Chapter 1

Basics in Computing

1.1 INTRODUCTION

We begin this chapter with some of the basic concepts of representation of numbers on computers and errors introduced during computation. Problem-solving using computers and the steps involved are also discussed in brief.

Many of the available digital computing systems fall mainly under four categories: personal computers, workstations, mainframe computers and super computers, based on their speed, cost and facilities. Mainframe and super computers being costly and are used only for research and development purposes. These computers involve large-scale programmes with huge data. In the new millennium, computers with storage capacities of several hundred billion words and capable of making 15 billion calculations a second are made available at select installations over the globe.

With the availability of such powerful digital computers and vastly improved numerical methods, scientists and engineers will be able to develop models that can be used for numerous purposes: weather prediction, effect of solar storms on Earth, the performance of an aircraft as a whole, some aspects of space flight simulations and many more practical problems meant for the welfare of the mankind.

In order to solve a given problem using a computer, the major steps involved are:

(i) Choosing an appropriate numerical method,

(ii) Designing an algorithm,

(iii) Programming and debugging, and

(iv) Computer execution.

These steps are briefly explained as follows.

We define the numerical method as a mathematical formula for finding the solution to a given problem. There may be many methods available to solve the same problem. For example, in Chapter 2, we shall present various computer- based numerical methods, such as, bisection method, regula-falsi method, method of iteration, Newton–Raphson method, Muller's method, Graeffe's root squaring method, etc., for solving an algebraic or transcendental

equation. One should choose an appropriate method, which suits best in the given situation, as a first step.

Once we chose a particular method for solving a problem, we should write down the sequence of steps to be followed in order, precisely and unambiguously, to obtain the solution. This is called *designing an algorithm.*

Now, the flow chart for the algorithm is drawn and then translated into a programming language, which we call *computer programme.* This programme should be debugged and free from coding errors. The choice of the languages is for the user to decide. It can be FORTRAN, BASIC, COBOL, Pascal, C, etc.

As a last step, the computer programme is fed to a personal computer or to a mainframe computer, with the necessary data through an input unit. Then, the central processing unit (CPU) of the computing system interprets the programme steps and executes them if the programme is free from coding errors. When it encounters output statement, the numerical answers to the problem are sent to the output unit chosen by the user; it may be a printer. This completes the problem-solving task using a computer.

1.2 REPRESENTATION OF NUMBERS

It is known that in our daily life, we use numbers based on the decimal system. In this system, we use ten symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and the number 10 is called the base of the system. Thus, when a base N is given, we need N different symbols 0, 1, 2, . . . , $(N-1)$ to represent an arbitrary number. The number systems commonly used in computers are shown as in Table 1.1.

Table 1.1 Number Systems

Base, N	Number
2	Binary
8	Octal
10	Decimal
16	Hexadecimal

Thus, if a number system has only two symbols 0 and 1, then, its base is 2 and so on. In general, an arbitrary real number, a can be written as

$$a = a_m N^m + a_{m-1} N^{m-1} + \ldots + a_1 N^1 + a_0 + a_{-1} N^{-1} + \ldots + a_{-m} N^{-m}$$

In binary system, it has the form,

$$a = a_m 2^m + a_{m-1} 2^{m-1} + \ldots + a_1 2^1 + a_0 + a_{-1} 2^{-1} + \ldots + a_{-m} 2^{-m}$$

In hexadecimal system, we write it as

$$a = a_m 16^m + a_{m-1} 16^{m-1} + \ldots + a_1 16^1 + a_0 + a_{-1} 16^{-1} + \ldots + a^{-m} 16^{-m}$$

For example, the value of the decimal number 1729 is represented and calculated as

$$(1729)_{10} = 1 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 9 \times 10^0$$

While the decimal equivalent of binary number 1.0011001 is

$$1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 0 \times 2^{-6} + 1 \times 2^{-7}$$

$$= 1 + \frac{1}{8} + \frac{1}{16} + \frac{1}{128} = (1.1953125)_{10}$$

Electronic computers use binary system whose base is 2. The two symbols used in this system are 0 and 1, which are called *binary digits* or simply *bits*. The internal representation of any data within a computer is in binary form. However, we prefer data input and output of numerical results in decimal system. Within the computer, the arithmetic is carried out in binary form. Infact, there is a built-in circuit design in every computer, which converts decimal input to binary and binary result to decimal output, and carry-out binary addition, subtraction, multiplication and division. For example, the method of converting decimal to binary equivalent can be seen as follows: we divide the given decimal number by 2 and the resulting successive quotients and continue to do so till the quotient becomes zero. Then, the binary equivalent of the decimal number is obtained as a string of remainders and the process can be seen through examples.

