

Chapter 3

FLOATING POINT NUMBERS AND ERRORS IN COMPUTATIONS

ch3

Background Material Needed

- Special matrices (Section 2.4)
- Matrix and vector norms (Section 2.5)

3.1 Floating Point Number Systems

Most scientific and engineering computations on a computer are performed using **floating-point arithmetic**. Computers may have different bases, though base 2 is most common. The other commonly used bases are 10 and 16. Most hand calculators use base 10, while IBM mainframes use base 16.

A t -digit **floating point** number in base β has the form:

$$x = \pm m \cdot \beta^e,$$

where m is a t -digit fraction, called **mantissa**, and e is called **exponent**. If the first digit of the mantissa is different from zero, then the floating point number is called **normalized**. Thus 0.3457×10^5 is a 4-digit normalized decimal floating number, whereas 0.03457×10^6 is a five-digit unnormalized decimal floating point number.

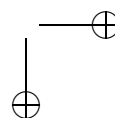
The number of digits in the mantissa is called the **precision**. On many computers, it is possible to manipulate floating point numbers so that a number can be represented with about twice the usual precision. Such a precision is called **double precision**.

Most computers nowadays conform to the IEEE floating point standard (ANSI/IEEE standard 754-1985). For a single-precision, IEEE standard recommends about 24 binary digits and for a double precision, about 53 binary digits.

IEEE Floating Point Standard

- Single precision

1	23	8
Sign	Mantissa	Exponent



- Double Precision

1	52	11
Sign	Mantissa	Exponent

Thus, IEEE standard for *single precision provides approximately seven decimal digits of accuracy*, since $2^{-23} \approx 1.2 \times 10^{-7}$, and *double precision provides approximately sixteen decimal digits of accuracy*, since $2^{-52} \approx 2.2 \times 10^{-16}$.

Note: Although computations with double precision increase accuracy, they require more computer time and storage.

On each computer, there is an allowable range of the exponent e : L , the minimum; U , the maximum. **L and U vary from computer to computer.**

If, during computations, the computer produces a number whose exponent is too large (too small), that is, it is outside the permissible range, then we say that an **overflow (underflow)** has occurred.

Overflow is a serious problem; for most systems, the result of an overflow is $\pm\infty$. Underflow is usually considered less serious. On most computers, when an underflow occurs, the computed value is set to zero, and then computations proceed. Unless otherwise stated, we will use only decimal arithmetic.

E2.1.1 Example 3.1 Examples of Overflow and Underflow

1. Let $\beta = 10$, $t = 3$, $L = -3$, $U = 3$.

$$a = 0.111 \times 10^3, \quad b = 0.120 \times 10^3$$

$$c = a \times b = .133 \times 10^5$$

will result in an **overflow**, because the exponent 5 is too large.

2. Let $\beta = 10$, $t = 3$, $L = -2$, $U = 3$

$$a = 0.1 \times 10^{-1},$$

$$b = 0.2 \times 10^{-1}$$

$$c = ab = 2 \times 10^{-4}$$

will result in an **underflow**. ■

Simple mathematical computations such as finding a **square root**, or **exponent** of a number or computing **factorials** can give overflow. For example, consider computing

$$c = \sqrt{a^2 + b^2}.$$

If a or b is very large, then we will get overflow while computing $a^2 + b^2$.

The IEEE standard also sets forth the results of operations with infinities and NaNs. All operations with infinities correspond to the limiting case in real analysis. Those ambiguous situations, such as $0 \cdot \infty$, result in NaNs, and all binary operations with one or two NaNs result in a NaN.

Avoiding overflow: An Example

Overflow and underflow can **sometimes** be avoided just by organizing the computations differently. Consider, for example, the task of computing the length of an n -vector x with components; denoted by $\|x\|_2$:

$$\|x\|_2^2 = x_1^2 + x_2^2 + \cdots + x_n^2.$$

If some x_i is too big or too small, then we can get overflow or underflow with the usual way of computing $\|x\|_2$. However, if we normalize each component of the vector by dividing it by $m = \max(|x_1|, \dots, |x_n|)$ and then form the squares and the sum, then overflow problem can be avoided. Thus, a better way to compute $\|x\|_2^2$ will be:

1. $m = \max(|x_1|, \dots, |x_n|)$
2. $y_i = x_i/m, i = 1, \dots, n$
3. $\|x\|_2 = m\sqrt{(y_1^2 + y_2^2 + \cdots + y_n^2)}$

3.2 Rounding Errors

If a computed result of a given real number is not machine representable, then there are two ways it can be represented in the machine. Consider

$$\pm \cdot d_1 \cdots d_t d_{t+1} \cdots$$

Then the first method, *chopping*, is the method in which the digits from d_{t+1} on are simply chopped off. The second method is **rounding**, in which the digits d_{t+1} through the rest are not only chopped off, but the digit d_t is also rounded up or down depending on whether $d_{t+1} \geq \beta/2$ or $d_{t+1} < \beta/2$.

Let $\text{fl}(x)$ denote the floating point representation of a real number x .

E2.2.1 Example 3.2 Rounding

Consider base 10. Let $x = 3.141596$

$$\begin{aligned} t = 2, \quad \text{fl}(x) &= 3.1 \\ t = 3, \quad \text{fl}(x) &= 3.14 \\ t = 4, \quad \text{fl}(x) &= 3.142 \end{aligned}$$

■

We now give an expression to measure the error made in representing a real number x on the computer, and then show how this measure can be used to give bounds for errors in other floating point computations.

D2-1 **Definition 3.3.** Let \hat{x} denote an approximation of x , then there are two ways we can measure the error:

$$\begin{aligned} \text{Absolute Error} &= |\hat{x} - x| \\ \text{Relative Error} &= \frac{|\hat{x} - x|}{|x|}, \quad x \neq 0. \end{aligned}$$

The relative error makes more sense than the absolute error. The following simple example shows this.

E2.2.2 **Example 3.4 (Relative Error Versus Absolute Error)** Consider

$$x_1 = 1.31, \hat{x}_1 = 1.30$$

$$\text{and } x_2 = 0.12, \hat{x}_2 = 0.11$$

The absolute errors in both cases are the same: $|\hat{x}_1 - x_1| = |\hat{x}_2 - x_2| = 0.01$. On the other hand, the relative error in the first case is: $\frac{|\hat{x}_1 - x_1|}{|x_1|} = 0.0076335$ and the relative error in the second case is: $\frac{|\hat{x}_2 - x_2|}{|x_2|} = 0.0833333$. Thus, the relative errors show that \hat{x}_1 is closer to x_1 than \hat{x}_2 is to x_2 , whereas the absolute errors give no indication of this at all. ■

The relative error also gives an indication of the number of significant digits in an approximate answer. *If the relative error is about 10^{-s} , then x and \hat{x} agree to about s significant digits.* More specifically,

D2-2 **Definition 3.5.** \hat{x} is said to approximate x to s significant digits if s is the largest non-negative integer for which the relative error $\frac{|x - \hat{x}|}{|x|} < 5(10^{-s})$; that is, s is given by $s = \left\lceil -\log \left(\frac{|x - \hat{x}|}{|x|} \right) + \frac{1}{2} \right\rceil$.

Thus, in the above examples, \hat{x}_1 and x_1 agree to **two** significant digits, while \hat{x}_2 and x_2 agree to about only **one** significant digit.

Round-off Error in Representation of a Real Number

We now give an expression for the relative error in representing a real number x by its floating point representation $\text{fl}(x)$:

T2.2.1 **Theorem 3.6.** Let $\text{fl}(x)$ denote the floating point representation of a real number x . Then

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \mu = \begin{cases} \frac{1}{2}\beta^{1-t} & \text{for rounding} \\ \beta^{1-t} & \text{for chopping} \end{cases}. \quad (3.1) \quad \text{eq3.2.1}$$

Proof. We establish the bound for rounding and leave the other part as an **[Exercise 3.1]**.

Let x be written as

$$x = (\cdot d_1 d_2 \cdots d_t d_{t+1} \cdots) \times \beta^e$$

where $d_1 \neq 0$ and $0 \leq d_i < \beta$. When we round off x we obtain one of the following floating point numbers:

$$x' = (\cdot d_1 d_2 \cdots d_t) \times \beta^e$$

$$x'' = [(\cdot d_1 d_2 \cdots d_t) + \beta^{-t}] \times \beta^e.$$

Obviously we have $x \in (x', x'')$. Assume, without any loss of generality, that x is closer to x' . We then have

$$|x - x'| \leq \frac{1}{2}|x' - x''| = \frac{1}{2}\beta^{e-t}.$$

Thus, the relative error

$$\begin{aligned} \frac{|x - x'|}{|x|} &\leq \left(\frac{1}{2} \frac{\beta^{-t}}{d_1 d_2 \cdots d_t \cdots} \right) \\ &\leq \frac{1}{2} \frac{\beta^{-t}}{\frac{1}{\beta}} \text{ (since } d_i < \beta) = \frac{1}{2} \beta^{1-t}. \end{aligned}$$

□

E2.2.3 **Example 3.7** Consider the three digit representation of the decimal number $x = 0.2346$ ($\beta = 10$, $t = 3$). Then, if **rounding** is used, we have:

$$\begin{aligned} \text{fl}(x) &= 0.235 \\ \text{Relative Error} &= 0.001705 < \frac{1}{2}10^{-2}. \end{aligned}$$

Similarly, if **chopping** is used, we have:

$$\begin{aligned} \text{fl}(x) &= 0.234 \\ \text{Relative Error} &= 0.0025575 < 10^{-2}. \end{aligned}$$

■

Definition 3.8. The number μ in [eq3.2.1](#) is called the machine precision or unit roundoff error. It is the smallest positive floating point number such that:

$$\text{fl}(1 + \mu) > 1.$$

The machine precision, μ , is usually between 10^{-16} and 10^{-7} (on most machines), for double and single precision, respectively. For the IBM 360 and 370, $\beta = 16$, $t = 6$, $\mu = 4.77 \times 10^{-7}$.

The machine precision is very important in scientific computations. If the particulars β , t , L and U for a computer are not known, the following simple FORTRAN program can be run to estimate μ for that computer (Forsythe, Malcolm and Moler (1977), p. 14.)

```
REAL MEU, MEU 1
MEU = 1.0
10 MEU = 0.5 * MEU
MEU 1 = MEU + 1.0
IF (MEU 1.GT.1.0) GOTO 10
```

The above FORTRAN program computes an approximation of μ which differs from μ by at most a factor of 2. This approximation is quite acceptable, since an exact value of μ is not that important and is seldom needed.

The book by Forsythe, Malcolm and Moler (1977) also contains an extensive list of L and U for various computers.

3.3 Laws of Floating Point Arithmetic

The formula

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \mu = \begin{cases} \beta^{1-t} & \text{for chopping} \\ \frac{1}{2}\beta^{1-t} & \text{for rounding} \end{cases}$$

can be written as

$$\text{fl}(x) = x(1 + \delta) \quad \text{where } |\delta| \leq \mu \quad (3.2) \quad \boxed{\text{eq3.3.1}}$$

Assuming that the IEEE standard holds, we can easily derive the following simple laws of floating point arithmetic.

T2.3.1 **Theorem 3.9.** *Let x and y be two floating point numbers, and let $\text{fl}(x + y)$, $\text{fl}(x - y)$, $\text{fl}(xy)$, and $\text{fl}(x/y)$ denote the computed sum, difference, product and quotient. Then*

1. $\text{fl}(x \pm y) = (x \pm y)(1 + \delta)$, where $|\delta| \leq \mu$.
2. $\text{fl}(xy) = (xy)(1 + \delta)$, where $|\delta| \leq \mu$.
3. if $y \neq 0$, then $\text{fl}(x/y) = (x/y)(1 + \delta)$, where $|\delta| \leq \mu$.

On computers that do not use the IEEE standard, the following floating point law of addition might hold:

4. $\text{fl}(x + y) = x(1 + \delta_1) + y(1 + \delta_2)$ where $|\delta_1| \leq \mu$, and $|\delta_2| \leq \mu$.

E2.3.1 **Example 3.10 Simple Floating Point Operations with Rounding**

Let

$$\beta = 10, \quad t = 3$$

in 1 through 3.

1. $x = 0.999 \times 10^2$, $y = 0.111 \times 10^0$.

$$\begin{aligned} x + y &= 100.0110 = .100011 \times 10^3 \\ \text{fl}(x + y) &= 0.100 \times 10^3 \end{aligned}$$

Thus, $\text{fl}(x + y) = (x + y)(1 + \delta)$, where

$$\delta = -1.0999 \times 10^{-4}, \quad |\delta| < \frac{1}{2}(10^{-2}).$$

$$2. x = 0.999 \times 10^2, y = 0.111 \times 10^0.$$

$$xy = 11.0889$$

$$\text{fl}(xy) = 0.111 \times 10^2$$

Thus, $\text{fl}(xy) = xy(1 + \delta)$, where

$$\delta = 1.00100 \times 10^{-3}, \quad |\delta| \leq \frac{1}{2}(10^{1-3}).$$

$$3. x = 0.999 \times 10^2, y = 0.111 \times 10^0.$$

$$\frac{x}{y} = 900$$

$$\text{fl}\left(\frac{x}{y}\right) = 0.900 \times 10^3$$

$$\delta = 0.$$

4. Let

$$\beta = 10, \quad t = 4$$

$$x = 0.1112, \quad y = 0.2245 \times 10^5$$

$$xy = 0.24964 \times 10^4,$$

$$\text{fl}(xy) = 0.2496 \times 10^4$$

Thus, $|\text{fl}(xy) - xy| = .44$ and

$$|\delta| = 1.7625 \times 10^{-4} < \frac{1}{2} \times 10^{-3}$$

■

Computing Without a Guard-Digit

Theorem ^{¶2.3.1} 3.9 and the examples following this theorem show that the relative errors in computing the sum, difference, product and quotient in floating point arithmetic are small. However, there are computers without guard digits in which additions and subtractions may not be accurate.

A guard digit is an extra digit on the lower end of the arithmetic register whose purpose is to catch the low order digit which would otherwise be pushed out of existence when the decimal points are aligned.

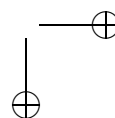
For computers with a guard digit

$$\text{fl}(x \pm y) = (x + y)(1 + \delta) \quad |\delta| \leq \mu$$

However, for those without a guard digit

$$\text{fl}(x \pm y) = x(1 + \delta_1) \pm y(1 + \delta_2),$$

$$|\delta_1| \leq \mu, \quad |\delta_2| \leq \mu.$$



Remark: Throughout this book, we will assume that the computations have been performed with a guard digit, as they are on almost all available machines.

