Chapter 1

Error Analysis

Numerical analysis is concerned with the process by which mathematical problems can be solved by the operations of arithmetics. It is also concerned with choosing that procedure which is best suited to the solution of a particular problem. A paramount goal in numerical analysis is to access the accuracy of the results of calculations. Errors contained in the numerical answers to problems generally arise in two areas:

1. Those inherent in the mathematical formulation of the problems, such as
   (a) The error incurred when the mathematical statement of a problem is only an approximation to the physical situation;
   (b) The error due to inaccuracies in the physical data.
2. Those incurred in the numerical computation process, such as
   (a) Programming blunder;
   (b) Truncation error, i.e., inexact evaluation of mathematical operators;
   (c) Roundoff errors, i.e., inexact arithmetic calculations.

Type (1) errors are beyond the control of the calculation and are usually negligible. It is understood, however, that the worth of a computed solution must be carefully weighed against these errors. Programming blunder which results in the correct calculation of the wrong result usually can be detected or verified. It is the last two sources of computational error that chiefly interest us and should be controlled by any feasible algorithm.

1.1 Measurement of Errors

In dealing with vectors, matrices and functions, the problem of measuring their exactness to a certain given quantity is usually done by the concept of norms.

Definition 1.1.1 *A norm $\|\cdot\|$ on a vector space V is a real-valued function on V such that for every $u, v \in V$,*

1. *$\|v\| \geq 0; \|v\| = 0$ if and only if $v = 0$;*
2. *$\|\alpha v\| = |\alpha|\|v\|$ for every scalar α;*
3. *$\|u + v\| \leq \|u\| + \|v\|$. (The triangle inequality)*

Examples. In the R^n space,

1. For $p \geq 1, \|v\|_p := (\sum_{i=1}^{n} |v_i|^p)^{1/p}$. ($L_p$-norm)
2. $\|v\|_\infty := \max_{1 \leq i \leq n} |v_i|$. ($L_\infty$-norm)
3. Given $w_i > 0$, $\|v\|_{p,w} := (\sum_{i=1}^{n} w_i |v_i|^p)^{1/p}$. (Weighted L_p-norm).

Examples. In the function space $C[a, b] :=$ The space of all functions continuous on $[a, b]$,

1. $\|f\|_p := (\int_a^b |f(x)|^p dx)^{1/p}$.
2. $\|f\|_\infty := \max_{x \in [a,b]} |f(x)|$. (Uniform norm)

Examples. In the matrix space $R^{n \times m}$,

1. $\|A\| := \sup_{\|x\|=1, x \in R^m} \|Ax\|$. (Induced norm)
   (a) $\|A\|_1 = \max_{1 \leq j \leq m} \sum_{i=1}^{n} |a_{ij}|$.
   (b) $\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{m} |a_{ij}|$.
2. $\|A\|_F := (\sum_{j=1}^{n} \sum_{i=1}^{m} |a_{ij}|^2)^{1/2}$. (Frobenius norm)

1.2 Representation of Numbers

There are two fundamentally different concepts of representing numbers in a computing machine:

1. On an analog computer, the numbers are represented by some physical quantities, e.g., the length of a bar or the intensity of a voltage whereas the arithmetics are simulated through some physical measurements. Therefore, precision in physical measurements limits the accuracy of analog devices.

2. On a digital computer, the numbers are represented by a sequence of digits where each digit is represented by a specific physical quantity. The arithmetic is similar to ordinary pencil-and-paper arithmetic. The accuracy of digital computers is limited by the number of places, the word length, used to internally represent a number.

Most of the modern computers are digital computers. For a word of length n, a number can be stored in several different fashions:

(a) Fixed-point representation where the position of the decimal (binary) point is fixed. This representation is very rarely used in nontrivial calculations.

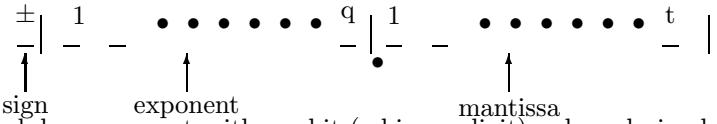
(b) Floating-point representation where the position of the decimal (binary) is not fixed at the outset; rather its position with respect to the first digit is indicated for each number separately.

