COMPUTER ORIENTED NUMERICAL METHODS

THIRD EDITION

V. RAJARAMAN

Honorary Professor,
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore

Prentice-Hall of India Private Limited
New Delhi - 110 001
2006

Rs. 95.00

COMPUTER ORIENTED NUMERICAL METHODS, 3rd Ed.
by V. Rajaraman

ISBN-81-203-0786-0

The export rights of this book are vested solely with the publisher.

Thirty-sixth Printing (Third Edition) ... ... September, 2006

Published by Asoke K. Ghosh, Prentice-Hall of India Private Limited, M-97, Connaught Circus, New Delhi-110001 and Printed by Jay Print Pack Private Limited, New Delhi-110015.

Contents

Preface to the Third Edition

The widespread use of digital computers in recent years in engineering design and scientific research has made the study of numerical methods as important as the study of calculus for engineers and scientists. Thus a number of universities have introduced a course on numerical methods and computer programming in the undergraduate science and engineering curricula. Engineers and scientists trained in the pre-computer days also need to be educated in this area. This book is intended for both these classes of students. It is a concise presentation of the basic concepts used in evolving numerical methods with special emphasis on developing computational algorithms for solving problems in algebra and calculus. It is written for students who have taken a first course in differential and integral calculus.

This book is not a handbook or a cookbook of recipes to solve problems in numerical analysis. The approach is to ensure conceptual understanding of numerical methods by relying on students' geometric intuition. The presentation assumes that the methods developed would provide a basis for evolving algorithms for implementation on a digital computer. Thus, the book begins with a discussion of the concept of stored program computers and the consequent need to formulate algorithms for problem solving. Formulation of algorithm is illustrated with a number of examples and an algorithmic language based on English (and similar to PASCAL) is used to express the numerical procedures. This approach is different from those used in most books which either use a programming language like FORTRAN or use flow charts to express algorithms. From the point of view of the student the approach where FORTRAN is used leads to the mixing up of two sets of concepts: one relating to the syntax of the FORTRAN language and the other relating to the logic of numerical algorithms. Further there is a tendency among students to 'lift' the FORTRAN programs from the book in situations where they are not suitable. Flow charts consume considerable space without a commensurate improvement in clarity. Very often the flow charts presented are a translation of FORTRAN programs using variable names, etc., relevant to FORTRAN.

The chapter on computer algorithms is followed by one on computer arithmetic. A good understanding of floating-point arithmetic used in computers is essential to appreciate problems of rounding in numerical calculations. The basis for appreciating the difference in approach needed between desk calculator oriented

numerical methods and those oriented towards the use of computers is laid in these two chapters.

This is followed by a chapter on iterative methods for solving algebraic and transcendental equations and another on direct and iterative methods of solving linear simultaneous algebraic equations. In both these chapters some algorithms are presented to illustrate the mental framework necessary when computer oriented procedures are evolved and to bring into focus the trade-offs between programming ease, computational time, computer storage, truncation and rounding errors.

A chapter on interpolation follows this chapter. This chapter in included not because of its relevance to computers (in fact classical interpolation is a rare operation when computers are used) but due to the conceptual foundation it lays for much of numerical calculus. In this third edition a section on cubic splines has been introduced. The problem of evolving algorithms to fit a curve to a set of points is discussed next. The least squares regression method is discussed in this chapter.

Computer-oriented interpolation which relies on approximation of functions by power series is taken up next. In this chapter also the emphasis is on building a conceptual understanding. Neither a detailed derivation of results nor a listing of approximation formulae is given.

The next chapter introduces numerical methods for differentiation and integration. The last chapter discusses the solution of ordinary differential equations with initial conditions. Both Runge-Kutta and predictor-corrector methods are presented. No attempt has been made to discuss boundary value problems with ordinary and partial differential equations. This decision was taken as the book is only a brief introduction to numerical methods and further because the readers are required to know only elementary differential and integral calculus to understand the book.

The first edition of this book was published in 1971 and the second edition in 1980. The book has now been thoroughly revised based on feedback from readers. In this edition all the chapters have been rewritten and more sections have been added. All algorithms have been rewritten in a PASCAL like language.

The author would like to express his thanks to Mr. K.V. Nori for help in improving the manuscript and Dr. C.N.R. Rao, Director, Indian Institute of Science, Bangalore, who has fostered the excellent academic climate at Bangalore which made this endeavour possible.

A number of people helped in the preparation of the manuscript for printing. Thanks are due to Ms. Mallika for typing corrections to the third edition and to Mr. Kamal K. Jain for solving some of the exercises. Thanks are due to the entire staff of the Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, for continued cooperation.

The author would finally like to express his affectionate appreciation to his wife Dharma who proof-read and corrected the manuscript and cheerfully devoted herself to the book-writing project.

V. RAJARAMAN

1

Computational Algorithms

1.1 Introduction

Assume that a set of instructions are to be given to a schoolboy to solve the following pair of equations for x and y given the values of a, b, c, p, q and r.

$$ax + by = c \tag{1.1}$$

$$px + qy = r. \tag{1.2}$$

Assume that a, b, c, p, q and r are non-zero. We have from Eq. (1.1)

$$x = (c - by)/a. \tag{1.3}$$

Substituting x in Eq. (1.2)

$$p(c - by)/a + qy = r$$

or

$$\left(q - \frac{pb}{a}\right)y = r - \frac{pc}{a}. \tag{1.4}$$

Thus if we compute $q - (pb/a)$ and *replace* the value of q by it and compute $r - (pc/a)$ and *replace* the value of r by it we have

$$qy = r \quad \text{or} \quad y = r/q$$

We can now substitute the value of y in Eq. (1.3) and find x. This method of solving the equations may be expressed as the following series of instructions.

Instruction 1: Read six numbers and assign them respectively to

$$a, b, c, p, q \text{ and } r.$$

Instruction 2: Compute $q - (pb/a)$. Replace the value of q by this value.
Instruction 3: Compute $r - (pc/a)$. Replace the value of r by it.
Instruction 4: If $q = 0$ then write 'No Solution' and stop. Else continue.
Instruction 5: Set $y = (r/q)$.
Instruction 6: Set $x = (c - by)/a$.
Instruction 7: Write the values of x and y as answers.
Instruction 8: Stop.

If a, b, c, p, q and r are respectively 2, 3, 4, 5, 6 and 7 then the following computations would be performed when the instructions are executed.

Instruction 1: $a\ b\ c\ p\ q\ r$
$2\ 3\ 4\ 5\ 6\ 7$
Instruction 2: $q \leftarrow 6 - (15/2) = -1.5$
Instruction 3: $r \leftarrow 7 - (20/2) = -3.$
Instruction 4: As $q \neq 0$ continue.
Instruction 5: $y = (-3/-1.5) = 2.$
Instruction 6: $x = (4 - 6)/2 = -1.$
Instruction 7: $x = -1, y = 2$ answers.
Instruction 8: Stop.

Observe that when instruction 2 is executed the previous value of q namely 6 is replaced by a new value, namely, -1.5. Similarly when instruction 3 is executed the value of r (namely, 7) is replaced by a new value -3. The values of the variables at the beginning and at the end of execution are given below:

a	b	c	p	q	r	x	y
2	3	4	5	6	7	—	—
2	3	4	5	−1.5	−3	−1	2

A digital computer is analogous to the schoolboy in the above example. It is necessary to give a detailed set of instructions to the computer for solving a problem. Such a set of instructions which leads to a step by step *procedure* for solving a problem is called an *algorithm*. Unlike a schoolboy a computer (as of today) does not have the capability of reading and understanding instructions written in a natural language like English. Thus it is necessary to express the algorithm in a language understood by the computer. An algorithm coded in a computer language is called a *program* and the language used for coding is called a *programming language*.

The algorithm given earlier in English is coded below in a programming language called BASIC.

```
 1  READ A, B, C, P, Q, R
 2  LET Q = Q - (P * B/A)
 3  LET R = R - (P * C/A)
 4  IF Q = 0 THEN 9
 5  LET Y = R/Q
 6  LET X = (C - B * Y)/A
 7  PRINT X, Y, "ANSWERS"
 8  STOP
 9  PRINT "NO SOLUTION"
10  STOP
11  DATA 2, 3, 4, 5, 6, 7
12  END
```

The idea in writing the above program is to illustrate the ease of coding a set of instructions in a programming language. The student is urged to compare the above program with the procedure given in English earlier. (The symbol * in the above program stands for the multiplication operator.)

1.2 The Structure of a Computer

In order to execute programs a computer has a number of interconnected units (see Figure 1.1). The function of each of these units is explained below taking the program written above as an illustration.

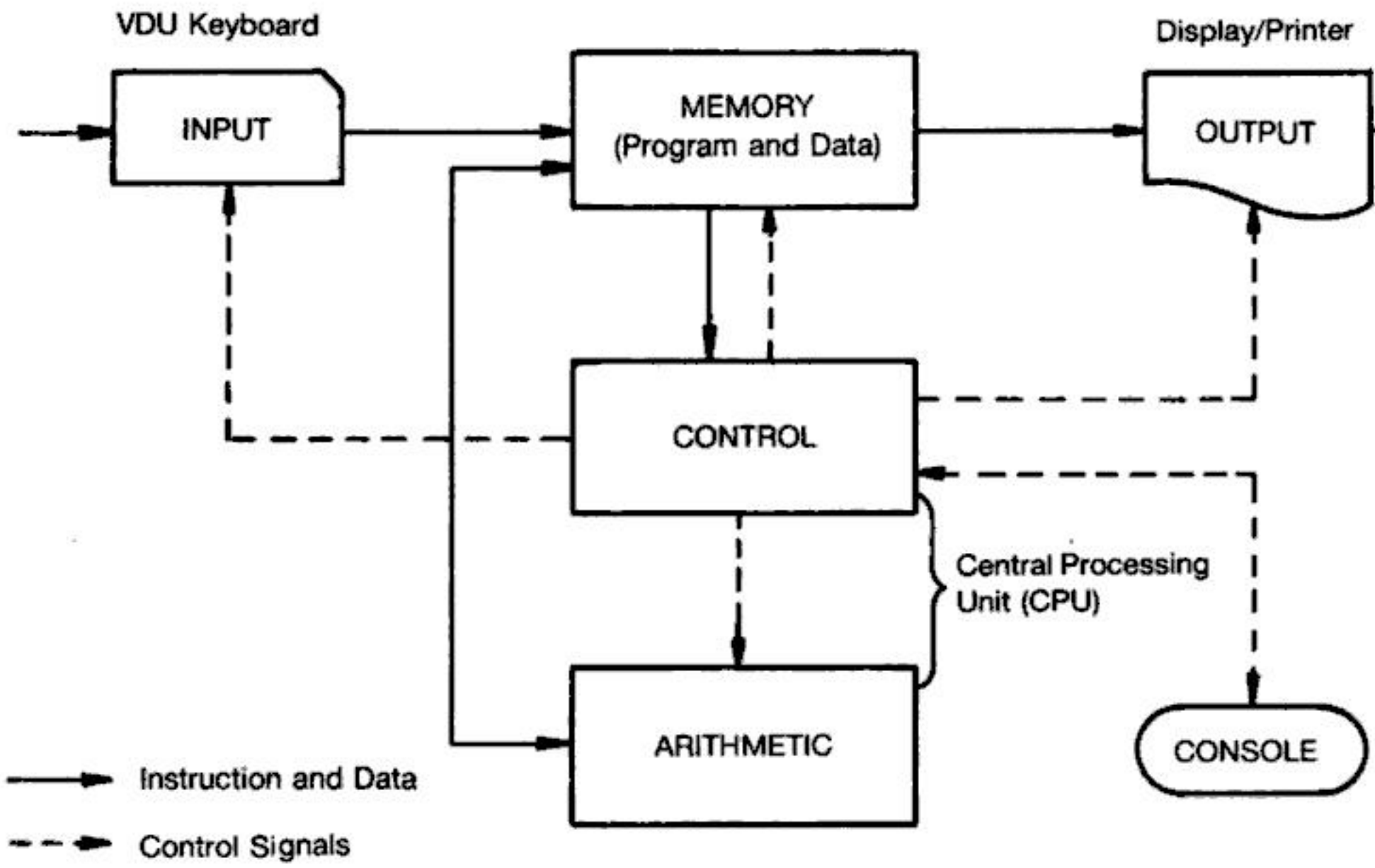

Fig. 1.1 Block diagram of a computer.

