NEERAJ

COMPUTER-ORIENTED NUMERICAL TECHNIQUES

By:
Ranveer, M.C.A.

Reference Book Cum Question Bank

New Edition

NEERAJ PUBLICATIONS
(Publishers of Educational Books)
(An ISO 9001 : 2008 Certified Company)
1507, Ist Floor, NAI SARAK, DELHI - 110 006
Ph. 011-23260329, 69414105 Fax 011-23285501
E-mail: info@neerajbooks.com
Website: www.neerajbooks.com

Price
₹ 160/-

Published by:

NEERAJ PUBLICATIONS

Admn. Office : **Delhi-110 007**
Sales Office : **1507, 1st Floor, Nai Sarak, Delhi-110 006**
E-mail: **info@neerajbooks.com** *Website*: **www.neerajbooks.com**

	Typesetting by: *Competent Computers*	*Printed at*: *Novelty Printers*

Notes:

1. *For the best & upto-date study & results, please prefer the recommended textbooks/study material only.*
2. *This book is just a Guide Book/Reference Book published by NEERAJ PUBLICATIONS based on the suggested syllabus by a particular Board /University.*
3. *The information and data etc. given in this Book are from the best of the data arranged by the Author, but for the complete and upto-date information and data etc. see the Govt. of India Publications/textbooks recommended by the Board/University.*
4. *Publisher is not responsible for any omission or error though every care has been taken while preparing, printing, composing and proof reading of the Book. As all the Composing, Printing, Publishing and Proof Reading etc. are done by Human only and chances of Human Error could not be denied. If any reader is not satisfied, then he is requested not to buy this book.*
5. *In case of any dispute whatsoever the maximum anybody can claim against NEERAJ PUBLICATIONS is just for the price of the Book.*
6. *If anyone finds any mistake or error in this Book, he is requested to inform the Publisher, so that the same could be rectified and he would be provided the rectified Book free of cost.*
7. *The number of questions in NEERAJ study materials are indicative of general scope and design of the question paper.*
8. *Subject to Delhi Jurisdiction only.*

How to get Books by V.P.P.

If you want to buy **NEERAJ® BOOKS** through V.P.P. then you can order the same at the Online Order Form of **www.neerajbooks.com** or send us an E-mail at **info@neerajbooks.com** or you may send us a Fax at **011-23285501** or write us a letter alongwith Rs. 100/- as advance by Bank Draft mentioning your complete requirement, class, course, subject code, medium, name, address and telephone no. which will help us in sending your required **NEERAJ® BOOKS** to you at the earliest.

23260329
69414105
Fax: 011-23285501

NEERAJ PUBLICATIONS

(Publishers of Educational Books)
(An ISO 9001 : 2008 Certified Company)
1507, 1st Floor, NAI SARAK, DELHI - 110 006
E-mail: **info@neerajbooks.com** *Website:* **www.neerajbooks.com**

CONTENTS

COMPUTER-ORIENTED NUMERICAL TECHNIQUES

■■

Sample Preview of The Chapter

Published by:

NEERAJ PUBLICATIONS

www.neerajbooks.com

COMPUTER-ORIENTED NUMERICAL TECHNIQUES

COMPUTER ARITHMETIC AND SOLUTION OF LINEAR AND NON-LINEAR EQUATIONS

Computer Arithmetic

1

INTRODUCTION

Computer arithmetic is the mathematical theory which underlines the way of calculational machines operate on integer numbers.

Computers manipulate *integer numbers* of a finite precision, internally represented as strings of bits of fixed length. A processor's hardware [Braun, 1963] is built to perform additions, multiplications, and other standard arithmetical operations along with *logical* operations like "or", "and", "not", "exclusive or" and so on.

The distinguishing features of computer arithmetic are:

- *Logical* operations, i.e. the ability to calculate bit per bit the conjunction, disjunction and negation of *integers*. For clarity, since we deal with a logical theory, we will refer to these operations with the adjective *bitwise*;
- Fixed precision, which means every representable number lies in a fixed, well-defined range of values and every operation must signal exceptions, when unable to provide a result which fits into that range, usually by means of carry and /or overflows bits.

The main features of a computer which influence the formulation of algorithms are:

(i) The algorithm is stored in the memory of the computer. This facilitates the repetitive execution of instructions.

(ii) Results of computation may be stored in the memory and retrieved when necessary for further computation.

(iii) The sequence of execution of instructions may be altered based on the results of computation. The facility is to test the sign of a number or test if it is zero coupled with the presence of the entire algorithm in the computer's memory enables alternate routes to be taken during the execution of the algorithm.