Similarly, for converting a decimal fraction into its binary equivalent, we multiply it by 2, the resultant integer part gives the most significant bit of the binary fraction. We keep multiplying by 2 and extract the next significant digit, and the process is continued until the fractional part becomes zero.

Example 1.1 Convert the decimal number 47 into its binary equivalent.

Solution

		Remainder
2	47	↓
2	23	1
2	11	1
2	5	1
2	2	1
2	1	0
	0	1 ⟵ Most significant bit

Thus,

$$(47)_{10} = (101111)_2$$

Example 1.2 Find the binary equivalent of the decimal fraction 0.7 ;25.

Solution

	Product	Integer part	
0.7625 × 2	1.5250	1	← Most significant digit
0.5250 × 2	1.0500	1	
0.05 × 2	0.1	0	
0.1 × 2	0.2	0	
0.2 × 2	0.4	0	
0.4 × 2	0.8	0	
0.8 × 2	1.6	1	
0.6 × 2	1.2	1	
0.2 × 2	0.4	0	Repeated hereafter

Therefore,

$$(0.7625)_{10} = (0.11000011(0011))_2$$

Suppose, we consider 8 bits only, that is, $(0.11000011)_2$; its decimal value is equal to $(0.7617187)_{10}$. While if we take 12 bits, its decimal equivalent is $(0.76245)_{10}$.

Example 1.3 Convert $(59)_{10}$ into binary and then into octal.

Solution

		Remainder	
2	59		
2	29	1	
2	14	1	
2	7	0	
2	3	1	
2	1	1	
	0	1	← Most significant bit

Thus,

$$(59)_{10} = (111011)_2$$

Now, if we group the binary digits such that each group contains three bits each, we can easily go from binary to octal. Thus,

$$(111011)_2 = 111\ 011 = (73)_8$$

1.2.1 Floating-point Representation

In general, two types of arithmetic operations are carried out in computers: integer arithmetic and floating point arithmetic. However, most scientific and engineering calculations are essentially carried out in floating point arithmetic. For example, an n digit floating point number in base b can be represented as

$$a = \pm(.d_1d_2, \ldots, d_n)_b b^e$$

where, $(.d_1d_2, \ldots, d_n)_b$ is called *mantissa* and e is the *exponent*. Let us consider a binary number as

$$.10110101 \times 2^{11} = .10110101E01011$$

Here, .10110101 is called the mantissa and 1011 is the exponent. The precision of floating point numbers on any computer is determined by the number of digits used in the mantissa, which in general, varies.

Computers having 48 bits to represent single precision real number, in general, allocate 1 bit for sign, 7 bits for exponent and 40 bits for mantissa. Thus, the range is limited from 2.939E−39 to 1.701E+38. The numerical precision in this case is 11 decimal digits. Similarly, those computers having 32 bits to represent a single precision real number, in general, allocate 1 bit for sign, 7 bits for exponent and 24 bits for mantissa. In this case, the limit ranges from 2.939E−39 to 1.701E+38. The numerical precision in this case is only six decimal digits.

When 64-bit double precision arithmetic is used, the computer output may be accurate even up to 16 decimal digits.

1.3 ERRORS IN COMPUTATIONS

Numerically, computed solutions are subject to certain errors. It may be fruitful to identify the error sources and their growth while classifying the errors in numerical computation. There are essentially three error sources: inherent errors, local round-off errors and local truncation errors.

1.3.1 Inherent Errors

It is that quantity of error which is present in the statement of the problem itself, before finding its solution. It arises due to the simplified assumptions made in the mathematical modelling of a problem. It can also arise when the data is obtained from certain physical measurements of the parameters of the problem.

1.3.2 Local Round-off Errors

Every computer has a finite word length, and therefore, it is possible to store only a fixed number of digits of a given input number. Since computers store information in binary form, storing an exact decimal number in its binary form into the computer memory gives an error. This error is computer-dependent. Also, at the end of computation of a particular problem, the final results in the computer, which is obviously in binary form, should be converted into decimal form — a form understandable to the user — before their print out. Therefore, an additional error is committed at this stage too. This error is called *local round-off error.*

For example, in Section 1.2, we have noted that

$$(0.7625)_{10} = (0.11000011\ (0011))_2$$

If a particular computer system has a word length of 12 bits only, then the decimal number 0.7625 is stored in the computer memory in binary form as