We shall call results 1 through 3 of Theorem 3.9 along with (3.2) the **fundamental laws of floating point arithmetic**. These fundamental laws form the basis for establishing bounds for relative errors in other floating point computations.

E3.3.2 **Example 3.11** Consider the floating point computation of $x(y + z)$:

$$\begin{aligned} \text{fl}(x(y + z)) &= [x \cdot \text{fl}(y + z)](1 + \delta_1) \\ &= x(y + z)(1 + \delta_2)(1 + \delta_1) \\ &= x(y + z)(1 + \delta_1\delta_2 + \delta_1 + \delta_2) \\ &\approx x(y + z)(1 + \delta_3), \end{aligned}$$

where $\delta_3 = \delta_1 + \delta_2$; since δ_1 and δ_2 are small, their product is neglected.

We can now easily establish the bound of δ_3 . Suppose $\beta = 10$, and that rounding is used. Then

$$\begin{aligned} |\delta_3| &= |\delta_1 + \delta_2| \leq |\delta_1| + |\delta_2| \\ &\leq \frac{1}{2} \times 10^{1-t} + \frac{1}{2} \times 10^{1-t} \\ &= 10^{1-t}. \end{aligned}$$

Thus, the relative error due to round-off in computing $\text{fl}(x(y + z))$ is about 10^{1-t} in the worst case. ■

3.4 Addition of n Floating Point Numbers

Consider adding n floating point numbers x_1, x_2, \dots, x_n with rounding. Define $s_2 = \text{fl}(x_1 + x_2)$. This gives

$$s_2 = \text{fl}(x_1 + x_2) = (x_1 + x_2)(1 + \delta_2), \quad (3.3) \quad \text{eq3.4.1}$$

where $|\delta_2| \leq \mu$. That is, $s_2 - (x_1 + x_2) = \delta_2(x_1 + x_2)$. Define s_3, s_4, \dots, s_n recursively by

$$s_{i+1} = \text{fl}(s_i + x_{i+1}), \quad i = 2, 3, \dots, n-1. \quad (3.4) \quad \text{eq3.4.2}$$

Then $s_3 = \text{fl}(s_2 + x_3) = (s_2 + x_3)(1 + \delta_3) = x_1(1 + \delta_2)(1 + \delta_3) + x_2(1 + \delta_2)(1 + \delta_3) + x_3(1 + \delta_3)$. Then

$$\begin{aligned} s_3 - (x_1 + x_2 + x_3) &= (x_1 + x_2)\delta_2 + (x_1 + x_2)(1 + \delta_2)\delta_3 + x_3\delta_3 \\ &\approx (x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3 \end{aligned} \quad (3.5) \quad \text{eq3.4.3}$$

(neglecting the term $\delta_2\delta_3$ which is small, and so on). Thus, by induction we can show that

$$\begin{aligned} s_n - (x_1 + x_2 + \dots + x_n) &\approx (x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3 \\ &\quad + \dots + (x_1 + x_2 + \dots + x_n)\delta_n \end{aligned} \quad (3.6) \quad \text{eq3.4.4}$$

(again neglecting the terms $\delta_i\delta_j$, which are small).

The Equation (3.6) can be written as

$$s_n - (x_1 + x_2 + \cdots + x_n) \approx \begin{aligned} &x_1(\delta_2 + \delta_3 + \cdots + \delta_n) \\ &+ x_2(\delta_2 + \cdots + \delta_n) + x_3(\delta_3 + \cdots + \delta_n) \\ &+ \cdots + x_n\delta_n \end{aligned} \quad (3.7) \quad \boxed{\text{eq3.4.5}}$$

where each $|\delta_i| \leq \frac{1}{2}\beta^{1-t} = \mu$. Defining $\delta_1 = 0$, we can write:

T2.4.1 **Theorem 3.12 (Rounding Error in Floating Point Addition).** *Let x_1, x_2, \dots, x_n be n floating point numbers. Then*

$$\begin{aligned} fl(x_1 + x_2 + \cdots + x_n) - (x_1 + x_2 + \cdots + x_n) \\ \approx x_1(\delta_1 + \delta_2 + \cdots + \delta_n) + x_2(\delta_2 + \cdots + \delta_n) + \cdots + x_n\delta_n, \end{aligned} \quad (3.8) \quad \boxed{\text{eq3.4.6}}$$

where each $|\delta_i| \leq \mu, i = 1, 2, \dots, n$.

Remark: From the above formula we see that we should expect smaller error in general when adding n floating point numbers in increasing order of magnitude:

$$|x_1| \leq |x_2| \leq |x_3| \leq \cdots \leq |x_n|.$$

If the numbers are arranged in increasing order of magnitude, then the larger errors will be associated with the smaller numbers (**Exercise 3.6**).

T3.13 **Theorem 3.13.** *Define the numbers n_{i1} by*

$$\begin{aligned} 1 + \eta_1 &= (1 + \delta_1)(1 + \delta_2) \cdots (1 + \delta_n) \\ 1 + \eta_2 &= (1 + \delta_2)(1 + \delta_3) \cdots (1 + \delta_n) \\ 1 + \eta_3 &= (1 + \delta_3) \cdots (1 + \delta_n) \\ &\vdots \\ 1 + \eta_{n-1} &= (1 + \delta_{n-1})(1 + \delta_n) \\ 1 + \eta_n &= (1 + \delta_n) \end{aligned}$$

Also define $\mu' = \frac{\mu}{0.9}$ and assume that $n\mu \leq 0.1$, then

$$\begin{aligned} fl(x_1 + x_2 + \cdots + x_n) = \\ x_1(1 + \eta_1) + x_2(1 + \eta_2) + \cdots + x_{n-1}(1 + \eta_{n-1}) + x_n(1 + \eta_n), \end{aligned} \quad (3.9) \quad \boxed{\text{eq3.4.7}}$$

where $|\eta_1| \leq (n-1)\mu'$ and $|\eta_i| \leq (n-i+1)\mu', i = 2, \dots, n$.

Proof. See Stewart (1998), pp. 130-132. \square

3.5 Multiplication of n Floating Point Numbers

Proceeding as in the case of addition of n floating point numbers in the last section, it can be shown that

T2.5.1 **Theorem 3.14.**

$$\text{fl}(x_1 \times x_2 \times \cdots \times x_n) \approx (1 + \epsilon) \prod_{i=1}^n x_i$$

where $\epsilon = |(1 + \delta_2)(1 + \delta_3) \cdots (1 + \delta_n) - 1|$ and $|\delta_i| \leq \mu$, $i = 1, 2, \dots, n$.

A bound for ϵ : Assuming that $(n - 1)\mu < .01$, it can be shown that

$$\epsilon < 1.06(n - 1)\mu \quad (3.10) \quad \boxed{\text{eq2.5.1}}$$

(This assumption is quite realistic; on most machines this assumption will hold for fairly large values of n).

Thus,

$$\epsilon \leq (1 + \mu)^{n-1} - 1 < (n - 1)\mu \left[1 + \frac{0.05}{1 - .05} \right] < 1.06(n - 1)\mu. \quad (3.11) \quad \boxed{2.5.1}$$

Thus, combining Theorem [T2.5.1](#) and [eq2.5.1](#), we can write

T2.5.2 **Theorem 3.15 (Rounding Error in Floating Point Multiplication).** *The relative error in computing the product of n floating point numbers is at most $1.06(n - 1)\mu$, assuming that $(n - 1)\mu < .01$.*

3.6 Inner Product Computation

A frequently arising computational task in numerical linear algebra is the computation of the **inner product** of two n -vectors x and y :

$$x^T y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \quad (3.12) \quad \boxed{2.6.1}$$

where x_i and y_i , $i = 1, \dots, n$, are the components of x and y .

Define

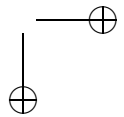
$$S_1 = \text{fl}(x_1 y_1), \quad (3.13) \quad \boxed{2.6.2}$$

$$S_2 = \text{fl}(S_1 + \text{fl}(x_2 y_2)), \quad (3.14) \quad \boxed{2.6.3}$$

\vdots

$$S_k = \text{fl}(S_{k-1} + \text{fl}(x_k y_k)), \quad (3.15) \quad \boxed{2.6.4}$$

$$k = 3, 4, \dots, n.$$



We then have, using Theorem [T2.5.1](#) [B.14](#),

$$S_1 = x_1 y_1 (1 + \delta_1), \quad (3.16) \quad \boxed{2.6.5}$$

$$S_2 = [S_1 + x_2 y_2 (1 + \delta_2)] (1 + \eta_2) \quad (3.17) \quad \boxed{2.6.6}$$

$$\vdots$$

$$S_n = [S_{n-1} + x_n y_n (1 + \delta_n)] (1 + \eta_n), \quad (3.18) \quad \boxed{2.6.7}$$

where each $|\delta_i| \leq \mu$, and $|\eta_i| \leq \mu$. Substituting the values of S_1 through S_{n-1} in S_n and making some rearrangements, we can write

$$S_n = \sum_{i=1}^n x_i y_i (1 + \epsilon_i) \quad (3.19) \quad \boxed{2.6.8}$$

where

$$\begin{aligned} 1 + \epsilon_i &= (1 + \delta_i)(1 + \eta_i)(1 + \eta_{i+1}) \cdots (1 + \eta_n) \\ &\approx 1 + \delta_i + \eta_i + \eta_{i+1} + \cdots + \eta_n \quad (\eta_1 = 0) \end{aligned} \quad (3.20) \quad \boxed{2.6.9}$$

(ignoring the products $\delta_i \eta_j$ and $\eta_j \eta_k$, which are small).

For example, when $n = 2$, it is easy to check that

$$S_2 = x_1 y_1 (1 + \epsilon_1) + x_2 y_2 (1 + \epsilon_2), \quad (3.21) \quad \boxed{2.6.10}$$

where $1 + \epsilon_1 \approx 1 + \delta_1 + \eta_2$, $1 + \epsilon_2 \approx 1 + \delta_2 + \eta_2$ (neglecting the products of $\delta_1 \eta_2$ and $\delta_2 \eta_2$, which are small).

From [\(3.19\)](#), and [\(3.20\)](#), we can write:

[3.6.1](#) Theorem 3.16.

$$\text{fl}(x_1 y_1 + x_2 y_2 + \cdots + x_n y_n) = x_1 y_1 (1 + \epsilon_1) + x_2 y_2 (1 + \epsilon_2) + \cdots + x_n y_n (1 + \epsilon_n)$$

where ϵ_i are given by [\(3.20\)](#).

From Theorem [B.6.1](#) [B.16](#) we have

$$|\text{fl}(x^T y) - x^T y| \leq \sum_{i=1}^n |x_i y_i| |\epsilon_i| \quad (3.22) \quad \boxed{3.6.12}$$

A bound for $|\epsilon_i|$ in terms of μ . Under the assumption that $n\mu < 0.01$, a bound for ϵ_i in terms of μ can be established (see Higham (1996), pp. 74-75).

Using this bound, we can write:

[T2.6.1](#) Theorem 3.17 (Rounding Error in Inner Product Computation).

$$|\text{fl}(x^T y) - x^T y| \leq n1.01\mu |x|^T |y|,$$

where $|x|$ stands for the vector with components $|x_i|$.

Remarks:

- (i) Note that high relative accuracy can not be guaranteed if $|x^T y| \ll |x^T| |y|$.
- (ii) Bounds given in Theorems [3.6.1](#) and [3.17](#) can be reduced by a factor n by using extended precision or some particular implementations (see Higham (1996), pp. 69- 71 for details).

3.7 Error Bounds for Floating Point Matrix Computations

T2.7.1 **Theorem 3.18.** Let $|M| = (|m_{ij}|)$. Let A and B be two floating point matrices and c a floating point number. Then

1. $\text{fl}(cA) = cA + E$, $|E| \leq \mu|cA|$
2. $\text{fl}(A + B) = (A + B) + E$, $|E| \leq \mu|A + B|$

If A and B are two matrices compatible for matrix-multiplication, then

3. $\text{fl}(AB) = AB + E$, $|E| \leq n\mu|A| |B| + O(\mu^2)$.

Proof. See Wilkinson (1965), p. 115. \square

Meaning of $O(\mu^2)$

Remark: In the above expression, the notation $O(\mu^2)$ stands for a complicated expression that is bounded by $c\mu^2$, where c is a constant, depending upon the problem. The expression $O(\mu^2)$ will be used frequently in this book.

Remark: The last result shows that the matrix multiplication in floating point arithmetic can be very inaccurate, since $|A| |B|$ may be much larger than $|AB|$ itself [**Exercise 3.9**].

Error Bounds in Terms of Norms

Traditionally, for matrix computations the bounds for error matrices are given in terms of the norms of the matrices, rather than in terms of absolute values of the matrices as given above. Here we rewrite the bound for error matrices for matrix multiplications using norms, for easy reference later in the book. *We must note, however, that entry-wise error bounds are more meaningful than norm-wise errors* (see remarks in **Section 4.3**).

In terms of a norm, we can write

$$\text{fl}(AB) = AB + E,$$

where

$$\|E\|_p \leq \eta\mu\|A\|_p\|B\|_p + O(\mu^2), p = 1, 2, F.$$

In particular, in terms of $\|\cdot\|_1$ norm, we have

T2.7.2 **Theorem 3.19.** $\|\text{fl}(AB) - AB\|_1 \leq n\mu\|A\|_1\|B\|_1 + O(\mu^2)$.

Two Important Special Cases

A. Matrix-vector multiplication If b is a vector, then from above we have

$$\|fl(Ab) - Ab\|_1 \leq n\mu \|A\|_1 \|b\|_1$$

B. Matrix multiplication by an orthogonal matrix Recall that a real matrix O is called an *orthogonal matrix* if $O^T O = O O^T = I$.

C2.7.1 **Corollary 3.20.** Let $A \in R^{n \times n}$ and $Q \in R^{n \times n}$ orthogonal. Then

$$\|fl(QA) - QA\|_2 \leq n\mu \|A\|_2$$

Implication of the above result

The result of Corollary ^{C2.7.1}3.20 says that, although matrix multiplication can be inaccurate in general, if one of the matrices is orthogonal then the floating point matrix multiplication gives only a small and acceptable error. As we will see in later chapters, this result forms the basis of many numerically viable algorithms discussed in this book.

