\kappa(B)$.

10.12 Let $A = \begin{bmatrix} 1 & a \\ a & 1 \end{bmatrix}$. For what values of a is A ill-conditioned? What happens as $a \to \infty$.

10.13 Prove that for any orthogonal matrix P, $\kappa(PA) = \kappa(AP) = \kappa(A)$ for any $n \times n$ matrix A.

10.14 There is an equivalent definition of $\kappa(A)$ [27, pp. 84-85].

The condition number $\kappa(A)$ of a nonsingular matrix A is

$$\frac{1}{\kappa(A)} = \min_{B \in S_n} \left\{ \frac{\|A - B\|_2}{\|A\|_2} \right\},$$

where S_n is the set of singular $n \times n$ matrices.

a. If A is ill-conditioned, what does this say about the relative distance of A from the subset of $n \times n$ singular matrices

b. If A is a well-conditioned matrix, answer the question of part (a).

10.11.1 MATLAB Problems

10.15

a. Write a MATLAB function that builds the matrix

$$A = \begin{bmatrix} n & n-1 & n-2 & \dots & 3 & 2 & 1 \\ n-1 & n-1 & n-2 & \dots & 3 & 2 & 1 \\ 0 & n-2 & n-2 & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & & & \ddots & \ddots & 2 & \vdots \\ \vdots & & & & 2 & 2 & 1 \\ 0 & \dots & \dots & \dots & 0 & 1 & 1 \end{bmatrix}.$$

b. For $n = 5, 10, 15, 20$:

i. Compute the vector of eigenvalues and assign it to the variable E1.

ii. Perturb $A\,(1, n-1)$ by 10^{-8} and assign the eigenvalues of the perturbed matrix to the variable E2.

iii. Compute $\|E1 - E2\|_2$.

c. Which eigenvalues appear to be perturbed the most?

d. The function `eigcond` in the software distribution with calling format

```
[c lambda] = eigcond(A)
```

computes the condition number c_i of eigenvalue λ_i. We will develop the function code in Section 18.11. Run `eigcond` for each of the matrices in part (b). Do the results confirm your conclusions of part (c)?

10.16 **a.** Let $x = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \end{bmatrix}$ and execute

```
A = gallery('sampling', x)
```

Is A ill-conditioned?

b. Execute the following code:

```
>> E = 0.0001*rand(8);
>> B = A+E;
>> eig(A)
>> eig(B)
```

c. What do you conclude from the code? Justify your answer using `eigcond` introduced in Problem 10.15.

10.17 Let `A = gallery(3)`.

a. Is A ill-conditioned?

b. Find the eigenvalues of A. Perturb A by a random matrix E whose values do not exceed 10^{-5}. Does it appear the eigenvalues of A are ill-conditioned?

c. Find eigenvectors for A using the command `[V E] = eig(A)`. Perturb the elements of A by 0.0001*rand(3,3) and compute the eigenvectors. What conclusion can you make?

10.18 `Let A = gallery(5)`. Answer questions (a)-(c) in Problem 10.17 for this matrix.

10.19 A magic square of order n is an $n \times n$ matrix with entries $1 \dots n^2$, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. MATLAB will create a magic square with the command `A = magic(n)`.

a. What is the largest eigenvalue of `magic(n)` for $n = 5, 8, 15$? In general, what is the largest eigenvalue of a magic square of order n? Prove it. Hint: Look at the Perron-Frobenius theorem, Theorem 5.6 in this book.

b. What is the largest singular value of `magic(n)` for $n = 5, 8, 15$? In general, what is the largest singular value of a magic square of order n?

10.20 The symmetric Pei matrix is defined by

```
alpha*eye(n) + ones(n)
```

Enter the anonymous function

```
p = @(alpha,n) alpha*eye(n) + ones(n);
```

of two variables that creates a Pei matrix.

Fix $n = 25$, and draw a graph of `alpha = 0.5:-.01:.01` vs. `cond(p(alpha(i),25))`. What do you conclude?