(iv) The computer has the capability to perform only the basic arithmetic operations of addition, subtraction, multiplication and division. In formulating algorithms all other mathematical operations should be reduced to these basic operations.

To summarise, in order to solve a mathematical problem (like say the solution of differential equations) on a computer, a step-by-step procedure utilizing the above characteristics of a computer should be evolved. In particular, it should be observed that only the elementary arithmetic operations may be used even when solving problems involving the operations of calculus (like differentiation and integration). Formulation of such algorithms is the main subject-matter of *numerical analysis*.

By the end of this chapter we will be able to define computer arithmetic includes: Floating Point Arithmetic and Errors, Floating Point Representation of Numbers, Sources of Errors, Non-Associativity of Arithmetic,

propagated errors, some pitfalls in computation, loss of significant digits, instability of algorithms.

CHAPTER AT A GLANCE

FLOATING POINT ARITHMETIC AND ERRORS

Here we first start with floating point representation of numbers.

Floating Point Representation of Numbers

In general, we are using two types of numbers in calculations:

(a) Integers: 1, …–3, –2, –1, 0, 1, 2, 3,….. and

(b) Other real numbers, such as numbers with decimal point.

Scientists and engineers have developed a compact notation for writing very large or very small numbers. If we wrote it out, the mass of the sun in grams would be a two followed by 33 zeroes. The speed of light in metres per second would be a three followed by eight zeroes. These same numbers, when expressed in scientific notation are 2×10^{33} and 3×10^{8}. Any number n can be expressed as

$$n = f \times 10^{e}$$

where f is a fraction and e is an exponent. Both f and e may be negative. If f is negative the number n is negative. If e is negative, the number is less than one.

The essential idea of scientific notation is to separate the significant digits of a number from its magnitude. The number of significant digits is determined by the size of f and the range of magnitude is determined by the size of e.

We wrote the speed of light as 3×10^{8} metres per second. If that is not precise enough, we can write 2.997×10^{8} to express the same number with four digits of precision.

Floating Point Numbers

Floating point number systems apply this same idea separating the significant digits of a number from its magnitude – to representing numbers in computer systems.

Relatively small numbers for the fraction and exponent part provide a way to represent a very wide range with acceptable precision.

An n-digit floating point number in base β (a given natural number), has the form $x = \pm (.d_1 d_2 \ldots d_n)_\beta \, \beta^e$, $0 \le d_i < \beta$, $m \le e \le M$; I = 1, 2, …n, $d_1 \ne 0$; where, $(.d_1 d_2 \ldots d_n)_\beta$ is a β-fraction called mantissa and its value is given by $(.d_1 d_2 \ldots d_n)_\beta = d_1 \times 1/\beta + d_2 \times 1/\beta^2 + + d_n \times 1/\beta^n$; e is an integer called exponent.

The exponent e is also limited to range $m < e < M$, where m and M are integers varying from computer to computer. Usually, $m = -M$.

In IBM 1130, $m = -128$ (in binary), –39 (decimal) and M = 127 (in binary), 38 (in decimal). For most of the computers $\beta = 2$(binary), on same computers $\beta = 16$ (hexadecimal) and in pocket calculators $\beta = 10$ (decimal).

Normalized Numbers

We represented the speed of light as 2.997×10^{8}. We could also have written 0.2997×10^{9} or 0.02997×10^{10}. We can move the decimal point to the left, adding zeroes as necessary, by increasing the exponent by one for each place the decimal point is moved. Similarly, we can compensate for moving the decimal point to the right by decreasing the exponent. However, if we are dealing with a fixed size fraction part, as in a computer implementation, leading zeroes in the fraction part cost precision. If we were limited to four digits of fraction, the last example would become 0.0299×10^{10}, a cost of one digit of precision. The same problem can occur in binary fractions. In order to preserve as many significant digits as possible, floating point numbers are stored such that the leftmost digit of the fraction part is non-zero. If, after a calculation, the leftmost digit is not significant (i.e. it is zero), the fraction is shifted left and the exponent decreased by one until a significant digit, for binary numbers, a one, is present in the leftmost digit. A floating point number in that form is called a **normalized** number. There are many possible unnormalized forms for a number, but only one normalized form.