Definition 1.2.1 *A normalized floating-point number is a number of the form*

$$\pm .d_1 \dots d_t \times \beta^e$$

where β = integer = fixed base; e = integer = exponent; t = number of digits; $1 \le d_1 \le \beta - 1; 0 \le d_i \le \beta - 1$ for $2 \le i \le t$. The fractional part $.d_1 \dots d_t = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_t}{\beta^t}$ is called the mantissa of the floating-point number.

In a digital computer with word length n, a floating-point number is stored as follows:

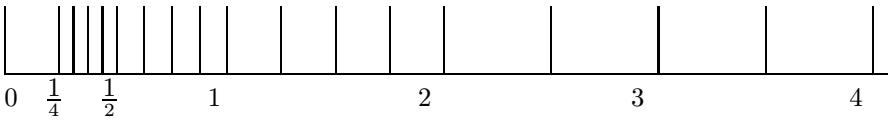


where each bar represents either a bit (a binary digit), a hexadecimal (base 16) or so on. Note that the exponent portion should be able to represent integers from 0 to $\beta^q - 1$ (why?), but with some shift, say by $-\beta^{q-1}$, the range of exponent would be from $-\beta^{q-1}$ to $\beta^{q-1}(\beta - 1) - 1$. In general, we denote this range by saying $-m \le e \le M$, and the system of all floating-point numbers by $F(\beta, t, m, M)$.

Example. Consider the system of floating-point numbers $F(2, 3, 1, 2)$:

	-1	0	1	2
.100	4/16	4/8	4/4	4/2
.101	5/16	5/8	5/4	5/2
.110	6/16	6/8	6/4	6/2
.111	7/16	7/8	7/4	7/2

Note that with each fixed e, in the interval $[\beta^{e-1}, \beta^e)$, each floating-point is equally spaced by β^{e-t}. The distribution on the real line is as follows:



Example. With single precision on an IBM machine, $\beta = 16,\ t = 6,\ m = -64,\ M = 63$, so the range of floating-point numbers is

$$5.398 \times 10^{-79} \approx 16^{-65} \leq |\text{ real }| \leq 16^{63} - 16^{57} \approx 7.237 \times 10^{75}.$$

Any number outside this range is said to be overflow or underflow, respectively.
Example. With single precision on a CRAY machine, $\beta = 2, t = 52, m = -975$, $M = 1071$, so the range of floating-point numbers is

$$1.50 \times 10^{-294} \leq |\text{ real }| \leq 2.5 \times 10^{322}.$$

Example. The IEEE Binary Floating Point Arithmetic Standard for a normalized single-precision 32-bit word specifies 8 bits for the exponent, 23 bits with one bit hidden for the mantissa. The exponent is biased by 127. The floating-point number is described by

$$(-1)^S * 2^{E-Bias} * 1.F$$

where S is the sign of the number, E is the exponent of the number in base 2, and F is the fractional part of the number's mantissa. On the other hand, for a 64-bit word, IEEE specifies 11 bits for the exponent, biased by 1023, 52 bits with one bit hidden for the mantissa.
Remark. The set $F(\beta, t, m, M)$ of floating-point numbers is not a continuum or an infinite set. In fact, it has exactly $2(\beta - 1)\beta^{t-1}(M + m + 1) + 1$ numbers in it.

A number $x \notin F(\beta, t, m, M)$ which is not a machine number has to be approximated by a number which is. Usually such a machine-number approximation $rd(x)$ of x is obtained by rounding:

Definition 1.2.2 *The rounded value m_r to t bits of a mantissa m of more than t bits, with base β, is defined to be*

$$m_r := \{\, \beta^{-t} \lfloor \beta^t m + 0.5 \rfloor,\ \textit{if } m > 0, \beta^{-t} \lceil \beta^t m - 0.5 \rceil,\ \textit{if } m < 0$$

where the ceiling function $\lceil x \rceil :=$ the smallest integer $\geq x$ and the floor function $\lfloor x \rfloor :=$ the largest integers$\leq x$.