The program is typed using a keyboard attached to a video display unit (VDU) which is an *input unit* (Figure 1.1). From the input unit the program is transferred to the *memory* of the computer. *Execution* of the program is initiated after the entire program is stored in the memory. To begin execution the first instruction is decoded by the *control unit*. This unit activates the appropriate units and supervises the execution of the instruction. In the above BASIC program the first instruction commands that six numbers be read as data and assigned to A, B, C, P, Q and R. The memory of the computer may be conceptually thought of as a collection of boxes with each box having a distinct *label* or *address*. A box in a memory is also known as a memory location. The READ instruction would take six boxes or locations in memory and assign respectively the labels, A, B, C, P, Q, R to them. The six numbers 2, 3, 4, 5, 6 and 7 will then be stored in these six locations in memory.

After the first instruction is executed the control unit fetches the next instruction in the program and decodes it. In this example the second instruction commands that the number in memory location Q be replaced by $Q-(P * B/A)$. To do this the number in P is read (or *copied*) from the memory into the *arithmetic unit* and temporarily stored in an *accumulator*. It is then multiplied by the number contained in B. The product is then divided by the number contained in A and the (intermediate) result stored in a register. The contents of Q is then copied into the accumulator and the intermediate result is subtracted from it. The final result is then taken back to the memory location with label Q and stored there. The previous number contained in Q is thus *replaced by* this new number and as a consequence the old value in Q is lost. It should be noted that all arithmetic operations, namely, addition, subtraction, multiplication and division, are performed in the arithmetic unit under the overall control of the control unit.

After the execution of the second instruction the control unit fetches the third instruction. This instruction when executed leads to the replacement of the number in R by a new number obtained by calculating $R-(P * C)/A$.

The fourth instruction commands the control unit to fetch the contents of memory location Q and check if it is zero. If it is zero the next instruction to be executed is instruction 9. If the contents of Q is not zero then the next sequential instruction, namely, instruction 5 is to be executed. Instruction 4 which alters the normal sequential order of executing instructions is called a *conditional branch* instruction and endows a decision making capability to a computational procedure.

If Q is not zero then instruction 5 is taken as the next instruction for execution by the control unit. This instruction causes the arithmetic unit to compute the quotient (R/Q) and store it in a memory location with label Y. Observe that Y is a new label which did not appear earlier in the program; thus the control unit acquires a memory location, gives it the name Y and stores the quotient in that location.

The next instruction, in a similar manner, stores the value of $(C-B * Y)/A$ in a memory location with label X.

The seventh instruction commands the control unit to print the values contained in X and Y as answers. This is achieved by activating an *output unit* which is usually a printer or a Video Display Unit.

The next instruction when decoded commands that computation be stopped.

If Q happens to be zero when instruction 4 is decoded by the control unit (this would happen, for example, when A, B, C, P, Q and R are respectively 2, 3, 4, 4, 6, 5) then it commands that instruction 9 be taken up for execution next. Instruction 9 would command the output unit to print 'NO SOLUTION'. After printing this, control unit will fetch the next instruction in sequence (namely, instruction 10) which would command that computation be stopped.

The example considered above illustrates a number of important points. These are:

(i) It is necessary to formulate a step by step procedure, called an *algorithm*, for solving a problem on a computer.

(ii) The algorithm should then be expressed as a set of instructions in a *programming language*. These instructions constitute a program.

(iii) A computer has a number of interconnected units, namely, input, memory, control, arithmetic and output which are designed to facilitate execution of programs. The control unit acts as the overall supervisor during the execution of a program.

(iv) The program is first stored in the memory. Thus all the instructions would be resident in the computer's memory during the execution of a program. This is useful particularly if a set of instructions are to be repeatedly executed.

(v) The instructions in a program are executed sequentially in the order in which they are written. The normal sequential order may be altered by a conditional branch instruction. Such an instruction endows a program with an elementary decision-making capability.

(vi) A set of numbers is required as *data* on which computations are performed. Data are read when the program commands that this be done. A clear distinction between a program and the data on which it operates should be made.

(vii) A computer's memory may be thought of as a collection of labelled boxes. A variable name used in a program is the label of a box which is also called a location in memory. The *value* of the variable is the *contents* of the memory location with the corresponding label. When the value of a variable is read from memory for use in a computation it is preserved for future use (for example in instruction 2 the values of A, B and P are used in a calculation). When a number is stored in memory it replaces the previous contents of that location. (In instruction 2, for example, the previous value in location Q is replaced by a freshly calculated value.)

1.3 Some Examples of Algorithms

Example 1.1

A Fibonnacci sequence of integers is given below:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Observe that a term in the sequence is obtained as the sum of the immediately preceding two terms. Thus we have the recurrence relation.

1. previous term = 0
2. current term = 1

} initial values

3. new term = current term + previous term
4. Replace previous term by current term
5. Replace current term by new term.

Now if we repeat steps 3, 4 and 5 we generate successive terms in the Fibonnacci sequence. This is illustrated in the following table:

Previous term	Current term	New term
0	1	1
1	1	2
1	2	3
2	3	5
3	5	8
5	8	13
8	•	•
•	•	•
•	•	•

If we want to generate the Fibonnacci sequence till the largest number in the sequence is less than or equal to 100 we may write the procedure as the following algorithm.

Algorithm 1.1 Generation of Fibonnacci Sequence

```
1  previous term ← 0
2  current term ← 1
3  new term ← current term + previous term
4  Write previous term, current term, new term
5  While new term ≤ 100 do
       begin
6          previous term ← current term
7          current term ← new term
8          new term ← current term + previous term
9          Write new term
       end
10 Stop
```

In the above algorithm the symbol ← is used as an abbreviation for the statement "the contents of the variable on the left hand side is replaced by the contents of the variable on the right hand side". Observe instruction 5 which commands that all instructions following it enclosed between *begin* and *end* are to be done again and again as long as (while) new term ≤ 100.

Instructions 6 to 9 are enclosed between *begin* and *end*; *begin* and *end* serve the same purpose as brackets. These brackets enable us to readily see

that this group of statements are to be repeatedly executed. When a new term exceeds 100 then instructions 6 to 9 will be skipped and instruction 10 will be carried out. The *while* instruction is a very useful instruction which sets up a repetitive cycle of computations called a *program loop*. This loop is called a *while loop*.

The above algorithm written as a program in Pascal language is given below. Observe their close resemblance. In Pascal also the words *begin* and *end* are used to group statements. All statements are not numbered. Instead of writing Stop one uses END. in Pascal. The algorithms written in this book will be similar to Pascal but our notation will be simpler to enable the student to understand the procedure without his having to memorise syntax rules of a computer language. However, it will be extremely simple to convert algorithms written in this book to equivalent Pascal programs.

```
VAR   PREVTERM, CURRTERM, NEWTERM: INTEGER;
BEGIN
  PREVTERM : = 0;
  CURRTERM: = 1;
  NEWTERM  : = CURRTERM + PREVTERM;
  WRITELN (PREVTERM, CURRTERM, NEWTERM);
  WHILE NEWTERM < = 100 DO
      BEGIN
            PREVTERM : = CURRTERM;
            CURRTERM: = NEWTERM;
            NEWTERM  : = CURRTERM + PREVTERM;
            WRITELN (NEWTERM)
      END
END.
```

Example 1.2

We will now rewrite the set of instructions given in Section 1.1 for solving the simultaneous linear equations in a more concise notation. This notation will be used in the rest of this book.

Algorithm 1.2 Solution of Simultaneous Equations

```
1  Read a, b, c, p, q, r
2  q ← q – (pb/a)
3  r ← r – (pc/a)
4  if q = 0 then
5            Write 'No solution'
             else begin
6                      y ← r/q
7                      x ← (c – by)/a
8                      Write x, y
                 end
9  Stop
```

Observe instruction 4. This is called an 'If then else' instruction. The functioning of this instruction is illustrated by the following diagram (Figure 1.2).

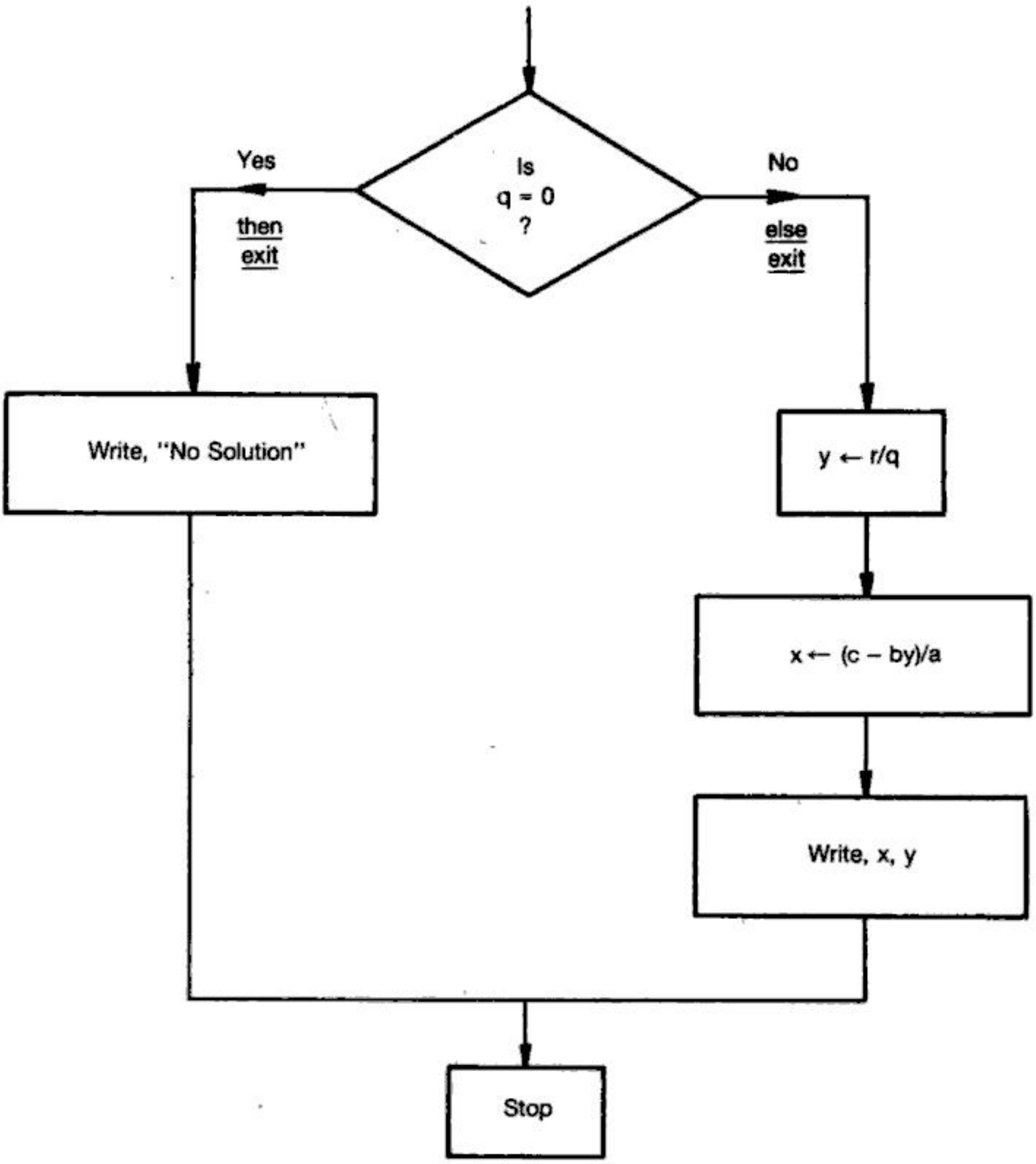

Fig. 1.2 Flow chart depicting if then else.

The indentation in the algorithm writeup is used for better readability. Observe the (brackets) *begin* and *end* used in the *else* clause. All the instructions within these brackets *begin* and *end* are to be executed one after another. In general the *if then else* statement will be of the type:

```
if condition
      then begin
                  instruction
                      ⋮
                  instruction
            end
      else begin
                  instruction
                      ⋮
                  instruction
            end
next instruction.
```

Example 1.3

We will now consider another example to illustrate the formulation of computational algorithms.