3.8 Round-off Errors Due to Cancellation and Recursive Computations

Intuitively, it is clear that if a large number of floating point computations is done, then the accumulated error can be quite large. **However, round-off error can be disastrous even at a single step of computation.** For example, consider the subtraction of two numbers:

$$x = 0.54617 \text{ and } y = 0.54601$$

The exact value is

$$d = x - y = 0.00016.$$

Suppose now we use four digit arithmetic with rounding. Then we have

$$\hat{x} = 0.5462 \text{ (Correct to four significant digits)}$$

$$\hat{y} = 0.5460 \text{ (Correct to four significant digits)}$$

$$\hat{d} = \hat{x} - \hat{y} = 0.0002.$$

How good is the approximation of \hat{d} to d ? The relative error is

$$\frac{|d - \hat{d}|}{|d|} = 0.25 \text{ (quite large!)}$$

What happened above is the following. In four digit arithmetic, the numbers 0.5462 and 0.5460 are of almost the same size. So, when the first one was

subtracted from the second, the most significant digits got canceled and the very least significant digit was left in the answer. This phenomenon, known as **catastrophic cancellation**, occurs when two numbers of approximately the same size are subtracted.

Remark: It is to be noted that in many cases, subtraction is performed rather accurately. It is not a cause of the error - *rather it reveals the errors made in earlier computations or even those in the data associated with the subtraction*. Indeed, it highlights the earlier errors.

Avoiding Cancellation:

Fortunately, in many cases catastrophic cancellation can be avoided. For example, consider the case of solving the quadratic equation:

$$ax^2 + bx + c = 0, \quad a \neq 0.$$

The usual way the two roots x_1 and x_2 are computed is:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

It is clear from above that if a , b , and c are numbers such that $-b$ is about the same size as $\sqrt{b^2 - 4ac}$ (with respect to the arithmetic used), then a catastrophic cancellation will occur in computing x_2 and as a result, the computed value of x_2 can be completely erroneous.

E2.8.1 **Example 3.21 (Cancellation in Root-Finding of the Quadratic)** Consider solving $ax^2 + bx + c = 0$, with $a = 1$, $b = -10^5$, $c = 1$ (Forsythe, Malcolm and Moler (1977), pp. 20-22). Then using $\beta = 10$, $t = 8$, $L = -U = -50$, we see that

$$x_1 = \frac{10^5 + \sqrt{10^{10} - 4}}{2} = 10^5 \text{ (true answer)}$$

$$x_2 = \frac{10^5 - 10^5}{2} = 0 \text{ (completely wrong).}$$

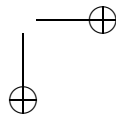
The true $x_2 = 0.000010000000001$ (correctly rounded to 11 significant digits). The catastrophic cancellation took place in computing x_2 , since $-b$ and $(\sqrt{b^2 - 4ac})$ are of the same order. Note that in 8-digit arithmetic, $\sqrt{10^{10} - 4} = 10^5$. ■

How Cancellation Can be Avoided in Finding Roots of the Quadratic?

Cancellation can be avoided if an equivalent pair of formulas is used:

$$x_1 = -\frac{b + \text{sign}(b)\sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{c}{ax_1}$$



where $\text{sign}(b)$ is the sign of b . Using these formulas, we easily see that:

$$\begin{aligned}x_1 &= 100000.00 \\x_2 &= \frac{1.0000000}{100000.00} = 0.000010000\end{aligned}$$

Remark: Cancellation may still take place during the subtraction $b^2 - 4ac$ and significant digits will be lost if $b^2 \approx 4ac$. In that case, it is advisable to use double precision in computing $b^2 - 4ac$.

E2.8.2 **Example 3.22** For yet another example consider the problem of evaluating

$$f(x) = e^x - x - 1 \quad \text{at } x = .01.$$

Using five digit arithmetic, the correct answer is 0.000050167. If $f(x)$ is evaluated directly from the expression, we have

$$\begin{aligned}f(.01) &= 1.0101 - (0.01) - 1 = 0.0001 \\ \text{Relative Error} &= \frac{0.0001 - 0.000050167}{0.000050167} \\ &= 0.99 \times 10^0,\end{aligned}$$

indicating that we cannot trust even the first significant digit.

Fortunately, cancellation can again be avoided using the convergent series for e^x :

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots$$

In this case we have

$$\begin{aligned}e^x - x - 1 &= \left(1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots\right) - x - 1 \\ &= \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} \dots\end{aligned}$$

For $x = 0.01$, this formula gives

$$\begin{aligned}&\frac{(.01)^2}{2} + \frac{(.01)^3}{3!} + \frac{(.01)^4}{4!} + \dots \\ &= .00005 + .000000166666 + .0000000004166 + \dots \\ &= .000050167 \text{ (correct up to five significant figures)}\end{aligned}$$

■

Remark: Note that if x were negative, then use of the convergent series for e^x would not have helped. For example, to compute e^x for a negative value of x , cancellation can be avoided by using

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots}$$

Recursive Computations

Recursive computations are those which are performed recursively so that the computation of one step depends upon the results of the previous steps. In such cases, even if the error made in the first step is negligible, due to the accumulation and magnification of error at every step, the final error can be quite large, giving a completely erroneous answer.

Certain recursions propagate errors in very unhealthy fashions. Consider the following example involving recursive computations, again from Forsythe, Malcolm, and Moler (1977), pp. 16-17.

E2.8.3 **Example 3.23** Suppose we need to compute the integral

$$E_n = \int_0^1 x^n e^{x-1} dx$$

for different values of n . Integrating by parts gives

$$E_n = \int_0^1 x^n e^{x-1} dx = (x^n e^{x-1})_0^1 - \int_0^1 n x^{n-1} e^{x-1} dx$$

or

$$E_n = 1 - nE_{n-1}, \quad n = 2, 3, \dots$$

Thus, if E_1 is known, then for different values of n , E_n can be computed, using the above recursive formula.

Indeed, with $\beta = 10$ and $t = 6$, and starting with $E_1 = 0.367879$ as a six-digit approximation to $E_1 = 1/e$, we have from above:

$$\begin{aligned} E_1 &= 0.367879 \\ E_2 &= 0.264242 \\ E_3 &= 0.207274 \\ E_4 &= 0.170904 \\ &\vdots \\ E_9 &= -0.068480 \text{ (wrong)} \end{aligned}$$

Although the integrand is positive throughout the interval $[0, 1]$, the computed value of E_9 is negative. This phenomenon can be explained as follows.

The error in computing E_2 was -2 times the error in computing E_1 , the error in computing E_3 was -3 times the error in E_2 (therefore, the error at this step was exactly six times the error in E_1). Thus, the error in computing E_9 was $(-2)(-3)(-4)\cdots(-9) = 9!$ times the error in E_1 . The error in E_1 was due to the rounding of $1/e$ using six significant digits, which is about 4.412×10^{-7} . However, this small error multiplied by $9!$ gave $9! \times 4.412 \times 10^{-7} = 0.1601$, which is quite large. ■

Rearranging the Recurrence

Again, for this example, it turned out that we could get a much better result by simply rearranging the recursion so that the error at every step, instead of being magnified, is reduced. Indeed, if we rewrite the recursion as

$$E_{n-1} = \frac{1 - E_n}{n}, n = \dots, 3, 2$$

then the error at each step will be reduced by a factor of $1/n$. Thus, starting with a large value of n (say, $n = 20$) and working backward, we will see that E_9 will be accurate to full six-digit precision.

To obtain a starting value, we note that

$$E_n = \int_0^1 x^n e^{n-1} dx \leq \int_0^1 x^n dx = \frac{1}{n+1}.$$

With $n = 20$, $E_{20} \leq \frac{1}{21}$. Let's take $E_{20} = 0$. Then, starting with $E_{20} = 0$, it can be shown (Forsythe, Malcolm, and Moler (1977), p.17) that $E_9 = 0.0916123$, which is correct to full six-digit precision.

The reason for obtaining this accuracy was that the error in E_{20} was at most $\frac{1}{21}$; this error was multiplied by $\frac{1}{20}$ in computing E_{19} , giving an error of at most $\frac{1}{20} \cdot \frac{1}{21} = 0.0024$ in the computation of E_{19} , and so on.

3.9 Review and Summary

The concepts of floating point numbers and rounding errors have been introduced and discussed in this chapter.

1. **Floating Point Numbers:** A normalized floating point number has the form

$$x = \pm r\beta^e,$$

where e is called **exponent**, r is the **significant**, and β is the **base** of the number system. The floating point number system is characterized by four parameters: β — the **base**; t — the **precision**; L, U — the **lower** and **upper limits of the exponent**.

2. **Errors:** The error(s) in a computation is measured either by *absolute error* or *relative error*.

The relative errors make more sense than absolute errors.

The relative error gives an indication of the number of significant digits in an approximate answer.

The relative error in representing a real number x by its floating point representation $\hat{f}(x)$ is bounded by a number μ , called the *machine precision* (**Theorem 3.6**).

3. Laws of Floating Point Arithmetic:

$$\text{fl}(x \odot y) = (x \odot y)(1 + \delta)$$

where \odot indicates any of the four basic arithmetic operations $+$, $-$, \times , or \div , and $|\delta| \leq \mu$.

4. **Addition, Multiplication, and Inner Product Computations.** The results of addition and multiplication of n floating point numbers and inner product computation are given in **Theorems 3.12**, **3.15** and **3.17** respectively.

- While adding n floating point numbers, it is advisable that they are added in increasing order of magnitude.

5. **Floating Point Matrix Multiplications.** The entry-wise and normalize error bounds for matrix multiplication of two floating point matrices are given in **Theorems 3.18** and **3.19**, respectively.

- Matrix multiplication in floating point arithmetic can be very inaccurate, unless one of the matrices is orthogonal (or unitary, if complex).
- The high accuracy in a matrix product computation involving an orthogonal matrix (**Corollary 3.20**) makes the use of orthogonal matrices in matrix computations so attractive.

6. **Round-off Errors Due to Cancellation and Recursive Computation.**

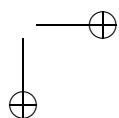
Subtractive cancellation or catastrophic cancellation (as it is commonly called) is a phenomenon in which a number of significant digits in a computation gets cancelled due to subtraction of two almost equal numbers. In most cases, however, subtractions are done exactly. *Catastrophic cancellation signals some errors made in previous steps.* In fact, it brings this error in prominence. Recursive computations are those which are performed recursively so that the computation of one step depends upon the results of the previous steps.

These have been discussed in some detail in **Section 3.8**.

Examples have been given to show how these errors come up in many basic computations. *An encouraging message here is that in several instances, computations can be reorganized so that cancellation can be avoided, and the error in recursive computations can be diminished at each step of computation.*

3.10 Suggestions for Further Reading

For details of IEEE standard, see the monograph “An American National Standard: IEEE Standard for Binary Floating-Point Arithmetic,” IEEE publication, 1985, IEEE standard for Radix-independent Floating Point Arithmetic, IEEE, New York, 1987, and “Numerical Computing with IEEE Floating Point Arithmetic” by M. Overton, SIAM, 2001.



For results on error bounds for basic floating point matrix operations, the classic books by James H. Wilkinson (i) *THE ALGEBRAIC EIGENVALUE PROBLEM (AEP)* and (ii) *ROUNDING ERRORS IN ALGEBRAIC PROCESSES* (Prentice-Hall, New Jersey, 1963) are extremely useful and valuable resources. *The most recent authoritative book on error analysis is the one by Higham (1996). Every researcher of numerical analysis must have a copy of this book.*

Discussion on basic floating point operations and rounding errors due to cancellations and recursive computations are given nowadays in many numerical analysis text books.

Exercises on Chapter ^{ch3}3

3.1 (a) Prove the following expression

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \mu = \begin{cases} \frac{1}{2}\beta^{1-t} & \text{for rounding} \\ \beta^{1-t} & \text{for chopping} \end{cases}$$

(b) Show that (a) can be written in the form

$$\text{fl}(x) = x(1 + \delta), \quad |\delta| \leq \mu.$$

3.2 Let x be a floating point number and k be a positive integer, then

$$\text{fl}\left(\frac{x^k}{k!}\right) = \frac{x^k}{k!}(1 + e_k),$$

where

$$|e_k| \leq 2k\mu + O(\mu^2).$$

3.3 Construct examples to show that the distributive law for floating point addition and multiplication does not hold. What can you say about the commutativity and associativity for these operations? Give reasons for your answers.

3.4 Let x_1, x_2, \dots, x_n be the n floating point numbers. Define

$$s_2 = \text{fl}(x_1 + x_2), \quad s_k = \text{fl}(s_{k-1} + x_k), \quad k = 3, \dots, n.$$

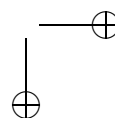
Then from Theorem ^{¶2.4.1}3.12 show that

$$\text{fl}(x_1 + x_2 + \dots + x_n) = x_1(1 + \eta_1) + x_2(1 + \eta_2) + \dots + x_n(1 + \eta_n).$$

3.5 (a) Give a proof of Theorem ^{¶2.5.1}3.14.

(b) Prove Theorem ^{¶2.5.2}3.15 by first establishing the result ^{¶2.5.1}(3.11).

3.6 (a) Construct an example to show that, when adding a list of floating point numbers, in general, the rounding error will be less if the numbers are added in order of increasing magnitude.



(b) Find another example to show that this is not always necessarily true.

3.7 Using Theorem [5.17](#), show that high relative accuracy is obtained in computing $x^T x$.

3.8 Show that

$$(a) \text{fl}(cA) = cA + E, \quad |E| \leq \mu|cA|$$

$$(b) \text{fl}(A + B) = (A + B) + E, \quad |E| \leq \mu(|A| + |B|)$$

$$(c) \text{fl}(AB) = AB + E, \quad |E| \leq n\mu|A| |B| + O(\mu^2)$$

(Consult **Wilkinson (1965)**, p. 115.)

3.9 Construct a simple example to show that the matrix multiplication in floating point arithmetic need not be accurate.

3.10 Prove that if Q is orthogonal then

$$\text{fl}(QA) = Q(A + E), \quad \text{where } \|E\|_2 \leq n^2\mu\|A\|_2 + O(\mu^2).$$

3.11 Let y_1, \dots, y_n be n column vectors defined recursively:

$$y_{i+1} = Ay_i, \quad i = 1, 2, \dots, n-1.$$

Let $\hat{y}_i = \text{fl}(y_i)$. Find a bound for the relative error in computing each y_i , $i = 1, \dots, n$.

3.12 Let $\beta = 10$, $t = 4$. Compute

$$\text{fl}(A^T A),$$

where

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}.$$

Repeat your computation with $t = 9$. Compare the results.