Remark. Considering the mantissa only, rounding results in an absolute error bounded by $|\epsilon| \leq \frac{1}{2}\beta^{-t}$. For any number x such that $|x|\epsilon[\beta^{-m-1}, \beta^M)$, if we write $x_r = x(1 + \delta)$, then $|\delta| \leq \frac{1}{2}\beta^{1-t}$. This is because for all x, there exists a unique e such that $\beta^{e-1} \leq x < \beta^e$. In $[\beta^{e-1}, \beta^e)$, numbers are uniformly spaced

by β^{e-t}, it follows that $|x_r - x| \leq \frac{1}{2}\beta^{e-t}$ and hence $\frac{|x_r-x|}{|x|} \leq \frac{\frac{1}{2}\beta^{e-t}}{\beta^{e-1}} = \frac{1}{2}\beta^{1-t}$. On an IBM machine ($\beta = 16$), for example, single precision ($t = 6$) gives $|\delta| \leq 2^{-21} \approx .477 \times 10^{-6}$ whereas double precision ($t = 14$) gives $|\delta| \leq 2^{-53} \approx .111 \times 10^{-15}$.

Remark. For each machine, there are numbers $\epsilon > 0$ such that $1 + \epsilon = 1$. Such number is called the machine precision eps.

1.3 Stability and Conditioning

The results of elementary arithmetic operations $x \pm y,\ x \cdot y,\ x/y$ need not belong to $F(\beta, t, m, M)$, even if both operands $x, y \in F(\beta, t, m, M)$ are machine numbers. In fact, the floating-point operations do not necessarily satisfy the well-known laws for arithmetic operations. It is therefore important to evaluate the performance of different schemes even if they are mathematically equivalent.

Example. (An example of roundoff error) Suppose we have a machine with $\beta = 10$ and $t = 5$. We evaluate the value $e^{-5.5}$ in two different ways:

$$\begin{aligned} e^{-5.5} &= 1.000 - 5.5000 + 15.125 - 27.730 + 38.129 - 41.942 \\ &\quad +38.446 - 30.208 + \dots \\ &= 0.00263363; \\ e^{-5.5} &= \tfrac{1}{e^{5.5}} = \tfrac{1}{1+5.5+15.125+27.730+\dots} = 0.0040865. \end{aligned}$$

It turns out the true value should be $e^{-5.5} \approx 0.00408677$. The wrongness in the first calculation originates from, for example, terms like 38.129 already have roundoff effor which is nearly as large as the final result. (38.12760417)

Definition 1.3.1 *A numerical method is said to be unstable if the roundoff errors introduced at one stage of the computation propagate with increasing magnitude in later stages.*

Example. (An example of instability of certain algorithm) Suppose $\beta = 10$ and $t = 6$. We compute the integrals

$$E_n = \int_0^1 x^n e^{x-1} dx,\ n = 1, 2, \dots$$

Using integration by parts, we find the recursive relationship

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

where $E_1 = 1/e$. Thus

$$\begin{aligned} E_1 &\approx 0.367879, E_6 \approx 0.127120, \\ E_2 &\approx 0.264242, E_7 \approx 0.110160, \\ E_3 &\approx 0.207274, E_8 \approx 0.118720, \\ E_4 &\approx 0.170904, E_9 \approx -0.0684800. \\ E_5 &\approx 0.145480, \end{aligned}$$

Although the integrand $x^9 e^{x-1}$ is positive through the interval $(0, 1)$, our computed value for E_9 is negative.

Observe that the only roundoff error made in the above calculations was in E_1 where $1/e$ was rounded to six significant digits. Since the recurrence formula is exact for real arithmetic, the error in E_9 is entirely due to the rounding error in E_1. Note that the error in E_1 is magnified by a factor of 2 in the calculation of E_2, then the error in E_2 is magnified by 3 in computing E_3, and so on. Thus, the error in E_9 is exactly the error in E_1 multiplied by 9!. Suppose the initial error is $\approx .441 \times 10^{-6}$. Then the error in E_9 should be $\approx .1601$ while the true value of E_9 is ≈ 0.0916.

Definition 1.3.2 *A mathematical problem is said to be well-conditioned if small changes in the data of a problem result in small changes in the solution.*

Example. (An example of sensitivity of certain problems). Suppose $\beta = 2, t = 30$. We want to solve find zeros of the polynomial

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

The zeros of $p(x)$ are obvious and are well separated. Upon entering a typical coefficient into the computer, we have to round it to 30 significant base-2 digits. Suppose a change in the 30-th significant base-2 digit is made only at the coefficient of x^{19}, i.e., suppose we are solving the polynomial

$$q(x) = p(x) + 2^{-23}x^{19}.$$

Then the zeros of $q(x)$ become

1.000000000	10.095266145 $\pm$ 0.643500904i
2.000000000	11.793633881 $\pm$ 1.652329728i
3.000000000	13.992358137 $\pm$ 2.518830070i
4.000000000	16.730737466 $\pm$ 2.812624894i
4.999999928	19.502429400 $\pm$ 1.940330347i
6.000000000	
6.999697234	
8.007267603	
8.917250249	
20.846908104.	