Suppose it is required to find the sum of the following series (which is an approximation for sin x)

$$S = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^{n-1} \frac{x^{(2n-1)}}{(2n-1)!} \tag{1.5}$$

The first step in evolving a procedure is to obtain a *recurrence relation* which gives the techniques of developing terms in the series. By inspection of the series the

$$i\text{th term} = (-1)^{i-1} \frac{x^{(2i-1)}}{(2i-1)!} \tag{1.6}$$

and the

$$(i+1)\text{th term} = (-1)^{i} \frac{x^{(2i+1)}}{(2i+1)!} \tag{1.7}$$

Thus the

$$(i+1)\text{th term} = (-1) \frac{x^2}{(2i+1)(2i)} (i\text{th term}) \tag{1.8}$$

The following algorithm implements the above technique to sum the series. We remind the reader that the symbol ← may be broadly interpreted as 'becomes' or 'is replaced by'. Thus $x \leftarrow x + y$ would be interpreted as x *becomes* previous contents of x + contents of y, or x is *replaced by* the previous contents of x + the contents of y.

Algorithm 1.3 Summation of Series

```
1 Read x, n
2 s ← x
3 term ← x
4 for i = 1 to (n − 1) in steps of 1 do
5                 term ← (− x^2) (term)/(2i(2i + 1))
6                 s ← s + term
  endfor
7 Write s
8 Stop
```

The interesting new feature in the above algorithm is instruction 4 which commands that all instructions following it upto *endfor* (upto and including 6) are to be executed $(n - 1)$ times. After instructions 5 and 6 are executed once with $i = 1$, i is to be incremented by 1 and these instructions are executed again. The set of instructions are to be executed finally with $i = (n - 1)$ after which the next instruction in the sequence (namely, instruction 7) is to be taken up for execution. Thus instruction 4 sets up a repetitive cycle of computations, namely, *a program loop*. Such a program loop is called a *for loop*.

The execution of the program loop in the above example may be traced as shown below. Such a trace is necessary to check whether the algorithm indeed does what we intended it to do.

Times in loop	i	*term*	s
1	1	$-x^3/3!$	$x - x^3/3!$
2	2	$+ x^5/5!$	$x - x^3/3! + x^5/5!$
3	3	$-x^7/7!$	$x - x^3/3! + x^5/5! - x^7/7!$
⋮	⋮	⋮	⋮
$n - 1$	$n - 1$	$\frac{(-1)^{n-1}x^{2n-1}}{(2n-1)!}$	$x - x^3/3! + \ldots + \frac{(-1)^{n-1}x^{2n-1}}{(2n-1)!}$

From the table it is seen that the algorithm develops the series as intended. There are some other interesting problems which arise when numbers are substituted for x and n and the algorithm executed on a computer. These problems will be considered later.

In general a *for* loop will be of the type

```
for  index = initial value to final value in steps of increment do
     instruction
          ⋮
     instruction
endfor
     next instruction
```

Example 1.4

In this example we will obtain an algorithm to evaluate the polynomial.

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \tag{1.9}$$

Evaluating this polynomial by a brute force technique is simple but will involve $n(n + 1)/2$ multiplications and n additions. In the algorithm which follows the number of multiplications is reduced to n.

The algorithm is based on the following alternate method of writing the expression.

$$\begin{aligned} p(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \\ &= a_0 + x(a_1 + a_2 x + \ldots + a_n x^{n-1}) \\ &= a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-2} + x(a_{n-1} + xa_n)))) \end{aligned} \tag{1.10}$$

The actual computational algorithm is given below:

Algorithm 1.4 Polynomial Evaluation

```
1 Read x, n
2 for i = 0 to n in steps of 1 do
3         Read a_i
  endfor
4 p ← a_n
5 for i = n to 1 in steps of – 1 do
6    p ← a_{i-1} + xp
  endfor
7 Write p
8 Stop
```

In the above algorithm the *for* loop set up by instruction 2 consists of only one statement, namely, Read a_i. This statement reads the *array* $a_0, a_1, \ldots, a_n$ which is used to store the coefficients of the polynomial. The *for* loop set up by instruction 5 also has only one instruction, namely, instruction 6. The algorithm is traced in the following table:

Times in loop	i	p
1	n	$a_{n-1} + xa_n$
2	$n - 1$	$a_{n-2} + x(a_{n-1} + xa_n)$
3	$n - 2$	$a_{n-3} + x(a_{n-2} + x(a_{n-1} + xa_n))$
⋮	⋮	⋮
n	1	$a_0 + x(a_1 + x(a_2 + \ldots x(a_{n-1} + xa_n)))$

The above algorithm illustrates the fact that in developing procedures for solving even simple problems a large amount of computational effort may be saved by a slight reformulation of the problem.

Example 1.5

As another illustration of the formulation of recurrence relation we will consider the problem of dividing a polynomial

$$a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0 \quad \text{by } (x - r)$$

where r is a root of the original polynomial. The long division method is illustrated below:

$$\begin{array}{r|l} x - r & a_n x^{n-1} + (a_{n-1} + ra_n)x^{n-2} + (a_{n-2} + r(a_{n-1} + ra_n))x^{n-3} + \ldots \\ & \overline{a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0} \\ & \underline{a_n x^n - ra_n x^{n-1}} \\ & \quad (a_{n-1} + ra_n)x^{n-1} + a_{n-2}x^{n-2} \\ & \quad \underline{(a_{n-1} + ra_n)x^{n-1} - r(a_{n-1} + ra_n)x^{n-2}} \\ & \qquad (a_{n-2} + r(a_{n-1} + ra_n))x^{n-2} + \ldots \end{array}$$

If we call the reduced polynomial

$$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \ldots + b_0$$

then equating the coefficients of the above polynomial with the quotient obtained by division above we get

$$\begin{aligned} b_{n-1} &= a_n \\ b_{n-2} &= a_{n-1} + ra_n = a_{n-1} + rb_{n-1} \\ b_{n-3} &= (a_{n-2} + r(a_{n-1} + ra_n)) \\ &= a_{n-2} + rb_{n-2}. \end{aligned}$$

In general

$$b_{n-(i+1)} = a_{n-i} + rb_{n-i}.$$

The above recurrence relation is expressed in the following algorithm:

Algorithm 1.5 Polynomial Division

```
1  for i = 0 to n in steps of 1 do
2          Read a_i
   endfor
3  b_{n-1} ← a_n
4  for i = 1 to (n - 1) in steps of 1 do
5          b_{n-(i+1)} ← a_{n-i} + r b_{n-i}
   endfor
6  for i = 0 to (n - 1) in steps of 1 do
7      Write b_i
   endfor
8  Stop
```

The above algorithm is very useful in reducing polynomials by removing the roots and may be extended to cases where the polynomial is to be divided by a quadratic factor like $x^2 + ax + b$. This is left as an exercise to the student.

References

1. Rajaraman, V., *Computer Programming in Pascal*, Prentice-Hall of India, New Delhi 1983.
2. Pennington, Ralph H., *Introductory Computer Methods and Numerical Analysis*, Collier-McMillan, London, 1966.
3. Beckett, R., and Hurt, J., *Numerical Calculations and Algorithms*, McGraw-Hill, New York,1967.
4. Forsythe, A.I., Keenan, T.A., Organick, E.I., Stenberg, W., *Computer Science: A First Course*, John Wiley, New York, 1969.
5. Wirth, N., *Systematic Programming: An Introduction*, Prentice-Hall, Inc., Englewood Cliffs (N.J.), 1973.

Exercises

1.1 Revise the instructions for solving the simultaneous equations given at the beginning of the chapter for the case when one or more of the coefficients a, b, c, p, q, r are zero.

1.2 In the program for solving simultaneous equations what is the physical significance of $(aq - bp)$ being zero?

1.3 Solve the simultaneous equations with the following values of a, b, c, p, q and r:

$$a = 2.500, \qquad b = 5.200, \qquad c = 6.200$$
$$p = 1.251, \qquad q = 2.605, \qquad r = 3.152$$

Give the answer correct to four decimal digits.

1.4 Obtain an algorithm to find the sum of the following series to the first n terms

(i) $\cos x = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 \ldots$

(ii) $\log_e(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots$

1.5 Obtain an algorithm to compute the sum S

$$S = \frac{1}{px + q} \sum_{i=0}^{n} a_i x^i.$$

1.6 Obtain an algorithm to evaluate

$$\frac{dp(x)}{dx}$$

where $p(x)$ is the nth degree polynomial

$$(a_n x^n + a_{n-1}x^{n-1} + \ldots a_1 x + a_0).$$

1.7 Develop an algorithm to read a vector, sum all its positive components, count such components and sum separately all the negative components of the vector.

1.8 Obtain an algorithm which given the coordinates of a point (x, y) will write a message whether it is in or not in the first quadrant of the unit circle.

2

Computer Arithmetic

2.1 Introduction

A number of algorithms for solving problems on computers were illustrated in the last chapter. 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 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 utilising 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.*

We will now examine the way arithmetic operations are carried out in a computer. This has a profound bearing on problem formulation for numerical solution as will be seen throughout this book.

2.2 Floating Point Representation of Numbers

There are two types of arithmetic operations available in a computer. These are:

(i) Integer arithmetic
(ii) Real or floating point arithmetic.

Integer arithmetic, as the name implies, deals with integer operands, that is, numbers without fractional parts. It is used mainly in counting and as subscripts.

Real arithmetic uses numbers with fractional parts as operands and is used in most computations.

Due to economic considerations computers are designed such that each location (also called a *word*) in memory stores only a finite number of digits. Consequently all operands in arithmetic operations have only a finite number of digits. For illustrative purpose we will assume that a (hypothetical) computer has a memory in which each location can store 6 digits and has provision to store one or more signs. One method of representing real numbers in such a computer would be to assume a fixed position for the decimal point and store all numbers (after appropriate shifting if necessary) with an assumed decimal point as shown in Figure 2.1.

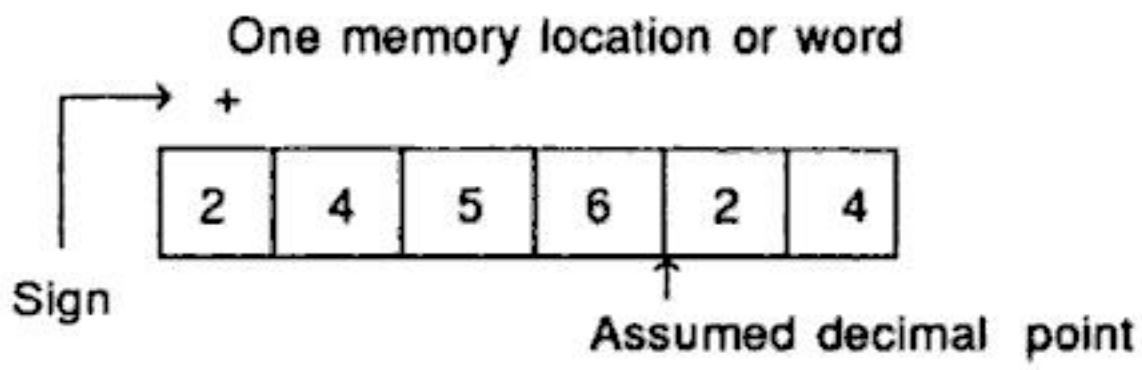

Fig. 2.1 A memory location storing the number 2456.24.

If such a convention is used the maximum and minimum (in magnitude) numbers that may be stored are 9999.99 and 0000.01 respectively. This range is quite inadequate in practice and a different convention for representing real numbers is adopted. This convention aims to preserve the maximum number of significant digits in a real number and also increase the range of values of real numbers stored. This representation is called the *normalized floating point* mode of representing and storing real numbers. In this mode a real number is expressed as a combination of a *mantissa* and an *exponent*. The mantissa is made less than 1 and greater than or equal to .1 and the exponent is the power of 10 which multiplies the mantissa. For example the number 44.85×10^6 is represented in this notation as .4485E8 (E8 is used to represent 10^8). The mantissa is .4485 and the exponent is 8. The number is stored in a memory location as shown in Figure 2.2.

