

CHAPTER 1: PRACTICAL ISSUES IN NUMERICAL METHODS: ERRORS, CONDITIONING AND STABILITY

1.1 INTRODUCTION

When a problem is treated by a numerical procedure (algorithm) on a computer, three requirements are needed in order to get numerically accurate and reliable results from the computer,:

- A 'good', i.e., well-conditioned problem.
- A 'good', i.e., stable numerical procedure.
- A good software for the implementation of the numerical procedure.

1.2 NUMERICAL ERRORS

The concepts of *conditioning* and *stability* are intimately related to numerical errors that are encountered in computations. Therefore in the beginning we give a short overview of the types of errors encountered in numerical applications.

1.2.1 Errors Definitions

When a real number x is approximated or represented by another number \bar{x} , two types of errors can be defined: The absolute error e_a defined by :

$$e_a = |x - \bar{x}| \quad (1.1)$$

and the relative error e_r defined by

$$e_r = \frac{|x - \bar{x}|}{|x|} \quad (1.2)$$

In numerical computations the relative error is more significant because it takes into account the order of magnitude of the value x under consideration. However, when a numerical procedure is used to find the value of a solution, the true value is obviously not known a priori. Since the errors e_a and e_r can not be calculated we can define only an error estimate. For example a number of numerical procedures use an iterative

$$e_a = \frac{|\text{present approximation} - \text{previous approximation}|}{|\text{previous approximation}|} \quad (1.3)$$

Example 1.1: Error in iterative sequence

Consider the following problem:

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right) \quad (1.4)$$

Starting with an initial condition x_0 it can be shown that this sequence converges to \sqrt{a} . Let us use this sequence to compute $\sqrt{9}$ which is 3. Starting from $x_0 = 1$ we compute at each iteration n , the value of x_n , the relative error estimate $e_{rn} = |(x_n - x_{n-1}) / x_{n-1}|$, and, since the true solution ($x = 3$) is known, we also compute the true relative error $e_{tn} = |(x_n - 3) / 3|$. Table 1.1 summarizes the results.

Table 1-1 Iteration for example 1.1

Iteration n	x_n	e_{rn}	e_{tn}
1	5.0000	0.4000E+01	0.6667E+01
2	3.4000	0.3200E+01	0.1333E+01
3	3.0235	0.1107E+01	0.7843E-02
4	3.0001	0.7752E-02	0.3052E-04
5	3.0000	0.3052E-04	0.4657E-09

Note that the error estimate (e_{rn}) is larger than the true error (e_{tn}). The iterations are generally continued until the error estimate falls below a pre-specified tolerance. In the next section we will discuss the types of errors that are encountered in numerical procedures.

1.2.2 Round-Off Errors

Computers are limited precision machines. Consequently numbers are stored in a limited and fixed number of digits that are represented by number of bits, or bytes (group of 8 bits). The rest of digits is ignored and discarded, and this leads to the occurrence of round-off errors. The number of significant figures for each computer depends on its architecture and on the compiler. Personal computers PC, for instance have 32 bits per register and are usually equipped with compilers that provide 7 significant digits under single precision and 15 significant figures under double precision.

Floating point representation

Any real number can be expressed in the following *floating point* form:

$$x = \pm r \times B^n \quad (1.5)$$

The number (r) is called the *mantissa*, the integer (n) is the *exponent* and (B) is the *base* for representation. The value $B = 10$ corresponds to the decimal system we are familiar with. Computers use instead the binary system i.e. $B = 2$. In the representation of Eq (1.5) we can also require that $1/B \leq r < 1$. This is called the *normalized floating point presentation*. Mantissa and exponent occupy a number of bits that depends on the machine architecture. For a 32-bit computer the exponent requires 7 bits. This means that the value of n is bounded by

$$|n| \leq 2^7 - 1 = 127 \quad (1.6)$$

Therefore the machine can handle numbers between $2^{-127} = 10^{-38}$ and $2^{+127} = 10^{+38}$. Smaller and larger numbers produce, respectively, an *underflow* and *overflow*, and the computations are halted.

When computations require better precision, a double-precision is needed. A floating-point number in double precision has a mantissa that has at least twice as many bits. Double precision calculations are however slower than single precision. The mantissa r requires, on the other hand, 24 bits on a 32-bits PC. Since 2^{-24} is roughly

equal to 0.59610^{-7} , this means that the computer have a limited precision of 7 decimal places. The number 2^{-24} is called *unit round-off* for the 32-bits machine.