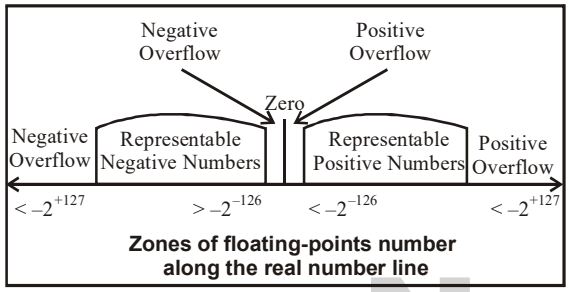
REPRESENTATION OF REAL NUMBERS IN THE COMPUTERS

Although the range of a single precision floating point number is $\pm 10^{-38}$ to $\pm 10^{38}$, it is important to remember that there are still only 2^{32} distinct values. The floating point system cannot represent every possible real number. Instead, it approximates the real numbers by a series of points. If the result of a calculation is not one of the numbers that can be represented exactly, what is stored is the nearest number that can be represented. This process is called **rounding**, and it introduces error in floating point calculations. Since rounding down is as likely as rounding up, the cumulative effect of rounding error is generally negligible.

The spacing between floating point numbers is not constant. Clearly, the difference between 0.10×2^{1} and 0.11×2^{1} is far less than the difference between 0.10×2^{127} and 0.11×2^{127}. If the difference between

numbers is expressed as a percentage of the number, the distances are similar throughout the range, and the relative error due to rounding is about the same for small numbers as for large.

Not only cannot all real numbers be expressed exactly, there are whole ranges of numbers that cannot be represented. Consider the real number line as shown below in Figure. The number zero can be represented exactly because it is defined by the standard. The positive numbers that can be represented fall approximately in the range 2^{-126} to 2^{+127}.



Numbers greater than 2^{+127} cannot be represented; this is called **positive overflow**. A similar range of negative numbers can be represented. Numbers to the left of that range cannot be represented; this is **negative overflow**. There is also a range of numbers near zero that cannot be represented.

The smallest positive number that can be represented in normalized form is 1.0×2^{-126}. The condition of trying to represent smaller numbers is called **positive underflow**. The same condition on the negative side of zero is called **negative underflow**.

Rounding and Chopping of a number and Associated Errors.

Rounding: A number x is rounded to t digits when x is replaced by a t digit number that approximates x with minimum error.

Example: $t = 5$, $x = 2.5873892874$ then rounding will be 2.5874.

Chopping: A number is *chopped* to t digits and all the digits past t are discarded.

Example: $t = 5$, $x = 2.5873892874$ then chopping will be 2.5874.

Overflow occurs when the result of a floating point operation is larger than the largest floating point number in the given floating point number system.

When this occurs, almost all computers will signal an error message.

Underflow occurs when the result of a computation is smaller than the smallest quantity the computer can store.

Some computers don't see this error because the machine sets the number to zero.

Relative Error: Let $fl(x)$ be floating point representation of real number x. Then $e_x = |x - fl(x)|$ is called round-off (absolute error).

$$r_x = \frac{x - fl(x)}{x} \text{ is called the relative error.}$$

Theorem: If $fl(x)$ is the n-digit floating point representation in base β of a real number x, then r_x the relative error in x satisfies the following:

(*i*) $|r_x| < 1/2\ \beta^{1-n}$ if rounding is used

(*ii*) $0 = |r_x| = \beta^{1-n}$ if chopping is used

Proof: Case 1. $d_{n+1} < \frac{1}{2}\beta$, then $fl(x) = \pm (.d_1 d_2 \ldots d_n)\beta^e$

$$|x - fl(x)| = d_{n+1}, d_{n+2} \ldots\ldots \beta^{e-n-1}$$
$$\leq \tfrac{1}{2}\,\beta.\beta^{e-n-1} = \tfrac{1}{2}\,\beta^{e-n}.$$

Case 2. $d_{n+1} < \frac{1}{2}\beta$, then $fl(x)$

$$= \pm \{(.d_1 d_2 \ldots . d_n)\beta^e + \beta^{e-n}\}$$
$$|x - fl(x)| = |-d_{n+1}, d_{n+2}.\beta^{e-n-1} + \beta^{e-n}|$$
$$= \beta^{e-n-1}|d_{+1}.d_{n+2} - \beta|$$
$$\leq \beta^{e-n-1} \times 1/2\ \beta = \tfrac{1}{2}\,\beta^{e-n}.$$

Associated Errors: In scientific computing, we never expect to get the exact answer. In exactness is practically the definition of scientific computing. Getting the exact answer, generally with integers or rational numbers, is symbolic computing, an interesting but distinct subject. Suppose we are trying to compute the number A. The computer will produce an approximation, which we call $\hat{A}$. This $\hat{A}$ may agree with A to 16 decimal places, but the identity $A = \hat{A}$ (almost) never is true in the mathematical sense, if only because the computer does not have an exact representation for A. For example, if we need to find x that satisfies the equation $x^2 - 175 = 0$, we might get 13 or 13.22876, depending on the computational method, but $\sqrt{175}$ cannot be represented exactly as a floating point number.