In the above case the 6 digits available in a memory location are arbitrarily divided into two parts. Four digits are used for the mantissa and two for the exponent. The mantissa and the exponent have their own independent signs. While storing numbers the leading digit in the mantissa is always made non-zero by appropriately shifting it and adjusting the value of the exponent. Thus the number .004854 would be stored as shown in Figure 2.3. The shifting of

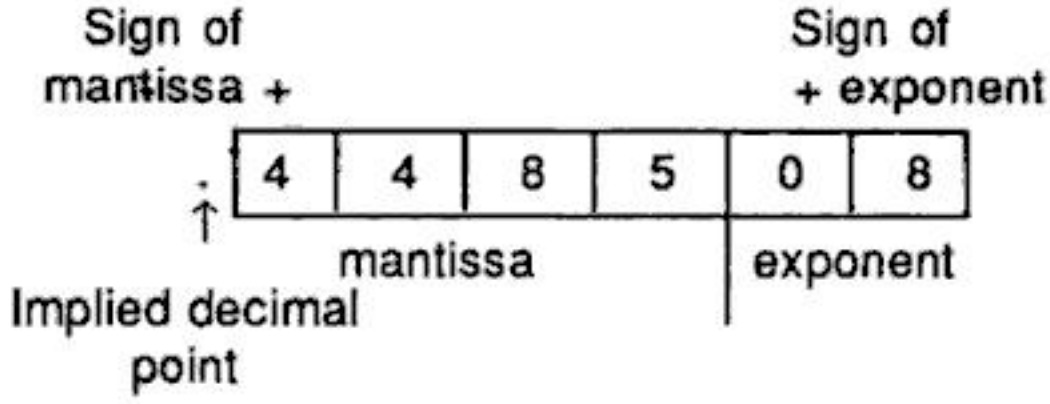

Fig. 2.2 Representation of 0.4485E8 in normalized floating point mode.

Fig. 2.3 Representation of .004854 in normalized floating point mode.

the mantissa to the left till its most significant digit is non-zero is called *normalization*. The normalization is done to preserve the maximum number of useful (information carrying) digits. The leading zeros in .004854 serve only to locate the decimal point. This information may thus be transferred to the exponent part of the number and the number stored as .4854 E–2.

When numbers are stored using this notation the range of numbers (magnitudes) that may be stored are $.9999 \times 10^{99}$ to $.1000 \times 10^{-99}$ which is obviously much larger than that used in the fixed decimal point notation described earlier. This increase in range has been obtained by reducing the number of significant digits in a number by 2.

We will now discuss how elementary arithmetic operations are performed on numbers expressed in this notation.

2.3 Arithmetic Operations with Normalized Floating Point Numbers

2.3.1 *Addition*

If two numbers represented in normalized floating point notation are to be added the exponents of the two numbers must be made equal and the mantissa shifted appropriately. The details will be clarified by some examples.

Example 2.1
Add .4546E5 to .5433E5. In this case the exponents are equal. Thus the mantissas are added. The sum is thus .9979E5.

Example 2.2
Add .4546E5 to .5433E7. In this case the two exponents are not equal. The operand with the larger exponent is kept as it is and the mantissa of the

operand with the smaller exponent is shifted *right* by a number of places equal to the difference in the two exponents. In this example the difference between the exponents is (7 – 5) = 2. Thus the mantissa of .4546E5 is shifted right two places. Each shift causes the last digit in the mantissa to be chopped off as the arithmetic unit in our hypothetical computer can accommodate only a 4 digit mantissa. Operands after the shift are as shown below:

$$\begin{array}{r} .5433E7 \\ .0045E7 \\ \hline .5478E7 \end{array}$$

Thus the sum is .5478E7.

Example 2.3
Add .4546E3 to .5433E7. In this case the first operand will be shifted 4 places to the right and .0000E7 will be added to .5433E7.

Example 2.4
Add .6434E3 to .4845E3. In this case the exponents are equal. When the mantissas are added the sum is 1.1279E3. As the mantissa has 5 digits and is greater than 1 it is shifted right one place before it is stored. When it is shifted the exponent is increased by one and the last digit of the mantissa is chopped off. Thus the answer would be .1127E4.

Example 2.5
Add .6434E99 to .4845E99. In this case again the sum of the mantissas exceeds 1. Thus the mantissa is shifted right and exponent increased by 1 resulting in a value of 100 for the exponent. As the exponent part cannot store more than two digits, in our hypothetical computer, the number is larger than the largest number that can be stored in a word. This condition is called an *overflow* condition and the arithmetic unit will intimate an error condition.

2.3.2 *Subtraction*

The operation of subtraction is nothing but adding a negative number. Thus the principles are the same. A few examples will now be considered to clarify some points.

Example 2.6
Subtract .9432E–4 from .5452E–3. As the exponents are not equal the number with the smaller exponent is shifted right and the exponent increased by 1 for each right shift. Thus the result is .5452E –3 –.0943E – 3 = .4509E – 3.

Example 2.7

Subtract .5424E3 from .5452E3. The exponents are equal. Thus the mantissas are subtracted. The result is .0028E3. As the most significant digit of the mantissa is 0 the mantissa is shifted *left* till the most significant digit is non-zero. (Remember that in normalized floating point the mantissa is ≥ .1 and < 1). For each left shift of the mantissa the exponent is reduced by 1. Thus the result is .2800E1. The trailing zeros in the results do not carry any information but are carried by the computer in all further calculations as though significant.

Example 2.8

The subraction .5452E −99 − .5424E −99 leads to the answer .0028E −99. Again the mantissa is shifted left (for normalization). In this process the exponent is reduced by 1. The exponent would thus become −100 with the first left shift. As the exponent in our hypothetical computer can store only two digits −100 cannot be accommodated in the exponent part of the number. In this case the result is smaller than the smallest number which could be stored in this (hypothetical) computer. This condition is called an *underflow* condition and the arithmetic unit will signal an error condition.

To recapitulate, if the result of an arithmetic operation gives a number smaller than .1000E −99 then it is called an underflow condition. Similarly any result greater than .9999E99 leads to an overflow condition.

2.3.3 *Multiplication*

Two numbers are multiplied in the normalized floating point mode by multiplying the mantissas and adding the exponents. After the multiplication of the mantissas the result mantissa is normalized as in addition/subtraction operation and the exponent appropriately adjusted. Some examples are given below to illustrate the procedure.

Example 2.9

$$+ .5543E12 \times .4111E - 15 = .22787273E - 3 = .2278E - 3$$
$$\text{Discarded}$$

Example 2.10

$$.1111E10 \times .1234E15 = .01370974E25 = .1370E24$$
$$\text{Discarded}$$

Example 2.11

$$.1111E51 \times .4444E50 = .04937284E101 = .4937E100$$

—Answer overflows.

Example 2.12

$$.1234E - 49 \times .1111E - 54 = .1370E - 104$$

— Result underflows.

2.3.4 *Division*

In dividing a number by another the mantissa of the numerator is divided by that of the denominator. The denominator exponent is subtracted from the numerator exponent. The quotient mantissa is normalized to make the most significant digit non-zero and the exponent appropriately adjusted. The mantissa of the result is chopped down to occupy 4 digits. Some examples are worked out below to clarify the procedure.

Example 2.13

.9998E1 ÷ .1000E − 99 = 9.9980E100 = .9998E101
— Result overflows.

Example 2.14

.9998E − 5 + .1000E98 = .9998E − 104 —Result underflows.

Example 2.15

.1000E5 + .9999E3 = 0.1000E2

2.4 Consequences of Normalized Floating Point Representation of Numbers

2.4.1 *Non-Associativity of Arithmetic*

In the last section methods of performing the four arithmetic operations with numbers in normalized floating point mode were presented. It was seen that numbers had to be truncated to fit into the 4 mantissa digits allowed in our hypothetical computer for each number. This truncation leads to a number of seemingly surprising results (namely, results which we are not used to in our experience with arithmetic). For instance $\frac{2}{3} \times 6 = 4$ as we all know. However, when the arithmetic is performed with floating point numbers .6667 added 6 times gives .3998E1 whereas .6667 × 6 gives .4000E1. In other words, the equation $6x = x + x + x + x + x + x$ is not true! (Check this using floating point arithmetic.)

Another consequence of the floating point representation is that the associative and the distributive laws of arithmetic are not always valid. In other words,

$$(a + b) - c \neq (a - c) + b$$

$$a(b - c) \neq (ab - ac).$$

These are illustrated with examples below.

Example 2.16

Let

$$a = .5665E1$$

$$b = .5556E - 1$$

$$c = .5644E1$$

$$(a + b) = .5665E1 + .5556E - 1$$

$$= .5665E1 + .0055E1$$

$$= .5720E1$$

$$(a + b) - c = .5720E1 - .5644E1$$

$$= .0076E1$$

$$= .7600E - 1$$

$$(a - c) = .5665E1 - .5644E1$$

$$= .0021E1 = .2100E - 1$$

$$(a - c) + b = .2100E - 1 + .5556E - 1$$

$$= .7656E - 1$$

It is thus seen that $(a + b) - c \neq (a - c) + b$.
In fact the correct answer if no number is truncated is .7656E – 1.

Example 2.17

Let

$$a = .5555E1$$

$$b = .4545E1$$

$$c = .4535E1$$

$$(b - c) = .0010E1 = .1000E - 1$$

$$a(b - c) = .5555E1 \times .1000E - 1$$

$$= .0555E0 = .5550E - 1$$

$$ab = .5555E1 \times .4545E1 = .2524E2$$

$$ac = .5555E1 \times .4535E1 = .2519E2$$

$$ab - ac = .0005E2 = .5000E - 1$$

Thus $a(b - c) \neq ab - ac$.

In fact, if the intermediate results are not truncated, the correct answer is .5555E – 1.

The above examples are intentionally chosen to illustrate the inaccuracies

that may build up due to shifting and truncation of numbers in arithmetic operations. The wide disparity in the results obtained in the examples is due to the fact that in each case the difference of two almost equal numbers is involved. Whenever such a condition occurs in practice one should be careful. If possible, subtraction operation should be eliminated altogether as illustrated in the following example.

Example 2.18

Evaluate $(1 - \cos x)$ at $x = .1396$ radians

$$\cos (.1396) = .9903$$

$$1 - \cos (.1396) = .1000E1 - .9903E0$$

$$= .1000E1 - .0990E1 = .1000E - 1.$$

If we rewrite $(1 - \cos x)$ as $2 \sin^2 x/2$ then the value is:

$$\sin\left(\frac{x}{2}\right) = \sin .0698 \simeq .6974E - 1$$

$$2 \sin^2 \frac{x}{2} = (.2000E1) \times (.6974E - 1) \times (.6974E - 1)$$

$$= .9727E - 2$$

The value obtained by the alternate formula is closer to the true value .9728E – 2.

Example 2.19

The solution of the quadratic equation

$$x^2 - 1000x + 25 = 0$$

is

$$(1000 \pm \sqrt{(10^6 - 10^2)})/2$$

using floating point arithmetic (with a 4 digit mantissa):

$$10^6 = .1000E7, \; 10^2 = .1000E3.$$

Thus

$$10^6 - 10^2 = .1000E7 - .1000E3$$

$$= .1000E7$$

Thus

$$\sqrt{10^6 - 10^2} = .1000E4$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Using 4 digit floating point arithmetic the recurrence relation of Eq. (2.1) gives

$P_1 = .3678E0$ (correct $P_1 = .367879$)
correct $P_2 = .264242$

$P_2 = 1 - 2P_1 = .1000E1 - .7356E0$

or

$= .1000E1 - .0735E1$

$= .0265E1$

$= .2650E0$

Error in $P_2 = (.2650 - .264242) = .000758$

$P_3 = 1 - 3P_2 = .2050E0$

$P_4 = 1 - 4P_3 = .1800E0$

$P_5 = 1 - 5P_4 = .1000E0$

$P_6 = 1 - 6P_5 = .4000E0$

Using 6 digit calculations $P_6 = .127120E0$

There is a big difference between these two values of P_6 and we should investigate the reason for this. By inspection of the arithmetic given above we see that the entire error in the result is due to the rounding error committed in approximating P_2. This error is .000758. The error in P_6 becomes very large as the error in P_2 is multiplied by 3 in the next step, 4 in the step next to that and so on. Thus the final error is approximately (.000758) (3) (4) (5) (6) = .27288. Thus the true value of P_6 calculated from the error is (.4000 – .27288) = .12712.