3.13 Show how to arrange computation in each of the following, so that the loss of significant digits can be avoided. Do one numerical example in each case to support your answer.

$$(a) e^x - x - 1, \quad \text{for negative values of } x.$$

$$(b) \sqrt{x^4 + 1} - x^2, \quad \text{for large values of } x.$$

$$(c) \frac{1}{x} - \frac{1}{x+1}, \quad \text{for large values of } x.$$

$$(d) x - \sin x, \quad \text{for values of } x \text{ near zero.}$$

$$(e) 1 - \cos x, \quad \text{for values of } x \text{ near zero.}$$

$$(f) \frac{e^x - 1}{x} \text{ for } |x| \ll 1.$$

(g) $\frac{(1 - \cos x)}{x^2}$ for small x

3.14 What are the relative and absolute errors in approximating

(a) π by $\frac{22}{7}$?

(b) $\frac{1}{3}$ by .333?

(c) $\frac{1}{6}$ by .166?

How many significant digits are there in each computation?

3.15 Let $\beta = 10$, $t = 4$. Consider computing

$$a = \left(\frac{1}{6} - .1666\right) / .1666.$$

How many correct digits of the exact answer will you get?

3.16 Consider evaluating

$$e = \sqrt{a^2 + b^2}.$$

How can the computation be organized so that overflow in computing $a^2 + b^2$ for large values of a or b can be avoided?

3.17 What answers will you get if you compute the following numbers on your calculator or computer?

(a) $\sqrt{10^8 - 1}$

(b) $\sqrt{10^{-20} - 1}$

(c) $10^{16} - 50$

Compute the absolute and relative errors in each case.

3.18 What problem do you foresee in solving the quadratic equations

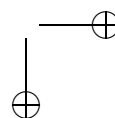
(a) $x^2 - 10^6x + 1 = 0$,

(b) $10^{-10}x^2 - 10^{10}x + 10^{10} = 0$

using the well-known formula:

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

What remedy do you suggest? Now solve the equations using your suggested remedy, with $t = 4$.



3.19 Show that the integral

$$y_i = \int_0^1 \frac{x^i}{x+5} dx$$

can be computed by using the recursion formula:

$$y_i = \frac{1}{i} - 5y_{i-1}.$$

Compute y_1, y_2, \dots, y_{10} using this formula, taking

$$y_0 = \ln(x+5)|_{x=0}^1 = \ln 6 - \ln 5 = \ln(1.2).$$

What abnormalities do you observe in this computations? Explain what happened.

Now rearrange the recursion so that the values of y_i can be computed more accurately.

Chapter 4

STABILITY OF ALGORITHMS AND CONDITIONING OF PROBLEMS

ch4

Background Material Needed

- Vector and matrix norms (Section 2.5).

4.1 Introduction

In this chapter, we introduce the basic concepts of *algorithm*, two of its important properties, *efficiency* and *stability* and an important property of the problem, called, *conditioning*.

We first define the concept of algorithm and state some basic algorithms for matrix computations in this section itself.

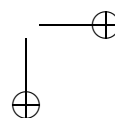
D3-1

Definition 4.1. *An algorithm is an ordered set of operations, logical and arithmetic, which when applied to a computational problem defined by a given set of data, called the **input data**, produces a solution to the problem. A solution is comprised of a set of data called the **output data**.*

In this book, for the sake of convenience and simplicity, we will very often describe algorithms by means of **pseudocodes** which can be translated into computer codes easily. Describing algorithms by pseudocodes has been made popular by Stewart (1973). Here are some examples.

4.1.1 Computing the Norm of a Vector

Given $x = (x_1, \dots, x_n)^T$, compute $\|x\|_2$.



A3.1.1

ALGORITHM 4.1. Computing the Norm of a Vector.**Input:** n, x_1, \dots, x_n .**Output:** $s = \|x\|_2$.**Step 1:** Compute $r = \max(|x_1|, \dots, |x_n|)$.**Step 2:** Compute $y_i = x_i/r$, $i = 1, \dots, n$ **Step 3:** Compute $s = \|x\|_2 = r\sqrt{(y_1^2 + \dots + y_n^2)}$.**Pseudocodes** $r = \max(|x_1|, \dots, |x_n|)$ $s = 0$ For $i = 1$ to n do $y_i = x_i/r$; $s = s + y_i^2$ $s = r(s)^{1/2}$

End

An Algorithmic Note

In order to avoid overflow, each entry of x was normalized before using the norm formula

$$\|x\|_2 = \sqrt{x_1^2 + \dots + x_n^2}.$$

4.1.2 Computing the Inner Product of Two Vectors

Given x and y two n -vectors, $x = (x_1, x_2, \dots, x_n)^T$ and $y = (y_1, y_2, \dots, y_n)^T$, compute the inner product $x^T y = x_1 y_1 + x_2 y_2 + \dots + x_n y_n$.

A3.1.2

ALGORITHM 4.2. Computing the Inner Product of Two Vectors.**Input:** A positive integer n and two sets of numbers $\{x_i\}_{i=1}^n$ and $\{y_i\}_{i=1}^n$.**Output:** Sum = Inner product $x^T y$

Step 1: Compute the partial products: $s_i = x_i y_i$, $i = 1, \dots, n$.

Step 2: Add the partial products: $\text{Sum} = \sum_{i=1}^n s_i$.

Pseudocodes

```
Sum = 0
For  $i = 1, \dots, n$  do
    Sum = Sum +  $x_i y_i$ 
End
```

4.1.3 Solution of an Upper Triangular System

Consider the system

$$Ty = b$$

where $T = (t_{ij})$ is a nonsingular upper triangular matrix and $y = (y_1, y_2, \dots, y_n)^T$. Specifically,

$$\begin{aligned} t_{11}y_1 + t_{12}y_2 + \cdots + t_{1n}y_n &= b_1 \\ t_{22}y_2 + \cdots + t_{2n}y_n &= b_2 \\ t_{33}y_3 + \cdots + t_{3n}y_n &= b_3 \\ &\vdots \\ t_{n-1,n-1}y_{n-1} + t_{n-1,n}y_n &= b_{n-1} \\ t_{nn}y_n &= b_n \end{aligned}$$

where each $t_{ii} \neq 0$ for $i = 1, 2, \dots, n$.

The last equation is solved first to obtain y_n , then this value is inserted in the last but one equation to obtain y_{n-1} , and so on. This process is known as **back substitution**. The algorithm can easily be written down.

ALGORITHM 4.3. Back Substitution Method for Upper Triangular System.

A3.1.3

Input: An $n \times n$ upper triangular matrix $T = (t_{ij})$ and an n -vector b .

Output: The vector $y = (y_1, \dots, y_n)^T$, such that $Ty = b$.

Step 1: Compute $y_n = \frac{b_n}{t_{nn}}$

Step 2: Compute y_{n-1} through y_1 successively:

For $i = n - 1, \dots, 2, 1$ do

$$y_i = \frac{1}{t_{ii}} \left(b_i - \sum_{j=i+1}^n t_{ij} y_j \right), \quad i = n - 1, \dots, 2, 1.$$

End

4.1.4 Solving a Lower Triangular System

A lower triangular system can be solved in an analogous manner. The process is known as the **forward elimination method**. Let $L = (l_{ij})$, and $b = (b_1, b_2, \dots, b_n)^T$.

ALGORITHM 4.4. The Forward Elimination Method for Lower Triangular System.

A4.1.4

Input: A $n \times n$ lower triangular matrix $L = (l_{ij})$ and an n -vector b .

Output: An n -vector $y = (y_1, y_2, \dots, y_n)^T$ such that $Ly = b$.

Step 1. Compute $y_1 = \frac{b_1}{l_{11}}$

Step 2. For $i = 2, 3, \dots, n$ do

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

End

MATCOM Notes: Algorithms [A3.1.3](#) and [A4.1.4](#) have been implemented in MATCOM programs, BACKSUB and FORELIM, respectively.

4.2 Efficiency of an Algorithm

Two most desirable properties of an algorithm are: **Efficiency and Stability**.

The efficiency of an algorithm is measured by the amount of computer time consumed in its implementation. A *theoretical and crude measure of efficiency* is the number of floating point operations (**flops**) needed to implement the algorithm.

D4-2-1

Definition 4.2. A **flop** is a basic floating point operation: $+$, $-$, $*$, or $/$.

Flop-Count for Algorithm [A3.1.3](#) and Algorithm [A4.1.4](#): Each of these algorithms requires n^2 flops.

The Big O Notation

An algorithm will be called an $O(n^p)$ algorithm if the dominant term in the operations count of the algorithm is a multiple of n^p . Thus, the solution of a triangular system is an $O(n^2)$ algorithm.

Notation for Overwriting and Interchange

We will use the notation

$$a \equiv b$$

to denote that "**b overwrites a**". Similarly, if two computed quantities a and b are interchanged, they will be written symbolically

$$a \leftrightarrow b.$$

4.3 Definition and Concept of Stability

The examples on catastrophic cancellations and recursive computations in the last chapter (Section 3.8) had one thing in common: **the inaccuracy of the computed result in each case was entirely due to the algorithm used**, because as soon as the algorithm was changed or rearranged and applied to the problem with the same data, the computed result became satisfactory. Thus, we are talking about two different types of algorithms for a given problem. The algorithms of the first type are examples of *unstable algorithms*, while the ones of the second type—giving satisfactory results—are *stable algorithms*.

There are two types of stable algorithms: *backward stable* and *forward stable*.

In this context, we first define **forward error** and **backward error**. Let $\hat{f}(x)$ be the computed approximate value of $f(x)$ with an input data x . Then

- **Forward Error** = $|f(x) - \hat{f}(x)|$.

On the other hand, backward errors relate the errors to the data of the problem rather than to the problem's solution.

- **Backward Error.** Here we ask for what value of the input data y , $f(y) = \hat{f}(x)$? *Backward Error* = $|y - x|$.

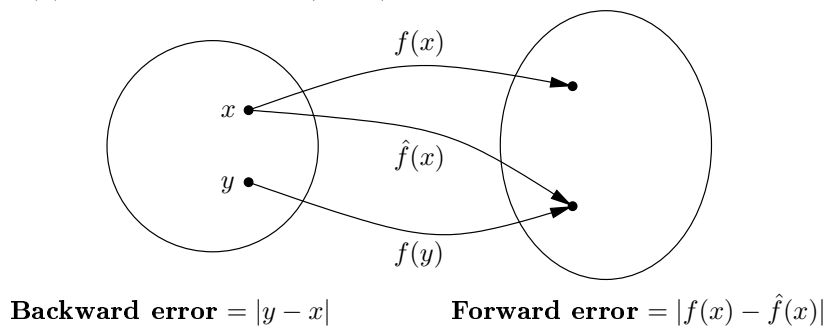


Figure 4.1. Backward Error vs. Forward Error

Example 4.3 (Backward vs. Forward Errors.)

Suppose we would like to estimate $f(x) = e^x$ at $x = 1$. Consider the truncated series

$$\hat{f}(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!}$$

Then

$$f(1) = 2.7183, \hat{f}(1) = 2.6667$$

Forward Error = $2.7183 - 2.6667 = 0.0516$.

To find backward error we must find y such that $f(y) = \hat{f}(1)$. For e^x , $y = \log(\hat{f}(1))$. At $x = 1$ we have $y = \log(\hat{f}(1)) = 0.9808$.

Backward Error: $|y - x| = 0.0192$.

Verify: $e^y = e^{0.9809} = 2.6667 = \hat{f}(1) = 2.6667$. ■

Example 4.4 (Forward Error Bound for the Inner Product).

Let x and y be two n -vectors. Then the error bound in Theorem 3.9: $|fl(x^T y) - x^T y| \leq n\mu|x^T y|$ is the *forward error bound* for inner product of x and y .

This result shows that high accuracy in inner product computation can not be guaranteed unless $|x^T y| \ll |x^T y|$ on the other hand, if $y = x$, then the result will be accurate. ■

Example 4.5 (Forward Error Bound for Matrix Multiplication).

The error bound in Theorem 3.19: $fl(AB) = AB + E$, where $\|E\|_1 \leq n\mu\|A\|_1\|B\| + O(\mu^2)$ gives the *forward error bound* for matrix multiplication. ■

Backward and Forward Stability

D3-2

Definition 4.6. An algorithm is called **backward stable** if for any x , it produces a value $\hat{f}(x)$ with a small backward error. In other words, *an algorithm is backward-stable if it produces an exact solution to a nearby problem. That is, an algorithm is backward stable if $\hat{f}(x) = f(y)$, for some y close to x .*

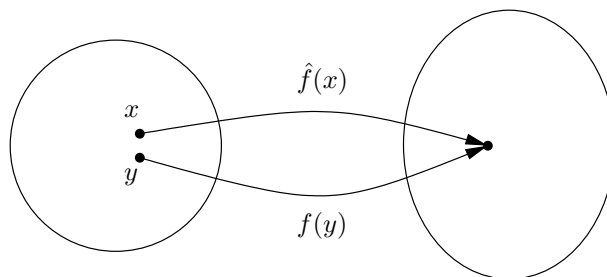


Figure 4.1 Backward Stable Algorithm.

Remark: The forward stability of the algorithm is defined in a similar way. In this book, by “stability”, we will imply backward stability. Thus, an algorithm will be called stable if it is backward stable.

Note that an algorithm can be forward stable without being backward stable; that is, a small error in $\hat{f}(x)$ may or may not correspond to a small perturbation of the data.

The process of analyzing the backward errors in a numerical computation is called the *backward error analysis*. Backward error analysis, introduced in the literature by J. H. Wilkinson,^{*} is nowadays widely used in matrix computations and using this analysis, the stability (or instability) of many algorithms in numerical linear algebra has been established in recent years.

Example 4.7 Backward Stability of Arithmetic Operations of Two Floating Point Numbers

Consider computing the sum of two floating point numbers x and y . We have seen before (Theorem 3.9) that

$$\begin{aligned} \text{fl}(x + y) &= (x + y)(1 + \delta) \\ &= x(1 + \delta) + y(1 + \delta) = x' + y' \end{aligned}$$

Thus, the computed sum of two floating point numbers x and y is the exact sum of another two floating point numbers x' and y' . Since $|\delta| \leq \mu$, both x' and y' are close to x and y , respectively. Thus we conclude that *the operation of adding two floating point numbers is backward stable*. Similar statements, of course, hold for other arithmetic operations of two floating point numbers. ■

Example 4.8 Backward Stability of Addition of n Floating-Point Numbers.

Recall from Chapter 3 (Theorem 3.13) that $\text{fl}(x_1 + x_2 + \cdots + x_n) = x_1(1 + \eta_1) + x_2(1 + \eta_2) + \cdots + x_n(1 + \eta_n)$, where each η_i is small. Thus, the *computed sum of n floating point numbers is the exact sum of n perturbed numbers with small perturbations*. ■

Example 4.9 Backward Stability and Instability of the Inner and Outer Products:

The inner product of two vectors x and y is backward stable. Theorem 3.16 shows that the computed inner product is the exact inner product of a perturbed set of data: x_1, x_2, \dots, x_n and $y_1(1 + \epsilon_1), \dots, y_n(1 + \epsilon_n)$, where each perturbation is small.