10.21 The *Wilkinson bidiagonal matrix* has the general form

$$A = \begin{bmatrix} n & n & & & \\ & n-1 & n & & \\ & & \ddots & \ddots & \\ & & & 2 & n \\ & & & & 1 \end{bmatrix},$$

and is often used for testing purposes.

a. Create an anonymous function using `diag` that builds an $n \times n$ Wilkinson bidiagonal matrix.

b. Graph the condition number of the Wilkinson bidiagonal matrices of orders 1 through 15. What are your conclusions?

c. Eigenvalues are tricky to compute accurately. The 20×20 *Wilkinson-bidiagonal matrix*, A, has eigenvalues $\lambda = 20, 19, \ldots, 1$. This matrix illustrates that even though the eigenvalues of a matrix are not equal or even close to each other, an eigenvalue problem can be ill-conditioned. Compute the eigenvalues of A, perturb $A\,(20, 1)$ by 10^{-10} and compute the eigenvalues. Comment on the results.

10.22 The symmetric rosser matrix is particularly nasty, intentionally. Investigate the conditioning of the matrix and its eigenvalues. If the eigenvalues appear well-conditioned, run an experiment that demonstrates this. Generate the Rosser matrix with the command

```
A = rosser;
```

10.23 The $n \times n$ Hilbert matrices have elements $h_{ij} = 1/(i+j-1)$ and are notoriously ill-conditioned.

a. Create `A = hilb(8)`. Perturb A(1,8) by 10^{-5}, and call the perturbed matrix B. Let `b = rand(8,1)`. Compute `x = A\b`, `y = B\b`, and follow by computing $\|x-y\|$ and $\|x-y\|/\|x\|$. Is ill-conditioning evident?

b. Compute the condition number of `hilb(8)` using the MATLAB command `cond(A)` and then use it to compute the theoretical upper bound given in Theorem 10.4, part (2). Do your calculations verify the inequality?

10.24 Let x be a known value, such as `ones(n,1)` and compute b = Ax. Theoretically, $\bar{x} = A\backslash b$ should be the known solution x. However, due to the nature of floating point computation the relative error $\|x-\bar{x}\|/\|x\|$ is not expected to be 0, but it should be small if A is not ill-conditioned. The idea for this problem is to observe ill-conditioning using the matrix `A = gallery('lotkin', n)`. Perform the steps presented in the following algorithm and comment on the results. The notation $\langle \ldots \rangle$ in a print statement indicates the value to print. For instance,

$$print\left('The\ condition\ number\ of\ A\ is',\ \langle condition\ number\ with\ 8\ significant\ digits\rangle\right)$$

```
procedure ERRORTEST
   for n = 2:11 do
      A = gallery('lotkin',n)
      $x = [1\ 1\ \ldots\ 1\ 1]^T$
      b = Ax
      $\hat{x} = A\backslash b$
      Print "The condition number of A is ⟨$\eta(A)$ with 15 significant digits⟩"
      Print "The relative error is ⟨$\frac{\|x-\bar{x}\|}{\|x\|}$ with 15 significant digits⟩"
   end for
end procedure
```

10.25 The goal of this exercise is to observe the behavior of $\kappa\,(H_n)$ as n goes to ∞, where H_n is the $n \times n$ Hilbert matrix, defined by $(H_n)_{ij} = 1/(i+j-1)$. In MATLAB, generate H_n using the command `H = hilb(n)`. Let n vary from 2 to 500, and make a plot of n versus $log_{10}\,(\eta\,(H_n))$. Draw conclusions about the experimental behavior. Hint: Use the MATLAB plotting function `semilogy`.

Problems 10.26 and 10.27 deal with non-square systems.

The technique of linear least-squares solves a system $Ax = b$ by minimizing the residual $\|b-Ax\|_2$. Chapter 12 introduces the topic and Chapter 16 discusses it in depth. Least-squares problems usually involve dealing with $m \times n$ systems, $m \neq n$. Since solutions are obtained using floating point arithmetic, they are subject to errors, and it certainly is reasonable to ask the questions

"Is there a definition of ill-conditioning for non-square systems?".

"Can perturbations in entries of an $m \times n$ system, $m \neq n$ cause large fluctuations in the solution?"