This inflation of a small initial error into a large one in the result is due to the nature of the algorithm. Such an algorithm is known as an *unstable algorithm*. Such an unstable algorithm should be reformulated into a stable algorithm. The algorithm $P_n = 1 - nP_{n-1}$ can be made into a stable one by rewriting

$$P_{n-1} = \frac{1 - P_n}{n} \qquad n = \ldots, 4, 3, 2 \tag{2.2}$$

This algorithm works backwards from large n towards small n. To obtain a starting value we observe that

$$P_n = \int_0^1 x^n e^{x-1} dx \leq \int_0^1 x^n dx = \frac{x^{n+1}}{n+1}\Big|_0^1 \leq \frac{1}{n+1} \tag{2.3}$$

P_n decreases towards zero as n increases.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The remainder after each division is either 0 or 1 and is noted on the right. After successive divisions when the final quotient is zero the procedure is stopped. The binary equivalent is the set of 1s and 0s obtained as remainders, the last remainder being the most significant binary digit (also called *bit*). Thus 14 = 1110. The procedure explained above is really a method of getting a polynomial equivalent of the decimal number with 2 as the base of the polynomial. This is evident from the polynomial shown at the bottom of p. 29.

The binary equivalent of the fractional part is obtained in a similar way. In this case instead of successively dividing by 2, the division is by $^1/_2$ (or equivalently it is multiplication by 2). The integer part of the product after multiplication is either 1 or 0 and these bits read from *top to bottom* give the binary equivalent. The procedure is stopped when the fractional part of the product is zero.

$$
\begin{aligned}
.875 \times 2 &= 1.750 \rightarrow 1 \qquad 2^{-1} \\
.75 \times 2 &= 1.5 \rightarrow 1 \qquad 2^{-2} \\
.5 \times 2 &= 1.0 \rightarrow 1 \qquad 2^{-3}
\end{aligned}
$$

$$
\begin{aligned}
.875 &= 2^{-1}\,(1 + 2^{-1}\,(1 + 2^{-1}\,(1))) \\
&= 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}. \\
&= .111 \text{ (binary)}
\end{aligned}
$$

Example 2.24
Convert 17.375 to binary.

$$
\begin{aligned}
.375 \times 2 &= .750 \quad 0 \\
.750 \times 2 &= 1.500 \quad 1 \\
.500 \times 2 &= 1.000 \quad 1
\end{aligned}
$$

2	17	1
2	8	0
2	4	0
2	2	0
2	1	1
	0	

Thus 17.375 = 10001.011

Example 2.25
Convert .2 to binary.

$$
\begin{aligned}
.2 \times 2 &= .4 \quad 0 \\
.4 \times 2 &= .8 \quad 0 \\
.8 \times 2 &= 1.6 \quad 1 \\
.6 \times 2 &= 1.2 \quad 1 \\
.2 \times 2 &= .4 \quad 0
\end{aligned}
$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

2.5 Calculate the value of the polynomial $x^3 - 4x^2 + 0.1x - .5$ for $x = 4.011$ using floating point arithmetic with 4 digit mantissa in two different ways, namely, straightforward determination and by using nested parenthesis method. Find the relative errors in the two methods.

2.6 Find the value of

$$(1 + x)^2 \text{ and } (x^2 + 2x) + 1$$

when

$$x = .5999\text{E} - 2.$$

Calculate the relative errors in the two methods of calculating the expression. Which is the preferred method?

2.7 Find the value of $\sin x = x - x^3/3! + x^5/5! \ldots$ with an absolute error smaller than .005 for $x = .2000\text{E}0$ and $x = .1276\text{E}2$ using normalized floating point arithmetic with 4 digit mantissa.
(Hint: Do not blindly substitute when $x = .1276\text{E}2$.)

2.8 Find the value of $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$ with an absolute error less than .005 for $x = .2500\text{E}0$ and $x = .5250\text{E}1$. It is given that $e = 2.7183$.

2.9 Express the following binary numbers in decimal, octal and hexadecimal:
(i) 1011.1101 (ii) 1110111.00111

2.10 Express the following decimal numbers in binary form:

(i) 22.625 (ii) 6.02 (iii) – 10.125 (iv) 15.8

2.11 Develop an algorithm to convert (i) a decimal integer into a binary number, (ii) a decimal fraction into an equivalent binary fraction. Combine the two algorithms to convert any decimal number to binary.

2.12 Convert the decimal number 28.4 into an equivalent number in (i) ternary system, (ii) octal system and (iii) hexadecimal system.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

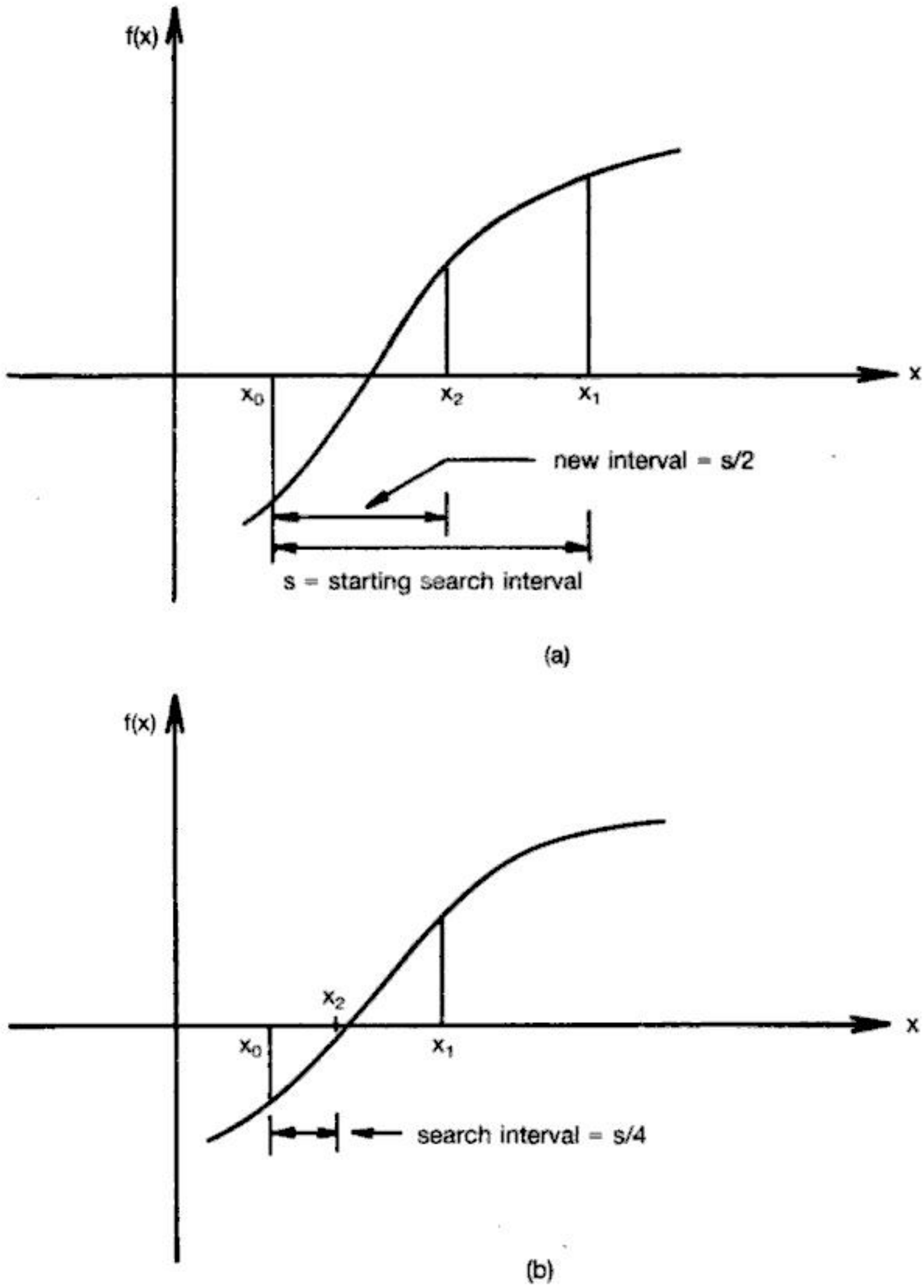

Fig. 3.2(a), (b) Illustrating bisection method.

Example 3.1

We will find the square root of a number by this method. The square root of a number d is evaluated by finding the root of the equation $x^2 - d = f(x) = 0$. Let

$$d = 25.0, \quad x_0 = 2.0, \quad x_1 = 7.0.$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

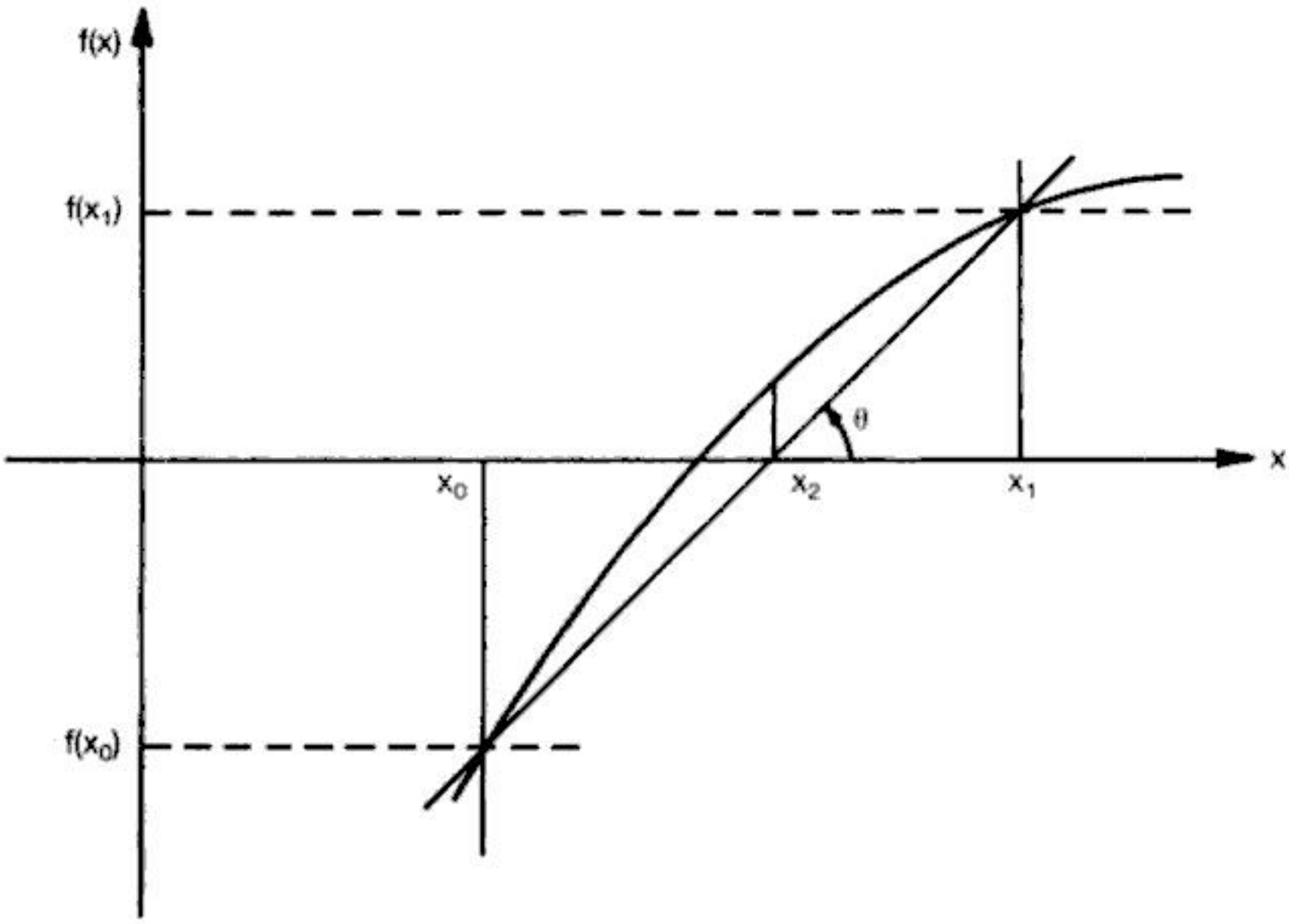

Fig. 3.3 Illustrating false position method.