Note that the small change in the coefficient -210 by a quantity $2^{-23} \approx 10^{-7}$ has caused ten of the zeros of $p(x)$ to become complex and that two have moved more than 2.81 units off the real axis.

We now analyze what has happened. We write

$$p(x, \alpha) = x^{20} - \alpha x^{19} + \ldots = 0.$$

Then

$$\begin{aligned} \frac{\partial p(x,\alpha)}{\partial x}\frac{\partial x}{\partial \alpha} + \frac{\partial p(x,\alpha)}{\partial \alpha} &= 0, \\ \frac{\partial x}{\partial \alpha} &= -\frac{\frac{\partial p}{\partial \alpha}}{\frac{\partial p}{\partial x}} = \frac{x^{19}}{\sum_{i=1}^{20}\prod_{j\neq i}(x-j)}. \end{aligned}$$

Evaluating this at each root gives $\frac{\partial x}{\partial \alpha}|_{x=i} = \frac{i^{19}}{\prod_{\substack{j=1 \\ j\neq i}}^{20}(i-j)}$.

These numbers give a direct measure of the sensitivity of each of the roots to the coefficient α. It turns out, for example, $\frac{\partial x}{\partial \alpha}|_{x=18} \approx 1.0 \times 10^9$.

Remark. An ill-conditioned problem can be solved accurately, if this possible at all, only by very careful calculation, quite aside from the method used. An unstable numerical method for a particular (even a well-conditioned) problem may give accurate results only in the early date, but inevitably will give useless results in the long run.

Remark. Let f represent a true algorithm and f^* a floating-point algorithm. Let x represent a true input value and x^* a floating-point input value. Then we have

$$\begin{aligned} & |f(x) - f^*(x^*)| \\ \leq & \underbrace{|f(x) - f(x^*)|}_{\text{conditioning}} + \underbrace{|f(x^*) - f^*(x)|}_{\text{stability}} + \underbrace{|f^*(x) - f^*(x^*)|}_{\text{truncation}}. \end{aligned}$$

1.4 Analysis of Error Propagation

An algorithm can briefly be described as a function $y = \varphi(x)$. Suppose that a function

$$\varphi : D \subset R^n \to R^n$$

is defined on an open subset D, and that its component functions φ_i have continuous first derivatives. Let $\overline{x}$ be an approximate value for x, and let $\overline{y} := \varphi a(\overline{x})$. We denote the absolute errors by $\Delta y := \overline{y} - y$ and $\Delta x := \overline{x} - x$. Then by a Taylor series expansion and disregarding higher-order terms, we obtain

$$\Delta y \approx D\varphi(x)\Delta x \tag{1.1}$$

where $D\varphi(x)$ is the Jacobian matrix

$$D\varphi(x) := \begin{bmatrix} \frac{\partial \varphi_1}{\partial x_1}, & \cdots & \frac{\partial \varphi_1}{\partial x_n} \\ \frac{\partial \varphi_m}{\partial x_1}, & \cdots & \frac{\partial \varphi_m}{\partial x_n} \end{bmatrix}.$$

Similarly, the relative error $\epsilon_{y_i} := \frac{\Delta y_i}{y_i}$ is related to $\epsilon_{x_i} := \frac{\Delta x_i}{x_i}$ by

$$\epsilon_{y_i} \approx \sum_{j=1}^{n} \left(\frac{x_j}{\varphi_i(x)} \frac{\partial \varphi_i}{\partial x_j}\right) \epsilon_{x_j} \tag{1.2}$$

Sometimes it is more convenient to replace (1.2) by an estimate

$$\frac{\|\varphi(\overline{x}) - \varphi(x)\|}{\|\varphi(x)\|} \leq c \frac{\|\overline{x} - x\|}{\|x\|}$$

where c is called the condition number.