Many real numbers are what called *machine numbers*. That is they are numbers that can be represented in normalized floating point form with a mantissa r and an exponent n as allowed by the machine architecture. However there are also many real numbers that are not machine numbers, and when attempting to approximate them, the computer introduces inevitable round-off errors. There are two basic ways for approximation: *chopping* and *rounding-up*. In chopping the computer simply discards the excess bits. In rounding-off the computer drops the excess bits as before, but increases the last remaining bit by one unit (10^{-7}). Chopping takes less time to store a number, but is less accurate in general than rounding-up

1.2.3 Truncation Errors

Truncation errors occur whenever an approximation of a mathematical procedure is used instead of the exact one. Truncation errors are made for instance when an infinite sum is replaced by a finite one. Consider, for instance, the Taylor series expansion of $f(x)$ around a point x_0 :

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2!} f''(x_0) + \dots + \frac{h^n}{n!} f^{(n)}(x_0) + \dots \quad (1.7)$$

Let replace this infinite sum by a partial one \tilde{f} that includes terms up to h^n

$$\tilde{f} = f(x_0) + hf'(x_0) + \frac{h^2}{2!} f''(x_0) + \dots + \frac{h^n}{n!} f^{(n)}(x_0)$$

The difference $E = f(x_0 + h) - \tilde{f}$ is a truncation error. The error (E) includes all the terms not included in the approximation,

$$E = \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(x_0) + \frac{h^{n+2}}{(n+2)!} f^{(n+2)}(x_0) + \dots \quad (1.8)$$

The Taylor's theorem provides a way to estimate this error. It can be shown that:

$$E = \frac{f^{(n+1)}(\zeta)h^{n+1}}{(n+1)!} \quad (1.9)$$

where $\zeta \in [x, x + h]$.

The error is usually written in the following form

$$E = O(h^{n+1}) \quad (1.10)$$

which indicates that the truncation error is of order h^{n+1} . A more detailed description of truncation errors is given in later chapters . Contrary to round-off errors which are inevitable and are characteristics of the computer hardware, truncation errors are dependent on the programmer and much of the art of numerical analysis relied upon the manipulation of truncation errors.

Having described the types of errors encountered in numerical computations, we turn our attention to discussing the concepts of the conditioning of a problem and the stability of a numerical method.

1.3 CONDITIONING OF A PROBLEM

A problem is said to be *well-conditioned* if small changes introduced in the problem data gives rise to changes of similar magnitude in the results. If, on the other hand, small changes in the data results in large variations in the solution, then the problem is said to be *ill-conditioned*. It should be noted that the conditioning of the problem is not related to the numerical procedure used to solve the problem . For a number of problems we can define a number called “*condition number*” that quantifies the conditioning of the problem. A problem is ill-conditioned if its condition number is “very large”. In the following we provide some examples of this concept.

Example 1.2: Evaluation of a function

We start with the simple problem of evaluating a real function f at a point x . To check the conditioning of this problem, let us perturb slightly the point x to $x+h$ and observe its effect on the output $f(x)$. The relative error introduced by this perturbation is

$$e = \frac{f(x+h) - f(x)}{f(x)} \quad (1.11)$$

The mean-value theorem allows to write

$$f(x+h) - f(x) = f'(\zeta)h \quad (1.12)$$

where $\zeta \in [x, x+h]$. For small perturbations (h), we can approximate $f(\zeta)$ by $f(x)$, and the relative error becomes:

$$e = \frac{hf'(x)}{f(x)} = \frac{xf'(x)}{f(x)} \frac{h}{x} \quad (1.13)$$

The term h/x represents the relative size of the perturbation i.e. $((x+h)-x)/x$. Thus the term:

$$\frac{xf'(x)}{f(x)} \quad (1.14)$$

can be defined as the condition number of the problem. Note that it is independent of the perturbation h . As an application, let us evaluate the condition number for the following function:

$$f(x) = \cos^{-1}(x) \quad (1.15)$$

Since the derivative is given by $f'(x) = \frac{-1}{\sqrt{1-x^2}}$, the condition number is therefore:

$$\frac{xf'(x)}{f(x)} = \frac{-x}{\cos^{-1}(x)\sqrt{1-x^2}} \quad (1.16)$$

It can be seen that the condition number becomes infinite when it is desired to evaluate $f(x)$ for values of x near 1, and the problem is therefore ill-conditioned.