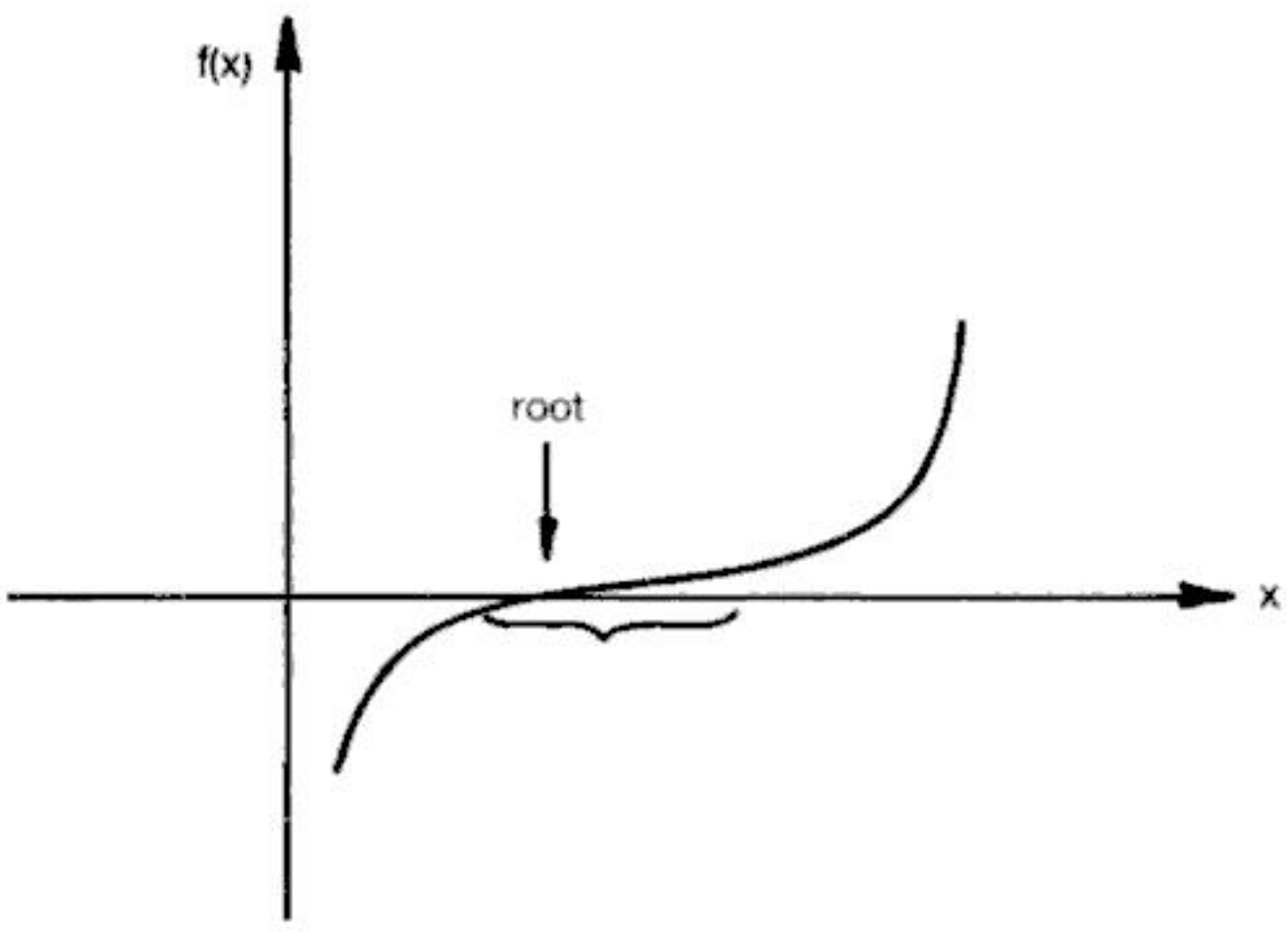

Fig. 3.4 Illustrating that the criterion $| f(x) | \leq e$ may lead to a large error in the root if slope of $f(x)$ is small near the root.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

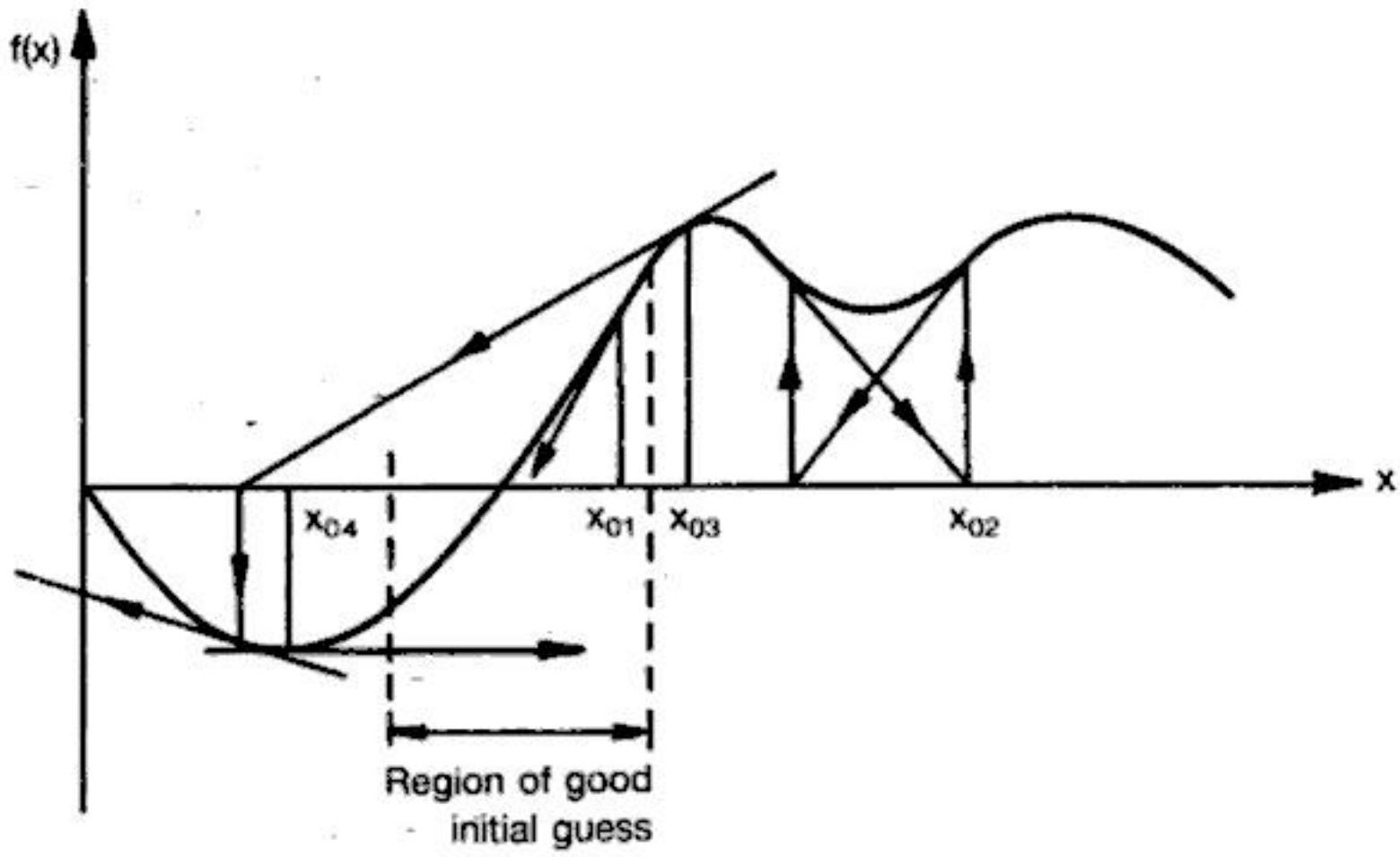

Fig. 3.6 Illustrating traps in the choice of initial guess of root in Newton-Raphson method.

estimate is available then the Newton-Raphson procedure may be safely applied to quickly converge to the root.

3.5.1 *Rate of Convergence of the Iterative Procedure*

Equation 3.6 which is reproduced again below gives the rule to proceed from the ith to the $(i + 1)$th step:

$$x_{i+1} = x_i - [f(x_i)/f'(x_i)]. \tag{3.6}$$

The convergence of the iterative procedure is judged by the rate at which the error between the true root and the calculated root decreases. The *order of convergence* of an iterative process is defined in terms of the errors e_i and e_{i+1} in successive approximations. An iterative algorithm is kth order convergent if k is the largest integer such that:

$$\lim_{i \to \infty} (e_{i+1}/e_i^k) \leq M \tag{3.7}$$

where M is a finite number. In other words the error in any step is proportional to the kth power of the error in the previous step.

The Newton-Raphson procedure is second order convergent as may be seen from the following derivation:

Let y be the root of the equation $f(x) = 0$. Let the ith iterate x_i differ from the true root by an amount e_i. Thus e_i is the error in the ith iteration. Equation 3.6 may be rewritten as:

$$y + e_{i+1} = y + e_i - [f(y + e_i)/f'(y + e_i)].$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

quantity which would lead to loss of significant digits. An iterative formula suitable for computation is:

$$x_{i+1} = \frac{x_{i-1}f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})}. \tag{3.13}$$

An algorithm to implement the secant method is given as Algorithm 3.5. The student should note the exit provided when $(f(x_i) - f(x_{i-1}))$ is too small. The criterion used to check convergence is to check if $f(x)$ is small. This criterion is more appropriate as in this method successive values of x_i's tend to fall on either side of the root when a root is approached and checking the value of $f(x_0)$ might lead to an earlier end of the iterative cycle. As in the Newton-Raphson method a limit is put on the number of iterations to be allowed.

Algorithm 3.5 Secant Method

```
1  Read x0, x1, e, delta, n
   Remarks: x0 x1 are two initial guesses to the root. e is the prescribed
            precision, delta the minimum allowed value of slope and
            n the maximum number of iterations to be permitted.
2  f0 ← f(x0)
3  f1 ← f(x1)
   Remarks: Statements 5 to 12 are repeated until the procedure converges
            to a root or iterations exceed n
4  for i = 1 to n in steps of 1 do
5        if |f1 − f0| < delta then GOTO 15
6        x2 ← (x0 f1 − x1 f0)/(f1 − f0)
7        f2 ← f(x2)
8        if |f2| < e then GOTO 17
9        f0 ← f1
10       f1 ← f2
11       x0 ← x1
12       x1 ← x2
   endfor
13 Write 'Does not converge', x0, x1, f0, f1
14 Stop
15 Write 'Slope too small', i, f0, f1, x0, x1
16 Stop
17 Write 'Convergent solution', i, x2, f2
18 Stop
```

Example 3.4

Find the smallest positive root of the following equation

$$f(x) = x^3 - 3x^2 + x + 1 = 0$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

By the mean value theorem[1]

$$\frac{g(x_{n-1}) - g(r)}{(x_{n-1} - r)} = g'(\lambda) \tag{3.25}$$

where λ is a point between x_{n-1} and r.

Thus

$$(x_n - r) = g'(\lambda)(x_{n-1} - r). \tag{3.26}$$

Let M be the maximum value of $g'(x)$ occurring at $x = \lambda$ in the entire range $[x_0, r]$.

Then we have

$$\begin{aligned} |x_n - r| &\leq |g'(\lambda)|\,|x_{n-1} - r| \leq M|x_{n-1} - r| \\ |x_{n-1} - r| &\leq M|x_{n-2} - r| \\ &\vdots \\ |x_1 - r| &\leq M|x_0 - r|. \end{aligned}$$

Thus

$$|x_n - r| \leq M^n |x_0 - r|. \tag{3.27}$$

Thus if $M < 1$, whatever the initial value of x_0, the value of x_n will tend towards r for a sufficiently large n. However, if $M \geq 1$ the process will not converge. Figure 3.9(a) and (b) illustrate the two situations.

An algorithm implementing the successive approximation procedure is given as algorithm 3.6. The student is urged to compare it with the algorithm for the Newton-Raphson method.