Example. For the elementary operations, we have

1. $\epsilon_{xy} \approx \epsilon_x + \epsilon_y$.
2. $\epsilon_{x/y} \approx \epsilon_x - \epsilon_y$.
3. $\epsilon_{x\pm y} \approx \frac{x}{x\pm y}\epsilon_x \pm \frac{y}{x\pm y}\epsilon_y$, if $x \pm y \neq 0$
4. $\epsilon_{j\overline{x}} \approx \frac{1}{2}\epsilon_x$.

It should be noted that the multiplication, division, and square root are not dangerous: The relative errors of the operands do not propagate strongly into the result. If, however, two operands of different sign are to be added, then at least one of the factors $\left|\frac{x}{x+y}\right|, \left|\frac{y}{x+y}\right|$ is bigger than 1, and at least one of the relative errors ϵ_x, ϵ_y will be amplified. This amplification is drastic if $x \approx -y$ holds and therefore cancellation is dangerous.

Generally an algorithm φ comprises a sequence of elementary arithmetic operations. Suppose this chain of computation is denoted as

$$x = x^0 \to \varphi^{(0)}(x^{(0)}) = x^{(1)} \to \ldots \to \varphi^{(r)}(x^{(r)}) = x^{(r+1)} = y.$$

For convenience, we define

$$\psi^{(i)} := \varphi^{(r)} \circ \varphi^{(r-1)} \circ \ldots \circ \varphi^{(i)},\ i = 0, 1 \ldots, r.$$

With floating-point arithmetic, input and roundoff errors will perturb all the intermediate exact results $x^{(i)}$. For the absolute errors

$$\Delta x^{(i)} := \overline{x}^i - x^{(i)},$$

we have

$$\Delta x^{(i+1)} = [fl(\varphi^{(i)}(\overline{x}^i)) - \varphi^{(i)}(\overline{x}^i)] + [\varphi^{(i)}(\overline{x}^i) - \varphi^{(i)}(x^{(i)})].$$

Now by (1.1),

$$\varphi^{(i)}(\overline{x}^i) - \varphi^{(i)}(x^{(i)}) \approx D\varphi^{(i)}(x^{(i)})\Delta x^{(i)}.$$

On the other hand,

$$fl(\varphi^{(i)}(\overline{x}^i)) = rd(\varphi^{(i)}(\overline{x}^i)) = (I + E_{i+1})\varphi^{(i)}(\overline{x}^i)$$

where $E_{i+1} := diag(\delta_1, \ldots, \delta_n\}$ with $|\delta_i| \leq \delta$. It is reasonable to assume that

$$E_{i+1}\varphi^{(i)}(\overline{x}^i) \approx E_{i+1}\varphi^{(i)}(x^{(i)})$$

since the error term $E_{i+1}(\varphi^{(i)}(\overline{x}^i) - \varphi^{(i)}(x^{(i)}))$ is of higher-order error term. Thus we find

$$\Delta x^{(i+1)} \approx E_{i+1}x^{(i+1)} + D\varphi^{(i)}(x^{(i)}\Delta x^{(i)}. \tag{1.3}$$

The quantity $\alpha_{i+1} := E_{i+1}x^{(i+1)}$ can be interpreted as the absolute roundoff error newly created when $\varphi^{(i)}$ is evaluated in floating-point arithmetic. Consequently,

$$
\begin{aligned}
\Delta x^{(1)} &\approx D\varphi^{(0)}(x)\Delta x + E_1 x^{(1)}, \\
\Delta x^{(2)} &\approx D\varphi^{(1)}(x^{(1)}(D\varphi^{(0)}(x)\Delta x + E_1 x^{(1)}) + E_2 x^{(2)}, \\
&\vdots \\
\Delta y &= \Delta x^{(r+1)} \approx D\varphi^{(r)} \dots D\varphi^{(0)}\Delta x + D\varphi^{(r)} \dots D\varphi^{(1)} E_1 x^{(1)} + \dots \\
&= D\varphi(x)\Delta x + D\psi^{(1)}(x^{(1)})E_1 x^{(1)} + \dots + \psi^{(r)}(x^{(r)})E_r x^{(r)} + E_{r+1} y.
\end{aligned}
\tag{1.4}
$$

The formula (1.4) describes the effect of the input error Δx and the roundoff error α_i on the result $y = \varphi(x)$. We should note how the size of the Jacobian matrix $D\varphi^{(i)}$ is critical for the effect of the intermediate roundoff errors in final result.