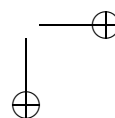
The outer product of the vectors x and y is however, not backward stable [Exercise 4.1(b)]. ■

Examples of Backward Stability and Instability of Linear Systems Solvers

Definition 4.10. *An algorithm for solving $Ax = b$ will be called **backward stable** if the computed solution \hat{x} is such that*

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

^{*}**James H. Wilkinson**, a British mathematician, is well known for his pioneering work on backward error analysis for matrix computations. He was affiliated with the National Physical Laboratory in Britain, and held visiting appointments at Argonne National Laboratory, Stanford University, etc. Wilkinson died an untimely death in 1986. A fellowship in his name has since been established at Argonne National Laboratory. Wilkinson's book THE ALGEBRAIC EIGENVALUE PROBLEM is an extremely important and very useful book for any numerical analyst.



with E and δb small.

How Do We Measure Smallness?

The “smallness” of a matrix or a vector is measured either by looking into its entries or by computing its norm.

Norm-wise vs. Entry-wise Errors

While measuring errors in computations using norms is traditional in matrix computations, component-wise measure of errors is becoming increasingly important. It really does make more sense.

An $n \times n$ matrix A has n^2 entries, but the norm of A is a single number. Thus the smallness or largeness of the norm of an error matrix E does not truly reflect the smallness or largeness of the individual entries of E . For example, if $E = (10, .00001, 1)^T$, then $\|E\| = 10.0499$. Thus the small entry .00001 was not reflected in the norm measure.

E4.11 Example 4.11 A Stable Algorithm — Solution of an Upper Triangular System by Back Substitution

Consider Algorithm 4.3 (the **back substitution** method). It can be shown (see Chapter 13) that the computed solution \hat{x} , obtained by this algorithm satisfies:

$$(T + E)\hat{x} = b,$$

where the entries of the error matrix E are quite small. In fact, if $E = (e_{ij})$ and $T = (t_{ij})$, then

$$|e_{ij}| \leq n\mu|t_{ij}| + O(\mu^2)$$

showing that the error can be even smaller than the error made in rounding the entries of T . Thus, *the back substitution process for solving an upper triangular system is stable.* ■

E4.12 Example 4.12 An Unstable Algorithm — Gaussian Elimination Without Pivoting

Consider solving the 2×2 system using the standard elimination method, called Gaussian elimination: $Ax = b$, where $A = \begin{pmatrix} 10^{-10} & 1 \\ 1 & 2 \end{pmatrix}$, $b = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$. That is,

$$\begin{aligned} 10^{-10}x_1 + x_2 &= 1 \\ x_1 + 2x_2 &= 3. \end{aligned}$$

Eliminating x_2 from the second equation, we obtain

$$\begin{aligned} 10^{-10}x_1 + x_2 &= 1 \\ (2 - 10^{10})x_2 &= 3 - 10^{10} \end{aligned}$$

In computer arithmetic, we will have

$$\begin{aligned} 10^{-10}x_1 + x_2 &= 1 \\ -10^{-10}x_2 &= -10^{10}, \end{aligned}$$

giving $x_2 = 1$, $x_1 = 0$, whereas the exact solution is $x_1 = x_2 = 1$.

Thus, **the above process is clearly unstable**. The readers are asked to verify themselves that the computed solution $\hat{x} = (1, 0)^T$ is the exact solution of the system $(A + E)\hat{x} = b$, where E is large. ■

If an algorithm is stable for a given matrix A , then one would like to see that the algorithm is stable for every matrix A in a given class. Thus, we may give a formal definition of stability as follows:

D3-3 **Definition 4.13.** An algorithm is stable for a class of matrices C if for every matrix A in C , the computed solution by the algorithm is the exact solution of a nearby problem.

Thus, for the linear system problem

$$Ax = b,$$

an algorithm is stable for a class of matrices C if for every $A \in C$ and for each b , it produces a computed solution \hat{x} that satisfies

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

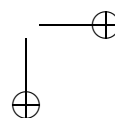
for some E and δb , where $(A + E)$ is close to A and $b + \delta b$ is close to b .

4.4 Conditioning of the Problem and Perturbation Analysis

From the preceding discussion we should not form the opinion that if a stable algorithm is used to solve a problem then the computed solution will be accurate. A property of the problem, called **conditioning**, also contributes to the accuracy or inaccuracy of the computed result.

The conditioning of a problem is a property of the problem itself. It is concerned with how the solution of the problem will change if the input data contains some impurities. This concern arises from the fact that in practical applications very often the data come from some experimental observations where the measurements can be subjected to disturbances (or “noise”) in the data. There are other sources of error also, for example, *round-off errors*, and *discretization errors*. Thus, when a numerical analyst has a problem in hand to solve, he or she must frequently solve the problem not with the original data, but with data that has been perturbed. *The question naturally arises: What effects do these perturbations have on the solution?*

A theoretical study done by numerical analysts to investigate these effects, which is independent of the particular algorithm used to solve the problem, is called



perturbation analysis. This study helps one detect whether a given problem is “bad” or “good” in the sense of whether small perturbations in the data will create a large or small change in the solution.

When the result of a perturbation analysis is combined with that of backward error analysis of a particular algorithm, an error bound in the computed solution by the algorithm can be obtained.

D3-4 **Definition 4.14.** A problem (with respect to a given set of data) is called an *ill-conditioned* or *badly-conditioned* problem if a small relative perturbation in data can cause a large relative error in the computed solution, regardless of the method of solution. Otherwise, it is called *well-conditioned*; that is, a problem is *well-conditioned* if all small perturbations in data produce only small relative errors in the solution.

Let x and y denote the original and the slightly perturbed data, and $f(x)$ and $f(y)$ be the respective solutions. Then

Well-Conditioned Problem: If y is close to x , then $f(y)$ is close to $f(x)$.

Ill-Conditioned Problem: Even if y is close to x , then $f(y)$ can depart from $f(x)$ drastically.

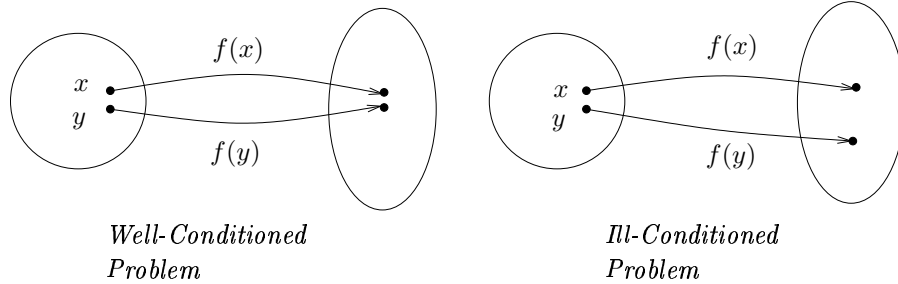


Figure 4.2 Well-Conditioned (left) and Ill-Conditioned (Right) Problems

Numerical analysts attempt to assign a number to each problem, called the *condition number*, to determine if the problem is ill-conditioned or well-conditioned. Formally, the relative condition number or simply the condition number can be defined as follows:

Definition 4.15. The condition number of the problem f with respect to the data x is defined as:

$$\frac{\text{Relative error in the solution}}{\text{Relative perturbation in the data}} = \frac{|f(x) - f(y)|}{|f(x)|} \bigg/ \left| \frac{x - y}{x} \right| \quad (4.1) \quad \boxed{\text{eq4.4.1}}$$

If $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, and x and $y \in \mathbb{R}^n$, then the condition number is formally defined as:

$$\limsup_{\epsilon \rightarrow 0} \left\{ \left(\frac{\|f(y) - f(x)\|}{\|f(x)\|} \right) \bigg/ \left(\frac{\|x - y\|}{\|x\|} \right) \mid \|x - y\| \leq \epsilon \right\} .$$

A problem is *ill-conditioned* if the condition number is $\gg 1$ (\gg stands for much greater than), for example, 10^{10} , 10^{15} , etc.

Example 4.16 Condition Number of a Function

If $f(x)$ is a differentiable function of one variable, then, it is easy to see [Exercise 4.10] that for small perturbations, the condition number of $f(x)$, denoted by $c(x)$, is given by

$$c(x) = \left| \frac{xf'(x)}{f(x)} \right|. \quad (4.2) \quad \boxed{\text{eq4.4.2}}$$

As an example, let $f(x) = e^x$. Then $c(x) = |x|$, which is large for large values of x . That means, *a small relative error in x can produce a large relative error in e^x , so this problem is ill-conditioned, when x is large.*

If f or x is a vector, then the condition number can be defined in the same way using norms instead of the absolute values.

Thus, the condition number of a function of several variables (or a vector) can be defined by replacing $f'(x)$ by its gradient. In this case $c(x) = \frac{\|x \nabla f\|}{\|f(x)\|}$, where ∇f is the gradient.

For example, if $f(x) = x_1 - x_2$, then $\nabla f = (1, -1)$, and the condition number $c(x)$ of f (with respect to the infinite norm) is given by:

$$c(x) = \frac{\|x \nabla f\|_\infty}{\|f(x)\|_\infty} = \frac{2 \max\{|x_1|, |x_2|\}}{|x_1 - x_2|} \quad (4.3) \quad \boxed{\text{eq4.4.3}}$$

which shows that $f(x)$ is **ill-conditioned** if $x_1 \simeq x_2$. ■

Example 4.17 Condition Number of the Matrix-Vector Product

Suppose $A \in \mathbb{R}$ and x is an n -vector. Then it can be shown [Exercise 4.12] that the condition number κ of Ax (with respect to perturbations of x) is given by

$$\kappa = \|A\| \frac{\|x\|}{\|Ax\|}, \quad (4.4) \quad \boxed{\text{eq4.4.4}}$$

where the matrix norm is the subordinate matrix norm.

If A is square and nonsingular, then

$$\kappa \leq \|A\| \|A^{-1}\|. \quad (4.5) \quad \boxed{\text{eq4.4.5}}$$

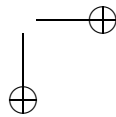
■

Example 4.18 Condition Number of the Polynomial Roots

The problem of finding roots of a polynomial can be highly ill-conditioned. We first illustrate this with a simple quadratic polynomial. Consider solving the quadratic equation:

$$f(x) = x^2 - 2x + 1 = 0.$$

The roots are $x = 1, 1$. Now perturb the coefficient 2 by 0.00001. The computed roots of the perturbed polynomial $\hat{f}(x) = x^2 - 2.00001x + 1$ are: $x_1 = 1.0032$ and



$x_2 = 0.9968$. The relative errors in x_1 and x_2 are 0.0032; on the other hand, the relative error in the data is 5×10^{-6} . Thus, a *small perturbation in the data changed the roots substantially*. ■

The Wilkinson Polynomial

The above example involved multiple roots. Multiple roots or roots close to each other invariably make the root-finding problem ill-conditioned; however, the *problem can be ill-conditioned even when the roots are very well separated*. Consider the following well-known example by Wilkinson:

$$\begin{aligned} p(x) &= (x-1)(x-2)\cdots(x-20) \\ &= x^{20} - 210x^{19} + \cdots \end{aligned}$$

The roots of $p(x)$ are $1, 2, \dots, 20$. Now perturb the coefficient of x^{19} from -210 to $-210 - 2^{-23}$, leaving other coefficients unchanged. This change amounts to approximately 1.12×10^{-7} , which is small. Several roots of the perturbed polynomial, carefully computed by Wilkinson, were found to be very different from the original roots. For example, the roots $x = 16$ and $x = 17$ became approximately equal to $16.73 \pm 2.81i$. This change can be easily explained by computing the condition numbers of the individual roots. It can be shown [Exercise 4.23(a)] that the condition number of the root $x = x_j$ with respect to the perturbation of the single coefficient a_i is

$$\text{cond}_j = \frac{|a_i x_j^{i-1}|}{|p'(x_j)|}$$

Using this definition it is easy to verify that cond_{16} and cond_{17} are both of order $O(10^{10})$, which are quite large.

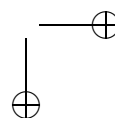
Note: The definition of conditioning is data-dependent. Thus, a problem which is ill-conditioned for *one* set of data could be well-conditioned for *another* set.

Root-finding and Eigenvalue Computation

The above examples teach us a very useful lesson: it is not a good idea to compute the eigenvalues of a matrix by explicitly finding the coefficients of the characteristic polynomial and evaluating its zeros, since the round-off errors in computations will invariably put some small perturbations in the computed coefficients of the characteristic polynomial, and these small perturbations in the coefficients may cause large changes in the zeros. The eigenvalues will then be computed inaccurately.

4.5 Conditioning of the Problem, Stability of the Algorithm, and Accuracy of the Solution

As stated in the previous section, the conditioning of a problem is a property of the problem itself, and has nothing to do with the algorithm used to solve the



problem. To a user, of course, the accuracy of the computed solution is of primary importance. However, the accuracy of a computed solution by a given algorithm is directly connected with both the stability of the algorithm and the conditioning of the problem. *If the problem is ill-conditioned, no matter how stable the algorithm is, the accuracy of the computed solution cannot be guaranteed.*

In general, if a backward stable algorithm is applied to a problem with the condition number κ , then the accuracy of the solution depends upon κ . *If it is small, the results will be accurate; but if it is large, the accuracy can not be guaranteed, it will depend how large the condition number is.*

Backward Stability and Accuracy

Note that the definition of backward stability does not say that the computed solution \hat{x} by a backward stable algorithm will be close to the exact solution of the original problem. However, when a stable algorithm is applied to a well-conditioned problem, the computed solution should be near the exact solution. Also, if a "stable" algorithm is applied to an ill-conditioned problem, it should not introduce more error than what the data warrants.

Stable Algorithm + Well-Conditioned Problem

\equiv Accurate Solution (the computed solution is near the exact solution).

Stable Algorithm + Ill-Conditioned Problem \equiv Accuracy not guaranteed.

Illustration. Suppose that an algorithm to solve the computational problem f defined by the input x produces the function \hat{f} as an approximation of f . Let y be close to x . Then the behavior of a stable algorithm in two cases, when the problem is well-conditioned and ill-conditioned, is illustrated in the following figure: (Stewart (1998, p.123))

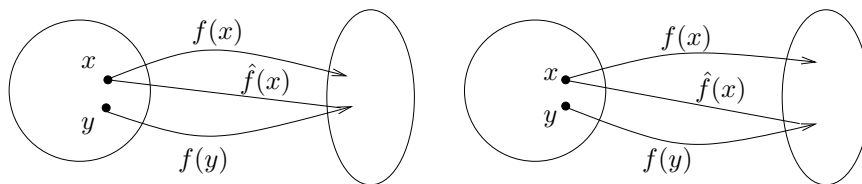


Figure 4.2. Performance of a Backward Stable Algorithm with Well-Conditioned (Left-Side) and Ill-Conditioned Problem (Right-Side).