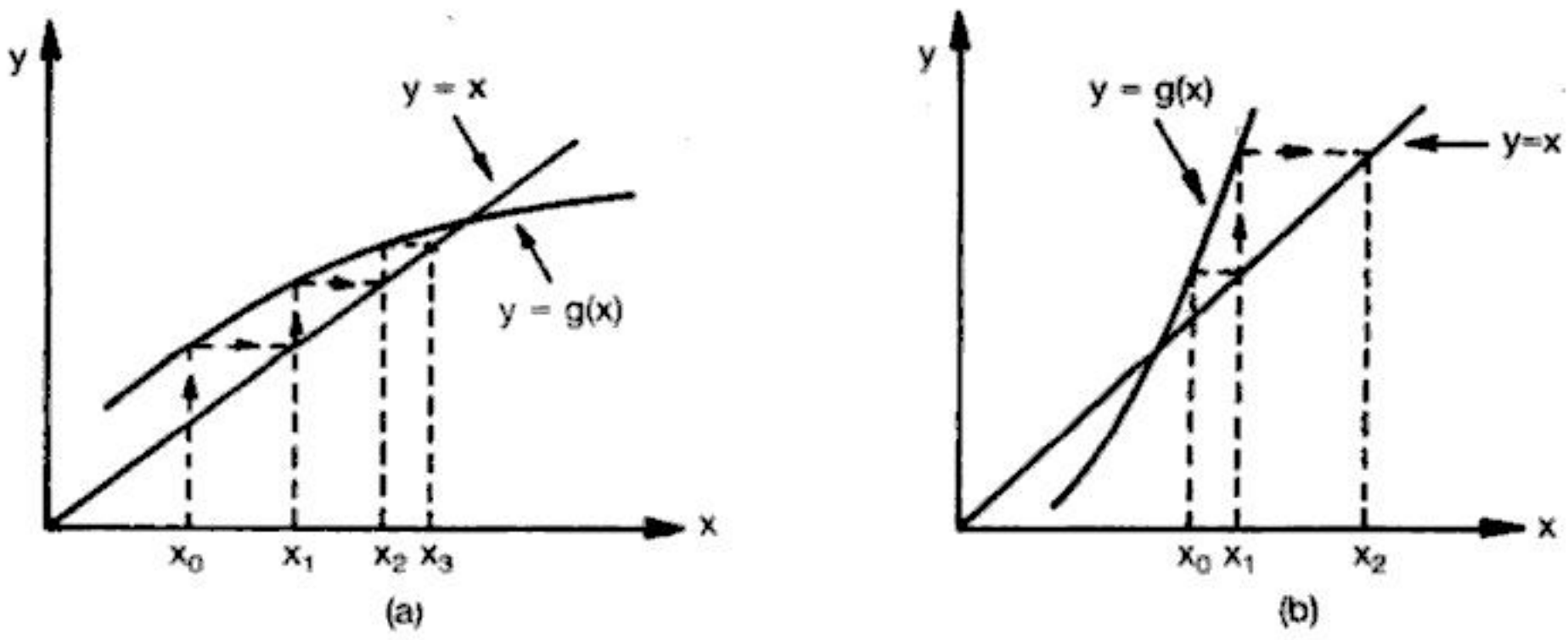

Fig. 3.9 Illustrating (a) convergent and (b) divergent solution in successive approximation method.

[1]Mean value theorem states that given a continuous function $g(x)$ with a continuous first derivative the slope of the chord to the curve drawn between two arbitrary points a, b on it, namely, $\frac{g(b) - g(a)}{(b - a)}$ is equal to the slope of a tangent to the curve at some point between a and b.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

This gives an upper bound for all the roots of the polynomial equation and we do not need to search for a root beyond this interval

$$-\sqrt{\left(\frac{a_{n-1}}{a_n}\right)^2 - 2\left(\frac{a_{n-2}}{a_n}\right)}, +\sqrt{\left(\frac{a_{n-1}}{a_n}\right)^2 - 2\left(\frac{a_{n-2}}{a_n}\right)} \tag{3.34}$$

The general strategy to find all the roots of the equation are:

1. Examine if all the coefficients are real. In this case all the roots of the equation are real or pairs of roots are complex conjugates.
2. Find the range of roots from Eq. (3.34). The function $f(x)$ is now plotted in this range. We should take care not to use too coarse an interval lest we miss some roots. In Figure 3.10, for example, root r_1 will be detected as $f(x)$ changes sign between successive plot points. Roots r_2 and r_3 will be missed as they lie within a plot interval. We can simultaneously plot $f'(x)$ which is anyway required. The change of sign of $f'(x)$ will warn of possible roots in an interval.
3. Having guessed the roots from the plots we can count them and see if all roots are accounted for (An nth order polynomial must have n roots).
4. If the number of roots counted is less than the order of the polynomial there could be multiple roots or complex pairs of roots. They have to be found after all the simple real roots are found.
5. The largest root r is found by using Newton-Raphson procedure which we explain in detail later in this section. The polynomial is divided by $(x - r)$. (The division algorithm was given as Algorithm 1.5 in Chapter 1). The reduced polynomial is used to find the next root and remove it. The procedure of reducing the order of a polynomial is known as *deflation*. When all simple roots are removed complex conjugate pairs and repeated roots are tackled.

We will first give the Newton-Raphson method for a polynomial equation to find simple real roots. We assume that the polynomial has real coefficients and all roots are real. One may wonder why yet another method is given for polynomials when Newton-Raphson method is applicable for any function. The main reason is efficiency. The computational labour is reduced with the special algorithm we will present now.

A polynomial equation may be written as:

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n = 0 \tag{3.35}$$

We may rewrite

$$f(x) = g(x)(x - x_0) + R \tag{3.36}$$

where $g(x)$ is a $(n - 1)$th order polynomial obtained by dividing $f(x)$ by

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The bounds of the roots of Eq. (3.49) are given by (3.34) and are:

$$\pm \sqrt{\left(\frac{-4}{1}\right)^2 - 2\left(\frac{3}{1}\right)} = \pm \sqrt{10} < \pm 3.2$$

hus we plot $f(x)$ in the range $\pm$ 3.2

$x =$	−3.2	−3	−2	−1	0	1	2	3	3.2
$f(x) =$	233.6	182	34	−10	−10	−2	2	14	22.8

From the above table we see that there are two real roots. One root is between − 2 and − 1 and the other between 1 and 2. We first find the root between 1 and 2 starting with a trial value of 1.5 using Algorithm 3.7. The following table shows the calculations.

Let $x_0 = 1.5$

i	a_i	b_i	S	$P = a_0 + b_0 x_0$
0	−10	6.875	3.6875	= −10 + (6.875)(1.5)
1	8	−7.5	−2.125	$x_1 \leftarrow x_0 - P/S$
2	3	−2.5	−1	= 1.5 − .3125/3.6875
3	−4	1	1	
4	1			= 1.4153

$x_0 = 1.4153$

i	a_i	b_i	S	
0	−10	7.0686	3.8121	$x_0 \leftarrow$ 1.4153 − .0041/3.8121
1	8	−0.6581	−2.3132	
2	3	−2.5847	−1.1694	= 1.4142
3	−4	1	1	e = 0.00077
4	1			

As calculations used are up to 4 significant digits only, we stop here as $e \leq 0.001$. Thus $x_0 = 1.414$ is one root

To find the other root which is between − 2 and − 1, we again start with a trial value of − 1.5 and obtain − 1.414 as a root using a similar calculation.

Now we remove root 1.414 from Eq. (3.49) using Algorithm 1.5. We get the values of b_i from the last table above as: $b_3 = 1$, $b_2 = -2.5847$, $b_1 = -.6581$ and $b_0 = 7.0686$

Thus the reduced polynomial is

$$x^3 - 2.5847\,x^2 - 0.6581\,x + 7.0686 = 0 \qquad (3.50)$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

4.3 Pivoting

In the triangularization Algorithm 4.6 instruction 4 is

$$u \leftarrow a_{ik}/a_{kk}.$$

In this instruction it has been assumed that a_{kk} is not zero. If it happens to be zero or nearly zero the algorithm will lead to no results or meaningless results. An example of this type was considered in Chapter 2, Sec. 2.5 for two simultaneous equations. It was seen there that the two equations when interchanged gave a better solution. Similarly if any of the a_{kk}s is small it would be necessary to reorder the equations. Observe that the value of a_{kk}s would be modified during the elimination process and there is no way of predicting their values at the start of the procedure.

Referring to Equations (4.1), (4.2) and (4.3) and Algorithm 4.1 it is seen that all elements except a_{11} in column 1 are made zero by that algorithm. The element a_{11} is called the *pivot* element. After a_{21} and a_{31} are eliminated the next step in triangularization is to eliminate all the terms below a_{22} in the second row (in this case a_{32} only). The pivot in this case is a_{22}.

In the elimination procedure the pivot should not be zero or a small number. In fact for maximum precision the pivot element should be the largest in absolute value of all the elements below it in its column. In other words a_{11} should be picked as the maximum of a_{11}, a_{21} and a_{31}. The element a_{22} should be greater than a_{32}. In general the pivot element a_{kk} should be the largest of

$$\{|a_{mk}|\} \text{ for } m = k + 1, n$$

Thus during the Gauss elimination procedure $|a_{mk}|$s should be searched and the equation with the maximum value of $|a_{mk}|$ should be interchanged with the current equation. For example if during elimination we have the following situation:

$$x_1 + 2x_2 + 3x_3 = 4 \tag{4.15}$$

$$0.3x_2 + 4x_3 = 5 \tag{4.16}$$

$$-8x_2 + 3x_3 = 6 \tag{4.17}$$

then as $|-8| > 0.3$, Equations (4.16) and (4.17) should be interchanged to yield

$$x_1 + 2x_2 + 3x_3 = 4 \tag{4.18}$$

$$-8x_2 + 3x_3 = 6 \tag{4.19}$$

$$0.3x_2 + 4x_3 = 5 \tag{4.20}$$

It should be remembered that interchange of equations does not affect the solution.

The algorithm for picking the largest element as the pivot and interchanging

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The next question we have to answer is whether we take the deviations between the tabulated values of y and those obtained from the straight line or the tabulated values of x and those obtained from the straight line. In the first case the deviations are:

$$\sum_{i=1}^{n} (y_i - \bar{y}_i)^2 = \sum_{i=1}^{n} \{y_i - (a_1 x_i + a_0)\}^2 \tag{6.8}$$

In the second case the assumed straight line would be represented by the equation

$$x = b_1 y + b_0 \tag{6.9}$$

and the deviations would be

$$\sum_{i=1}^{n} (x_i - \bar{x}_i)^2 = \sum_{i=1}^{n} \{x_i - (b_1 y_i + b_0)\}^2 \tag{6.10}$$

The straight lines obtained in the two cases are different.

In obtaining the equation $y = a_1 x + a_0$ it is assumed that x is the *independent* or controlled variable and thus has no errors in it. The variable y is the *dependent* or measured variable and is assumed to have statistical errors. The deviation $(y_i - \bar{y}_i)$ measures the error between measured and fitted value along the y direction. The linear-least squares fit equation obtained in this case is called as *linear regression of y on x.*

In writing the equation $x = b_1 y + b_0$ it is assumed that y is the independent or controlled variable and x the dependent or measured variable. Thus it is assumed that y has no errors and the errors are in x. The deviation $(x_i - \bar{x}_i)$ measures the error between measured and fitted values in the x direction. The linear least squares fit equation obtained in this case is known as *linear regression of x on y.*

Given a problem we have to decide which is the independent and which is the dependent variable. If it is not possible to identify a variable as such and if both variables have inherent errors in them then one may take the deviation as the perpendicular distance between the straight line and the point (x, y) and minimise the sum of square of these deviations. Detailed discussion of this is outside the scope of this book. Further, this is rarely used.