The answer to both of these questions is "yes." Define the condition number of an $m \times n$ matrix as σ_1/σ_k, where σ_1 is the largest and σ_k the smallest nonzero singular values of A. Of course if A is square, this is $\eta(A)$. As we have noted, never compute singular values by finding the eigenvalues of $A^{\mathrm{T}}A$. The MATLAB command

```
S = svd(A)
```

returns a vector S containing the singular values of A:

$$\sigma_1 = S(1) \geq \sigma_2 = S(2) \geq \ldots \sigma_k = S(k).$$

10.26 For each matrix, use `svd` to compute the condition number and verify your result by using MATLAB's `cond`. Note that you must find the smallest nonzero singular value, since is possible that there will be zero singular values. Explain the differences between $\kappa(A)$, $\kappa(B)$ and $\kappa(C)$.

a. $A = \begin{bmatrix} 1 & 3 & -1 \\ 8 & -4 & 12 \\ 1 & 9 & 0 \\ -1 & 7 & -8 \\ 5 & 6 & 1 \end{bmatrix}$

b. $B = \begin{bmatrix} 1 & 9 & 0 & 1 & -1 & 15 & 7 \\ 1 & -5 & 3 & -2 & 1 & -18 & 0 \\ 27 & 1 & 7 & -1 & 1 & 9 & -2 \\ 1 & 5 & -6 & 20 & 33 & 55 & 98 \end{bmatrix}$

c. $C = \begin{bmatrix} 1 & 2 & 8 & 19 \\ 5 & -1 & 29 & 62 \\ -1 & 3 & -3 & -4 \\ 3 & 5 & 23 & 54 \\ 1 & 6 & 12 & 31 \end{bmatrix}$

10.27 The QR decomposition of an $m \times n$ matrix is a decomposition of a matrix A into a product $A = QR$, where Q is an orthogonal matrix and R is an upper-triangular matrix. We will extensively discuss this decomposition in Chapters 14 and 17. There are two types of QR decomposition, the full and the reduced:

Full $A^{m\times n} = Q^{m\times m}R^{m\times n}$
Reduced $A^{m\times n} = Q^{m\times n}R^{n\times n}$

For $m > n$, the reduced decomposition saves space. To obtain a reduced QR decomposition, use the MATLAB function `qr` as follows:

```
[Q R] = qr(A,0).
```

For reasons we will explain in Chapter 16, if $m > n$, and $rank(A) = n$, one method of solving a least-squares problem is to follow these steps:

```
procedure SOLVELQ(A,b)
   Compute the reduced QR decomposition of A
   $c = Q^{\mathrm{T}}b$
   Solve the upper-triangular system $Rx = c$.
   return x
end procedure
```

a. Implement solveq as a MATLAB function. Use `backsolve` from Chapter 9 to solve the upper-triangular system $Rx = c$.

b. Solve the system $\begin{bmatrix} 2 & 7 & 1 \\ -1 & 6 & 2 \\ 0 & 1 & 8 \\ 5 & 9 & 10 \\ 4 & 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \\ 0 \\ -1 \\ 5 \end{bmatrix}.$

10.28 A *Toeplitz matrix* is a matrix whose entries are constant along each diagonal. The MATLAB command

```
T = toeplitz(c,r)
```

returns a Toeplitz matrix T having c as its first column and r as its first row. If the first elements of c and r are different, a message is printed and the column element is used.

a. Build the 100×100 pentadiagonal matrix

$$A = \begin{bmatrix} 6 & -4 & 1 & 0 & \dots & 0 \\ -4 & 6 & -4 & 1 & \dots & 0 \\ 1 & -4 & 6 & -4 & \ddots & \vdots \\ 0 & 1 & -4 & \ddots & \ddots & 1 \\ \vdots & \vdots & \ddots & -4 & 6 & -4 \\ 0 & 0 & \dots & 1 & -4 & 6 \end{bmatrix}.$$

b. Verify that the matrix is ill-conditioned.

c. Run the following experiment to observe the behavior of the eigenvalues of A as it is slightly perturbed. The MATLAB program computes the eigenvalues of a randomly perturbed A and outputs the norm of the difference between the eigenvalues of A and those of the perturbed matrix. The perturbations to the elements of A are in the range $0 < a_{ij} < 10^{-8}$.

```
E1 = eig(T);
E1 = sort(E1);
for i = 1:3
    deltaT = 1.0e-8*rand(100,100);
    B = T + deltaT;
    E2 = eig(B);
    E2 = sort(E2);
    norm(E1-E2)
end
```

d. Are the eigenvalues of A ill-conditioned? For more information about this problem, see Ref. [29].