Why Inaccurate Answer? Based on our discussions here, we can state some of the following reasons why the answer might be inaccurate (see Higham (1996), p.35 for details).

- The problem might be ill-conditioned

- The algorithm might be unstable
- The test examples may be too special
- The algorithm, though successful, might have failed in the particular circumstances

4.6 Perturbation Analysis of the Linear System Problem

Consider the following linear system:

$$\begin{aligned}x_1 + 2x_2 &= 3 \\ 2x_1 + 3.999x_2 &= 5.999\end{aligned}$$

The exact solution is: $x_1 = x_2 = 1$. Now make a small perturbation in the right-hand side obtaining the system:

$$\begin{aligned}x_1 + 2x_2 &= 3 \\ 2x_1 + 3.999x_2 &= 6\end{aligned}$$

The solution of the perturbed system, obtained by Gaussian elimination with partial pivoting (considered to be a stable method in practice) is: $x_1 = 3, x_2 = 0$.

Thus, a very small change in the right hand side changed the solution altogether.

In this section we study the effect of small perturbations of the input data A and b on the computed solution x of the system $Ax = b$.

Since in the linear system problem $Ax = b$, the input data are A and b , there could be impurities either in b or in A or in both. We will therefore consider the effect of perturbations on the solution x in each of these cases separately. We will see that in all these cases, a number called the *condition number* of the matrix A plays an important role.

4.6.1 Effect of Perturbation in the Right-Hand Side Vector b

ss4.6.1

We assume here that there are impurities in b but the matrix A is exact.

T6.6.1

Theorem 4.19. (Right Perturbation Theorem) If δb and δx , are, respectively, the perturbations of b and x in the linear system $Ax = b$, A is nonsingular and $b \neq 0$, then

$$\frac{\|\delta b\|}{\text{Cond}(A)\|b\|} \leq \frac{\|\delta x\|}{\|x\|} \leq \text{Cond}(A) \frac{\|\delta b\|}{\|b\|}.$$

Proof. Since

$$Ax = b,$$

and

$$A(x + \delta x) = b + \delta b,$$

We have

$$A\delta x = \delta b.$$

That is,

$$\delta x = A^{-1}\delta b.$$

Taking a subordinate matrix-vector norm we get

$$\|\delta x\| \leq \|A^{-1}\| \|\delta b\|. \quad (4.6) \quad \boxed{6.6.1}$$

Again, taking the same norm on both sides of $Ax = b$, we get $\|Ax\| = \|b\|$ or

$$\|b\| = \|Ax\| \leq \|A\| \|x\| \quad (4.7) \quad \boxed{6.6.2}$$

Combining $\boxed{6.6.1}$ and $\boxed{6.6.2}$, we have

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|\delta b\|}{\|b\|}. \quad (4.8) \quad \boxed{6.6.3}$$

On the other hand, $A\delta x = \delta b$ gives

$$\|\delta x\| \geq \frac{\|\delta b\|}{\|A\|} \quad (4.9) \quad \boxed{6.6.4}$$

Also, from $Ax = b$, we have

$$\frac{1}{\|x\|} \geq \frac{1}{\|A^{-1}\| \|b\|}. \quad (4.10) \quad \boxed{6.6.5}$$

Combining $\boxed{6.6.4}$ and $\boxed{6.6.5}$, we have

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

- The other part can be similarly proved.

□

Definition 4.20. The number $\|A\| \|A^{-1}\|$ is called the condition number of A and is denoted by $\text{Cond}(A)$.

Interpretation of Theorem $\boxed{6.6.1}$ $\boxed{4.19}$

Theorem $\boxed{4.19}$ $\boxed{6.6.1}$ says that a relative change in the solution can be as large as $\text{Cond}(A)$ multiplied by the relative change in the vector b . Thus, *if the condition number is not too large, then a small perturbation in the vector b will have very little effect on the solution. On the other hand, if the condition number is large, then even a small perturbation in b might change the solution drastically.*

Remark: In view of Theorem ^{¶6.6.1}4.19, what happened with the above example can be easily explained. Note that for this example, $\text{Cond}(A) = O(10^4)$.

E4.20 Example 4.21 An ill-conditioned linear system problem

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4.0001 & 2.002 \\ 1 & 2.002 & 2.004 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 8.0021 \\ 5.006 \end{pmatrix}$$

The exact solution $x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. Change b to $b' = \begin{pmatrix} 4 \\ 8.0020 \\ 5.0061 \end{pmatrix}$.

Relative Perturbation:

$$\frac{\|b' - b\|}{\|b\|} = \frac{\|\delta b\|}{\|b\|} = 1.879 \times 10^{-5} \quad (\text{small}).$$

If we solve the system $Ax' = b'$, we get $x' = x + \delta x = \begin{pmatrix} 3.0850 \\ -0.0436 \\ 1.0022 \end{pmatrix}$.

(x' is completely different from x)

Relative Error in the solution: $\frac{\|\delta x\|}{\|x\|} = 1.3461$. It is easily verified that the inequality in Theorem ^{¶6.6.1}4.19 is satisfied: $\text{Cond}(A) \cdot \frac{\|\delta b\|}{\|b\|} = 96.5920 > 1.3461 = \frac{\|\delta x\|}{\|x\|}$.

The predicted change was, however, overly estimated. ■

Example 4.22 A well-conditioned problem

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 7 \end{pmatrix}.$$

The exact solution $x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. Let $b' = b + \delta b = \begin{pmatrix} 3.0001 \\ 7.0001 \end{pmatrix}$.

The relative change in b : $\frac{\|b' - b\|}{\|b\|} = 1.875 \times 10^{-5}$ (**small**). Note that $\text{Cond}(A) = 14.9330$ (**small**). *Thus a drastic change in the solution x is not expected.* In fact x' satisfying $Ax' = b'$ is

$$x' = \begin{pmatrix} 0.9999 \\ 1.0001 \end{pmatrix} \approx x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \quad \text{Note: } \frac{\|\delta x\|}{\|x\|} = 10^{-5} \quad (\text{small}).$$

■

4.6.2 Effect of Perturbation in the matrix A

Here we assume that there are impurities only in A and as a result we have $A + \Delta A$ in hand, but b is exact.

T6.6.2

Theorem 4.23. (Left Perturbation Theorem) Assume A is nonsingular and $b \neq 0$. Suppose that ΔA and δx are, respectively, the perturbations of A and x in the linear system $Ax = b$. Furthermore, assume that ΔA is such that $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$.

Then

$$\frac{\|\delta x\|}{\|x\|} \leq \text{Cond}(A) \frac{\|\Delta A\|}{\|A\|} / \left(1 - \text{Cond}(A) \frac{\|\Delta A\|}{\|A\|}\right).$$

Proof. We have

$$(A + \Delta A)(x + \delta x) = b,$$

or

$$(A + \Delta A)x + (A + \Delta A)\delta x = b. \quad (4.11) \quad \boxed{6.6.6}$$

Since $Ax = b$, we have from (4.11)

$$(A + \Delta A)\delta x = -\Delta Ax \quad (4.12) \quad \boxed{6.6.7}$$

or

$$\delta x = -A^{-1}\Delta A(x + \delta x). \quad (4.13) \quad \boxed{6.6.8}$$

Taking the norm on both sides, we have

$$\begin{aligned} \|\delta x\| &\leq \|A^{-1}\| \|\Delta A\| \cdot (\|x\| + \|\delta x\|) \\ &= \frac{\|A^{-1}\| \|A\| \|\Delta A\|}{\|A\|} (\|x\| + \|\delta x\|) \end{aligned} \quad (4.14) \quad \boxed{6.6.9}$$

that is,

$$\left(1 - \frac{\|A\|^{-1} \|A\| \|\Delta A\|}{\|A\|}\right) \|\delta x\| \leq \frac{\|A\| \|A^{-1}\| \|\Delta A\|}{\|A\|} \|x\|. \quad (4.15) \quad \boxed{6.6.10}$$

Since $\|A^{-1}\| \|\Delta A\| < 1$, the expression under parenthesis of the left hand side is positive. We can thus divide both sides of the inequality by this number without changing the inequality. After this, if we also divide by $\|x\|$, we obtain

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A\| \|A^{-1}\| \|\Delta A\|}{\|A\|} / \left(1 - \frac{\|A\| \|A^{-1}\| \|\Delta A\|}{\|A\|}\right) = \text{Cond}(A) \frac{\|\Delta A\|}{\|A\|} / \left(1 - \text{Cond}(A) \frac{\|\Delta A\|}{\|A\|}\right) \quad (4.16) \quad \boxed{6.6.11}$$

which proves the theorem. \square

Remarks: Because of the assumption that $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$ (which is quite reasonable to assume), the denominator on the right hand side of the inequality in Theorem 4.23 is less than one. Thus even if $\frac{\|\Delta A\|}{\|A\|}$ is small, then there could be a drastic change in the solution if $\text{Cond}(A)$ is large.

Example 4.24 Consider Example ^{E4.20}4.21 once more. Change $a_{23} = 2.002$ to 2.0021; keep b fixed. Thus

$$\Delta A = -10^{-4} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad (\text{small}).$$

Now solve the system: $(A + \Delta A)x' = b$:

$$x' = \begin{pmatrix} 3.0852 \\ -0.0437 \\ 1.0011 \end{pmatrix}, \quad \delta x = x' - x = \begin{pmatrix} 2.0852 \\ -1.0437 \\ 0.0021 \end{pmatrix}$$

$$\text{Relative Error} = \frac{\|\delta x\|}{\|x\|} = 1.3463 \quad (\text{quite large}).$$

Note that $\text{Cond}(A) = O(10^5)$. ■

4.6.3 Effect of Perturbations in both the matrix A and the vector b

Finally, we assume now that both the input data A and b have impurities. As a result we have the system with $A + \Delta A$ as the matrix and $b + \delta b$ as the right hand side vector.

T6.6.3 **Theorem 4.25. (General Perturbation Theorem)** Assume that A is nonsingular, $b \neq 0$, and $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$. Then

$$\frac{\|\delta x\|}{\|x\|} \leq \left(\frac{\text{Cond}(A)}{1 - \text{Cond}(A) \cdot \frac{\|\Delta A\|}{\|A\|}} \right) \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

Proof. Subtracting

$$Ax = b$$

from

$$(A + \Delta A)(x + \delta x) = b + \delta b$$

we have

$$(A + \Delta A)(x + \delta x) - Ax = \delta b$$

or

$$(A + \Delta A)(x + \delta x) - (A + \Delta A)x + (A + \Delta A)x - Ax = \delta b$$

or

$$(A + \Delta A)(\delta x) - \Delta Ax = \delta b$$

or

$$A(I - A^{-1}(-\Delta A))\delta x = \delta b + \Delta Ax. \quad (4.17) \quad \boxed{6.6.12}$$

Let $A^{-1}(-\Delta A) = F$. Then

$$\|F\| = \|A^{-1}(-\Delta A)\| \leq \|A^{-1}\| \|\Delta A\| < 1 \quad (\text{by assumption}).$$

Since $\|F\| < 1$, $I - F$ is invertible (see Theorem [1.7.7](#)) and then from [6.6.12](#) we have

$$\delta x = (I - F)^{-1} A^{-1} (\delta b + \Delta A x).$$

Again, using Theorem [1.7.7](#), we can write

$$\|(I - F)^{-1}\| \leq \frac{1}{1 - \|F\|} \quad (4.18) \quad \boxed{6.6.13}$$

Thus,

$$\|\delta x\| \leq \frac{\|A^{-1}\|}{1 - \|F\|} (\|\delta b\| + \|\Delta A\| \|x\|)$$

or

$$\begin{aligned} \frac{\|\delta x\|}{\|x\|} &\leq \frac{\|A^{-1}\|}{(1 - \|F\|)} \cdot \left(\frac{\|\delta b\|}{\|x\|} + \|\Delta A\| \right) \\ &\leq \frac{\|A^{-1}\|}{(1 - \|F\|)} \left(\frac{\|\delta b\| \|A\|}{\|b\|} + \|\Delta A\| \right) \quad (\text{Note that } \frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|}). \end{aligned} \quad (4.19) \quad \boxed{6.6.14}$$

That is,

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\| \|A\|}{(1 - \|F\|)} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right). \quad (4.20) \quad \boxed{6.6.15}$$

Again

$$\|F\| = \|A^{-1}(-\Delta A)\| \leq \|A^{-1}\| \|\Delta A\| = \frac{\|A^{-1}\| \|A\|}{\|A\|} \cdot \|\Delta A\|. \quad (4.21) \quad \boxed{6.6.16}$$

Since $\|F\| \leq 1$, we can write from [6.6.15](#) and [6.6.16](#)

$$\begin{aligned} \frac{\|\delta x\|}{\|x\|} &\leq \left(\frac{\|A^{-1}\| \|A\|}{(1 - (\frac{\|A^{-1}\| \|A\|}{\|A\|}) \cdot \|\Delta A\|)} \right) \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right) \\ &= \left(\frac{\text{Cond}(A)}{(1 - \frac{\text{Cond}(A)}{\|A\|} \cdot \|\Delta A\|)} \right) \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right). \end{aligned} \quad (4.22) \quad \boxed{6.6.17}$$

■ □

Remarks: We again see from (4.22) that even if the relative perturbations $\frac{\|\delta_b\|}{\|b\|}$ and $\frac{\|\Delta A\|}{\|A\|}$ are small, there might be a drastic change in the solution, if $\text{Cond}(A)$ is large. Thus, $\text{Cond}(A)$ plays the crucial role in the sensitivity of the solution.

Notation for Condition Numbers

Unless otherwise stated, when we write $\text{Cond}(A)$, we will mean $\text{Cond}_2(A)$, that is, the condition number with respect to 2-norm. The condition number of a matrix A with respect to a subordinate p norm ($p = 1, 2, \infty$) will be denoted by $\text{Cond}_p(A)$, that is, $\text{Cond}_1(A)$ will stand for the condition number of A with respect to 1-norm, etc.

4.7 Some Properties of the Condition Number of a Matrix

The following are some important (but easy to prove) properties of the condition number of a matrix.