We will now discuss a method of calculating a_0 and a_1 in the linear equation

$$y = a_1 x + a_0 \tag{6.7}$$

by minimising the sum of squares of error between measured values and that given by (6.7). Thus we want to minimise S given by

$$\min S = \min \sum_{i=1}^{n} (y_i - \bar{y}_i)^2 = \min \sum_{i=1}^{n} [y_i - (a_1 x_i + a_0)]^2 \tag{6.8}$$

S is a function of the two unknown variables a_0 and a_1. Thus to minimise S we take the partial derivative of S with respect to a_0 and a_1 and set these to zero. Thus

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

6.10 Fit a curve of the type ae^{-bx} for the data given in Table 6.6.

TABLE 6.6

x	.25	.5	.75	1.0	1.25	1.5	1.75	2.00	2.25	2.5
y	3.1	1.7	1.0	.68	.42	.26	.14	0.09	0.04	0.03

6.11 Obtain equations to find the parameters a and b for fitting by least squares method the following curve to a set of n points (x_i, y_i).

$$y = a \sinh bx.$$

6.12 Write a computational procedure for fitting an exponential curve ae^{bx} to a set of points (x_i, y_i).

6.13 Fit a curve of the type $1/(a + bx)$ to the points given in Table 6.7.

TABLE 6.7

x	0	1	2	3	4	5	6	7	8	9
y	11.0	3.33	2.20	1.52	1.00	0.91	0.82	0.66	0.56	0.49

6.14 Derive the equations for the coefficients a and b if the curve $(1/(a + bx^2))$ is to be fitted to the set of points (x_i, y_i).

6.15 Obtain an appropriate transformation and the equations for finding the coefficients of the curve $y = a(b^x)$ to be fitted to a set of points (x_i, y_i).

6.16 Fit a curve of the type

$$y = A \cos (\omega t + \varphi)$$

for the data of Table 6.8. Assume $\omega = 1.0$.

TABLE 6.8

t	0	.2	.4	.6	.8	1.0	1.2	1.4	1.6
y	0.05	.57	1.17	1.68	2.13	2.52	2.79	2.94	3.07

t	1.8	2.0	2.2	2.4	2.6	2.8	3.0	3.2
y	2.97	2.82	2.64	2.22	1.80	1.31	.75	.25

6.17 Find equations for the coefficients a and b of the curve $y = a \sin \omega t + b \sin 2\omega t$ by least squares. Assume ω is known.

6.18 Find equations for the coefficients a and b of the curve $y = ae^{-bx^2}$ by least squares.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

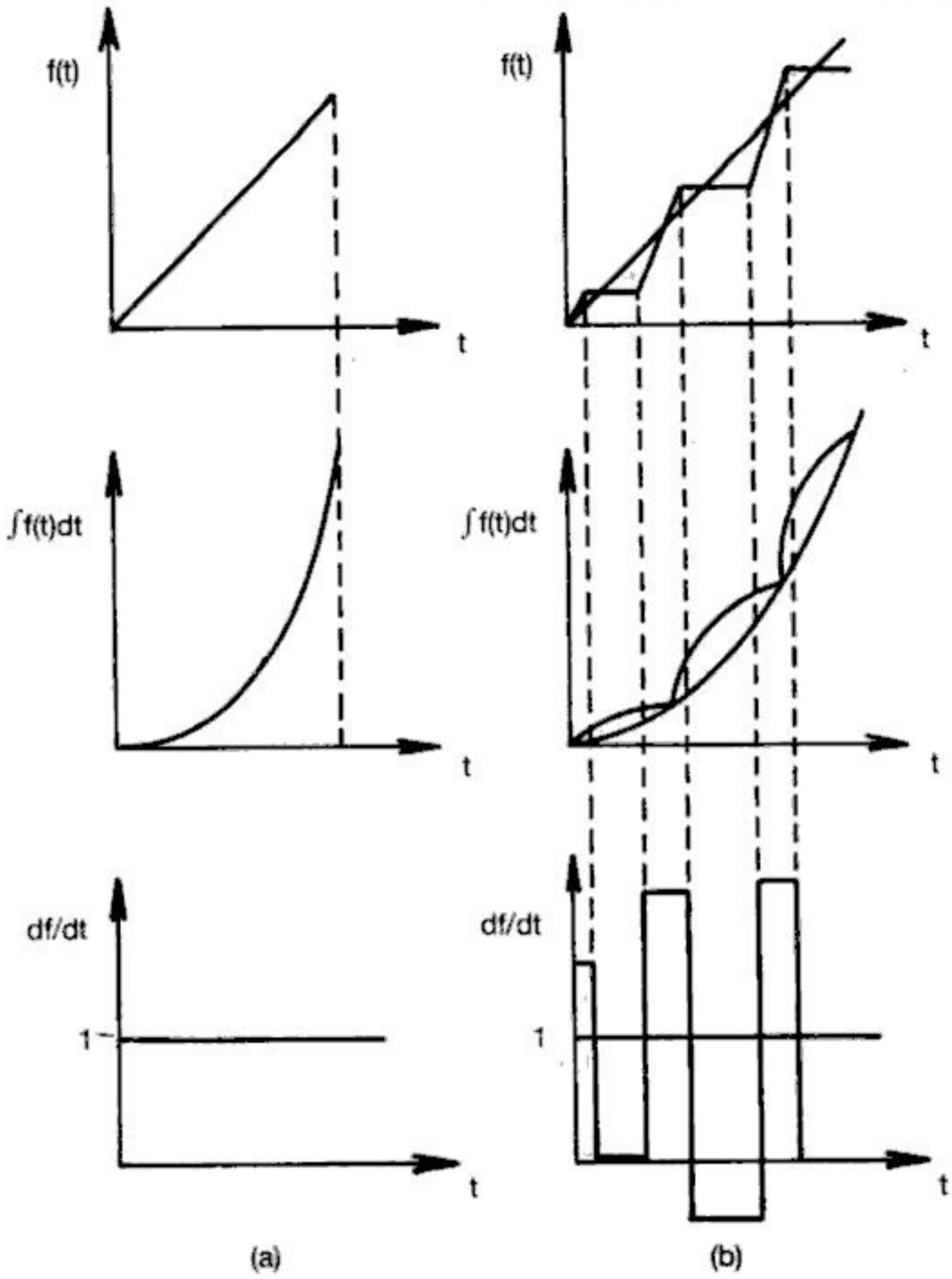

Fig. 8.1 Illustrating integration and differentiation of a function: (a) Function (b) Function $f(t)$ with superposed error.

that any errors in approximating a function are amplified while taking the derivative whereas they are smoothed out in integration.

Thus numerical differentiation should be avoided if an alternative exists.

8.2 Formulae for Numerical Differentiation

Formulae for numerical differentiation are obtained by using an interpolating polynomial through the appropriate set of points. For instance if the values of a function are available at $(x_1, x_2, \ldots, x_n)$ and its derivative at $x_1 < x < x_3$ is required we may use the polynomial

$$f(x) = f(x_1) + \frac{\Delta f_1}{h}(x - x_1) + \frac{\Delta^2 f_1}{2!h^2}(x - x_1)(x - x_2) + R(x) \qquad (8.1)$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

By Trapezoidal rule the integral is $\frac{h}{2}[f(0) + f(h)]$. Expanding $f(h)$ in a Taylor series:

$$\frac{h}{2}[f(0) + f(h)] = \frac{h}{2}\left[f(0) + \left(f(0) + hf'(0) + \frac{h^2}{2!}f''(0) + \ldots\right)\right]$$

$$= hf(0) + \frac{h^2}{2}f'(0) + \frac{h^3}{4}f''(0) + \frac{h^4}{12}f'''(0) + \ldots \qquad (8.19)$$

The difference between Equations (8.18) and (8.19) gives the error in applying Trapezoidal rule. The error is:

$$E = -\frac{h^3}{12}f''(0) - \frac{h^4}{24}f'''(0) \qquad (8.20)$$

As h is much smaller than 1 the dominant error term is the h^3 term. Thus if the interval at which $f(x)$ is tabulated is halved the error would be reduced by an eighth.

The error term in Simpson's rule may be calculated in a similar way. It is given by:

$$\frac{-h^5}{90}f^{iv}(0) \qquad (8.21)$$

Thus if the tabulated interval h is halved, the error is reduced by a factor of 32. This rapid reduction in error makes Simpson's rule a favourite one for integration.

8.6 Algorithms for Integration of Tabulated Function

There are two cases in which we may integrate numerically. In the first case the values of the function are given as a table for $x = x_1, x_1 + h, x_1 + 2h \ldots x_1 + nh$. In this case we use either Trapezoidal rule or Simpson's rule depending on the tabulation interval h and the desired accuracy. Algorithm 8.1 uses Trapezoidal rule.

Algorithm 8.1 Integrating a Tabulated Function Using Trapezoidal Rule

Remarks: $f_1, f_2, \ldots, f_{n+1}$ are the tabulated values at $x_1, x_1 + h, x_1 + 2h \ldots x_1 + nh$ $(n + 1$ *points*$)$

```
1 for i = 1 to n + 1 Read f_i endfor
2 sum ← (f_1 + f_{n+1})/2
3 for j = 2 to n do
4       sum ← sum + f_j
  endfor
5 Integral ← h · sum
6 Write Integral
7 Stop
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

TABLE 8.2 Showing Weights Attached to Functions in Successive Iterations when Simpson's Rule is Used

Iteration Number	$f(x_1)$	$x_1+\frac{h}{4}$	$x_1+\frac{h}{2}$	$x_1+\frac{3h}{4}$	$f(x_1+h)$	$x_1+\frac{5h}{4}$	$x_1+\frac{3h}{2}$	$x_1+\frac{7h}{4}$	$f(x_1+2h)$
0	1				(4)				1
1	1		(4)		2		(4)		1
2	1	(4)	2	(4)	2	(4)	2	(4)	1

Algorithm 8.5 Evaluation of Integral by Simpson's Rule

1 Read x_1, x_2, e
2 $h \leftarrow (x_2 - x_1)/2$
3 $i \leftarrow 2$
4 $S_1 \leftarrow f(x_1) + f(x_2)$
5 $S_2 \leftarrow 0$
6 $S_4 \leftarrow f(x_1 + h)$
7 $I_0 \leftarrow 0$
8 $I_n \leftarrow (S_1 + 4S_4) \cdot (h/3)$
repeat
9 $S_2 \leftarrow S_2 + S_4$
Remarks: S_2 stores already computed functional value and S_4 the value computed in the new iteration
10 $S_4 \leftarrow 0$
11 $x \leftarrow x_1 + h/2$
12 *for* $j = 1$ to i *do*
13 $S_4 \leftarrow S_4 + f(x)$
14 $x \leftarrow x + h$
endfor
15 $h \leftarrow h/2$
16 $i \leftarrow 2i$
17 $I_0 \leftarrow I_n$
18 $I_n \leftarrow (S_1 + 2S_2 + 4S_4) \cdot (h/3)$
19 *until* $|I_n - I_0| \leq e \cdot |I_n|$
20 Write I_n, h, i
21 Stop

Higher order Newton-Cotes formulae do give a smaller error per iteration but writing a general algorithm to economise computational effort in successive iterations is not easy. Thus in practice Simpson's rule is the most popular integration technique.

8.8 Gaussian Quadrature Formulae

In deriving Trapezoidal and Simpson's rules the values of the function are

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

COMPUTER ORIENTED NUMERICAL METHODS

THIRD EDITION

V. RAJARAMAN

This book is a concise introduction to numerical methods with special emphasis on evolving computational algorithms for solving non-linear algebraic equations, sets of linear equations, curve-fitting, integration, differentiation, and solving ordinary differential equations.

Outstanding Features:

- Elementary presentation of numerical methods from first principles for students who do not have any mathematics background.
- A conceptual understanding of numerical methods built on students' geometrical intuition.
- Numerical methods evolved with particular relevance to implementation on digital computers.
- All important numerical methods illustrated with computer algorithms written in an English based algorithmic language similar to PASCAL.
- A large number of solved examples are included.
- Solutions to selected problems at the end of the book.

V. Rajaraman, Ph.D. (Wisconsin) is Honorary Professor, Supercomputer Education & Research Centre, Indian Institute of Science, Bangalore. Earlier Professor Rajaraman taught at the Indian Institute of Technology Kanpur, from 1963 to 1982.

A pioneer in computer science education and research in India, Professor Rajaraman was awarded the prestigious Shanti Swarup Bhatnagar Award in 1976. He is also the recipient of the Homi Bhabha Award for Research in Applied Sciences, the U.P. Government National Award for Excellence in Teaching and Research and Rustom Choksy Award for excellence in engineering research. He is a fellow of the Indian Academy of Sciences, the Indian National Science Academy, and the Indian National Academy of Engineering. An author of several well established and highly successful computer books, Professor Rajaraman has published many research papers in reputed national and international journals.

Rs. 95.00

ISBN 81-203-0786-0

9 788120 307865

Prentice-Hall of India

New Delhi

www.phindia.com