0.110000110011. However, it is equivalent to 0.76245. Thus, in storing the number 0.7625, we have committed an error equal to 0.00005, which is the round-off error; inherent with the computer system considered. Thus, we define the *error* as

$$\text{Error} = \text{True value} - \text{Computed value}$$

Now, in order to determine the accuracy of an approximate solution, errors are measured in different ways. *Absolute error*, denoted by |Error|, while, the *relative error* is defined as

$$\text{Relative error} = \frac{|\text{Error}|}{|\text{True value}|}$$

For example, consider the value of $\sqrt{2} = (1.414213\ldots)$ up to four decimal places, then

$$\sqrt{2} = 1.4142 + \text{Error}$$

Hence, we get

$$\text{Absolute error} = |\text{Error}| = 0.00001$$

$$\text{Relative error} = \frac{0.00001}{1.4142}$$

We are aware of the fact that $\sqrt{2}$ is irrational. However, widely used value up to four decimal digits is taken as the true value for the computation of relative error. These error measures are generally used in numerical analysis for measuring the accuracy of the results.

When a number N is written in floating point form with t digits, say, in base 10 as

$$N = (.d_1 d_2 \ldots d_t)10^e$$

we say that the number N has t significant digits. Here, d_1 is called the *most significant digit*. For example, 0.3 agrees with 1/3 to one significant digit, while 0.3333 agrees with 1/3 to four significant digits.

1.3.3 Local Truncation Error

It is generally easier to expand a function into a power series using Taylor series expansion and evaluate it by retaining the first few terms. For example, we may approximate the function $f(x) = \cos x$ by the series

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots + (-1)^n \frac{x^{2n}}{(2n)!} + \cdots \qquad (1.1)$$

In fact, it is an infinite series expansion. If we use only the first three terms to

compute cos x for a given x, we get an approximate answer. Here, the error is due to truncating the series. Suppose, we retain the first n terms, the *truncation error* (TE) is given by

$$\text{TE} \leq \frac{x^{2n+2}}{(2n+2)!} \tag{1.2}$$

It may be noted that the TE is independent of the computer used.

If we wish to compute cos x for $|x| < \pi/2$ accurate with five significant digits, the question is, how many terms in the expansion (1.1) are to be included? In this situation

$$\frac{x^{2n+2}}{(2n+2)!} < .5 \times 10^{-5} = 5 \times 10^{-6}$$

Taking logarithm on both sides, we get

$$(2n+2)\log x - \log[(2n+2)!] < \log_{10} 5 - 6\log_{10} 10 = 0.699 - 6 = -5.3$$

or

$$\log[(2n+2)!] - (2n+2)\log x > 5.3$$

We can observe that, for x in the interval $[-\pi/2, \pi/2]$, the above inequality is satisfied for $n = 7$. Hence, seven terms in the expansion (1.1) are required to get the value of cos x, with the prescribed accuracy, in the interval $[-\pi/2, \pi/2]$. In this example, the truncation error is given by

$$\text{TE} \leq \frac{x^{16}}{16!}$$

1.4 PROBLEM-SOLVING USING COMPUTERS

Nowadays, most of the students and teachers browse the internet to find literature on scientific topics which they want to understand better and to locate animations or simulations. If a student attempts to program a simulation from scratch, he may get bogged down with the intricacies of the syntax of the programming language. However, many tools for programming and visualisation are available on the Internet. For example, MATLAB, a software package for high performance numerical computation and visualisation, is one of the most widely used tools in Science and Engineering fields today. Its main attraction is its interactive environment, with many built-in functions for numerical computation, graphics, and animation. This software package makes the job of programming very simple, instead of writing in a language like C. MATLAB also provides visualisation tools for creating two-D and Three-D plots which enhance our understanding of various concepts. In fact, this package is freely available on the Internet under Windows environment. For illustration, consider a projectile which is thrown with initial velocity v at an angle 0, we want to find out—(i) the instance when the projectile hits the

ground, and (ii) the range and the trajectory of the projectile. This problem can be easily solved using computer simulation MATLAB (see Rudra Pratap, 2010).

While browsing the Internet, we also find many software packages for computing a variety of two-D, three-D scientific problems with graphing capabilities in areas such as Computational Fluid Dynamics, Structural Mechanics and several topics of current interest. Finally, it may be noted that learning to solve any scientific problem using only paper, pencil and a scientific calculator, does have its own value. However, combining it with computer simulations may enlarge the scope for better understanding of any project work in a real-life situation.