10.29 Use MATLAB's function `condest` to estimate the condition number of the 12×12 Hilbert matrix. Find the true 1-norm condition number and compare the results. Explain the warning message you receive when computing the true 1-norm condition number.

10.30 The MATLAB statement `A = rand(1000,1000)` creates a 1000×1000 random matrix. In order to observe the computational time difference between finding the condition number of A and estimating the condition number, execute

```
>> A = rand(1000,1000);
>> tic;cond(A);toc
>> tic;condest(A);toc
```

Comment on the results.

10.31 The $n \times (m+1)$ Vandermonde matrix $V = \begin{bmatrix} 1 & t_1 & \dots & t_1^m \\ 1 & t_2 & \dots & t_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & \dots & t_n^m \end{bmatrix}$ is created from the vector $t = \begin{bmatrix} t_1 & t_2 & \dots & t_{n-1} & t_n \end{bmatrix}^{\mathrm{T}}$ and integer m. It has uses in polynomial approximation and other areas, and we will formally introduce it in Chapter 12. The function `vandermonde(t,m)` in the software distribution constructs the *Vandermonde matrix*, which is square if $m = n - 1$.

a. Construct the 11×11 Vandermonde matrix using $t = 1.0 : 0.1 : 2.0$. Compute `cond(A)` and `condest(A)`.

b. Construct the 21×21 Vandermonde matrix using $t = 1.0 : .05 : 2.0$. Compute `cond(A)` and `condest(A)`.

c. Are these matrices ill-conditioned? Is it safe to use these matrices in an application involving Gaussian elimination?

10.32 This problem deals with the instability of polynomial root finding. First, note how MATLAB handles polynomial operations by doing some simple computations. A polynomial, p, in MATLAB is an $n+1$-dimensional vector, where $p(1)$ is the coefficient of x^n, $p(2)$ is the coefficient of x^{n-2}, ..., $p(n+1)$ is the coefficient of x^0.

a. Show how to represent the polynomial $p(x) = x^4 - x^3 + x - 1$ in MATLAB.

b. To multiply two polynomials, use the MATLAB function `conv`. For instance, to compute the coefficients of $\left(x^2+1\right)(x+3)$, proceed as follows:

```
>> factor1 = [1 0 1];
>> factor2 = [1 3];
>> p = conv(factor1,factor2)

p =
     1     3     1     3
% analytical result is x^3 + 3x^2 + x + 3
```

Using `conv` find the coefficients of $(x-1)^4(x-2)(x+1)$ and assign them to the vector `p`.

c. Using the MATLAB function `roots`, compute the roots of $(x-1)^4(x-2)(x+1)$. Explain the results.

Let a_i be the coefficient of x^i in polynomial $p(x)$ and r be a simple root of p. Suppose roundoff error perturbs a_i by an amount δa_i, so the root now becomes $r+\delta r$. Does a small δa_i result in a small δr? We can answer that question if we have a formula for the condition number,

$C_{a_i}(r)$, for the computation. By applying Theorem 2.1 in Ref. [30],

$$C_{a_i}(r) = \frac{\left|a_i r^{i-1}\right|}{|p'(r)|}.$$

Use this result in Problems 10.33 and 10.34.

10.33 If `p` is the MATLAB representation of a polynomial, the function `polyder` computes the derivative of p.

a. Using `polyder`, compute the derivative of the function in part (b) of Problem 10.32.

b. Compute the condition number for the roots $x_2 = 2$ and $x_3 = -1$ of the polynomial in part (b) of Problem 10.32 at the coefficient of x^5. Do your results confirm your explanation in part (c) of Problem 10.32?

10.34 The function `wilkpoly` in the software distribution computes the MATLAB representation of the Wilkinson polynomial

$$p(x) = (x-1)(x-2)(x-3)\dots(x-19)(x-20).$$

a. Compute the roots of the polynomial after a perturbation of -2^{-23} in the coefficient of x^{19}.

b. Using the definition of $C_{a_i}(x_k)$, compute the sensitivity of each root $x_k = k$, $1 \le k \le 20$ relative to a change in the coefficient of x^{19}. Use the function `polyder` discussed in Problem 10.33. Do your results agree with the errors in the roots you computed in part (a)?