Four primary sources of error are:

(i) Round off error,

(ii) truncation error,

(iii) termination of iterations, and

(iv) statistical error in Monte Carlo.

We will estimate the sizes of these errors, either a priori from what we know in advance about the solution, or a posteriori from the computed (approximate) solutions themselves. Software development requires

distinguishing these errors from those caused by outright bugs. In fact, the bug may not be that a formula is wrong in a mathematical sense, but that an approximation is not accurate enough.

Scientific computing is shaped by the fact that nothing is exact. A mathematical formula that would give the exact answer with exact inputs might not be robust enough to give an approximate answer with (inevitably) approximate inputs. Individual errors that were small at the source might combine and grow in the steps of a long computation. Such a method is unstable. A problem is ill conditioned if any computational method for it is unstable. Stability theory, which is modelling and analysis of error growth, is an important part of scientific computing.

TRUNCATION ERRORS

Truncation errors are defined as those errors that result from using an approximation in place of an exact mathematical procedure. Truncation error results from terminating after a finite number of terms known as formula truncation error or simply truncation error.

Let a function $f(x)$ is infinitely differentiable in an interval which includes the point $x = a$. Then the Taylor series expansion of $f(x)$ about $x = a$ is given by

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)(x-a)^k}{k!} \quad ...(i)$$

where $f^{(k)}(a)$ denotes the kth derivative of $f(x)$ evaluated at $x = a$

$$f^{(k)}(a) \equiv \frac{d^k f(x)}{dx^k} x = 0 \quad ... (ii)$$

If the series is truncated after n terms, then it is equivalent to approximating $f(x)$ with a polynomial of degree n–1.

$$f_n(x) \equiv \sum_{k=0}^{n-1} \frac{f^{(k)}(a)(x-a)^k}{k!} \quad ... (iii)$$

The error in approximating $E_n(x)$ is equal to the sum of the neglected higher order terms and is often called the tail of the series. The tail is given by

$$E_n(x) \equiv f(x) - f_n(x) = \frac{f^{(x)}(\xi)(x-a)^n}{n!} \quad ... (iv)$$

It is possible sometimes to place an upper bound on the x of $E_n(x)$ depending on the nature of function $f(x)$.

If the maximum value of $|f_n(x)|$ over the interval $[a, x]$ is known or can be estimated, then

$$M_n(x) \equiv \max_{a \le \xi \le x} \left[\left| f^{(n)}(\xi) \right| \right] \quad ... (v)$$

From Eqs. (*iv*) and (*v*), the worst bound on the size of the truncation error can be written as

$$|E_n(x) \le \frac{M_n(x)|x-a|^n}{n!} \quad ... (vi)$$

If $h = x - a$, then the truncation error $E_n(x)$ is said to be of order O (h^n). In other words, as $h \to 0$, $E_n(x) \to 0$ at the same rate as h^n.

Hence, $O(h^n) \approx ch^n |h| << 1$ ... (vii)

where c is a non-zero constant.

The *total numerical error* is the summation of the truncation and round-off errors. The best way to minimise round-off errors is to increase the number of significant figures of the computer. It should be noted here that round-off error increases due to subtractive cancellation or due to an increase in the number of computations in an analysis. The truncation error can be reduced by decreasing the step size. In general, the truncation errors are *decreased* as the round-off errors are *increased* in numerical differentiation.

There exists no systematic and general approaches in evaluating numerical errors for all problems. In most cases, error estimates are based on experience and judgement of the engineer or scientist.

Model errors relate to bias that can be ascribed to incomplete mathematical models. Errors also enter into the analysis due to uncertainty in the physical data on which a model is based.

Example: Given the trigonometric function $f(x) = \sin x$,

(a) expand $f(x)$ about $x = 0$ using Taylor series

(b) truncate the series to $n = 6$ terms

(c) find the relative error at $x = \pi/4$ due to truncation in (*b*)

(d) determine the upper bound on the magnitude of the relative error at $x = \pi/4$ and express it as a per cent.

Sol.

(a) Using Eq. $\frac{X_E - X_A}{|X_E|} < 5 \times 10^{-t}$, the Taylor series expansion is given by