- (I) $\text{Cond}(A) \geq 1$.
- (II) $\text{Cond}_2(A) = 1$ if and only if A is a nonzero scalar multiple of an orthogonal matrix, i.e., $A^T A = \alpha I$, where $\alpha \neq 0$.
(Note that this property of an orthogonal matrix A makes the matrix so attractive for its use in numerical computations.)
- (III) $\text{Cond}(\alpha A) = \text{Cond}(A)$, where α is a nonzero scalar.
- (IV) $\text{Cond}_2(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$, where σ_{\max} and σ_{\min} are the largest and smallest singular values of A .
- (V) $\text{Cond}_2(A^T A) = (\text{Cond}_2(A))^2$.
- (VI) $\text{Cond}_2(A) = \text{Cond}_2(A^T)$; $\text{Cond}_1(A) = \text{Cond}_\infty(A^T)$.
- (VII) $\text{Cond}(AB) \leq \text{Cond}(A) \text{Cond}(B)$, if A and B are compatible for matrix multiplication.

We now formally define the ill-conditioning and well-conditioning in terms of the condition number.

D6-2 **Definition 4.26.** *The system $Ax = b$ is ill-conditioned if $\text{Cond}(A)$ is quite large. Otherwise, it is well-conditioned.*

A Convention: Unless otherwise stated, by $\text{Cond}(A)$ we will mean $\text{Cond}_2(A)$.

Remarks: Though the condition number, as defined above, is norm-dependent, the condition numbers with respect to two different norms are related (see Golub and Van Loan (1996), p. 26). (For example, it can be shown that if A is an $n \times n$ matrix, then $\frac{1}{n} \leq \frac{\text{Cond}_2(A)}{\text{Cond}_\infty(A)} \leq n$.) In general, *if a matrix is well-conditioned or ill-conditioned with respect to one norm, it is also ill-conditioned or well-conditioned with respect to some other norms.*

Example 4.27 ■

(a) Consider

$$A = \begin{pmatrix} 1 & 0.9999 \\ 0.9999 & 1 \end{pmatrix}; \text{ then } A^{-1} = 10^3 \begin{pmatrix} 5.0003 & -4.99997 \\ -4.9997 & 5.0003 \end{pmatrix}.$$

1. The condition numbers with respect to the infinity norm and 1-norm are

$$\|A\|_\infty = \|A\|_1 = 1.9999; \quad \|A^{-1}\|_\infty = \|A^{-1}\|_1 = 10^4$$

$$\text{Cond}_\infty(A) = \text{Cond}_1(A) = 1.9999 \times 10^4$$

2. The condition number with respect to the 2-norm is

$$\|A\|_2 = \sqrt{\rho(A)} = 1.9999, \quad \|A^{-1}\|_2 = \sqrt{\rho(A^{-1})} = 10^4$$

$$\text{Cond}_2(A) = 1.9999 \times 10^4.$$

Remark: For the above example, it turned out that the condition number with respect to any norm is the same. This is, however, not always the case. In general, however, they are closely related. (See below the condition number of the Hilbert matrix with respect to different norms.)

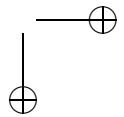
4.7.1 Some Well-known Ill-conditioned Matrices

1. **The Hilbert Matrix**

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \vdots & & & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \cdots & \frac{1}{2n-1} \end{pmatrix}$$

For $n = 10$, $\text{Cond}_2(A) = 1.6025 \times 10^{13}$; $\text{Cond}_\infty(A) = 3.5353 \times 10^{13}$; $\text{Cond}_1(A) = 3.5353 \times 10^{13}$.

2. **The Pei Matrix** $A = (a_{ij})$ with $a_{ii} = \alpha, a_{ij} = 1$ for $i \neq j$. The matrix becomes ill-conditioned when α is close to 1 or $n - 1$. For example, when $\alpha = 0.9999$ and $n = 5$, $\text{Cond}(A) = 5 \times 10^4$.



3. **Vandermonde Matrix** $A = (a_{ij})$, where $a_{ij} = v_i^{n-j}$; $v_i = i$ -th component of an n -vector v . For $n = 5$, $v = (1, 2, 3, 4, 5)^T$, $\text{Cond}(A) = 2.6170 \times 10^4$. This matrix arises in several practical applications, including polynomial interpolation.

4.7.2 How Large Must the Condition Number be for Ill-Conditioning?

A frequently asked question is: how large $\text{Cond}(A)$ has to be before the system $Ax = b$ is considered to be ill-conditioned? We will use Theorem 4.25 to answer the question.

Suppose for simplicity

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|\delta b\|}{\|b\|} = 10^{-d}.$$

Then, from Theorem 4.25, it follows that $\frac{\|\delta x\|}{\|x\|}$ is *approximately* less than or equal to $2 \times \text{Cond}(A) \times 10^{-d}$.

This says that if the data has a relative error of 10^{-d} and if the relative error in the solution has to be guaranteed to be less than or equal to 10^{-t} , then $\text{Cond}(A)$ has to be less than or equal to $\frac{1}{2} \times 10^{d-t}$. Thus, *whether a system is ill-conditioned or well-conditioned depends on (i) the accuracy of the data and, (ii) how much error in the solution can be tolerated.*

For example, suppose that the data have a relative error of about 10^{-5} and an accuracy of about 10^{-3} is sought, then $\text{Cond}(A) \leq \frac{1}{2} \times 10^2 = 50$. On the other hand, if the accuracy of about 10^{-2} is sought, then $\text{Cond}(A) \leq \frac{1}{2} \times 10^3 = 500$. Thus, in the first case the system will be well-conditioned if $\text{Cond}(A)$ is less than or equal to 50, while in the second case, the system will be well-conditioned if $\text{Cond}(A)$ is less than or equal to 500.

Estimating Accuracy from the Condition Number

In general, if the data are approximately accurate and if $\text{Cond}(A) = 10^s$, then there will be only about $t - s$ significant digit accuracy in the computed solution when the solution is computed in t -digit arithmetic.

For better understanding of conditioning, stability and accuracy, we again refer the readers to the paper of Bunch (1987).

4.7.3 The Condition Number and Nearness to Singularity

The condition number also gives an indication when a matrix A is computationally close to a singular matrix: *if $\text{Cond}(A)$ is large, A is close to singular.*

This measure of nearness to singularity is a more accurate measure than the determinant of A . For example, consider the well-known $n \times n$ upper triangular matrix $A = (a_{ij})$ with $a_{ii} = 1$, and $a_{ij} = -1$ if $j > i$. The matrix has the determinant equal to 1, however, it is nearly singular for large n . Note that

$$\text{Cond}_\infty(A) = n2^{n-1}.$$

Similarly, *the smallness of the determinant of a matrix does not necessarily mean that A is close to a singular matrix.* For example, consider $A = \text{diag}(0.1, 0.1, \dots, 0.1)$ of order 1000. $\det(A) = 10^{-1000}$, which is a small number. However, A is considered to be perfectly nonsingular, because $\text{Cond}_2(A) = 1$.

4.7.4 Examples of Ill-conditioned Eigenvalue Problems

Perturbation analysis of the eigenvalue problem will be discussed in Chapter ^{ch9}9. The conditioning of the eigenvalues and eigenvectors will be introduced there. Here we just present a few examples of the well-known ill-conditioned eigenvalue problems.

E4.7.2 **Example 4.28** Consider the 10×10 matrix:

$$A = \begin{pmatrix} 1 & 1 & & & & & & & & \\ & 1 & 1 & 0 & & & & & & \\ & & \ddots & \ddots & & & & & & \\ & 0 & & \ddots & \ddots & & & & & \\ & & & & \ddots & 1 & & & & \\ & & & & & & 1 & & & \end{pmatrix}.$$

The eigenvalues of A are all 1. Now perturb the (10,1) coefficient of A by a small quantity $\epsilon = 10^{-10}$. Then the eigenvalues of the perturbed matrix computed using the MATLAB function `eig` (that uses a numerically effective eigenvalue-computation algorithm) were found to be:

$$\begin{array}{l} 0 \\ 1.0184 + 0.0980i \\ 0.9506 + 0.0876i \\ 1.0764 + 0.0632i \\ 0.9051 + 0.0350i \\ 1.0999 + 0.00i \\ 1.0764 - 0.0632i \\ 0.9051 - 0.0350i \\ 1.0184 - 0.0980i \\ 0.9506 - 0.0876i \end{array}$$

(Note the change in the eigenvalues.) ■

Example 4.29 The Wilkinson-Bidiagonal Matrix

Again, it should not be thought that an eigenvalue problem can be ill-conditioned only when the eigenvalues are multiple or are close to each other. An eigenvalue problem with well-separated eigenvalues can be very ill-conditioned too. Consider the 20×20 triangular matrix (known as the Wilkinson-bidiagonal matrix):

$$A = \begin{pmatrix} 20 & 20 & & & \\ & 19 & 20 & & 0 \\ & & \ddots & \ddots & \\ & 0 & & \ddots & 20 \\ & & & & 1 \end{pmatrix}.$$

The eigenvalues of A are $1, 2, \dots, 20$. Now perturb the $(20,1)$ entry of A by $\epsilon = 10^{-10}$. If the eigenvalues of this slightly perturbed matrix are computed using MATLAB function `eig`, it will be seen that some of them will change drastically; they will even become complex, as shown in the following graph. Again, this can be explained by using the definition of condition number of an individual eigenvalue given in Chapter 9.

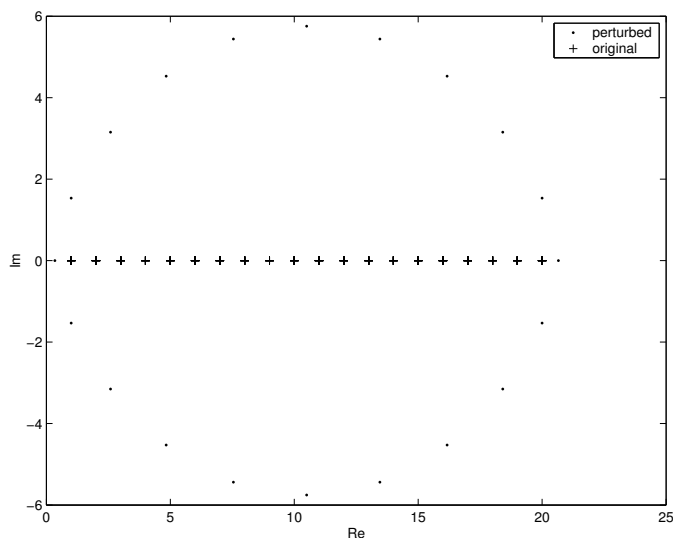


Figure 4.3. The Eigenvalues of a Slightly Perturbed Wilkinson Matrix.

■

E4.7.4 Example 4.30 (J. Wilkinson (1965), p. 92)

$$A = \begin{pmatrix} n & (n-1) & (n-2) & \cdots & 3 & 2 & 1 \\ (n-1) & (n-1) & (n-2) & \cdots & 3 & 2 & 1 \\ 0 & (n-2) & (n-2) & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & & & \ddots & \ddots & 2 & \vdots \\ \vdots & & & & & 2 & 2 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 1 \end{pmatrix}$$

As n increases, the smallest eigenvalues become progressively ill-conditioned. For example, when $n = 12$, the condition numbers of the first few eigenvalues are of order unity while those of the last three are of order 10^7 . ■

4.8 Some Guidelines for Designing Stable Algorithms

Following Higham (1996, pp. 30-31), and based on our discussions in this chapter, we state a few helpful guidelines for designing a stable algorithms: However note “*There is no simple recipe for designing stable algorithms*” (Higham (1996), p. 30)

- Avoid catastrophic cancellations, if possible.
- Avoid unnecessary overflow and underflow.
- In transforming the problem to another mathematically equivalent problem, use only well-conditioned transformation, such as orthogonal matrix multipliability.
- If a numerical scheme appears to be unstable, look for different formulations, which are mathematically equivalent, but not numerically (see the use of modified Gram Schmidt processes versus classical Gram Schmidt processes in solving least-squares problems in Chapter 8).
- Arrange your computational scheme (if possible) in such a way that the intermediate quantities are much smaller than the final answer.
- Update the solution only by using a small correction; that is, update as new solution = old solution + small correction, if the *correction can be computed with sufficient figures*

4.9 Review and Summary

In this chapter we have introduced two of the most important concepts in numerical linear algebra, namely, the **conditioning** of the problem and **stability** of the algorithm, and have discussed how they effect the **accuracy** of the solution.

4.9.1 Conditioning of the Problem

The conditioning of the problem is a property of the problem. A problem is said to be **ill-conditioned** if a small change in the data can cause a large change in the solution, otherwise it is **well-conditioned**.

The examples of ill-conditioned problems:

- Wilkinson's polynomial of degree 20 for the root-finding problem
- Wilkinson's bidiagonal matrix for the eigenvalue problem
- The Hilbert matrix for the algebraic linear system problem

The conditioning of a problems is data dependent. A problem can be ill-conditioned with respect to one set of data while it may be quite well-conditioned with respect to another set.

Ill-conditioning or well-conditioning of a matrix problem is generally measured by means of a number called the **condition number**.

In the linear system problem $Ax = b$, the input data are A and b . There may exist impurities either in A or in b , or in both.

We have presented perturbation analyses in all the three cases. The results are contained in Theorems [4.19](#), [4.23](#), and [4.25](#). Theorem [4.25](#) is the most general theorem.

In all these three cases, it turns out that

$$\text{Cond}(A) = \|A\| \|A^{-1}\|$$

is the deciding factor. If this number is large, then a small perturbation in the input data may cause a large relative error in the computed solution. In this case, the system is called an **ill-conditioned system**, otherwise it is **well-conditioned**. The matrix A having a large condition number is called an **ill-conditioned matrix**.

Some important properties of the condition number of a matrix have been listed ([Section 4.7](#)).

The condition number, of course, has a noticeable effect on the accuracy of the solution.

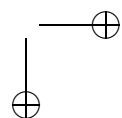
A frequently asked question is: *How large does $\text{Cond}(A)$ have to be before the system $Ax = b$ is considered to be ill-conditioned?*

The answer depends upon the accuracy of the input data and the level of tolerance of the error in the solution.

In general, if the data are approximately accurate and if $\text{Cond}(A) = 10^s$, then there are about $(t - s)$ significant digits of accuracy in the solution, if it is computed in t -digit arithmetic.

4.9.2 Stability of an Algorithm

An algorithm is said to be a **backward stable algorithm** if it computes the exact solution of a nearby problem. Some examples of stable algorithms are (as we will see later in the book) are:



- Backward substitution and forward elimination for triangular systems
- Gaussian elimination with complete pivoting for linear systems
- QR factorization using Householder and Givens transformations
- QR iteration algorithm for eigenvalue computations, etc.