Example 1.3: Linear system

Consider the following system of linear equations:

$$0.78x_1 + 0.563x_2 = 0.217 \quad (1.17)$$

$$0.913x_1 + 0.659x_2 = 0.254 \quad (1.18)$$

It can be checked that the true solution of this system is:

$$x_1 = 1 \text{ and } x_2 = -1 . \quad (1.19)$$

To check the conditioning of this problem, a small perturbation is added to the system as follows:

$$(0.78 + 0.001)x_1 + (0.563 + 0.001)x_2 = 0.217 \quad (1.20)$$

$$(0.913 - 0.002)x_1 + (0.659 - 0.001)x_2 = 0.254 \quad (1.21)$$

The system becomes:

$$0.781x_1 + 0.564x_2 = 0.217 \quad (1.22)$$

$$0.911x_1 + 0.658x_2 = 0.254 \quad (1.23)$$

It can be checked that the solution of the new system is $x_1 = -5.0$ and $x_2 = 7.3085$. Notice how small changes in the data resulted in significant changes in the solution. The problem is therefore ill-conditioned. A number of methods are presented in later chapters to quantify the conditioning of matrices involved in linear systems calculations.

1.4 STABILITY OF AN ALGORITHM

When a numerical procedure is used to solve a given problem, attention should be given to the stability of the procedure. A numerical algorithm is said to be stable if small errors made at one stage of the process are not magnified in later stages. Otherwise the numerical procedure is said to be unstable. It should be noted that an

unstable algorithm can produce poor solutions even to well-conditioned problems. In the following some examples are provided.

Example 1.4: Stability of sequence

Consider the following sequence of real numbers

$$x_0 = 1; x_1 = \frac{1}{5} \tag{1.24}$$

$$x_{n+1} = \frac{31}{5}x_n - \frac{6}{5}x_{n-1} \tag{1.25}$$

First we prove that the sequence of numbers generate the series

$$x_n = \left(\frac{1}{5}\right)^n \tag{1.26}$$

Equation (1.25) is true for $n = 0$ and $n = 1$. Assume it is valid up to n and let check the validity for $n + 1$.

$$x_{n+1} = \frac{31}{5}x_n - \frac{6}{5}x_{n-1} = \frac{31}{5}\left(\frac{1}{5}\right)^n - \frac{6}{5}\left(\frac{1}{5}\right)^{n-1} \tag{1.27}$$

Equivalently

$$x_{n+1} = \left(\frac{1}{5}\right)^{n-1} \left(\frac{31}{25} - \frac{6}{5}\right) = \left(\frac{1}{5}\right)^{n+1} \tag{1.28}$$

which concludes the proof. The following table (Table 1.2) shows the results for some numbers generated through the scheme (Eq 1.25) and through the direct formula (Eq 1.26).

It can be observed that the relative error for x_{10} is of order 10^5 . The algorithm is therefore unstable. The instability of the procedure can be seen by the fact that any

error present in x_n is multiplied in Eq. (1.25) by $31/5$ which is larger than 1. These errors propagate from x_2, x_3, \dots up to x_{10} .

Table 1-2: Iterations for Example 1.4

n	x_n Eq(1.25)	x_n Eq(1.26)	<i>Relative error</i>
2	0.40000E-01	0.40000E-01	0.0000E+00
3	0.80001E-02	0.80000E-02	0.13825E-04
4	0.16007E-02	0.16000E-02	0.41608E-03
5	0.32400E-03	0.32000E-03	0.12486E-01
6	0.87973E-04	0.64000E-04	0.37458E+00
7	0.15664E-03	0.12800E-04	0.11238E+02
8	0.86560E-03	0.25600E-05	0.33713E+03
9	0.51788E-02	0.51200E-06	0.10114E+05
10	0.31070E-01	0.10240E-06	0.30341E+06

Example 1.5: Stability of a linear systems solution procedure

Consider the solution of the following linear system

$$0.0001x_1 + x_2 = 1.0 \tag{1.29}$$

$$x_1 - x_2 = 0.0 \tag{1.30}$$

It can be checked that the true solution of the system can be approximated well by

$$x_1 = 1 \text{ and } x_2 = 1 \tag{1.31}$$

One way to solve the problem numerically is by eliminating one variable from one equation, i.e. the so-called Gauss elimination method with no pivoting to be discussed in a later chapter.

Let us multiply row 1 by 10000 and subtracting it from row 2. This results in the following system

$$0.0001x_1 + x_2 = 1.0 \tag{1.32}$$

$$0x_1 - 10000 x_2 = -10000 \quad (1.33)$$

Note that the coefficient multiplying x_2 in the second equation should be -10001 but because of round off errors becomes -10000 . We compute from the second equation $x_2 = 1.0$, which a good approximation of the true solution but when substituting x_1 we get

$$0.0001x_1 = 1.0 - (0.0001)(10000) \quad (1.34)$$

which yields

$$x_1 = 0.000 \quad (1.35)$$

which is a value far from the true solution. The numerical procedure is therefore unstable although the problem can be proved to be quite well-conditioned.