The *Gaussian elimination algorithm without row changes is unstable for arbitrary matrices. It is stable for special matrices such as strictly diagonally dominant, Hessenberg and symmetric positive definite. Gaussian elimination with partial pivoting is stable in practice.*

4.9.3 Effects of conditioning and stability on the accuracy of the solution

The conditioning of the problem and the stability of the algorithm both have effects on accuracy of the solution computed by the algorithm.

Stable Algorithm + Well-Conditioned Problem \equiv Accurate Solution

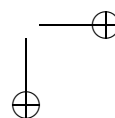
Stable Algorithm + Ill-Conditioned Problem \equiv Accuracy not guaranteed.

4.10 Suggestions for Further Reading

The basic concepts and results of stability and conditioning can be found in most numerical linear algebra books (e.g., Golub and Van Loan (1996), Stewart (1973), Trefethan and Bau (1997)). The two most authoritative books on these topics are the classical book by Wilkinson (1965) and the most recent one by Higham (1996). Stewart's recent books (1998a and 1998b) also give a fair amount of coverage of these topics. A book devoted entirely to the perturbation analysis is by Stewart and Sun (1990). An advanced book containing a fair amount of matrix perturbation results is by Bhatia (1991). For a condensed review of material of this chapter, see the article of Byers and Datta in *Handbook of Linear Algebra* (2006, pp. 37-1–37-32).

Exercises on Chapter ^{ch4}4

- 4.1 (a) Show that the floating point computations of the sum, product and division of two numbers are backward stable.
- (b) Show that the floating computation of the inner product of two vectors is backward stable, on the other hand, the outer product is not.
- 4.2 Are the following floating point computations backward stable? Give reasons for your answer in each case.
- (a) $fl(x + 1)$
- (b) $fl(x(y + z))$
- (c) $fl(x_1 + x_2 + \cdots + x_n)$



- (d) $\text{fl}(x_1 x_2 \cdots x_n)$
- (e) $\text{fl}(x^T y / c)$, where x and y are vectors and c is a scalar
- (f) $\text{fl}\left(\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\right)$

4.3 Show that the roots of the following polynomials are ill-conditioned and give reasons for your answers.

- (a) $x^3 - 3x^2 + 3x + 1$
- (b) $(x - 1)^3(x - 2)$
- (c) $(x - 1)(x - 0.99)(x - 2)$

4.4 Work out the flop-counts for the following simple matrix operations:

- (i) Multiplication of matrices A and B of orders $n \times m$ and $m \times p$, respectively.
- (ii) Multiplication of a matrix A of order $m \times n$ by a vector b .
- (iii) Multiplication of a column vector u by a row vector v .
- (iv) Computation of $\|u\|_2$.
- (v) Multiplication of row vector u by a column vector v .
- (vi) Computation of the matrix $A = \frac{uv^T}{u^T v}$, where u and v are m column vectors.
- (vii) Computation of the matrix $B = A - uv^T$, where A and B are two $n \times n$ matrices and u and v are two column vectors.

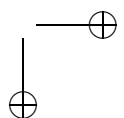
4.5 Develop an algorithm to compute the product $C = AB$ in each of the following cases. Your algorithm should take advantage of the special structure of the matrices in each case. Give flop-count and show storage requirement in each case.

- (a) A and B are both lower triangular matrices.
- (b) A is arbitrary and B is lower triangular.
- (c) A and B are both tridiagonal.
- (d) A is arbitrary and B is upper Hessenberg.
- (e) A is upper Hessenberg and B is tridiagonal.
- (f) A is upper Hessenberg and B is upper triangular.

4.6 A square matrix $A = (a_{ij})$ is said to be a **band matrix** of bandwidth $2k + 1$ if

$$a_{ij} = 0 \quad \text{whenever } |i - j| > k.$$

Develop an algorithm to compute the product $C = AB$, where A is arbitrary and B is a band matrix of bandwidth 2, taking advantage of the structure of the matrix B . Overwrite A with AB and give flop-count.



- 4.7 Let A and B be two symmetric matrices of the same order. Develop an algorithm to compute $C = A + B$, taking advantage of symmetry for each matrix. Your algorithm should overwrite B with C . What is the flop-count?
- 4.8 Let a_r and b_r denote, respectively, the r^{th} columns of the matrices A and B . Then develop an algorithm to compute the product AB from the formula

$$AB = \sum_{i=1}^n a_i b_i^T.$$

Give flop-count and storage requirement of the algorithm.

- 4.9 Consider the matrix

$$A = \begin{pmatrix} 12 & 11 & 10 & \cdots & 3 & 2 & 1 \\ 11 & 11 & 10 & \cdots & 3 & 2 & 1 \\ 0 & 10 & 10 & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & & & \ddots & \ddots & 2 & \vdots \\ \vdots & & & & & 2 & 2 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 1 \end{pmatrix}$$

Find the eigenvalues of this matrix using MATLAB command `eig`. Now perturb the (1,12) element to 10^{-9} and compute the eigenvalues of this perturbed matrix. What conclusion do you make about the conditioning of the eigenvalues?

- 4.10 If $f(x)$ is a real-valued differentiable function of a real variable x , then prove that $\frac{|f'(x)||x|}{|f(x)|}$ is the condition number of $f(x)$ at x .
- 4.11 (a) Show that if $f(x) = \log x$, then the condition number, $c(x) = \left| \frac{1}{\log x} \right|$.
 (b) Using the above result (or otherwise), show that $\log x$ is ill-conditioned near $x = 1$.
- 4.12 Show that the condition number κ for the product Ax (with respect to the perturbation of x) is $\kappa = \|A\| \frac{\|x\|}{\|Ax\|}$.
- 4.13 Show, by computing the condition number, that the problem of computing \sqrt{x} for $x > 0$ is a well-conditioned problem.
- 4.14 Work out a bound for the relative error when a backward stable algorithm is applied to a problem with the condition number κ .

- 4.15** let A be nonsingular and ΔA be such that $\frac{\|\Delta A\|}{\|A\|} < \text{Cond}(A)$. Then prove that $A + \Delta A$ is nonsingular.
- 4.16** (a) How are $\text{Cond}_2(A)$ and $\text{Cond}_2(A^{-1})$ related?
 (b) Show that
- i. $\text{Cond}_2(A) \geq 1$
 - ii. $\text{Cond}_2(A^T A) = (\text{Cond}_2(A))^2$.
- 4.17** (a) Let A be an orthogonal matrix. Then show that $\text{Cond}_2(A) = 1$.
 (b) Show that the $\text{Cond}_2(A) = 1$ if and only if A is a scalar multiple of an orthogonal matrix.
- 4.18** Let $U = (u_{ij})$ be a nonsingular upper triangular matrix. Then show that

$$\text{Cond}_2(U) \geq \frac{\max(u_{ii})}{\min(u_{ii})}.$$

Hence construct a simple example of an ill-conditioned non-diagonal symmetric positive definite matrix.

- 4.19** Let $A = LDL^T$ be a symmetric positive definite matrix. Let $D = \text{diag}(D_{ii})$. Then show

$$\text{Cond}_2(A) \geq \frac{\max(d_{ii})}{\min(d_{ii})}.$$

Hence construct an example for an ill-conditioned non-diagonal symmetric positive definite matrix.

- 4.20** Prove that for a given norm, $\text{Cond}(AB) \leq \text{Cond}(A) \cdot \text{Cond}(B)$.
- 4.21** (a) Find for what values of a the matrix $A = \begin{pmatrix} 1 & a \\ a & 1 \end{pmatrix}$ is ill-conditioned?
 (b) What is the condition number of A ?
- 4.22** Given an example to show that a stable algorithm applied to an ill-conditioned problem can produce an inaccurate solution.
- 4.23** (a) Let a_i be the i th coefficient of a polynomial $p(x)$ and let ∂a_i and ∂x_j denote small perturbations of a_i and the j th root x_j . Then show that the condition number of the root x_j with respect to perturbations of the coefficient a_i is

$$\frac{|a_i x_j^{i-1}|}{|p'(x_j)|}.$$

- (b) Using *MATLAB* function **polyval**, compute the condition numbers of the roots $x = i$, $i = 1, 2, \dots, 20$ of the Wilkinson polynomial

$$p(x) = (x-1)(x-2)\dots(x-20) = x^{20} - 210x^{19} + \dots$$

with respect to perturbation of the coefficient x^{19} from -210 to $-210 + 2^{-23}$. Present your results in tabular form and write your conclusion on the ill-conditioning of the roots of the Wilkinson polynomial. Explain why certain roots are more ill-conditioned than the others.

MATLAB AND MATCOM PROGRAMS AND PROBLEMS ON CHAPTER 4

M.4.1 Using the MATLAB function ‘**rand**’, create a 5×5 random matrix and then print out the following outputs:

$A(2,:)$, $A(:,1)$, $A(:,5)$,
 $A(1, 1: 2 : 5)$, $A([1, 5])$, $A(4: -1: 1, 5: -1: 1)$.

M.4.2 Using the function ‘**for**’, write a MATLAB program to find the **inner product** and **outer product** of two n -vectors u and v .

$$[s] = \mathbf{inpro}(u,v)$$

$$[A] = \mathbf{outpro}(u,v)$$

Test your program by creating two different vectors u and v using `rand(4,1)`.

M.4.3 Learn how to use the following MATLAB commands to create special matrices:

compan	Companion matrix
diag	Diagonal matrices or the diagonals of a matrix
ones	Matrix with all entries equal to 1
zeros	Zero matrix
rand	Random matrix
wilkinson	Wilkinson’s eigenvalue test matrix
hankel	Hankel matrix
toeplitz	Toeplitz matrix
hilb	Hilbert matrix
triu	Extract the upper triangular part of a matrix
tril	Extract the lower triangular part of a matrix
vander	Vandermonde matrix
rand(n)	Matrix with random entries, chosen from a normal distribution with mean zero, variance one and standard deviation one.

M.4.4 Learn how to use the following MATLAB functions for *basic matrix computations* (you will learn about the algorithms of these functions later in this book):

a\b	Linear equation solution of $Ax = b$.
inv	Matrix inverse
det	Determinant
cond	Condition number
eig	Eigenvalues and eigenvectors
norm	Various matrix and vector norms
poly	Characteristic polynomial
polyval	The value of a polynomial at a given number
plot	Plotting various functions.
rank	Rank of a matrix.
lu	LU factorization.
qr	QR factorization.
svd	Singular Value Decomposition.

M.4.5 Write MATLAB programs to create the following well-known matrices:

- (a) $[A] = \mathbf{wilk}(n)$ to create the Wilkinson bidiagonal matrix $A = (a_{ij})$ of order n :

$$\begin{aligned} a_{ii} &= n - i + 1, \quad i = 1, 2, \dots, 20 \\ a_{i-1,i} &= n, \quad i = 2, 3, \dots, n \\ a_{ij} &= 0, \quad \text{otherwise.} \end{aligned}$$

- (b) $[A] = \mathbf{Pei}(n)$ to create the Pei matrix $A = (a_{ij})$ of order n :

$$\begin{aligned} a_{ij} &= \alpha, \quad \alpha \text{ is a parameter near } 1 \text{ or } n - 1. \\ a_{ii} &= 1 \text{ for } i \neq j. \end{aligned}$$

- (c) Print the condition numbers of each of these two matrices with $n = 10, 20, 50$, and 100 , using the MATLAB function **cond**, with respect to the matrix norms $\|\cdot\|_2$, $\|\cdot\|_F$, and $\|\cdot\|_\infty$.

M.4.6 Using “help” commands for “**clock**” and “**etime**”, learn how to measure timing for an algorithm.

M.4.7 Using MATLAB functions ‘**for**’, ‘**size**’, ‘**zero**’, write a MATLAB program to find the product of two upper triangular matrices A and B of order $m \times n$ and $n \times p$, respectively. Test your program using

$$\begin{aligned} A &= \mathbf{triu}(\mathbf{rand}(4,3)), \\ B &= \mathbf{triu}(\mathbf{rand}(3,3)). \end{aligned}$$

M.4.8 *The purpose of this exercise is to test that the Hilbert matrix is ill-conditioned with respect to solving the linear system problem.*

- (i) Create $A = \mathbf{hilb}(10)$. Perturb the $(10,1)$ entry of A by 10^{-5} . Call the perturbed matrix B . Let $b = \mathbf{rand}(10,1)$. Compute $x = A \setminus b$, $y = B \setminus b$. Compute $\|x - y\|$ and $\frac{\|x - y\|}{\|x\|}$. What conclusion you draw from here?

- (ii) Compute the condition numbers of both A and B : $\text{Cond}(A), \text{Cond}(B)$.
- (iii) Compute the condition number of A using the MATLAB command $\text{Cond}(A)$ and then use it to compute the theoretical upper bound given in **Theorem 4.23**. Compare this bound with the actual relative error.

M.4.9 Perform the respective experiments stated in Section 4.7 on the Examples 4.28-4.30 to show that the eigenvalue problems for these matrices are ill-conditioned.

M.4.10 Write a MATLAB program to

- (a) Construct the $n \times n$ lower triangular matrix $A = (a_{ij})$ as follows:

$$\begin{aligned} a_{ij} &= 1 && \text{if } i = j \\ a_{ij} &= -1 && \text{if } i > j \\ a_{ij} &= 0 && \text{if } i < j \end{aligned}$$

- (b) Perform an experiment to show that solution of $Ax = b$ with A as above and the vector b created such that $b = Ax$, where $x = (1, 1, \dots, 1)^T$, becomes more and more inaccurate as n increases due to the increasingly ill-conditioning of A . Let \hat{x} denote the computed solution.

Present your results in the following form:

n	$\text{Cond}(A)$	$\hat{x} = A \setminus b$	Relative error $\frac{\ x - \hat{x}\ _2}{\ x\ _2}$	Residual norm $\frac{\ b - A\hat{x}\ _2}{\ b\ _2}$
10				
20				
30				
40				
50				

M.4.11 Using MATLAB function **vander** (v), where $v = \text{rand}(20, 1)$, create a 20×20 Vandermonde matrix A . Now take $x = \text{ones}(20, 1)$ and $b = A * x$. Compute now $y = A \setminus b$. Compare y with x by computing $y - x$ and $\|y - x\|$. What conclusions do you draw?

M.4.12 (*Higham's Gallery of Test Matrices*)

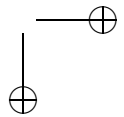
Learn how to use Higham's Gallery of test matrices in MATLAB (type *help gallery* for a complete list).

M.4.13 (*Computing the Sample Variance [Higham(1996)]*).

Consider computing **sample variance** of n numbers x_1, \dots, x_n defined by

$$S_n^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2,$$

where $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.



Describe various mathematically equivalent ways of computing this quantity and discuss their different numerical stability properties.

